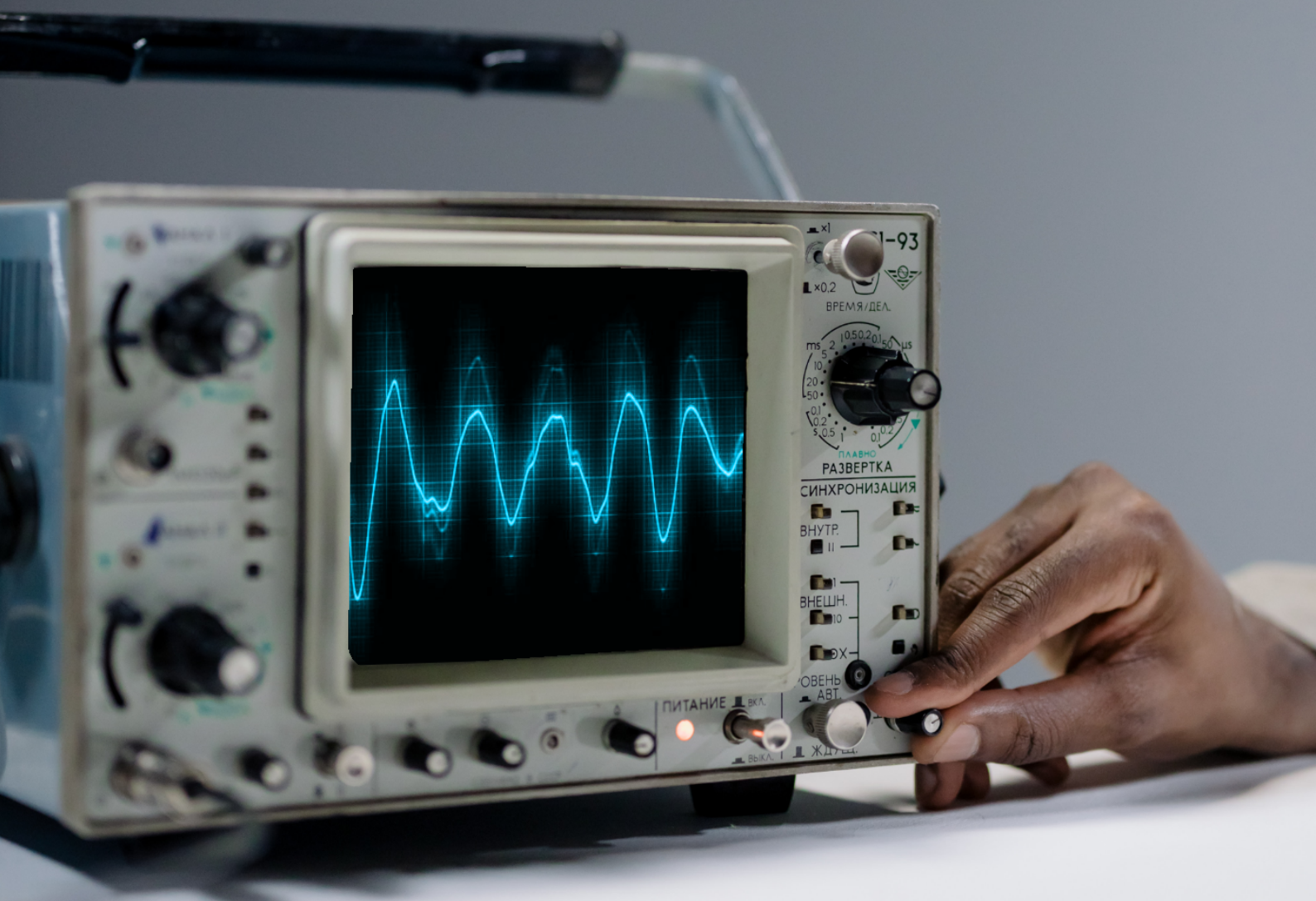
# Loss functions for profiled side-channel analysis

## An analysis of loss functions and the application of multi-loss functions for deep-learning in the SCA domain




TU Delft

Maikel Kerkhof

# Loss functions for profiled side-channel analysis

## An analysis of loss functions and the application of multi-loss functions for deep-learning in the SCA domain

by

M.M.H.A. Kerkhof

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 12th, 2021 at 14:00.


| | | |
|---|---|---|
| Student number: | 4297806 | |
| Project duration: | November 18, 2020 – August 12, 2021 | |
| Thesis committee: | Dr. S. Picek, | TU Delft, supervisor |
| | Prof. Dr. Ir. R.L. Lagendijk, | TU Delft |
| | Dr. E. Isufi, | TU Delft |


An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

# Abstract

Deep learning techniques have become the tool of choice for side-channel analysis. In recent years, neural networks like multi-layer perceptrons and convolutional neural networks have proven to be the most powerful instruments for performing side-channel analysis. Recent work on this topic has focused on different aspects of these techniques, either to improve the performance of the resulting models, reduce the complexity or find new well-performing architectures. A part of these neural networks that has received relatively little attention is the loss function. Loss functions play a key role in how these networks learn since each of the weights in the network is updated to minimise the loss calculated by the loss function. Work on loss functions in other fields where deep learning is used shows that the choice of loss function impacts the performance of the resulting models. While there are two novel functions proposed specifically for side-channel analysis, no broad analysis of the performance of different functions has been done in this context.

In this work, we provide such a broad comparison between different loss functions in the context of side-channel analysis and how they impact the performance of the resulting models. We show that novel, application-specific loss functions almost always outperform the current standard categorical cross-entropy. Besides that, we also show that state-of-the-art (multi-)loss functions from other domains can be successfully applied to side-channel analysis. Finally, we provide an overview of the strengths and weaknesses of the different loss functions in various side-channel analysis scenarios and use those to introduce our own novel loss function, the focal loss ratio. We show that this new loss function based on characteristics of other, well-performing loss functions, outperforms the previous best function in most SCA scenarios.

# Preface

This thesis is the final product of almost nine months of work. While I have mostly enjoyed working on the topic, I always imagined the master thesis differently than a project done completely at home. However, in the end, I am just happy it was possible to continue my studies from home without any further delays to my already long journey at the TU Delft. And after all, the extra projects, jobs, experiences and also parties throughout the years have been nothing but worth it.

Many people have directly or indirectly helped me throughout the course of this project. First of all, I would like to thank my supervisor Stjepan for his weekly support and guidance. Although we unfortunately have not met a single time in person since we started the thesis, I have never felt that it negatively impacted the process and I am very happy to have had you as my supervisor for this project. I also would like to thank Guilherme and Lichao, for providing me with a lot of helpful feedback in the final stages of writing this report. And last but certainly not least, I would like to thank Nicole, Marjolein, Jorn, Tim, Gijs, Emiel and Rowdy and others for providing the necessary moral support. Without all the coffee or cycling breaks, helpful feedback and endless conversations about just about anything but the thesis, finishing this project would have been a lot harder.

*Maikel Kerkhof*
*Delft, August 2021*

# Contents

# Chapter 1

# Introduction

In recent years, micro-controllers have found their way into an increasing number of aspects of our lives. Many of the things we interact with daily now contain some sort of micro-controller: from our phones and watches, smart cards such as our credit or ID card, to the routers and access points that provide our homes with Wi-Fi. For many devices, it holds true that we expect some form of confidentiality, integrity, and availability from them. For example, when we pay with a smart card, we want to be able to trust the transaction happens securely. To that end, many of those devices use some form of encryption to secure the processed data. The implementations of those cryptographic functions, however, are not always flawless.

To research the security of these devices and implementations a technique called side-channel analysis (SCA) is used. SCA is the act of analysing properties of the implementation of those cryptographic functions in an attempt to learn more about how it works and potentially what secret key is used. The implementation and nature of the processes used to create the micro-controller and cryptographic function sometimes cause the unintended leakage of information via a side-channel. This leakage can for example be in the form of a timing difference [31], power usage [30] or electromagnetic emanation [17] caused by the device when running a cryptographic function.

A subset of SCA is called profiled SCA. With profiled SCA, we assume that we have control over a profiling device, a device identical to the device we are targeting. This means that we can use this device to encrypt any data we want and know the key used for this encryption. With this knowledge, a profile can be build of the device. One approach to profiled SCA is the template attack [61], in which the relation between the noise in the measurements from the profiling device and the key is used to create a profile. Later, machine learning techniques were employed to perform SCA [2, 24, 33] against various implementations of AES.

More recently, the focus has shifted towards employing deep learning techniques for profiled SCA [4, 39, 40]. Deep learning architectures often have many hyperparameters and there is no straightforward way or a generic architecture that works best in every situation. The work in this context has therefore focused on improving certain aspects of the used techniques. For example on what optimisers work best for SCA [46], specific settings against countermeasures [7], the introduction of SCA specific metrics [60, 77] and attempts to create a methodology for finding suitable architectures [69, 75].

The problem we try to solve in this thesis is the relatively little attention loss functions have had in the context of SCA. The loss function is an important component of a deep learning model. It is used to calculate the error, or loss, between the output of a deep learning model and the true values that correspond to the given input. This function is then used to update the model with the aim of reducing this loss, i.e. it helps the model to learn the relation between the input and the expected output.

By comparing loss functions and analysing the impact they have on the performance in SCA, we aim to improve the tools that researchers have for performing SCA with deep learning. In other areas where deep learning has been used, such as face recognition, comparisons between various loss functions have already been made [26, 67, 72]. These works have shown that the choice of loss function influences the performance of the resulting model. Besides that, numerous proposals for application-specific loss functions have been made, often further improving the resulting deep learning models [3, 16, 19, 20, 36, 37, 57].

So far, no analysis of or comparison between commonly used loss functions in deep learning in

various SCA scenarios has been done. The difficulty of such a comparison in the context of SCA lies in the number of different settings there are. There are numerous datasets from different implementations and devices, leakage models, loss functions and architectures to compare. While two loss functions specifically for deep learning SCA have been proposed [74, 77], the comparison these works provide between these novel functions and commonly used loss functions is limited. Only a single architecture, leakage model or other loss function is considered.

To summarise, deep learning techniques have become the most powerful tools for SCA. By providing an analysis of loss functions in the context of SCA, we try to improve the tools researchers have for improving the security of cryptographic implementations. In this work, we will therefore analyse the influence of the loss function on the performance of deep learning models in various SCA scenarios. To do so, we will answer the following three research questions:

1. How do commonly used loss functions compare to novel, application-specific loss functions when deep learning is applied to side-channel attacks?
2. How do novel loss functions from other fields and multi-loss functions perform when applied to deep learning for side-channel analysis?
3. Can we construct a new loss function that improves the performance of deep learning models when used specifically for side-channel analysis?

The structure of this work is as follows. First, Chapter 2 provides the relevant background information required for this thesis. In Chapter 3, work related to this topic is discussed and our research questions are reformulated into sub-questions. Following that, we present our experiments and results on commonly used and novel loss functions in Chapter 4. In Chapter 5, we discuss our work on the application of (multi-)loss functions from other fields to SCA. The lessons learned from these two chapters are then used to construct a novel loss function for SCA in Chapter 6. Finally, we conclude our work by answering our three main research questions in Chapter 7 and providing insight into the limitations of this work and ideas for future work.

# Chapter 2

# Background

This chapter provides an introduction to the techniques and theory required for this thesis. First, an introduction on side-channel attacks (SCA) is given, focusing on profiled SCA. Then, concepts from the field of machine learning are introduced, such as deep-learning and loss functions. Finally, background information about the targeted cryptographic functions and the used datasets is given.

## 2.1 Side-Channel Attacks

Side-channel attacks (SCA) are attacks that use unintentional information leakages caused by the implementation of a cryptographic function to recover a secret key. These leakages could be in the form of timing differences [31], electromagnetic emanations [1] or power usage [30]. Finding and exploiting the correlation between the secret key and the observed side-channel can be done in several ways. In this section, we will discuss the different types of SCA, the leakage models used to model the relation between the secret key and the measurements and the metrics related to SCA.

### 2.1.1 Non-profiled Side-Channel Attacks

The first class of SCA are the non-profiled attacks. The scenario for these attacks is that for example the power consumption of a targeted device that is performing some cryptographic operation is measured directly. Kocher et al. first introduced such a method, called simple power analysis (SPA) [30]. The idea behind SPA is that observing the power usage of such a device directly reveals information about the performed operations. In turn, this information in combination with knowledge about the device could in some cases be used to break the implementation and retrieve the secret key.

More powerful non-profiled attacks have also been introduced. Kocher et al. also introduce a method called differential power analysis (DPA) [30]. For DPA, less or no additional knowledge is necessary, and only ciphertexts and power traces are required. The idea behind DPA is to create a differential trace based on the difference between the averages of two subsets of the available traces. These subsets are created using a selection function, which, if there is a correlation between the function and the targeted value, results in a differential trace that can reveal the key.

Another technique used to perform non-profiled attacks is called correlation power analysis (CPA) [6]. CPA uses a correlation factor between a Hamming distance-based leakage model and the available traces to test key hypotheses. Even more recently, non-profiled attacks based on deep learning have also been proposed [63]. This technique, which the authors named deep learning power analysis (DPLA), uses key hypotheses in a similar fashion. They calculate labels based on each hypothesis and train deep learning models with these traces and labels. They show that by looking at several metrics, they can discriminate between the correct key and other hypotheses, and train models which outperform CPA attacks.

### 2.1.2 Profiled Side-Channel Attacks

The second class of SCA are the profiled attacks. In a profiled attack, attackers use a second device that is identical to a targeted device they might have limited access to. Since the attackers have full control over this profiling device, they can use it to create a set of profiling traces and corresponding labels. These, in turn, can be used to create a model of the profiling device and attack the targeted device.

The template attack is the earliest example of such a profiled attack [61]. The profiling phase of the template attack consists of the attackers constructing templates for each of the possible key values, consisting of the mean signal and probability distributions of the noise. For each of these possible key values, a covariance matrix is then calculated. Next, in the attacking phase, one or more traces from the targeted device are used. For each of these traces, the probability that they originate from an operation with one of the possible key values is calculated using the constructed templates. The authors use their method to effectively attack RC4 and the DES cipher. Other works later improved upon the efficiency of template attacks [12] or demonstrated its potential against other implementations such as AES [8].

Profiled attacks based on machine learning techniques have also proven to be very successful. For this type of attack, SCA is seen as a classification problem, where a model is trained on a set of profiling traces and labels gathered from the profiling device. The trained model is then given traces gathered from the targeted device and a set of probabilities for each possible key value is predicted. Different types of machine learning techniques, such as support vector machines [2, 33] and random forest classifiers [33], have been shown to outperform template attacks while also requiring fewer profiling traces.

Deep learning techniques have also been used in profiled SCA. First explored by [40], multi-layer perceptrons (MLP) and convolutional neural networks (CNN) have been empirically shown to be even more powerful in specific SCA scenarios [7]. Picek et al. does mention that there are several potential drawbacks, such as the introduction of numerous hyperparameters which have to be tuned and the need to create new architectures for each different SCA scenario. Methodologies to find these architectures have been proposed, however [75], and there also seems to be potential in using architectures that have proven to work well in other domains [4, 27]. The work in this thesis will focus on profiled SCA using models based on MLP and CNN architectures.

### 2.1.3 Metrics

There are several metrics commonly used to measure the performance of machine learning models. Well-known examples are a models' accuracy, precision and recall and other such metrics based on the amount of true and false positives and negatives. Earlier work, however, has shown that such metrics do not work well to indicate the performance of deep learning models in the context of SCA [60, 77].

To be able to estimate, measure and compare the performance of deep learning models in the SCA domain, two metrics are commonly used: the guessing entropy and success rate. The output of the machine learning models is a list of probabilities for each possible key value. This output is sorted by probability from most likely to least likely key value and is called the key guess vector. Both the guessing entropy and success rate relate to the position or key rank of the correct key in this key guess vector.

#### Guessing Entropy

The partial guessing entropy (PGE) is calculated as the average key rank of the correct key byte in the key guess vector over a number of attacks. Commonly, partial guessing entropy is interchangeably used with the term guessing entropy (GE). In this work, we define both the GE and PGE to be the guessing entropy for a single key byte. In practice, the GE is calculated over 100 attacks and reported by plotting the GE versus the amount traces used in the attack [70, 74]. The amount of traces used is then increased and an attack can be considered successful if the GE reaches $1$, meaning that the average key rank over the number of attacks is $1$ and the correct key was chosen in each of the individual attacks. In this work, we will report the GE in a similar fashion. The PGE will be plotted

versus the amount of traces used in the attack and averaged over 100 attacks. When attacks are successful, i.e. reach a GE of 1, we will also report the amount of traces required, denoted as $\overline{N}_{T_{GE}}$.

#### Success Rate

The second metric often used for the evaluation of deep learning models in the SCA domain is the success rate (SR). The SR is calculated by choosing a low value such as 1, the order, and calculating for how many of the 100 attacks, the key rank of the correct key is equal or lower than the chosen value. If for example, an attack using $x$ traces is successful 20 out of 100 attacks, the SR for $x$ is 0.2. When reporting success rates, we plot the success rates versus the amount of traces used. We use the first-order success rate, meaning that for a single attack to be counted as successful, the key rank of the correct key has to be 1.

### 2.1.4 Leakage Models

The goal of SCA is to retrieve the key that is being used on the targeted device. To this end, we usually target an intermediate value in the process of encryption that is dependent on the key and either the plaintext or ciphertext. In the case of AES, further explained in Section 2.4, we target the intermediate value after the S-box operation is applied in the first round of AES. This value can be denoted as follows:

$$z_i = sbox(pt_i[j] \oplus k[j])$$

where $pt_i$ is the plaintext used during encryption, $sbox$ is the S-box substitution and $j$ is the key byte we want to attack. In this work, we use two so-called leakage models to model this value. The first one is the identity (ID) leakage model. When using the identity value, we directly use $z_i$ as a target. This results in 256 possible key byte values and classes. The second leakage model is the Hamming weight (HW) model. For the HW leakage model, we calculate the Hamming weight for each $z_i$ and use that as a label and target. The Hamming weight for a byte $z_i$ is defined as the number of bits set to 1 in the binary representation of the byte. For example the byte $A1 = 161$ in decimal representation and $A1 = 10100001$ in binary representation, so $HW(A1) = 3$. Using the HW leakage model results in 9 possible values and classes. The assumption behind the HW leakage model is that data leakage from a power trace is dependent on the number of bits that switch state [6].

## 2.2 Machine Learning

Machine learning is the process of letting an algorithm analyse different examples to try and create a model that approximates a certain process or task. By giving the algorithm more examples, a process called training, the model should learn and become more accurate in performing the task. Machine learning has been applied to many different fields. Some notable examples are playing various games [58], credit card fraud detection [38] or more cyber security related topics like penetration testing [9] or intrusion detection [64]. In general, three different approaches to machine learning exist. Learning can be done unsupervised, where a dataset without labels is used as training data. Here, learning is done by letting the algorithm process training data and letting it form a model based on the patterns present in the data.

Another approach is reinforcement learning. With reinforcement learning, an algorithm is faced with a certain scenario and has to make decisions or perform actions. Based on those actions, a reward or score is calculated. The idea is that the algorithm needs to learn to make decisions that maximise that score, getting better at making the right decisions and the underlying task such as playing a game [58].

The third approach is supervised learning, which is the approach we will consider in this work. In supervised learning, a training dataset consisting of the actual data and a label for each sample is used. Often, the task at hand is classification and the label is the class that the training sample belongs to. During the training phase, the model is given a sample as input and predicts a score or probability for each of the possible classes. The difference between the output and the actual class the sample belongs to is calculated via a loss function. This loss is then used to update the model and improve the predictions on the following inputs. The goal of this process is that after training, a generalised model is created that can also predict the correct class for inputs it has not seen during training.

### 2.2.1 Deep Learning

A subset of machine learning algorithms is called deep learning algorithms. Deep learning models consist of multiple layers of artificial neurons that are interconnected. Each of these connections has a weight, and the model learns by updating the weights based on the loss calculated on the output of the model. These techniques are called deep learning for the multiple layers and huge amounts of connections and therefore weights they contain. In recent years, these complex deep learning models have been shown to perform well in the context of SCA [4, 7, 27, 40, 43].

The building blocks of these networks are perceptrons: artificial neurons. Such a neuron takes an input $x_i$ from each of the neurons it is connected to in the previous layer. Each of these connections has a weight, $w_i$, which is multiplied with the input $x_i$. Some activation function $f$ is applied to the sum of these sets of weights and inputs and calculates an output $y$. So for a neuron with $n$ inputs, the output is given by Equation 2.1. For all but the last layer, the output of a neuron acts as the input to the neurons in the next layer.

$$y = f(\sum_{i=1}^{n} x_i w_i) \tag{2.1}$$

There are different activation functions commonly used in deep learning. When deep learning is applied in SCA, most often the Rectified Linear Unit (ReLU) [4, 43], Exponential Linear Unit (ELU) [47, 50], Scaled Exponential Linear Unit (SELU) [47, 50] or hyperbolic tangent (tanh) [4, 40] functions are used. The ReLU function is commonly used in deep learning and was shown to result in better performing models that trained faster than the previously used sigmoid activation function [18, 45].

$$relu(x) = max(0, x) \tag{ReLU}$$

The SELU activation function was introduced as an effort to create self-normalising neural networks [29]. The normalising refers to the activation outputs: the SELU function aims to output activations across different training samples which have a mean and variance between predefined intervals. The purpose of this normalisation is to allow for deeper networks, i.e. networks with more layers, to be trained by avoiding the vanishing or exploding gradient problem. For the parameters, the default values of $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$ are used.

$$selu(x) = \lambda \begin{cases} x & \text{if x > 0} \\ \alpha e^x - \alpha & \text{if x} \leq 0 \end{cases} \tag{SELU}$$

The ELU function is introduced by Clevert et al. to further improve the training speed and classification accuracy in comparison to ReLU [13]. According to Clevert et al., the possibility of negative outputs of the function improves the generalisation of the resulting model and propose to use $\alpha = 1.0$

$$elu(x) = \lambda \begin{cases} x & \text{if x > 0} \\ \alpha(e^x - 1) & \text{if x} \leq 0 \end{cases} \tag{ELU}$$

The tanh function maps the input to a value between $-1$ and $1$. Due to this, networks using the tanh activation do not suffer from gradients getting too large, i.e. exploding. However, when using deeper networks with more layers, the tanh activation function might suffer from vanishing gradients, i.e. gradients getting too small [22].

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{tanh}$$

Finally, the activation function of the last layer in the deep learning network is often different when classification problems are considered. Instead of any of the aforementioned functions, the softmax activation function is used. The softmax function transforms the outputs of the network, the vector **x**, to a probability distribution over the possible classes. This means that the resulting outputs of the model

are between 0 and 1 for each of the classes and the sum of the probabilities for all classes is 1.

$$softmax(\mathbf{x})_i = \frac{e_i^x}{\sum_{j=1}^{n} e_j^x} \text{ for } i = 1, .., n \quad \text{(Softmax)}$$

Besides the activation function, another part important part of a deep learning model is the optimiser. The activation function in each neuron is used to propagate the input through the layers of the model to the final output. The optimiser, in turn, is responsible for updating each of the weights based on the output of the model. Various algorithms to do this exist [56]. When SCA is considered, two optimisers are often considered: Adam, introduced by Kingma and Ba, [28, 43, 46] and RMSprop [4, 46]. In this work, we will treat the activation function and optimiser as configurable hyperparameters.

### Multi-layer Perceptron

A multi-layer perceptron (MLP) is a type of neural network that consists of an input layer, one or more hidden layers and an output layer. The number of neurons in the input layer is equal to the number of features in the data, while the output layer contains as many neurons as there are possible classes or outputs. The number of hidden layers, also called dense layers, is configurable as a hyperparameter. The same is true for the number of neurons in the dense layers. Figure 2.1 shows the basic structure of an MLP with an input with three features, a single hidden layer and four possible classes.

MLPs have been used successfully in the context of SCA. Simple MLPs with a single dense layer with 20 neurons have been able to break unprotected implementations of AES [40]. Besides those small models, deeper architectures with more layers have been used to successfully attack various protected implementations [4, 39].

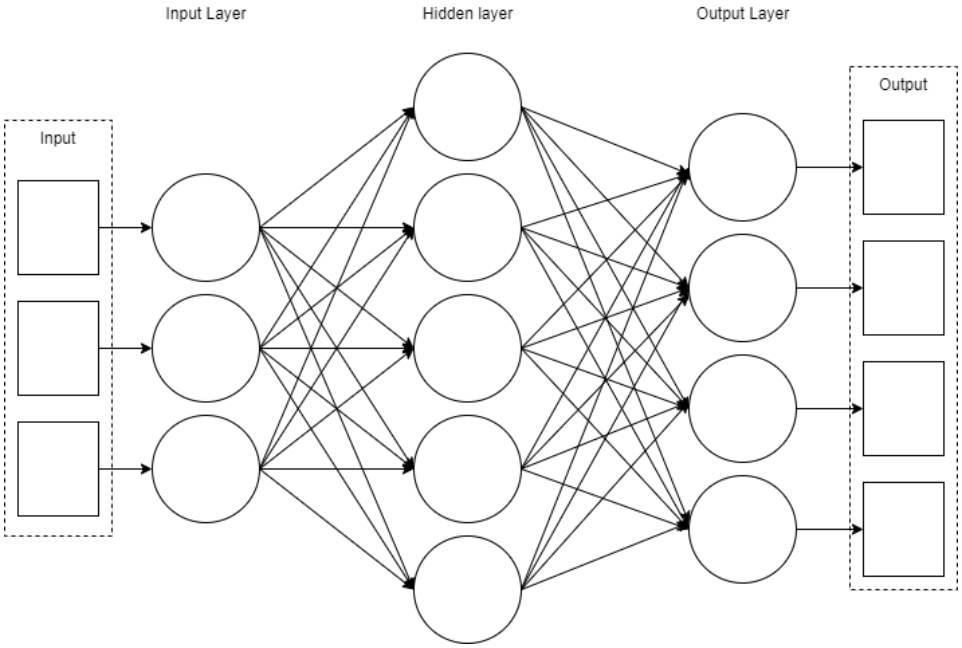



Figure 2.1: The structure of an MLP with an input with three features, a single hidden layer with five neurons and four possible classes.

### Convolutional Neural Network

Convolutional neural networks (CNNs) have gained popularity in various applications. They have proven to perform well in settings where the input data contains many features, such as in various computer vision applications [34, 35, 52]. CNNs work well in these cases with high-dimensional data because of their structure. Besides fully connected layers of neurons, CNNs also have different types of layers called convolutional and pooling layers. Unlike the layers in an MLP, convolutional layers

consist of filters. Similarly to the input weights of neurons, filters also contain sets of weights. Instead of being fully connected to other layers, filters convolve the input, i.e. they create a feature map from the features in the input. This is done by 'moving' the filters of size $k$, the kernel size, over the input data, multiplying the weights with the inputs and summing the respective values. The step size with which each filter moves over the input is called the stride. A lower stride, $s = 1$, is commonly used for the largest overlap between feature maps. A visual example of how input data is processed by a convolutional layer with a filter is shown in Figure 2.2. Similar to MLPs, a non-linear activation function is applied to the output of convolutional layers.

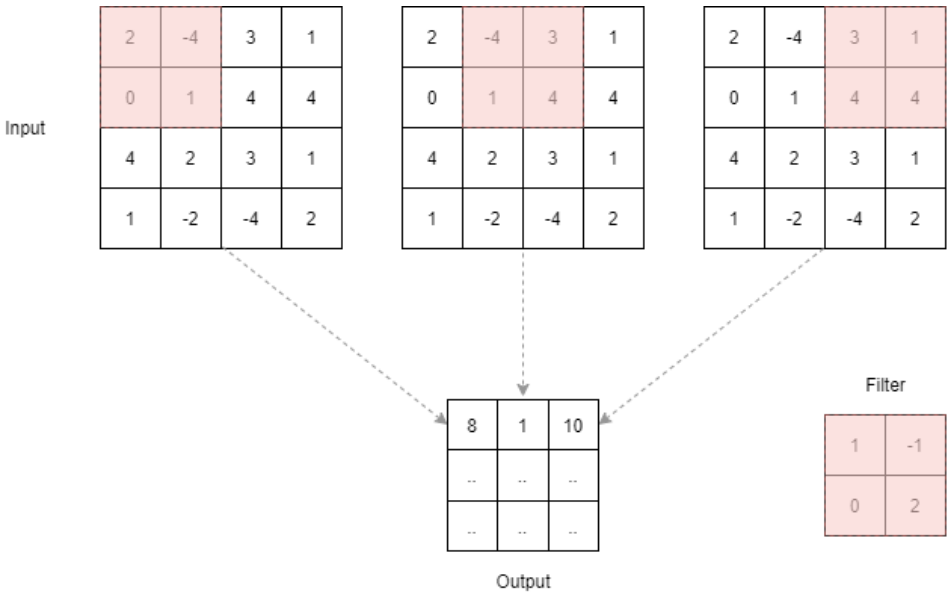



Figure 2.2: An example of how a convolutional layer calculates its output. The calculation of the output for the first three strides is shown. The kernel/filter size is 2 and the stride is equal to 1, meaning that with each stride, the filter is moved one place to the right.

Another part of CNNs is the pooling layers. Pooling layers reduce the output of convolutional layers further. Instead of containing weights, pooling layers combine their input by taking the maximum values of subsets of the data (max pooling) or the average value of these subsets (average pooling). Similarly to the filter in the convolutional layers, a sliding window with a pool size $k$ is moved with a stride $s$ over the input.

Finally, the last layers in a CNN are fully connected layers. These are the same as the layers of an MLP, with the number of neurons in the last layer is equal to the number of classes considered. Where the convolutional and pooling layers are used for feature extraction, e.g. distilling relevant features from the input data, the fully connected layers perform the classification part. An example of a CNN architecture with one convolutional block consisting of a convolutional layer and pooling layer, and one fully connected layer is shown in Figure 2.3.

CNNs have also been used successfully in the SCA domain. Their potential in this context was first shown by Maghrebi et al. with later work improving the design of CNN architectures and increasing their performance for SCA [4, 27, 40, 69, 75]. CNNs seem to do especially well in comparison with MLPs when the data has lots of features and samples [50] or if there is desynchronisation present [4].

### 2.2.2 Loss Functions

Both the MLPs and CNNs in this work are used in a supervised learning setting, which learns by training on a set of input and class labels. The difference between the output of the model and the actual class label belonging to the input is called the loss. This loss is calculated by a loss function. The optimiser tries to minimise this loss by using the derivative of the loss function to update the weights in the network. Loss functions are therefore a key part of how a neural network learns since they provide a target to be minimised. Many different loss functions exist, in this subsection, we will introduce the functions considered in this work.

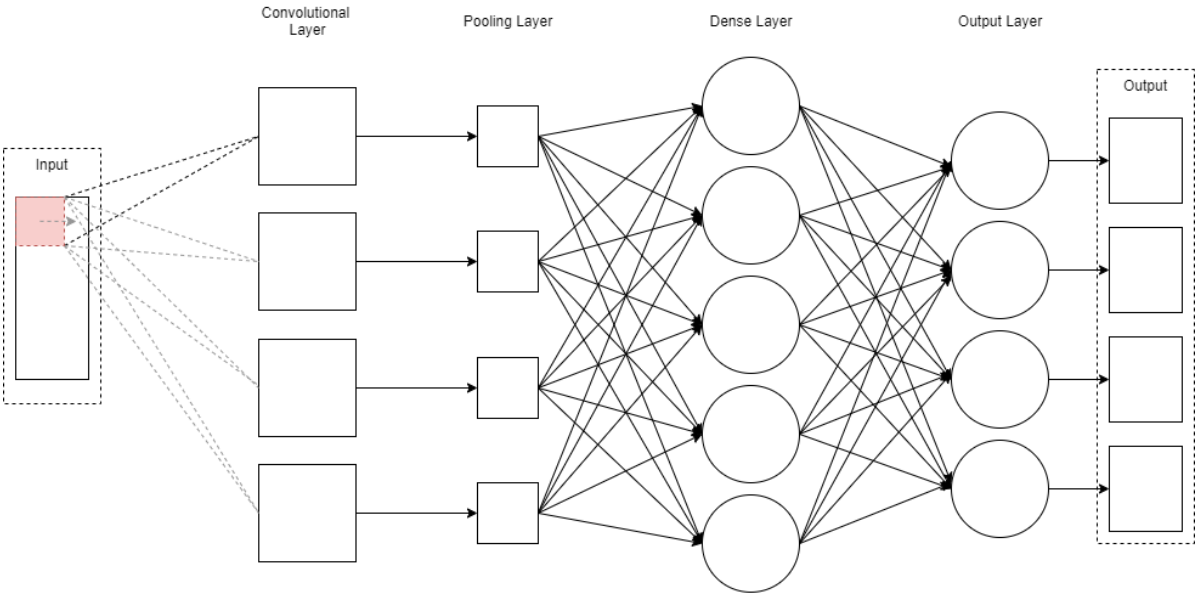



Figure 2.3: The structure of a CNN with a convolutional block consisting of one convolutional layer, one pooling layer, and a single hidden layer with five neurons and four possible classes. The red square is a single filter moving over the input with the stride marked by the grey arrow. The output of each filter is calculated by moving the filter over the complete input with a step size equal to the stride.

### Mean Squared Error

One of the simplest examples of a loss function is the mean squared error (MSE). The MSE is calculated by taking the mean of the pairwise squared differences between the elements of the prediction vector $\hat{\mathbf{y}}$ and the vector $\mathbf{y}$ with the true values.

$$mse(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{(MSE)}$$

The MSE and variations that also use the difference between the prediction and truth vectors have typically been used as loss functions when solving regression problems. The loss is calculated evenly for each of the samples, regardless of which class a sample belongs to. By minimising the loss, we minimise the distance between each of the output scores and the relevant true score. MSE is also usable for classification problems [26] and has been used in the context of SCA.

One variation of the MSE is the mean squared logarithmic error (MSLE). Instead of using the difference between de vectors directly, the MSLE is calculated by taking the difference of the natural logarithm applied to the true $\mathbf{y}$ and predicted $\hat{\mathbf{y}}$ values.

$$msle(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \qquad \text{(MSLE)}$$

The difference in practice is that MSLE is less sensitive to outliers in the data. When using MSE, a large prediction error on a single value can increase the overall loss substantially. With MSLE, this effect is less visible.

Finally, we also consider the logarithm of the hyperbolic cosine (log cosh) as a loss function. Log cosh loss, like MSLE, is also less sensitive to outliers [67].

$$log\_cosh(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\log(\cosh(\hat{y}_i - y_i)) \qquad \text{(Log cosh)}$$

### Classification Losses

The de facto standard loss function when doing classification is the categorical cross-entropy, sometimes also called the negative log-likelihood, softmax loss, log loss or just cross-entropy. It has been used in various classification tasks [23, 32, 73] and is also commonly used in SCA [4, 27, 40]. Cross-entropy is a measure of the difference between two distributions. When used as a loss function, the two underlying distributions are the predictions and the true classes of the samples. Minimising the cross-entropy, so the difference between the distribution modelled by the deep learning model and the true distribution of the classes should therefore improve the predictions of the neural network.

$$cce(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{CCE}$$

Another loss function used for classification is the (categorical) hinge loss. The hinge loss is designed to increase the margin between the predicted probability for the correct class and the predicted probability of the wrong class with the highest predicted probability.

$$cat\_hinge(\mathbf{y}, \hat{\mathbf{y}}) = max(1 - y_i * \hat{y}_i, 0) \tag{Cat Hinge}$$

### Novel Losses

More recently, two application-specific loss functions for SCA have been proposed. One of them is the ranking loss function proposed by Zaid et al.. The ranking losses uses both the output score of the model and the probabilities produced by applying the softmax activation function to these scores. The idea behind the ranking loss is to compare the rank of the correct key byte and the other key bytes in the score vector before the softmax function is applied.

$$rkl(\mathbf{s}) = \sum_{\substack{k \in \mathcal{K} \\ k \neq k^*}} \left( \log_2 \left( 1 + e^{-\alpha(s(k^*) - s(k))} \right) \right) \tag{Ranking loss}$$

where $\mathbf{s}$ is the vector with scores for each key hypothesis generated by processing the training samples by the model, $K$ is the set of all possible key values, $k*$ is the correct key and $s(k)$ is the score for key guess $k$, calculated by looking at the rank of $k$ in $\mathbf{k}$. Finally, $\alpha$ is a parameter that needs to be set dependent on the size of the used profiling set. The implementation of the ranking loss function is provided by Zaid et al. on Github [1].

The other novel loss function is the cross-entropy ratio (CER) loss [77]. Zhang et al. introduce the CER as a metric to estimate the performance of a deep learning model in the context of SCA. They also show that their metric can be used as a loss function directly by using a shuffled set of labels.

$$cer(\mathbf{y}, \hat{\mathbf{y}}) = \frac{CE(\mathbf{y}, \hat{\mathbf{y}})}{\frac{1}{n} \sum_{i=1}^{N} CE(\mathbf{y}_{r_i}, \hat{\mathbf{y}})} \tag{CER}$$

where CE is the categorical cross-entropy and $\mathbf{y}_{r_i}$ denotes the vector with the true probabilities ,1 for the correct class and 0 for all others, for each class but shuffled. The variable $N$ denotes the number of shuffled sets to use. Zhang et al. do not provide a value for $N$ but state that increasing $N$ should increase the accuracy of the metric. No comment is given on the value of $N$ in the CER loss function. As a starting point for our experiments, $N = 10$ is used. In Chapter 4, we will give more details about the influence of the value of $N$.

## 2.3 Datasets

For our experiments, we use several datasets commonly used in the SCA domain. These datasets contain power measurements from devices running an implementation of AES. Each dataset has different characteristics such as the number of features, traces and countermeasures. In this section, we

[1] https://github.com/gabzai/Ranking-Loss-SCA

briefly describe each of the datasets used in this work. Table 2.1 gives an overview of all the datasets and their relevant details.

### ASCAD

The first dataset considered is the ASCAD dataset introduced by Benadjila et al.[4]. The ASCAD dataset is generated by taking measurements from an ATMega8515 running masked AES-128, and is proposed as a benchmark dataset for SCA. The dataset consists of 50.000 profiling traces and 10.000 attack traces, each trace consisting of 700 features. The profiling and attacking set both use the same, fixed key. We will denote this dataset as ASCAD_fixed. The dataset is provided on the ASCAD GitHub repository [2].

The second dataset used in this thesis is also part of ASCAD, but uses variable keys. The ASCAD_variable dataset consists of 200.000 profiling and 100.000 attack traces, each consisting of 1.400 features. These traces are not synchronised and the keys used in the profiling set are variable. For our experiments, we use the first 50.000 traces from the profiling dataset to train our models. The ASCAD_variable dataset is available on the ASCAD GitHub repository [3].

As mentioned, the ASCAD dataset is constructed of measurements from a masked implementation of AES. The masks used during encryption are also included in the dataset and can be used to effectively remove the mask. By doing this, we create a new dataset that is similar to measurements taken from an unprotected implementation of AES. For our new dataset, denoted ASCAD_plain, the targeted sensitive value will then become:

$$z_i = sbox(pt_i[3] \oplus k[3]) \oplus m_{out_i}$$

where $m_{out_i}$ is the known mask for the targeted key byte.

Furthermore, we also use a variant of the ASCAD_fixed dataset but with a random desynchronisation, i.e. the features of the traces are randomly shifted up to 50 places [4].

### CHES_CTF

The second dataset is the CHES_CTF dataset [55]. The CHES_CTF dataset is generated by taking measurements from a masked AES-128 implementation and consists of 45.000 profiling and 5.000 attack traces, each consisting of 2.200 features. The profiling and attack set both have a different, but fixed key. The used dataset is provided by the TU Delft AISYLab [5].

Table 2.1: Datasets considered in this work and their relevant details. For each dataset, the number of features and profiling and attack traces, and the key settings and countermeasures are listed.

| Dataset | # Features | # Profiling | # Attack | Keys | Countermeasures |
|---|---|---|---|---|---|
| ASCAD_fixed | 700 | 50.000 | 10.000 | Same and fixed | Masking |
| ASCAD_variable | 1400 | 200.000 | 100.000 | Random | Masking |
| ASCAD_plain | 700 | 50.000 | 10.000 | Same and fixed | None |
| ASCAD_desync50 | 700 | 50.000 | 10.000 | Same and fixed | Masking and desync |
| CHES_CTF | 2200 | 45.000 | 5000 | Fixed but different | Masking |

## 2.4 Advanced Encryption Standard

The direct goal of side-channel analysis is to retrieve information or (a part of) the secret key used during some cryptographic operation. Implementations of various cryptographic functions have been shown to be vulnerable for side-channel analysis. The first timing attacks targeted RSA [31], while the

[2] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key
[3] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key
[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key
[5] http://aisylabdatasets.ewi.tudelft.nl/

later introduced DPA targeted DES [30]. Later on, template attacks were used as tools to attack many different cryptographic functions, such as RC4, DES and RSA [61].

In most of the recent work on SCA, the targeted implementation is that of the Advanced Encryption Standard (AES) [4, 40, 43, 46, 50]. AES is a commonly used symmetric block cipher with three different variations using either 128, 192 or 256-bit keys [15]. Symmetric ciphers use the same key for both encryption and decryption. AES is a block cipher since it encrypts chunks of data in a block-wise manner. These blocks of data, often with a size of 128 bits, are processed by performing the same operations for a number of rounds. The number of rounds is 10, 12 or 14 for the 128, 192 and 256-bit key variants respectively. The operations performed by the AES encryption algorithm are as follows.

1. **KeyExpansion** - Round keys for each of the rounds are derived from the secret key following the AES key schedule.
2. **AddRoundKey** - The XOR operation is applied between the first round key and the plaintext.
3. **Intermediate Rounds** - For 9, 11 or 13 rounds depending on the key bit size, the following operations are performed.
   (a) **SubBytes** - The AES S-box, a substitution table, is used to substitute each value of the state with another value. Daemen and Rijmen use the term state to define the intermediate values denoted as a matrix.
   (b) **ShiftRows** - The bottom three rows of the state are shifted according to a predefined value dependent on the used block size.
   (c) **MixColumns** - The columns of the state are mixed by multiplying them with a predefined polynomial.
   (d) **AddRoundKey** - The XOR operation is applied between the state and the corresponding round key.
4. **Last Round** - For the last round, the following operations are applied.
   (a) **SubBytes**
   (b) **ShiftRows**
   (c) **MixColumns**

AES has proven to be a strong cipher. The best-known attack on the 128-bit key variant still has a computational complexity of $2^{126.1}$, down from $2^{128}$ for a brute force attack [5]. This is still too complex to pose a real threat to the security of AES. The only feasible attacks remain side-channel attacks against implementations of AES. All of the datasets we use in this work, as introduced in Section 2.3, contain measurements taken from an implementation of AES-128, with a key length of 128 bits. The key byte that we target in this work and later in this work denoted just by 'key', is the third key byte of the first round key. To this end, we use the intermediate value after the first application of the SubBytes operation, i.e. the S-box. This intermediate value is commonly used for SCA since the non-linear S-box operation has shown to be more useful when discriminating between the correct and incorrect key guesses [33, 51].

# Chapter 3

# Related Work

Much work has been done in the field of deep learning in recent years. Many new applications have been proposed and researched [21] and frameworks such as Tensorflow [42] and Keras [11] enable rapid creation and testing of new solutions. After the successful application of machine learning techniques to the field of side-channel attacks [24] [2] [33], attention shifted towards deep learning [40] [7]. Some of this work focuses on the ability of deep learning models to deal with countermeasures [27] [7], other work goes more in detail about optimising core deep learning characteristics, like which architecture works best for side-channel attacks [50], hyper-parameter optimisation [70] and evaluation metrics [71]. Even more recently, several novel loss functions specifically for the application of deep learning to side-channel attacks were proposed [74] [77]. In this section, related work on these topics is discussed together with the place our work has among the already existing research.

## 3.1 Deep Learning for Side-Channel Attacks

Deep learning, in general, has found its way into many fields of study, the domain of side-channel attacks being one of them. Different approaches have been tried, for example by using multi-layer perceptrons (MLP) [4, 40, 70], convolutional neural networks (CNN) [27, 40, 70, 75] or auto-encoders [40]. Maghrebi et al. first studied the potential of these techniques by comparing them with the template attack [61] and machine learning approaches like support vector machines (SVM) and random forest (RF) models [40]. In their experiments, they used these different models to attack masked and unmasked implementations of AES, and compare the guessing entropy that the models achieved. Besides using a CNN and MLP, they also use a stacked auto-encoder, a long-short term memory model as other examples of deep learning techniques. This work is a good basis for further research: it clearly shows that deep learning techniques have the potential to be successful solutions for side-channel attacks. Although the used MLP architecture consists of only a single hidden layer, both the CNN and MLP show a consistently good performance, on both the masked and unmasked datasets. However, in relation to our own research question, little information is given about the used loss functions. The authors state that the mean squared error (MSE) or negative log-likelihood (NLL) are often used, though they never specify which function they use.

The work of Cagli et al. showed very promising results for the performance of convolutional neural networks in comparison with template attacks, specifically when random delay, or jitter, countermeasures are considered [7]. However, they did not compare their CNN with other machine learning techniques. Picek et al. fill this gap and provide such a comparison [49]. Their results show that, while considered a very powerful technique in general, CNNs are not always the optimal choice when side-channel attacks are considered. Interestingly, the other approaches seem to outperform the CNN in some of the tested scenarios, while being less computationally intensive. Even in a scenario where a CNN should perform well such as with the random delay dataset, according to Cagli et al., a simpler method like Naive Bayes seems to perform better. The authors do note that there might still be scenarios where using a CNN would be beneficial, such as when large datasets are considered, or when a dataset with a masking countermeasure is considered, and they propose that future work should be done on this aspect. Furthermore, while many of the hyperparameters are tuned for the MLP and CNN,

only the categorical cross-entropy (CCE) is mentioned as a loss function. It might therefore be valuable to continue on this, and see how CNNs perform when different loss functions are involved.

Picek et al. set out to compare the performance of convolutional neural networks to other machine learning techniques when considering side-channel attacks [50]. The methods they use are Naive Bayes, which showed promise in their earlier work [49], XGBoost, a well-known gradient boosting algorithm [10], and a Random Forest model. They compare these methods with deep learning techniques, namely a CNN architecture optimised for the DPAv4 dataset and a multi-layer perceptron. For each of these methods, the relevant parameters are tuned with random search. The experiments involve training and testing the different models on three different datasets: the DPAv2, DPAv4 and a dataset consisting of traces from a software implementation of AES, protected with the random delay countermeasure. They compare the resulting accuracy, guessing entropy and success rate [50]. In their work, they conclude that complex CNNs are not required for smaller datasets, but do show advantages in cases where the number of features and traces is high. However, they state that more experiments in this direction are necessary to properly assess the performance of CNNs for SCA.

The work done by Benadjila et al. further reinforces the idea that MLP and CNN architectures are capable deep learning techniques when applied to side-channel attacks [4]. The goal of their research is to solve two main limitations they see in previous works, namely the lack of details given on hyperparameters and their optimisation, and following from that, the reproducibility of those works. They provide a step-by-step explanation of how they tuned the different hyperparameters for their CNN and MLP architectures, and compare their best-found models for both types of architectures with a template attack and the VGG-16 architecture, a 16-layer CNN created for image recognition [59]. The four different approaches are used to attack an AES implementation with a masking countermeasure and different levels of desynchronisation. Their results show that both the CNN and MLP architectures are highly effective against synchronised traces and that the CNN performance is less sensitive to desynchronisation than that of the MLP. Although the process of optimising the different hyperparameters is described extensively, only a single loss function is tested. The authors acknowledge this, and state that using other functions could lead to different performance and may be investigated in the future. Besides describing their process of hyperparameter optimisation, Benadjila et al. also introduce a new format to store side-channel traces in, and datasets in that format [1]. To enable a better comparison to previous work on this topic, the ASCAD database will also be used in this work.

## 3.2 Loss Functions

As explained in section 2.2.2, loss functions play an important role in the way deep learning algorithms learn. Loss functions have therefore been studied extensively in different scenarios and for different applications.

Janocha and Czarnecki investigate in their work how the choice for certain loss functions affect deep learning models and their learning dynamics, and the robustness of the resulting classifiers [26]. They compare twelve different loss functions by training models using these functions on several toy datasets, the MNIST database[2] and the CIFAR-10 dataset [3]. The resulting models are compared on accuracy and the time it took to train them, and based on those results, the researchers argue that other losses might be preferable over the log loss. The loss functions that they categorise as expectation losses, like the L1 and L2 losses and other loss functions derived from them, seem to perform well when a lot of noise is involved. They also suggest loss functions developed specifically for an application might perform better than these classical loss functions. This is the premise for our work: a comparison of several aspects between classical and application-specific, novel loss functions.

Such a comparison has already been done for other specific applications of deep learning. In their work, Yash Srivastava et al. [72] perform a comparison between more classical loss functions like the cross-entropy and marginal loss, and loss functions specifically developed for face recognition, such as angular-softmax loss [66] and ArcFace loss [16]. They use two different CNN architectures and train these architectures with several combinations of datasets and loss functions, and test the resulting models on a third dataset. Their results show that the application-specific loss functions outperform the

[1] The ASCAD Database `https://github.com/ANSSI-FR/ASCAD`
[2] MNIST Database `http://yann.lecun.com/exdb/mnist/`
[3] CIFAR-10 Dataset `https://www.cs.toronto.edu/~kriz/cifar.html`

classical loss functions in terms of accuracy and rate of convergence. A similar setup of experiments, comparing the influence of loss functions on different architectures and datasets, can be considered for this work.

To the author's knowledge, such a broad comparison of the influence of loss functions has not been researched yet for the application of deep learning to side-channel analysis. However, several application-specific loss functions have been proposed.

Zhang et al. introduce a new metric for evaluating deep learning-based side-channel attacks called the cross-entropy ratio (CER), and a new loss function based on this metric [77]. They argue that the calculation of typical side-channel attack metrics like guessing entropy and success rate is computationally expensive, and therefore not suitable for direct implementation into a deep learning algorithm. Therefore, the new CER metric is proposed, which is a ratio between the cross-entropy calculated over the correct key, and the expected value of the cross-entropy over all other key hypotheses. They compare their loss function based on this metric with on an MLP and a CNN architecture, on three different datasets, namely ASCAD, AES_HD and AES_RD. Their results seem to show that their loss function outperforms the cross-entropy loss in every single scenario. There are, however, some remarks to be placed by these results. First of all, Zaid et al. also performed a comparison between cross-entropy and the CER loss for the ASCAD and AES_HD datasets. Their results differ from what Zhang et al. showed, and in their work, the models using cross-entropy loss function often perform better, needing fewer traces than the ones trained with CER loss to converge to a guessing entropy of 1. The authors also claim that the CER loss works well for the synchronised traces in the ASCAD database, showing graphs of the guessing entropy converging to 1 faster than when cross-entropy loss is used. However, in the work of Zaid et al., the model trained with CER loss does not even converge when the 100ms desynchronised set is considered. This clearly shows that a more in-depth analysis of these loss functions is necessary.

In their work, Zaid et al. also propose a loss function specifically for deep learning applied to side-channel analysis called ranking loss [74]. The ranking loss function aims to minimise the rank of the correct key among all other key hypotheses, thereby maximising the success rate. This is achieved by a pairwise comparison between the rank of the correct key and all the other key guesses, and increasing the loss value if the relative ordering of the pair is incorrect. Furthermore, the authors provide a proof from an information-theoretical perspective as to why the ranking loss should be more effective and should require fewer traces than the typically used cross-entropy loss. A downside to the RankingLoss approach is the introduction of the parameter $\alpha$. The authors consider this parameter as a learning rate and provide some details of how they tuned it for different scenarios and datasets. But they do state that the $\alpha$ parameter should be carefully tuned for each different scenario, which introduces extra work when using this function. The ranking loss, as a recently introduced loss function specifically for side-channel attacks, shows promising results, and will therefore also be considered in this work.

## 3.3 Multi-loss Functions

Like using ensembles of models to gain better performance than single deep learning models [47, 65], previous work has also looked into combinations of various loss functions [3, 19, 20] or adapting existing loss functions [14, 36]. In various settings, these multi-loss functions have improved the performance of the resulting models [19, 20, 57] .

First of all, Lin et al. introduced the focal loss [36]. Focal loss is an adaptation of the categorical cross-entropy introduced to work well for object detection in images. By nature, the object detection problem deals with imbalanced data: a small amount of pixels represents the interesting object, while the majority of the pixels is considered to be less interesting background samples. Focal loss introduces two additional parameters to the categorical cross-entropy, $\alpha$ and $\gamma$. The $\alpha$ parameter is a vector of weights used to balance the influence of samples from each of the classes. By giving classes with large numbers of samples a smaller weight and rarely occurring classes a higher weight, their contribution to the total loss is balanced. The $\gamma$ parameter is used to give easy examples, i.e. correctly classified with a high probability, less influence on the loss in comparison to hard examples. When $p$ gets larger, so when a sample is correctly classified with a larger probability, the contribution to the loss gets significantly smaller due to the $\gamma$. In their work, they show that they can perform similarly or better than the previous state of the art while using a less complex neural network. The focal loss performs especially well on

imbalanced datasets. Lin et al. also show that alternate forms of the focal loss work similarly well and suggest that similar adaptations to other loss functions might also work well.

In other applications of deep learning, several works show promising results by combining two or more loss functions to increase the performance of the resulting deep learning models [3, 19, 20, 57]. First of all, Barz and Denzler use the cosine similarity and a combination of the cosine similarity and categorical cross-entropy as loss functions. They show that the combination of those functions leads to models that outperform either of the loss functions separately on several image recognition datasets [3]. The increase in performance is largest when there are less than 200 samples per class. In the context of SCA, this also regularly occurs when considering the ID leakage model. The larger amount of 256 classes and for example 50.000 profiling traces uniformly distributed over the classes gives an average of 195 samples per class. To see if this performance increase noted by Barz and Denzler is also possible with SCA datasets, we will consider the combination of cosine similarity and categorical cross-entropy in our work.

Finally, Hajiabadi et al. use a combination of three loss functions for text classification and breast cancer prediction [19, 20]. They state that loss functions have different properties, such as increasing the margin between the correct class and other classes and robustness against outliers in the data. They identified three loss functions that have such specific characteristics. First, they use multi-class hinge loss, which is identical to the categorical hinge loss described in Subsection 2.2.2. Hinge losses also result in a (small) loss for correctly classified examples, thereby increasing the margin between the correct class and other classes. The second loss function they use is correntropy [68]. Correntropy bounds the loss between 0 and 1 for each sample, thereby making it more robust to outliers. The third function used is the categorical cross-entropy, as introduced in Subsection 2.2.2. [20] use a linear combination of these three functions, and use a training process to assign each of the functions a weight. Since this multi-loss function has already been shown to improve the performance in two other domains, we will use it in our comparison.

## 3.4 Our contribution

As is apparent from the discussed research, different aspects of the application of deep learning to side-channel attacks have been studied extensively. However, it seems that loss functions in the context of side-channel analysis have not been the sole focus of research yet. Our contribution will be such an analysis of loss functions in the context of side-channel analysis. The research discussed in this chapter allows us to expand our research questions into sub-questions and reformulate them as follows:

1. How do commonly used loss functions compare to novel, application-specific loss functions when deep learning is applied to side-channel attacks?
    - How does the choice of loss function impact the performance of a side-channel attack in terms of guessing entropy and success rate with different datasets and leakage models?
    - What is the influence of loss functions when different architectures like multi-layer perceptrons and convolutional neural networks are considered?
    - How does the choice of loss function impact the training time?
    - How does the choice of loss function impact the performance when countermeasures are involved?

Furthermore, we will also look into the application of novel (multi-)loss functions from other domains to the topic of SCA. Our second research question and corresponding sub-questions are:

2. How do novel loss functions from other fields and multi-loss functions perform when applied to deep learning for side-channel analysis?
    - Are there similarities between SCA data and data from other domains where novel loss functions improve the performance?
    - How do models trained with these loss functions perform in terms of guessing entropy and success rate?

- Do these loss functions provide a viable alternative to commonly used loss functions and SCA specific loss functions?

Finally, with the knowledge gained by answering the first two questions, we will define the relevant components of well-performing loss functions and try to create our own loss function. The aim of this new loss function is to provide a function that can be used in various SCA scenarios and improves the performance in comparison with the previously best function. Our third and final research question is:

3. Can we construct a new loss function that improves the performance of deep learning models when used specifically for side-channel analysis?

    - What are the characteristics of previously best loss functions that make them perform well in the context of SCA?
    - How can we use these characteristics to create a new loss function?

# Chapter 4

# Comparing Loss Functions

The choice of loss function used in training can influence the performance of the resulting deep learning model [26, 72]. In this section, we compare commonly used loss functions with novel application-specific loss functions in the context of side-channel analysis. We use several approaches for this comparison, for each approach we compare the guessing entropy and success rate, the number of trainable parameters and the required training time. First, we define and select a median model in terms of performance and use that to compare the different loss functions. Next, we optimise models for each specific loss function and scenario. Finally, several state-of-the-art architectures that are trained with different loss functions are compared.

## 4.1 Motivation

Loss functions play a central role in training a deep learning model. They are used to calculate the error, or loss, between the actual output and the desired output. The resulting loss is used to learn, i.e. update the weights associated with the connections between the neurons or filters of the deep learning network. Earlier work where deep learning is used for other applications such as face recognition and image classification has shown that the choice of loss function influences the performance of the resulting model [26, 72].

In recent years, the usage of deep learning has become more popular in the context of SCA [7, 27, 40, 43, 50, 63, 75]. Many of these works focus on improving certain aspects of the used MLP or CNN architectures. The goal of those improvements is to increase the performance of the model, by decreasing the amount of traces required to reach a guessing entropy of $1$ for the correct key. However, all of these recent works seem to have in common that no considerations about the used loss function are made. When they first explored the usage of deep learning techniques for SCA, Maghrebi et al. mentioned that categorical cross-entropy or the mean squared error are commonly used loss functions. Later work on deep learning for SCA seems to exclusively use either categorical cross-entropy [4, 47, 75] or mean squared error [44, 63]. Indeed, in [67], the authors show that minimising the categorical cross-entropy loss is equivalent to increasing the Perceived Information (PI) [53], a metric commonly used in the context of SCA.

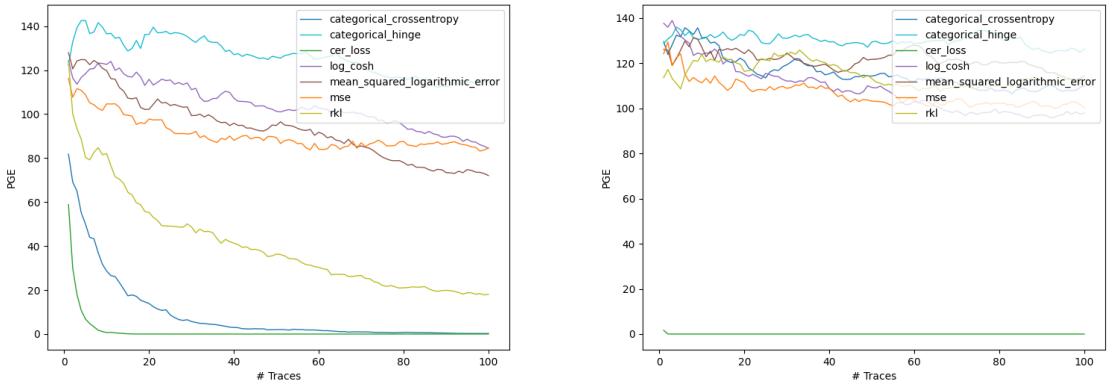
More recently, two novel loss functions specifically for usage in the context of SCA have been proposed. Zaid et al. propose ranking loss (RKL), a loss function that uses a pairwise comparison between the possible different key hypotheses to maximise the models' success rate. Zhang et al. propose the cross-entropy ratio (CER), which is the ratio between the categorical cross-entropy of the original profiling traces and a set of profiling traces with shuffled labels. The CER loss function should, according to the authors, be better suited for imbalanced profiling data [77].

In both of these papers, the newly proposed loss functions are compared to the categorical cross-entropy. However, the extent of these comparisons is limited, and only a single architecture or leakage model is tested. To the best of the author's knowledge, no broad comparison has been done between several commonly used loss functions such as categorical cross-entropy, mean squared error or hinge loss and these novel SCA-based loss functions on different architectures, leakage models and datasets. In this work, we perform such a broad comparison between commonly used loss functions and the novel

CER and RKL. We do so by testing the loss functions on various architectures, leakage models and datasets.

### 4.1.1 Median Model & Hyperparameter Optimisation

The training process of a deep learning model is influenced by several different hyperparameters. These hyperparameters are for example the number of layers and neurons per layer, the activation function each neuron uses and the loss function. Using CNNs introduces even more hyperparameters, such as the number of convolutional blocks and filters used. By picking a single random model, there is a possibility that we end up with certain hyperparameters that influence one loss function more than they do others. An example of this can be seen in Figure 4.1. In this scenario, each model is trained with the same hyperparameters, except for the loss function and learning rate. When the learning rate is set to $0.00001$ (Figure 4.1a), most of the loss functions seem to be converging towards a GE of $1$. However, when the learning rate is increased to $0.001$ (Figure 4.1b), none of them converge except the CER loss, for which the performance is actually increased.




(a) Learning rate 0.00001

(b) Learning rate 0.001

Figure 4.1: All models are trained with the same hyperparameters, except the learning rate. It is clear that the learning rate influences the performance of models with some losses more than others. In the scenario with the learning rate set to $0.01$, the performance of the CER loss is increased while the other losses fail to result in a model converging to a GE of $1$.

To reduce this possible effect of certain combinations of loss function and other hyperparameters, we use a median model to do our comparison. As defined in 4.1, we select such a model for each scenario by generating 100 random models and select the median performing model in terms of guessing entropy. We choose the median model to get a more representative model to compare the functions on and reduce the influence of other hyperparameters on the performance. We then use this model to compare the different loss functions.

**Definition 4.1. Median model**: A model selected from a set of $N$ models such that its performance is the median GE using all of the attack traces of the final guessing entropy of the $N$ models.

Certain hyperparameter values that are very different from the median model might lead to well-performing models when a specific loss function is used. To compare the loss functions when the other hyperparameters are optimised, we also perform hyperparameter optimisation via random search. We perform a random search for each scenario with each of the loss functions fixed. The exact setup of these experiments is described in Subsection 4.2.1.

### 4.1.2 Countermeasures & State-of-the-art Architectures

To protect hardware implementations from side-channel attacks, countermeasures are commonly used. These countermeasures are often a form of hiding or masking, which aims to hide or mask the statistical

relation between the intermediate value and the used key. This can be done by adding noise in the case of the hiding countermeasure or using an additional secret value in the case of masking [62]. To compare the robustness of models trained with different loss functions against several countermeasures, we test these models on datasets with and without these countermeasures.

Besides focusing on the resistance of different loss functions to countermeasures, several recent papers have proposed different MLP and CNN architectures that perform well on the considered dataset and leakage model [4, 75]. In those works, only the categorical cross-entropy is considered as a loss function. The performance of these models with other loss functions is not tested. These architectures might only work with the categorical cross-entropy, provide good performing models regardless of the chosen loss functions or perform even better with a different loss function. We therefore train models based on these state-of-the-art architectures with several classical and novel loss functions and compare their performance.

## 4.2 Experiment Setup

In this section, we describe the setup of the experiments we have performed to answer our research question. All the experiments have been performed on Nvidia GeForce GTX 1080Ti GPUs with 11GB of memory, part of the TU Delft HPC cluster [1].

### 4.2.1 Median Model & Hyperparameter Optimisation

To perform a broad comparison between the different loss functions, we define 12 different scenarios in which to do the comparison. Each of these scenarios is a combination of a dataset, a leakage model and an architecture type.

**Datasets and Leakage Models**
The three datasets used for these experiments are the ASCAD_fixed, ASCAD_variable and CHES_CTF datasets introduced in Section 2.3. In all three of the datasets, the targeted sensitive value is the output of the S-box in the first round of AES. In all of the considered datasets, masking is used as a countermeasure. For each trace $i$, the sensitive value $z_i$ corresponding to the third key byte can be calculated as follows:

$$z_i = sbox(pt_i[3] \oplus k[3])$$

where $pt_i$ is the plaintext used during encryption. We consider two different leakage models, the identity model and the Hamming weight (HW) model. In the identity model, we use $z_i$ directly as a label and target, resulting in 256 possible values and classes. In the HW model, we calculate the Hamming weight for each $z_i$ and use that as a label and target. This results in 9 possible values and classes.

**Architecture Types**
We consider the two different architecture types introduced in Subsection 2.2.1: multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs). Both of these types of deep learning architectures are commonly used for SCA and have shown excellent results in previous work [4, 40, 75].

For both the median model and hyperparameter optimisation approach, random hyperparameters for both of these architecture types need to be selected. In their work, Perin et al. specify a search space for both the MLP and CNN hyperparameters that is balanced between good performance in previous work and still allowing a broad range of possible values per parameter. Their search space is chosen as a basis for this experiment. Benadjila et al. perform several experiments with different amounts of epochs and learning rates on the ASCAD datasets. They propose to use up to 800 epochs and a learning rate of $10^{-5}$ in combination with the RMSProp optimiser. To balance computational cost and performance, each model is trained for 200 epochs. Both the Adam and RMSProp optimisers have been shown to perform well [4, 46]. In addition to the hyperparameter ranges proposed by Perin and

[1] http://insy.ewi.tudelft.nl/content/hpc-cluster

Picek, both these optimisers are added as an option and the range of learning rates is broadened. The possible values for each hyperparameter for the MLP models are given in Table 4.1.

Table 4.1: Hyperparameter space for multi-layer perceptrons.

| Hyperparameter | Min | Max | Step size |
|---|---|---|---|
| Dense layers | 2 | 8 | 1 |
| Neurons per layer | 100 | 1000 | 100 |
| Learning rate | 0.000001 | 0.001 | 0.00001 |
| Batch size | 100 | 1000 | 100 |
| | **Options** | | |
| Activation function | [ReLu, SELU, ELU, Tanh] | | |
| Optimiser | [Adam, RMSProp] | | |

For the CNN hyperparameters, the search space is again based on the work of Perin et al. with the same changes to the learning rate, amount of epochs and optimisers as for the MLPs. Additionally, a batch normalisation layer, as introduced by Ioffe and Szegedy, is applied after the input layer and after each convolutional block, as is done in earlier work to improve the performance of CNNs [4, 7, 47]. The possible values for each hyperparameter for the MLP models are given in Table 4.2.

Table 4.2: Hyperparameter space for convolutional neural networks.

| Hyperparameter | Min | Max | Step size |
|---|---|---|---|
| Convolutional layers | 1 | 2 | 1 |
| Convolutional filters | 8 | 32 | 4 |
| Kernel size | 10 | 20 | 2 |
| Pooling size | 2 | 5 | 1 |
| Pooling stride | 2 | 10 | 1 |
| Dense layers | 2 | 3 | 1 |
| Neurons per layer | 100 | 1000 | 100 |
| Learning rate | 0.000001 | 0.001 | 0.00001 |
| Batch size | 100 | 1000 | 100 |
| | **Options** | | |
| Activation function | [ReLu, SELU, ELU, Tanh] | | |
| Optimiser | [Adam, RMSProp] | | |
| Pooling type | [Max pooling, Average pooling] | | |

**Loss Functions**

The loss functions that are tested are functions commonly used in different deep learning applications and novel loss functions specifically developed for SCA, introduced in Subsection 2.2.2. In almost all recent work on deep learning for SCA, the categorical cross-entropy or mean squared error (MSE) are used as loss functions. Besides those commonly used functions, several others are also considered. The hyperbolic cosine loss, also called the log cosh loss, and mean squared logarithmic error (MSLE) are used because they are similar to MSE but more robust when faced with outliers. We also consider another loss function typically used for classification tasks, the categorical hinge loss. Furthermore, ranking loss [74] and cross-entropy ratio (CER) [77], two recently proposed novel loss functions specifically for the SCA domain are used.

**Preprocessing**

For each of the mentioned datasets, the full set of features from each trace is used and no further selection of points-of-interest is done. Earlier work suggests that scaling SCA features to values between 0 and 1 works well [39, 74]. Therefore, a similar method is applied in this work, and for every

experiment we perform, the features are normalised to values between 0 and 1. This is done by using the MinMaxScaler [2] from the scikit-learn Python module [3].

### Experiment Phases

We compare the loss functions in 12 different scenarios for both the median model and hyperparameter optimisation experiments. Each of the scenarios is a combination of a dataset, architecture type and leakage model. Table 4.3 shows an overview of these scenarios.

Table 4.3: Overview of the considered scenarios.

| **Dataset** | ASCAD_fixed | | | | ASCAD_variable | | | | CHES_CTF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Architecture** | MLP | | CNN | | MLP | | CNN | | MLP | | CNN | |
| **Leakage model** | ID | HW | ID | HW | ID | HW | ID | HW | ID | HW | ID | HW |

In the first phase, we have to find a median model to test each loss function on. To find such a model, we generate 100 models and take the median model in terms of guessing entropy. We then use the hyperparameters of the median model to train new models with each of the loss functions.

To summarise, for each of the scenarios, we perform the following steps:

1. Generate, train and test 100 random models
2. Select the median model in terms of guessing entropy
3. For each loss function, train and test the median model 10 times due to random initialisation of the trainable parameters
4. From those 10 models, select the median model per loss function based on guessing entropy
5. Compare the resulting models on guessing entropy, success rate, training time and the number of trainable parameters.

The second experiment phase with these scenarios consists of doing hyperparameter optimisation via a random search for each loss function. With each of the loss functions fixed, 100 random models are trained and the best model in terms of guessing entropy is selected. The best models for each loss function are then compared on guessing entropy, success rate, training time and the number of trainable parameters.

To summarise, for each of the scenarios from Table 4.3, we perform the following steps:

1. Generate, train and test 100 random models per loss function
2. For each loss function, select the best model based on guessing entropy
3. For each loss function, train and test the best model 10 times due to random initialisation of the trainable parameters
4. From those 10 models, select the median model per loss function based on guessing entropy
5. Compare the resulting models on guessing entropy, success rate, training time and the number of trainable parameters.

In both phases, we train 10 new models with the found median or best performing model. Since models with the same hyperparameters sometimes perform differently due to randomness in for example the random initialisation of the weights, outliers might occur in terms of performance or the training might fail. We therefore choose the median of those 10 models for our comparison.

This setup allows us to compare the loss functions on the same architecture, namely the median model in the first phase, and on the architectures optimised for each loss function in the second phase.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
[3] https://github.com/scikit-learn/scikit-learn

### 4.2.2 State-of-the-art Architectures in Profiled SCA

Different state-of-the-art deep learning models proposed in recent papers only consider the categorical cross-entropy loss function [4, 75]. For this experiment, we will use the $MLP_{best}$ architecture proposed by Benadjila et al., and the CNN architecture used in [74], which we will denote by $CNN_{methodology}$. Although there are some remarks to be made by the process used to create the $CNN_{methodology}$ model [69], it is still one of the best performing CNN models on the ASCAD fixed key dataset for which the used hyperparameters are available. Therefore, it is a suitable model for our experiments. Both these models are trained and tested on the ASCAD_fixed, ASCAD_variable and CHES_CTF datasets with each of the loss functions. Although the architecture is optimised for the ASCAD datasets, for consistency, we also test the performance on the CHES_CTF dataset. Similarly as before, for each of the datasets, leakage models and loss functions, we train the same model 10 times and take the median performing model in terms of guessing entropy. As specified before, the comparison metrics are the final GE, SR, training time and the number of trainable parameters.

### 4.2.3 Countermeasures

We compare the robustness of models trained with different loss functions against different countermeasures, namely masking and a random desynchronisation added to the traces. We use the previously found best-performing architectures per loss function for the ASCAD_fixed dataset. These models are trained on the ASCAD_plain dataset introduced in Section 2.3. The targeted sensitive value will then become:

$$z_i = sbox(pt_i[3] \oplus k[3]) \oplus m_{out_i}$$

where $m_i$ is the known mask for the targeted key byte. This will allow us to compare the performance in GE on an unprotected implementation of AES versus the performance on the regular ASCAD_fixed dataset, which is a masked implementation of AES. Furthermore, we test the performance of these models on the ASCAD_desync50. This dataset is similar to the ASCAD_fixed dataset but with a random desynchronisation, i.e. the features of the traces are randomly shifted up to 50 places. A comparison is then made on the performance in terms of guessing entropy and success rate. We again train 10 models with each loss function and use the median performing model per loss function in our comparison.

## 4.3 Results

In this section, we discuss the results for each of the aforementioned experiments. We will look at the performance of the loss functions on median models and models optimised via random search, their resilience against countermeasures and their performance when used with state-of-the-art architectures. The chosen median and optimised hyperparameters for each of the scenarios can be found in Appendix A and B.

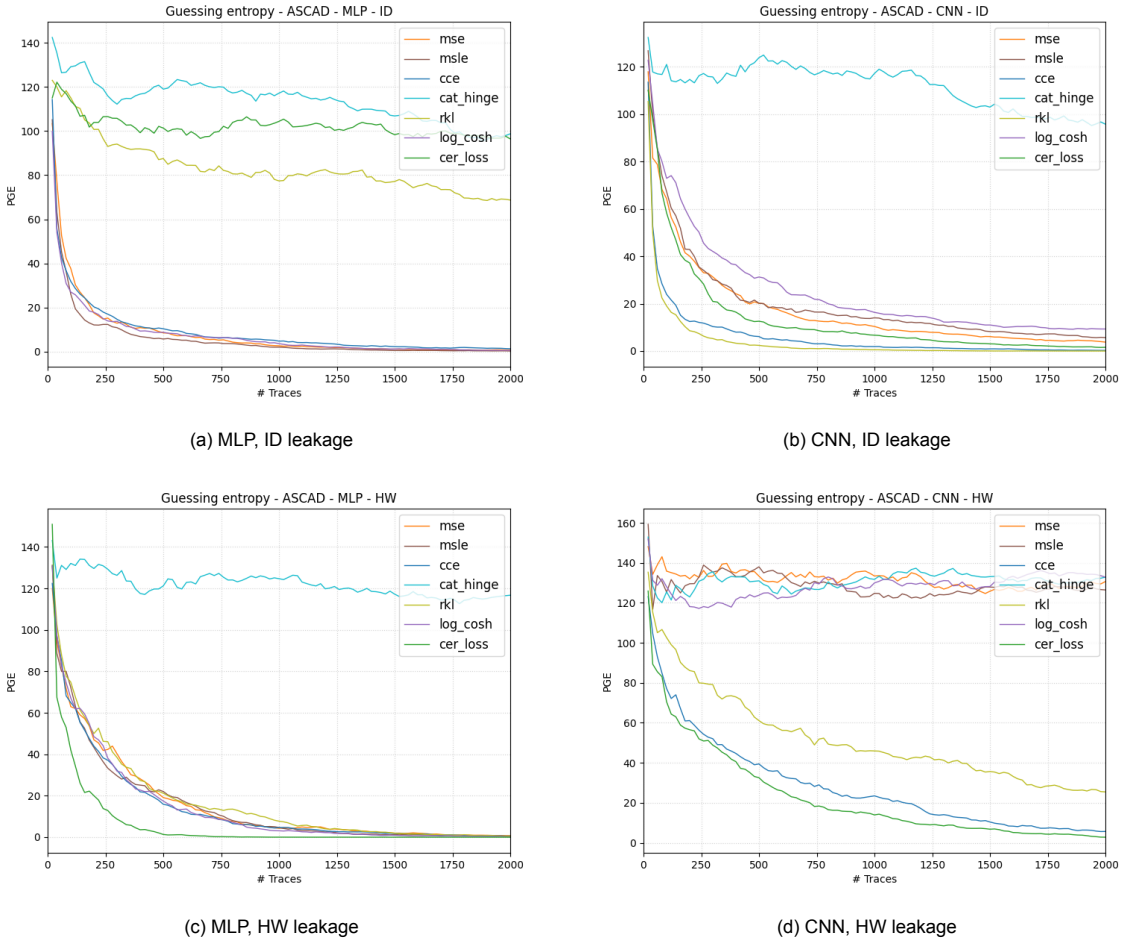
For all of the experiments, the same attack settings are used. The amounts of profiling traces used are 50.000 for the ASCAD datasets and 45.000 for the CHES_CTF dataset. In the attacking phase, we use up to 2000 traces for the ASCAD_fixed dataset and up to 3000 traces for the ASCAD_variable and CHES_CTF datasets.

### 4.3.1 Median Model & Hyperparameter Optimisation

#### ASCAD_fixed

We first consider the performance of the different loss functions on the ASCAD_fixed dataset. Figure 4.2 shows the guessing entropy over 100 attacks for each of the scenario's median models. Figure 4.3 shows the guessing entropy for each of the optimised models.

First of all, we notice that the ASCAD_fixed dataset can be considered relatively easy to attack. Most of the loss functions lead to a model which can retrieve the correct key in less than 2000 traces when a median MLP architecture is used. When we look at the optimised models, we see that even




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

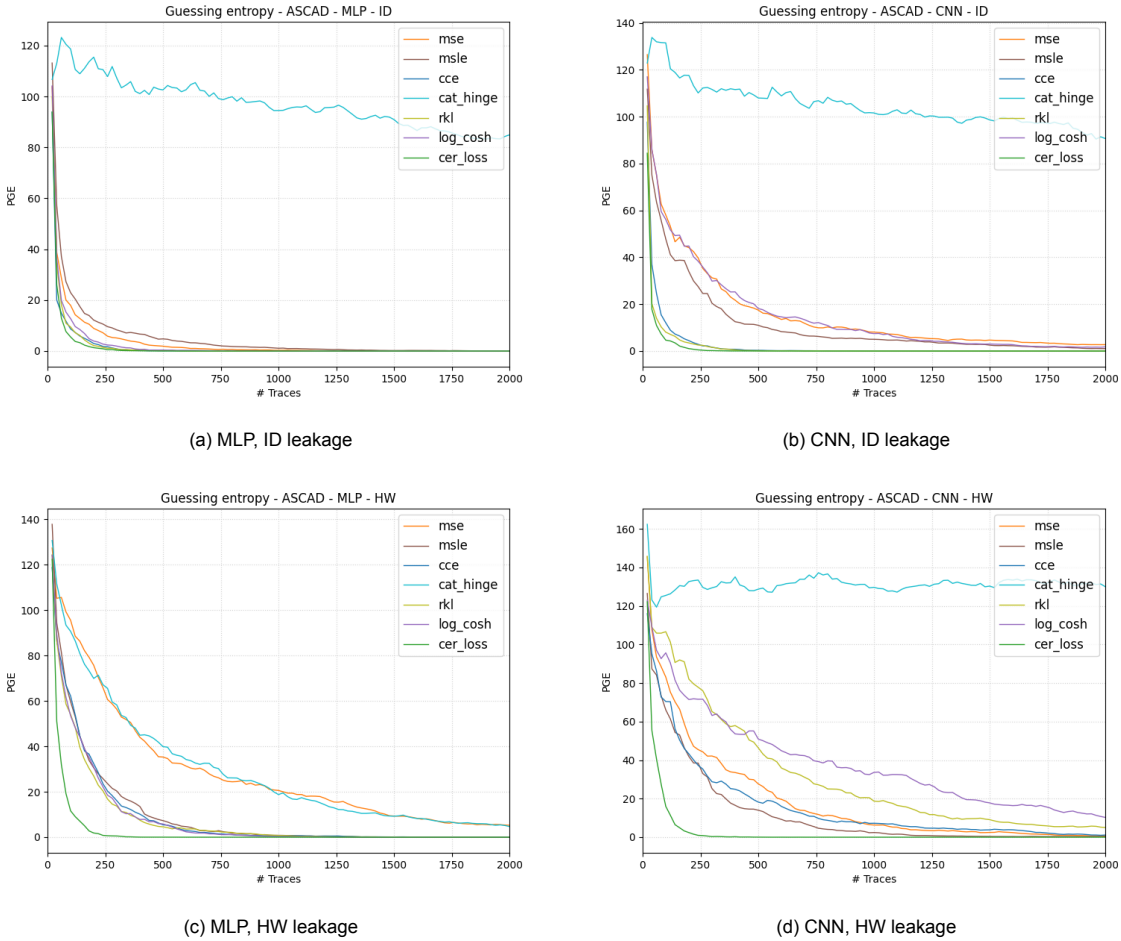
(d) CNN, HW leakage

Figure 4.2: GE of the median MLP and CNN models on the ASCAD_fixed dataset.

a simple parameter optimisation via random search results in models reaching a GE of 1 in less than 500 traces.

We also see that the commonly used categorical cross-entropy does indeed perform very consistently in these scenarios. It also shows to be quite robust to different hyperparameter choices, performing well with a broader range of different combinations of hyperparameters. Figure 4.4 shows the 100 models generated for the hyperparameter optimisation experiment with a CNN architecture and the ID leakage model, for both the categorical cross-entropy and MSE.

Looking at the first of the two novel loss functions, CER loss, we also see some interesting behaviour. First of all, CER loss outperforms every other function in all but two scenarios. It is only outperformed on the scenarios with a median model and ID leakage. Zhang et al. introduce the CER loss function to improve the performance of deep learning models on imbalanced SCA data, i.e. when the Hamming weight leakage model is considered [77]. Our results confirm that the function indeed performs well in those scenarios, outperforming the MLP models of the original CER paper and performing much better than the other loss functions. Furthermore, Zhang et al. show that their CER metric is a good estimator for the performance of a deep learning model regardless of the data being balanced or imbalanced. However, they only test their CER loss function on the HW leakage model, i.e. imbalanced data. Our results show that when the rest of the hyperparameters are optimised, the CER loss is also very suitable for the ID leakage model. Figure 4.3a and Figure 4.3b show that in these scenarios, the models trained with CER loss outperform the models trained with categorical cross-entropy. We can therefore conclude that when the ASCAD_fixed dataset is considered, the best choice of loss functions is the CER loss. Especially when the other hyperparameters are optimised, it significantly reduces the




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

(d) CNN, HW leakage

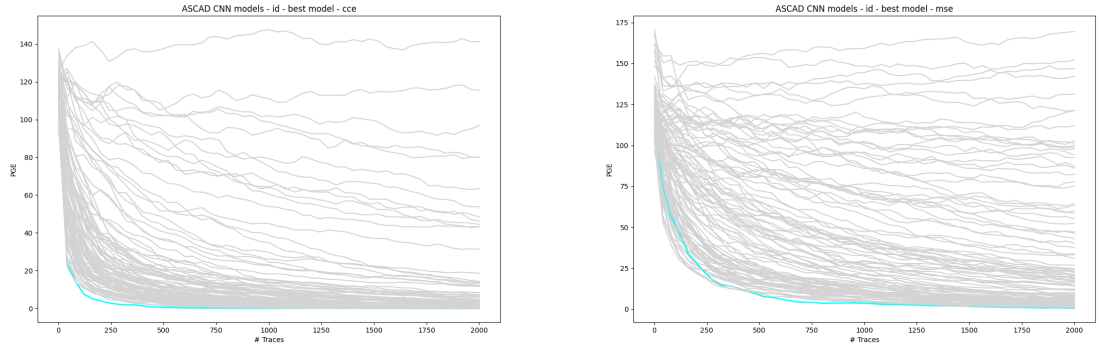
Figure 4.3: GE of the optimised MLP and CNN models on the ASCAD_fixed dataset.

amount of traces needed to perform a successful attack in comparison to the categorical cross-entropy and other loss functions.

The second novel loss function, ranking loss (RKL), performs less consistent. [74] compare the RKL function to categorical cross-entropy and CER loss, stating that the RKL outperforms both those functions. However, they only do their comparison with a single CNN architecture and only consider the ID leakage model[74]. If we look at our CNN median and optimised model results for the ID leakage, we indeed see that in those scenarios, RKL performs similarly or slightly better than the categorical cross-entropy and CER loss. However, in all the other scenarios, RKL performs worse than these loss functions.

Another remark that has to be made when discussing these results is the required training time. Figure 4.5 shows the training times for the median model with each of the loss functions, on both the HW and ID leakage models. This shows that when all other hyperparameters are equal, both the RKL and CER loss functions are significantly slower than other functions. In the case of RKL, the cause for the slower training time is the pairwise comparison that is part of the loss. This part of the loss is calculated by comparing the rank of the correct key with all the other key guesses. This causes an impact on the training time when the HW leakage is considered, where the output consists of 9 classes, and an even larger impact when the ID leakage is used, where there are 256 output classes.

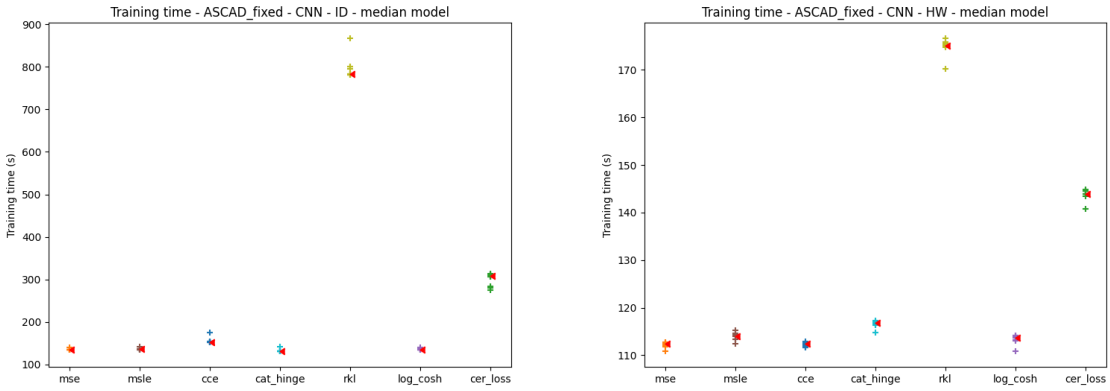
The increased training time in case of the CER loss is also due to the way the function is constructed. CER loss, as explained in Subsection 2.2.2, is calculated by dividing the cross-entropy over the profiling traces by the average of $N$ times the set of profiling traces with shuffled labels. Calculating the cross-entropy over the shuffled traces $N$ times causes the slower training in comparison with other




(a) Models trained with categorical cross-entropy

(b) Models trained with MSE

Figure 4.4: The 100 random models generated for hyperparameter optimisation with categorical cross-entropy and MSE as loss functions. Almost all models trained with the categorical cross-entropy perform well and converge towards a GE of 1 relatively fast, while there is more variation in the models trained with MSE. The blue line indicates the model with the lowest $\overline{N}_{T_{GE}}$.



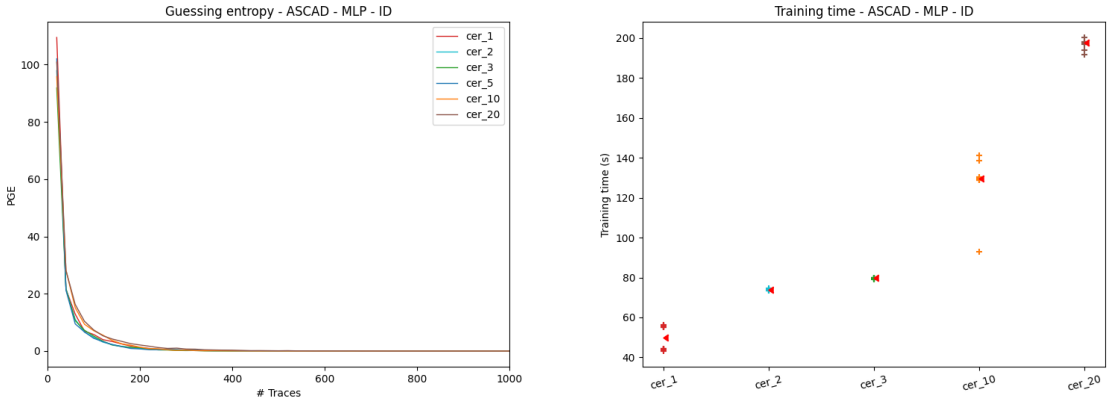

(a) ID leakage model

(b) HW leakage model

Figure 4.5: For each of the loss functions, ten models were trained with the hyperparameters of the median model. These are the training times for each of these models, for the HW and ID leakage models. For each loss function, the 10 models with the same hyperparameters are plotted. The red triangles mark the median training times.

loss functions. Since [77] do not analyse the impact of different values for $N$, we chose $N = 10$ for these experiments. However, as shown in Figure 4.6, all different values of $N$ except $N = 20$ result in a $\overline{N}_{T_{GE}}$ of approximately 500. For lower values like $N = 1$ or $N = 2$, there is no noticeable difference in training time in comparison with for example the categorical cross-entropy, while there is still the increase in performance in GE. For consistency, we have used $N = 10$ for all the following experiments.

The only function that does not often lead to a converging model is the categorical hinge loss. Only in two scenarios, namely the optimised MLP and CNN models with HW leakage, the usage of the categorical hinge loss leads to a converging model. A possible reason for this could be the combination of a low learning rate and the low amount of classes when HW leakage is considered. The median models for these scenarios all have a learning rate between $0.0001$ and $0.0007$, while the optimised models with categorical hinge loss tend to have a higher learning rate. Furthermore, if we look at the definition of the categorical hinge loss as described in Subsection 2.2.2, we see that the negative part of the loss is calculated based on the wrong class with the highest probability, i.e. the largest mistake. With the ID leakage model, we have too many wrong classes (255) and one correct class. Due to random initialisation of the weights, the loss coming from wrongly classified traces will stay approximately $1$ at the start of training and the main contribution to change of the loss has to come from a correctly classified example. This will not occur often enough since with the ID leakage, there are 256 classes. With the HW leakage model, there are only nine classes to consider. A correct classification will, even




(a) Guessing entropy

(b) Training time

Figure 4.6: Guessing entropy and training time for the optimised model with CER loss using different values of $N$.

when random guessing, happen more often, impacting the loss and allowing the model to learn. So due to the difference in the number of classes between the ID and HW leakage model and a low learning rate, the categorical hinge loss works better with the HW leakage model in this scenario.

The success rates of the discussed scenarios in Figure 4.7 and Figure 4.8 show similar results for the loss functions as the guessing entropy. Figure 4.8 shows more clearly that when the other hyperparameters are optimised, the CER loss performs best when the ASCAD_fixed dataset is considered.

Finally, we look at the number of trainable parameters for the scenarios with optimised hyperparameters. A smaller, less complex model in terms of the number of parameters is generally faster to train since there are fewer weights to be updated. Table 4.4 shows the number of trainable parameters that each of the optimised models per loss function has. Here we see that the optimised models trained with the categorical cross-entropy often have the least amount of trainable parameters in comparison to the other loss functions. Other functions, like MSE and CER loss, also seem to perform well with smaller models, while the categorical hinge and MSLE loss only perform well with larger models.

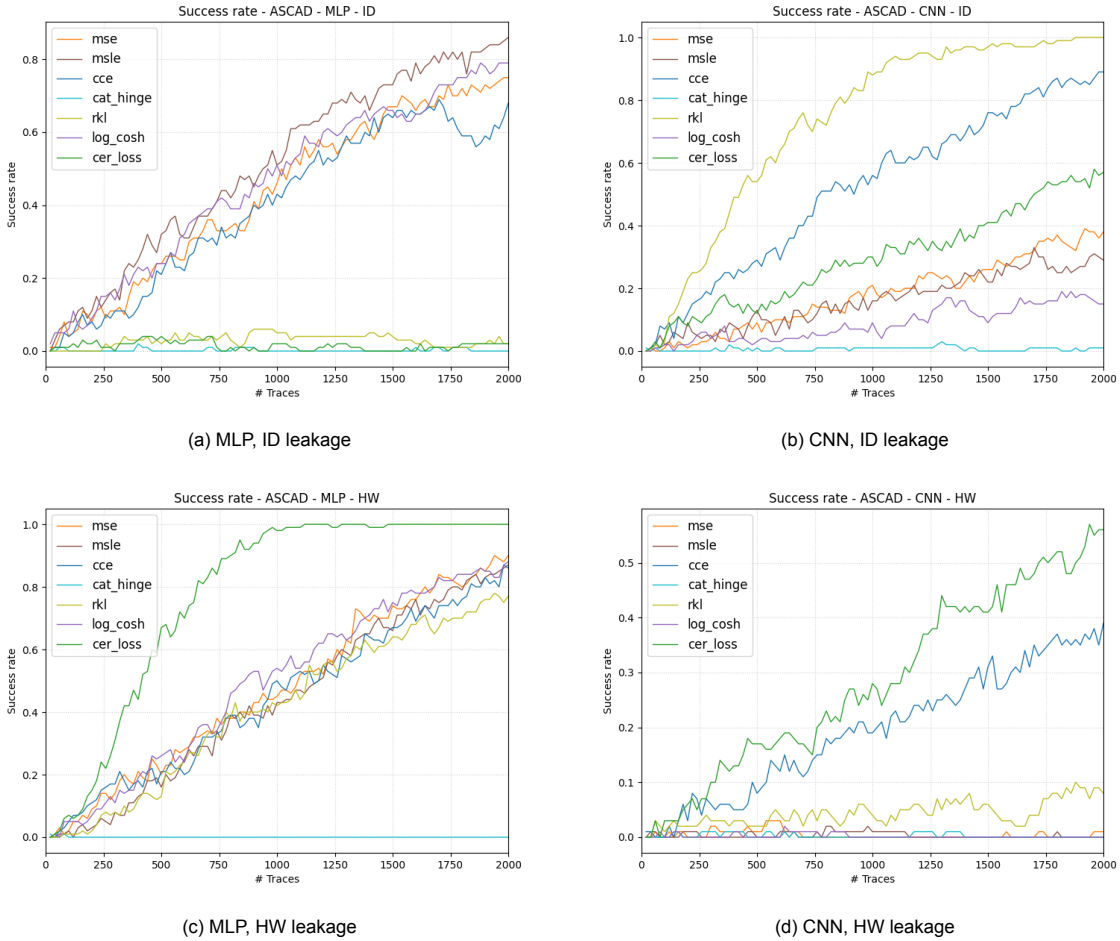
Table 4.4: The number of trainable parameters for the optimised models per loss function and scenario. The lowest number of trainable parameters for each scenario is marked blue, the highest orange.

| Loss function | MLP ID | MLP HW | CNN ID | CNN HW |
|---|---|---|---|---|
| Categorical cross-entropy | 116,156 | 302,809 | 53,244 | 1,124,565 |
| Categorical hinge | 1,295,656 | 3,882,609 | 1,022,920 | 2,852,165 |
| CER loss | 467,956 | 483,909 | 356,424 | 598,853 |
| Log cosh | 126,256 | 856„009 | 335,824 | 1,011,657 |
| MSLE | 543,456 | 1,449,909 | 5,738,736 | 1,335,177 |
| MSE | 166,656 | 754,809 | 214,564 | 291,409 |
| RKL | 543,456 | 604,809 | 186,616 | 1,056,421 |

Overall, when considering the ASCAD_fixed dataset, we can conclude that CER loss seems to be the best choice for the loss function. It significantly outperforms models with categorical cross-entropy, ranking loss and other loss functions. The resulting models still have a relatively low amount of trainable parameters and when using $N = 1$, are still as fast during training.

### ASCAD_variable

Next, we look at the results on the ASCAD dataset but with random keys used during the profiling phase. Figure 4.9 shows the performance in GE on the median models, Figure 4.10 shows the performance for the optimised models.




(a) MLP, ID leakage

(b) CNN, ID leakage

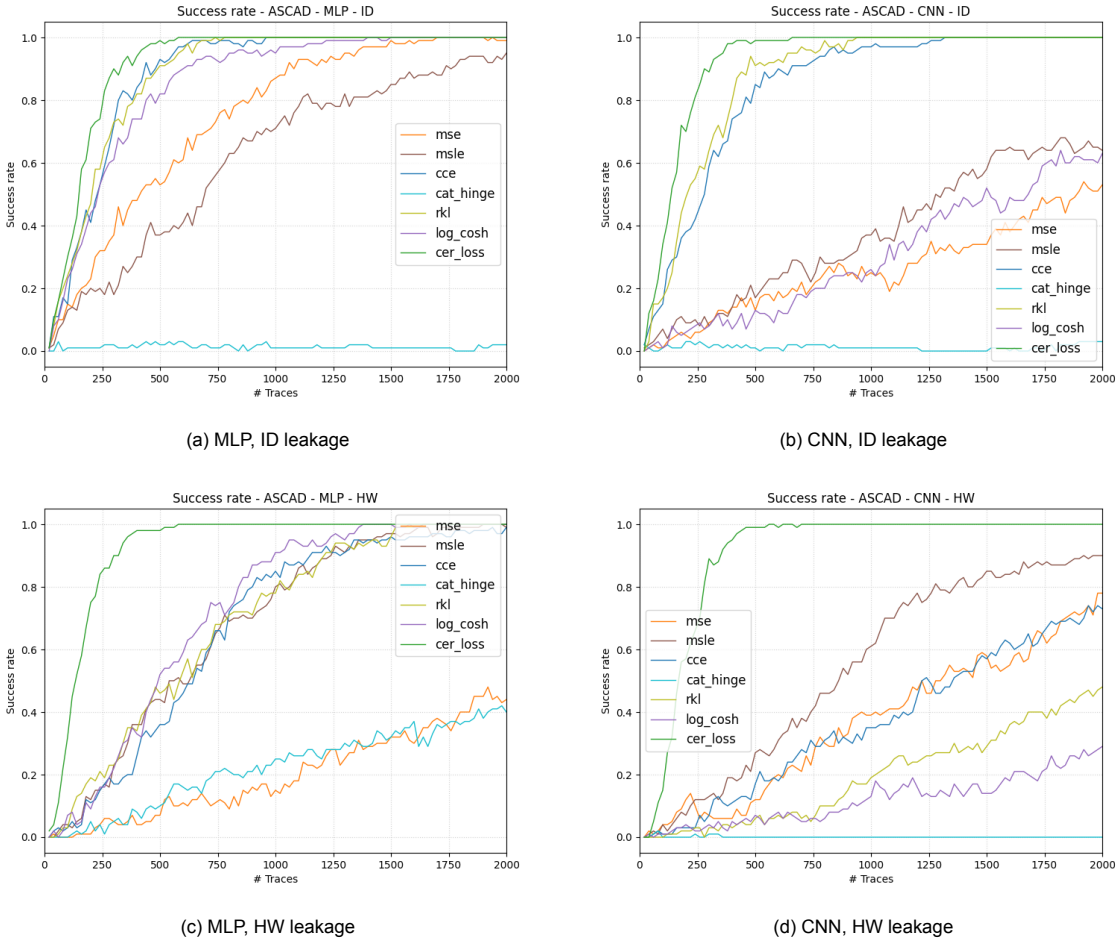
(c) MLP, HW leakage

(d) CNN, HW leakage

Figure 4.7: Succes rate of the median MLP and CNN models on the ASCAD_fixed dataset.

For the experiments performed on the ASCAD_variable dataset, we see results comparable to those on the ASCAD_fixed dataset. In most of the scenarios, we see that the models trained with CER loss perform the best, followed closely by the models trained with categorical cross-entropy. An exception to these similarities is visible in the scenarios with a median model. In the experiment with MLPs and HW leakage shown in Figure 4.9c, the CER loss model does not converge. In comparison with other scenarios with HW leakage, the model with CER loss performs extremely poorly in this case. The median model in this scenario consists of four dense layers of 200 neurons, uses the ELU activation function and RMSprop optimiser, and a batch size of 300. Looking closer at the training process of models with these parameters and CER loss reveals that during training, the loss sometimes becomes a NaN value. The underlying cause of this problem turns out to be the exploding gradients problem: gradients getting too large or small, causing the learning process to fail [48]. Figure 4.11 shows the largest and smallest gradient of the input layer for each of the epochs during the training process of one of these models. The reason these specific choices result in the exploding gradient problem appears to be the combination of several hyperparameters. Specifically, the ELU activation function, the loss function and the 0-1 normalisation used during preprocessing.

By normalising all feature values to values between 0 and 1, we remove any negative values from the profiling traces. If we look at the definition of the ELU function in Equation 4.1, we see that the output of the activation function is equal to the input for all $x > 0$. This means that the output is unbounded, i.e. there is no limit on how large it can get. This also means that, since we normalised to values between 0 and 1, the output of the activation function will always be positive. Something similar is true for the ReLU activation function. This, in combination with the CER loss function, and in some cases also the ranking loss, causes the gradients to get too large, leading to a poorly performing model or even a




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

(d) CNN, HW leakage

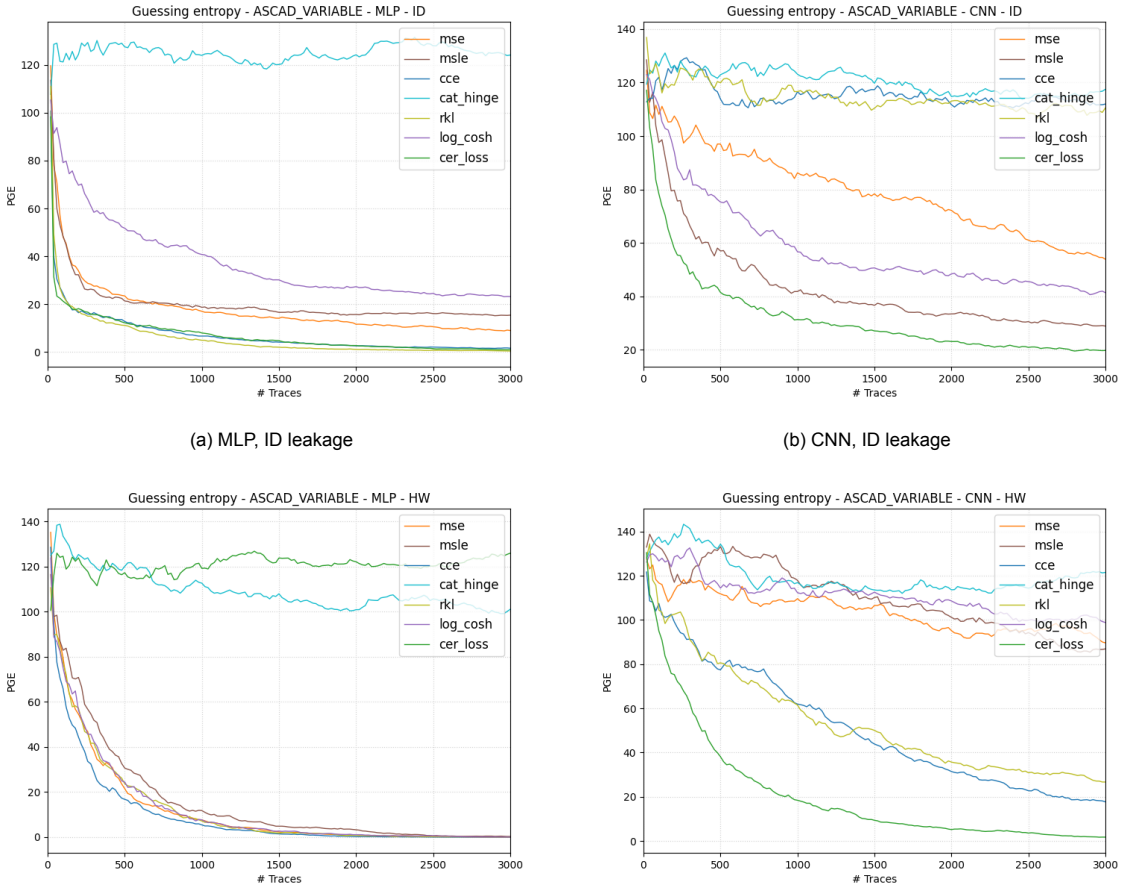
Figure 4.8: Success rate of the optimised MLP and CNN models on the ASCAD_fixed dataset.

failed training.

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{4.1}$$

Some solutions to mitigate this problem do exist. We could use a different combination of loss function, activation function and preprocessing method. Figure 4.11b shows for example the same gradients but with the profiling traces normalised by Z-score normalisation (standardisation) [69]. The attack performance is similar to when 0-1 normalisation is used, so when using CER loss and RKL in future work with activation functions such as ELU, using standardisation instead of 0-1 normalisation is to be preferred. Another possible solution might be clipping the gradients when they get too large or too small. For our experiments, we have kept the preprocessing similar for each experiment as described in Table 4.5.

The success rates and training times show similar results as with the ASCAD_fixed dataset. The success rates show a similar ordering in performance as the guessing entropy, and looking at the training times, ranking loss and CER loss are both slower than the other functions. However, some differences are visible if we compare the number of trainable parameters of the optimised models. Table 4.5 shows the number of trainable parameters of the optimised model for each of the loss functions. In contrast with the results on the ASCAD_fixed dataset, there is no clear function that works well with smaller models in general. The log cosh loss does seem to perform well with smaller CNN models, but the best performing MLP models with log cosh tend to be very large. Similarly, the optimised categorical cross-entropy and ranking loss MLP models are relatively small, while their CNN counterparts are larger.




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

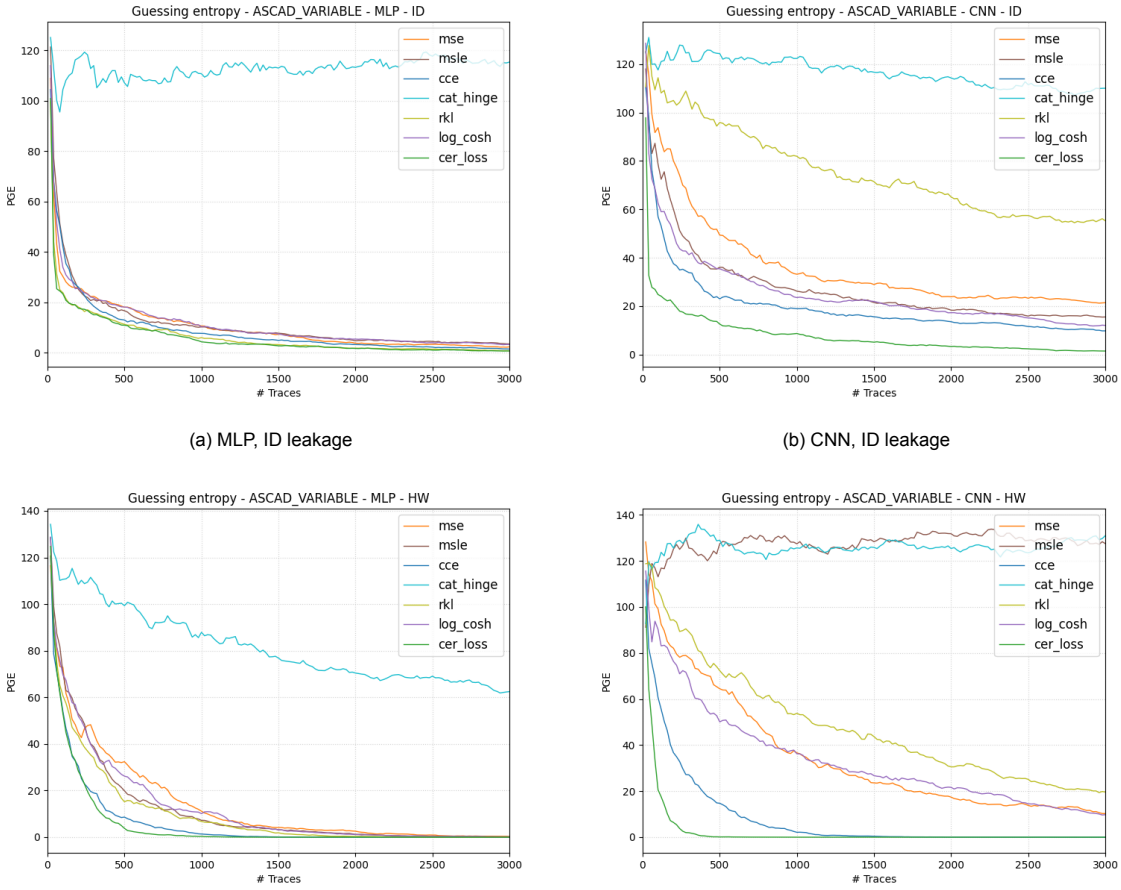
(d) CNN, HW leakage

Figure 4.9: GE of the median MLP and CNN models on the ASCAD_variable dataset.

Table 4.5: The number of trainable parameters for the optimised models per loss function and scenario. The lowest number of trainable parameters for each scenario is marked blue, the highest orange.

| Loss function | MLP ID | MLP HW | CNN ID | CNN HW |
|---|---|---|---|---|
| Categorical cross-entropy | 1,830,756 | 402,609 | 3,869,540 | 3,844,221 |
| Categorical hinge | 2,582,256 | 4,512,609 | 1,007,956 | 5,071,253 |
| CER loss | 1,966,656 | 2,079,909 | 1,602,260 | 937,253 |
| Log cosh | 6,662,256 | 3,701,709 | 314,616 | 245,481 |
| MSLE | 1,129,456 | 4,413,009 | 4,499,560 | 4,371,285 |
| MSE | 1,625,456 | 1,927,809 | 1,768,264 | 179,265 |
| RKL | 371,856 | 1,477,709 | 4,591,236 | 3,634,445 |

Similarly, as to the ASCAD_fixed dataset, we can conclude that the CER loss is also the preferred loss function when the ASCAD_variable dataset is considered. It again outperforms the categorical cross-entropy in terms of guessing entropy and success rate in most of the scenarios. It sometimes reduces the required traces for a guessing entropy of 1, $\overline{N}_{T_{GE}}$, more than threefold. When considering for example the optimised CNN models, the categorical cross-entropy has a $\overline{N}_{T_{GE}}$ of 2520, while for CER loss this is reduced to 720.




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

(d) CNN, HW leakage

Figure 4.10: GE of the optimised MLP and CNN models on the ASCAD_variable dataset.

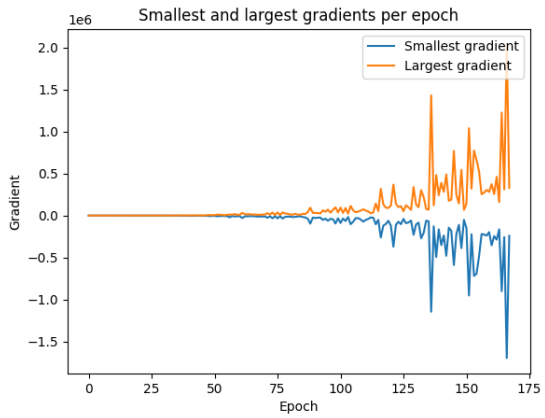
### CHES_CTF

Finally, we look at the performance of the median and optimised models on the CHES_CTF dataset. Figure 4.12 shows the guessing entropy in the scenarios with a median model, while the results of the optimised models are shown in Figure 4.13.

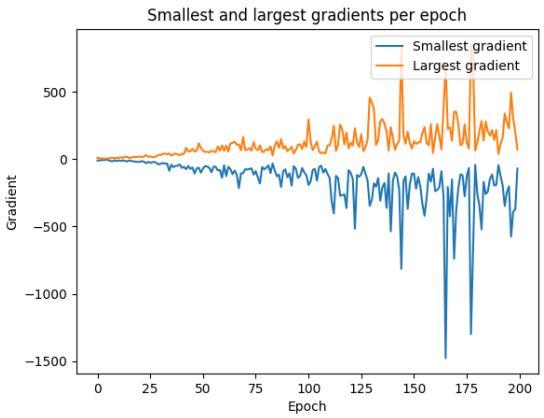
Right away, it is clear that the CHES_CTF is a more difficult target than the ASCAD datasets. Only in some scenarios are models capable of reaching a GE of 1 with less than 3000 traces. Other works that use the CHES_CTF dataset confirms this, where more optimised models or ensembles of models perform similarly or only slightly better when the HW leakage is considered [47, 70]. Another interesting result is the failure of the median model trained with ranking loss visible in Figure 4.12a. All 10 of the models trained with the ranking loss in this scenario fail to train due to the gradient problem explained in Section 4.3.1. This causes (some) weights of the resulting model to become NaN values and always output the same predictions. This explains the constant GE visible in Figure 4.12a.

In contrast with the ASCAD datasets, there is also no single function that clearly outperforms the others in all of the scenarios. When the ID leakage is considered, none of the models is able to perform a successful attack with less than 3000 traces. For the HW leakage scenarios, the best choice again is the CER loss.

Interestingly, the median CNN model trained with the log cosh loss seems to perform better than the other functions, shown in Figure 4.12b. It also performs better than the best model with log cosh loss in the same scenario but with the other hyperparameters optimised via random search visible in Figure 4.13. The median model for this scenario has three dense layers with 1000 neurons each, uses average pooling and the hyperbolic tangent (tanh) activation function. The optimised model has the same hyperparameters, except it uses the SELU activation function. A similar model, with approxi-

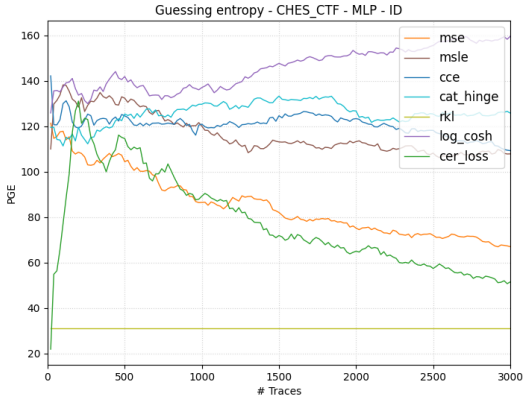



(a) 0-1 Normalisation

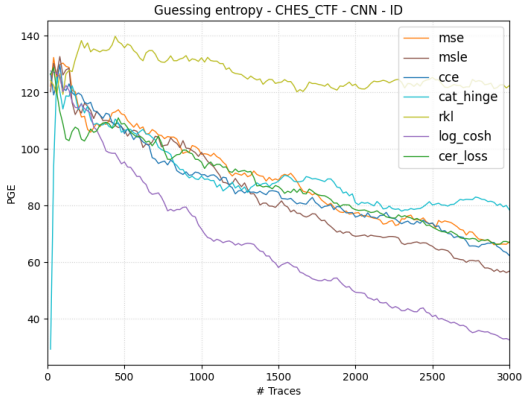



(b) Z-score normalisation

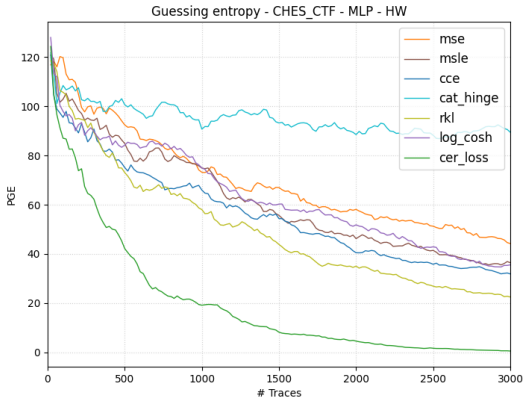
Figure 4.11: The largest and smallest gradient of the input layer during training of a model with CER loss in the median MLP HW leakage scenario when different preprocessing is done. The gradients explode to large values with 0-1 normalisation, while they do not when for example Z-score normalisation is applied.



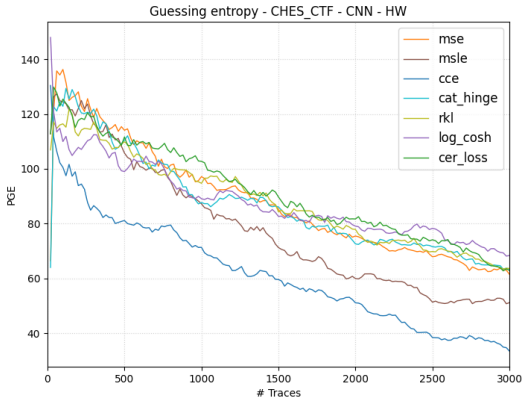

(a) MLP, ID leakage




(b) CNN, ID leakage



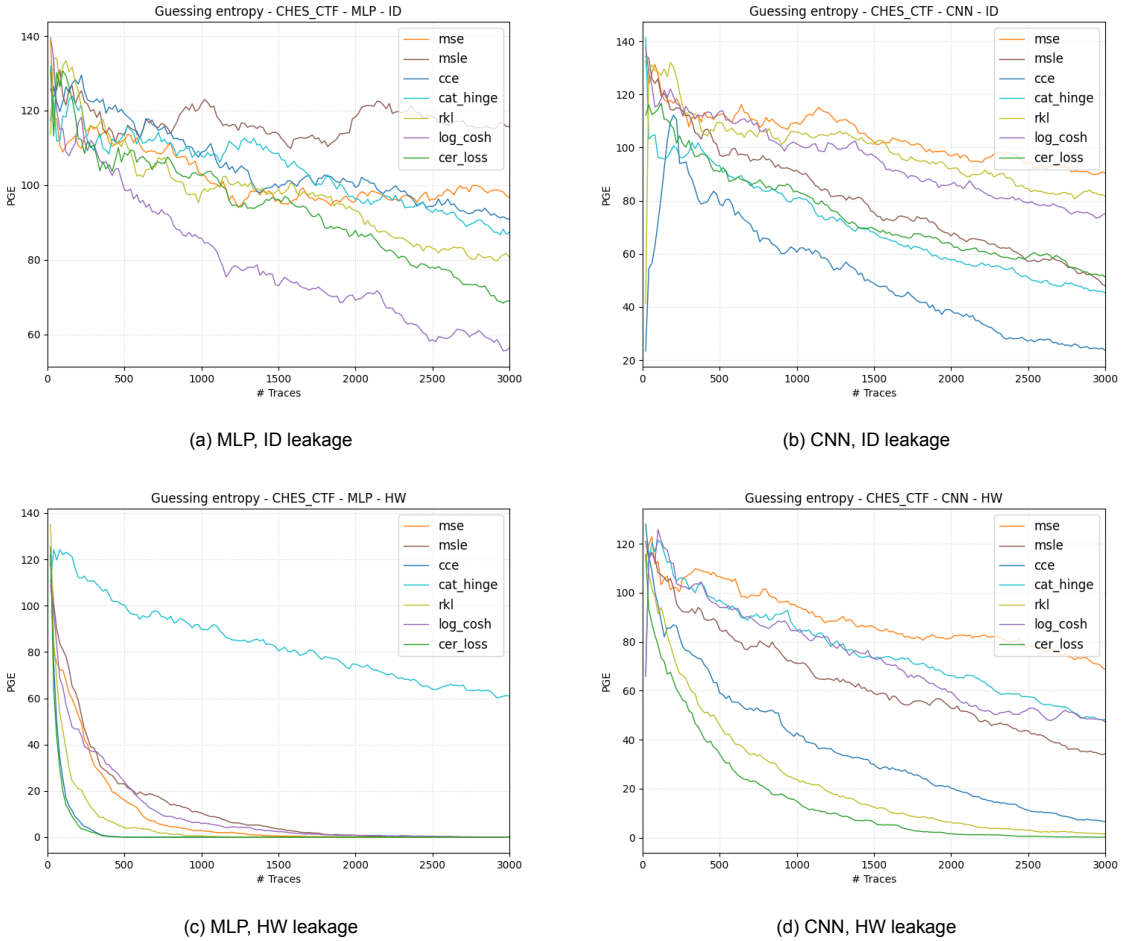

(c) MLP, HW leakage




(d) CNN, HW leakage

Figure 4.12: GE of the median MLP and CNN models on the CHES_CTF dataset.

mately the same numbers of dense layers and neurons, with the tanh activation function was not among the 100 models generated for the hyperparameter optimisation. This shows the limit of hyperparame-




(a) MLP, ID leakage

(b) CNN, ID leakage

(c) MLP, HW leakage

(d) CNN, HW leakage

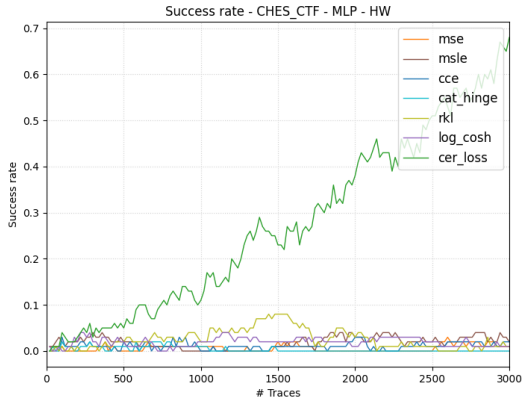
Figure 4.13: GE of the optimised MLP and CNN models on the CHES_CTF dataset.

ter optimisation via random search. For a search space with multiple dimensions and possible values, other optimisation strategies might be required to find the actual best model. Future work could look into further finding optimal models for each of the loss functions, for example with the method proposed by Wu et al.

Looking at the scenarios in which we can attack the CHES_CTF dataset successfully with less than 3000 traces, we again see that both the categorical cross-entropy and CER loss models perform the best. Figure 4.14 and Figure 4.15 show the success rates for these scenarios. For all the other scenarios, the success rates at 3000 traces stay lower than 0.05.

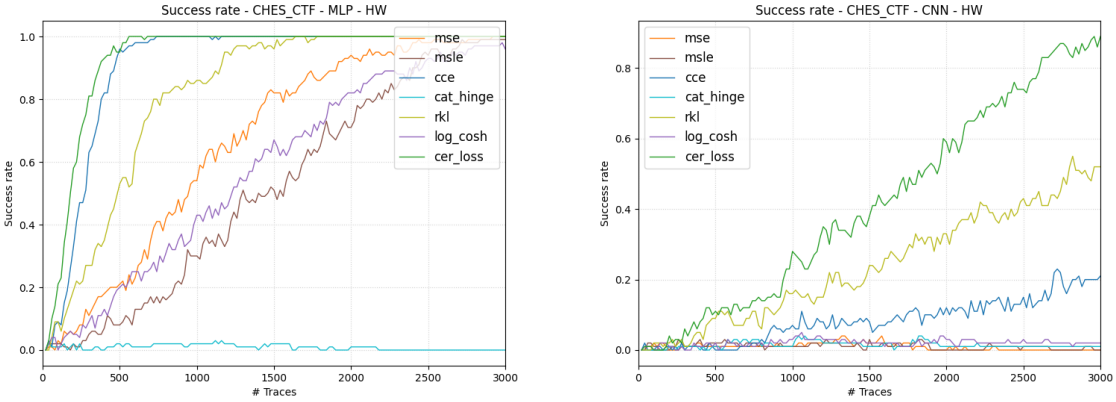
So far we mainly have looked at differences in performance between the leakage models and the median and optimised scenarios. One of our subquestions is related to the difference in the performance of the functions on different architecture types. For the CHES_CTF dataset, we see that our MLP models seem to perform better when the HW leakage model is considered. However, going back to the ASCAD datasets, we see different behaviour. In Figure 4.3 for example, we see that the performance decreases for all the loss functions when CNNs are considered, except for the CER loss. The same effect is visible in for the ASCAD_variable dataset, shown in Figure 4.10. This could be due to the way we have set up our hyperparameter ranges resulting in relatively better MLP architectures and less optimal CNN architectures. Earlier work, for example by [4], shows that their $CNN_{best}$ model outperforms their $MLP_{best}$ model slightly using categorical cross-entropy as a loss function, although they use a much more complex CNN architecture.

Finally, we look at the amounts of trainable parameters in Table 4.6. While the categorical hinge loss models are the least complex for the scenarios, they also do not perform very well. In contrast, the




(a) MLP, ID leakage

Figure 4.14: Succes rate of the median MLP on the CHES_CTF dataset.




(a) MLP, HW leakage

(b) CNN, HW leakage

Figure 4.15: Succes rate of the optimised MLP and CNN models on the CHES_CTF dataset.

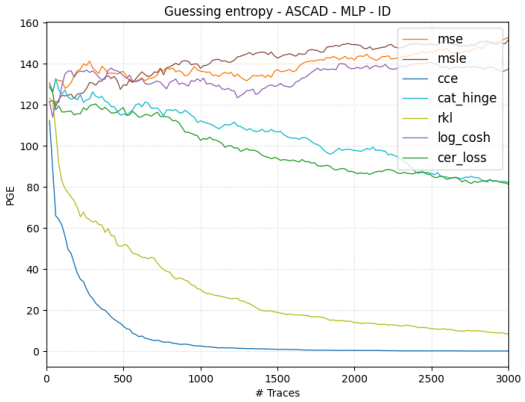
models with ranking loss perform reasonably, not the best or worst, and do so with the smallest models in the scenarios where the HW leakage is considered.

Table 4.6: The number of trainable parameters for the optimised models per loss function and scenario. The lowest number of trainable parameters for each scenario is marked blue, the highest orange.

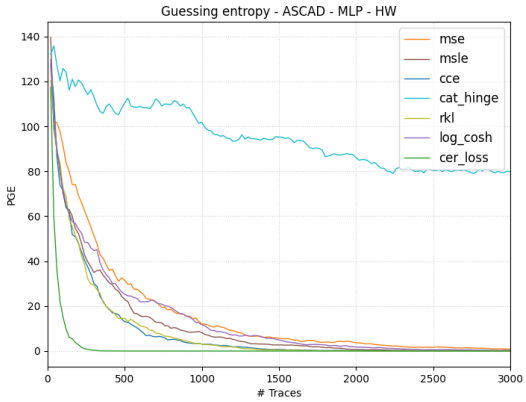
| **Loss function** | **MLP ID** | **MLP HW** | **CNN ID** | **CNN HW** |
|---|---|---|---|---|
| Categorical cross-entropy | 2,210,856 | 683,209 | 1,963,312 | 3,367,549 |
| Categorical hinge | 917,956 | 3,610,809 | 205,816 | 3,146,165 |
| CER loss | 1,464,256 | 4,491,209 | 2,393,676 | 197,345 |
| Log cosh | 3,998,656 | 522,409 | 2,376,008 | 1,187,777 |
| MSLE | 1,188,856 | 4,212,009 | 967,576 | 2,276,685 |
| MSE | 6,461,256 | 723,409 | 983,100 | 291,409 |
| RKL | 3,192,256 | 231,109 | 6,659,716 | 187,245 |

### 4.3.2 State-of-the-art Architectures

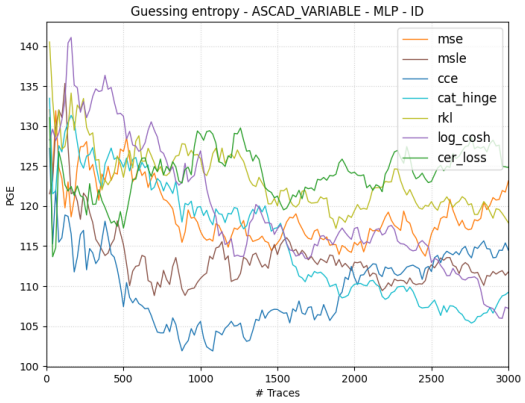
In this subsection, we will lay out the results of the experiments where two state-of-the-art architectures are considered, as described in Subsection 4.2.2. First, we will look at the results from the $MLP_{best}$ model, introduced by Benadjila et al., followed by the results of the $CNN_{methodology}$ model introduced by Zaid et al. Figure 4.16 shows the performance in guessing entropy for the models tested on the three different datasets.



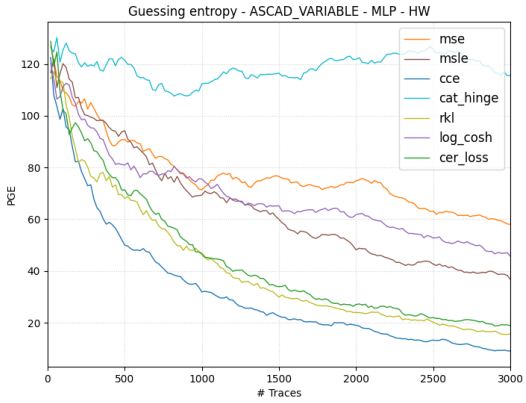

(a) ASCAD_fixed, ID leakage



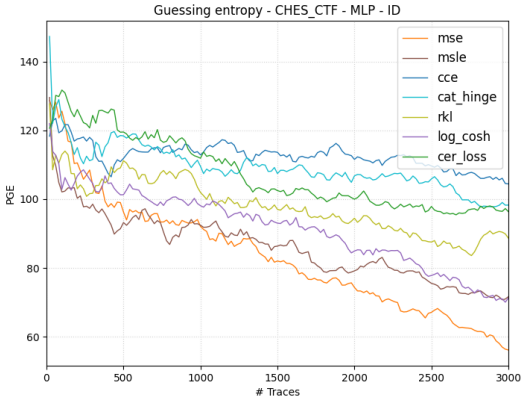

(b) ASCAD_fixed, HW leakage



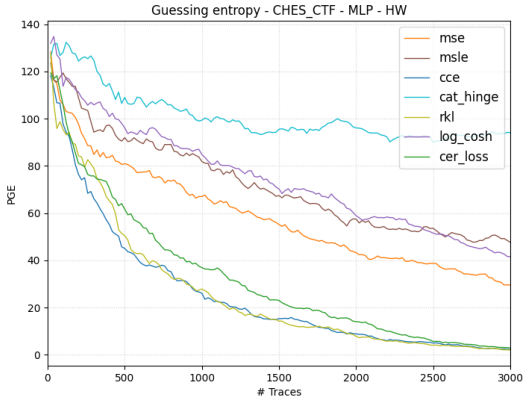

(c) ASCAD_variable, ID leakage




(d) ASCAD_variable, HW leakage




(e) CHES_CTF, ID leakage




(f) CHES_CTF, HW leakage

Figure 4.16: GE of the $MLP_{best}$ models on the ASCAD_fixed, -variable and CHES_CTF datasets.

First of all, we can see that in general, the $MLP_{best}$ architecture is not performing very well. Although the performance with the categorical cross-entropy is indeed similar to that in the original paper, the median and optimised models from the previous experiments outperform the $MLP_{best}$ in every scenario. The authors do manage to achieve a successful attack with this model on the ASCAD_fixed dataset and ID leakage, but only when they train the model for 400 epochs or more. In the context of the original paper, however, similar to the scenario in Figure 4.16a, the categorical cross-entropy does perform the best. Our results show that using another loss does not improve the performance presented in their work. The success rates in Figure 4.17 lead to a similar conclusion, i.e. that the $MLP\_best$ architecture does not perform well in several datasets, leakage models and loss functions. The model is only able to reach success rates higher than 50% on the ASCAD_fixed dataset with HW leakage.

The training times are in line with what we previously saw when comparing the different loss functions, although it takes significantly longer than the optimised and median models from our earlier experiments. Figure 4.18 shows for example the training times for the models on the ASCAD_fixed dataset with ID leakage. Each of the optimised models is trained 2-3 times faster than the $MLP_{best}$ models. A probable explanation for this is the combination of more trainable parameters and the small batch size of 100 that the authors proposed for the $MLP_{best}$ model. In comparison, the median and optimised models almost all have fewer layers (2-4 instead of 5) reducing the complexity, and use a larger batch size of 800-1000. This is also visible in the number of trainable parameters. The $MLP_{best}$ has 352,456 trainable parameters, where most of the optimised models had less than 200,000.
Next, we look at the performance of the $CNN_{methodology}$ model. In their work, Zaid et al. introduce a methodology to create small but well-performing CNN architectures. For our experiment, we have used their CNN model created for the ASCAD_fixed dataset. This is also the model that was used by Zaid et al. to demonstrate the performance of the ranking loss function. Figure 4.19 shows the results in terms of guessing entropy, while Figure 4.20 shows the success rate.

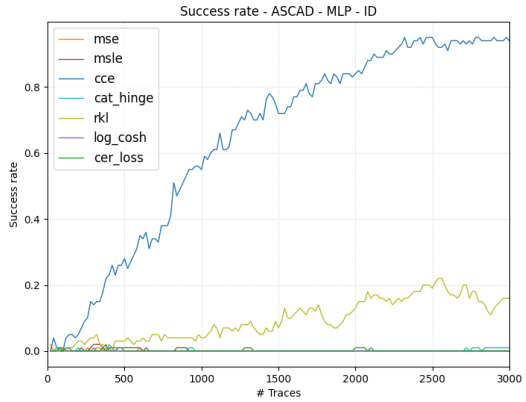
Since the CNN architecture was created specifically for the ASCAD_fixed dataset and ID leakage, it is no surprise that the performance is best in that scenario. Both the categorical cross-entropy and ranking loss are able to perform a successful attack with less than 200 traces. This is comparable to the performance of the model in the original paper [75]. However, in our experiments, the ranking loss did not outperform the categorical cross-entropy. In contrast with the ranking loss paper, the categorical cross-entropy performs slightly better. This difference in performance is better visible in Figure 4.20a. There the categorical cross-entropy reaches an SR of $1.0$ slightly earlier than the ranking loss model. One possible explanation of this difference with the paper by Zaid et al. is the way they present their results. They take the average $\overline{N}_{T_{GE}}$ over 10 converging models and compare those averages. In our experiments, however, we take the median of 10 models. In our experiments, the models with ranking loss are less consistent, e.g. there is a higher chance that one does not converge to a GE of $1$ for the correct key.

Looking at the performance on the other datasets, we see that the model is not very successful when the ID leakage model is used. Since the model is optimised for usage on the ASCAD datasets with the categorical cross-entropy in mind, this is no surprise. In the case of the HW leakage model, the models trained with CER loss outperform the other functions by quite a margin. They retrieve the correct key for the ASCAD_variable and CHES_CTF with less than 1500 and 2000 traces, where the models with other functions are not successful or need more than 3000 traces.

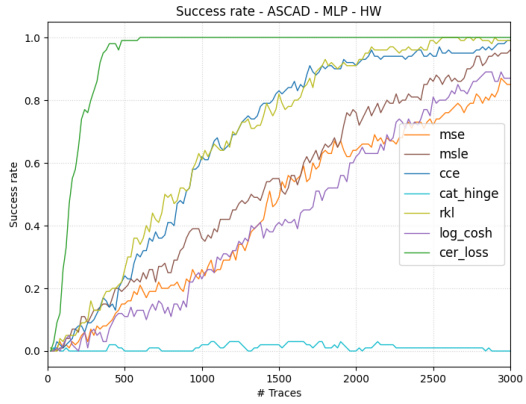
### 4.3.3 Countermeasures

In this subsection, we will discuss the results of the experiments with the ASCAD_plain and ASCAD_desync50 datasets introduced in Section 2.3. These results are compared against the performance of the same models on the ASCAD_fixed dataset, described in Subsection 4.3.1. Figure 4.21 shows the performance of the optimised MLP and CNN models in terms of guessing entropy on the unprotected implementation, while Figure 4.22 shows the performance on the dataset with desynchronised traces.

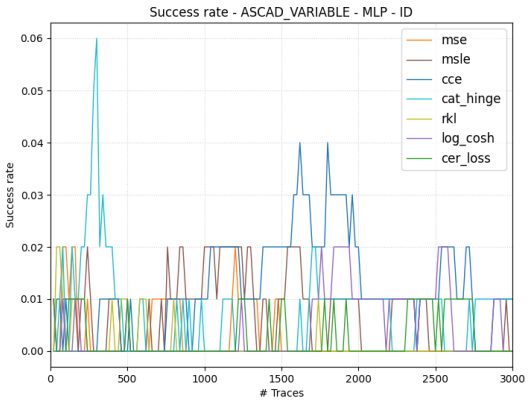
Since the ASCAD_plain dataset is equal to a set of measurements taken from a device running an unprotected version of AES, we see that every loss function except for the categorical hinge loss can retrieve the correct key in at most seven traces. This holds for both the ID and HW leakage, with both MLP and CNN models. For this unprotected implementation, it seems that every loss function besides the categorical hinge loss is a good choice.



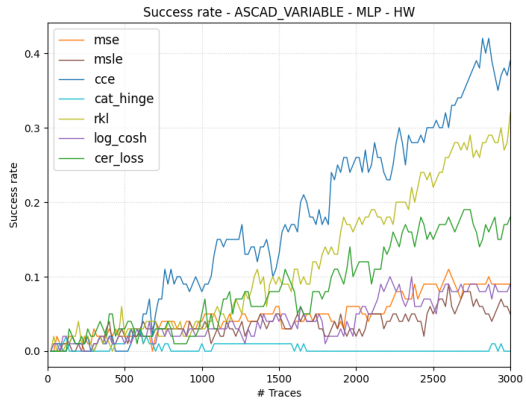

(a) ASCAD_fixed, ID leakage



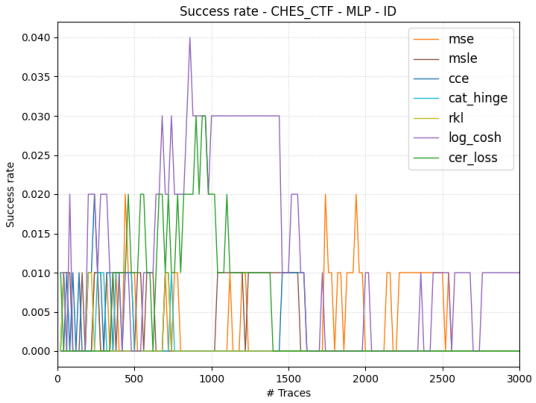

(b) ASCAD_fixed, HW leakage



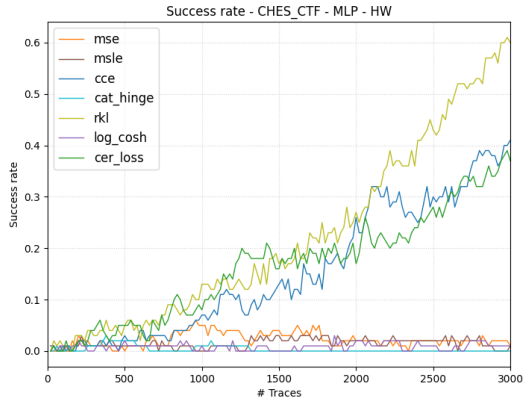

(c) ASCAD_variable, ID leakage




(d) ASCAD_variable, HW leakage



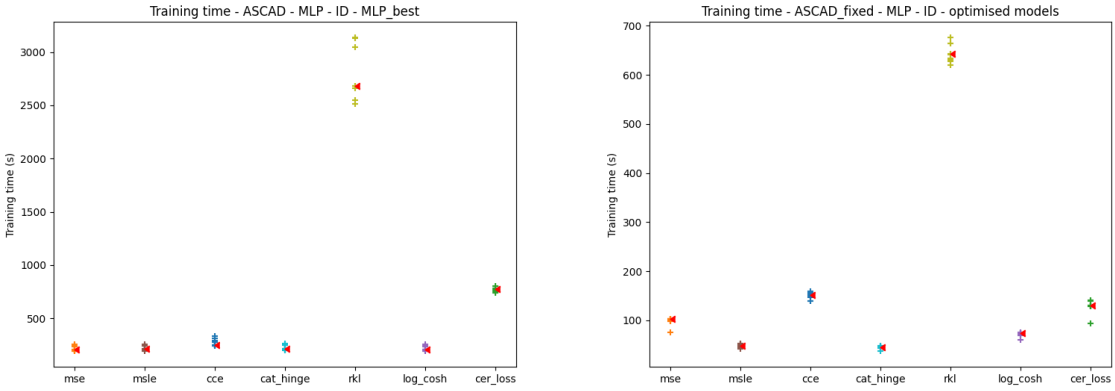

(e) CHES_CTF, ID leakage




(f) CHES_CTF, HW leakage

Figure 4.17: SR of the $MLP_{best}$ models on the ASCAD_fixed, -variable and CHES_CTF datasets.

The results when attacking the ASCAD_desync50 dataset, however, show different results. The extra countermeasure in the form of a random desynchronisation impacts the performance of the models heavily. The MLP models struggle the most. In the case of the HW leakage, only the model with CER loss seems to be somewhat converging. However, after 3000 traces, the GE is still only approximately




(a) $MLP_{best}$ models

(b) Optimised models

Figure 4.18: Training times for the $MLP_{best}$ and optimised models on the ASCAD_fixed dataset.

75. In the case of the ID leakage, the same holds for the categorical cross-entropy.

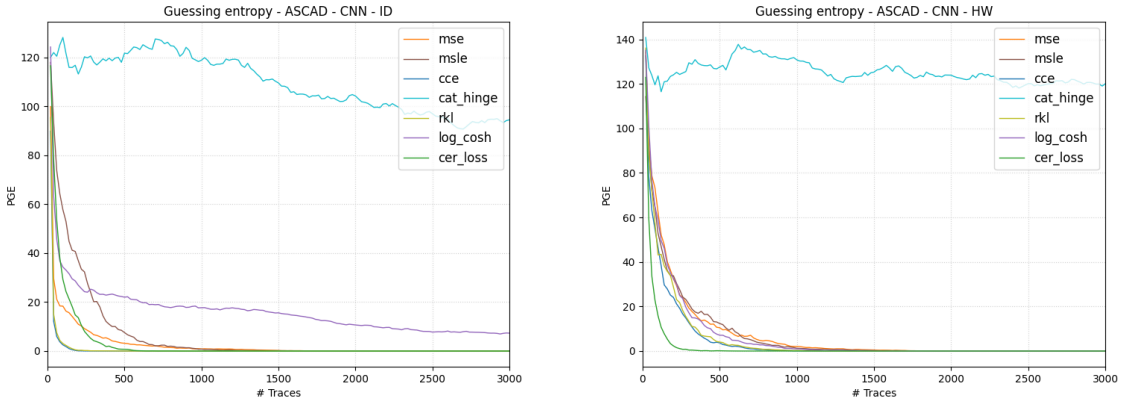
The CNN models seem to perform better than the MLP models on the desynchronised dataset. This is to be expected, as earlier work has shown the potential of CNNs against datasets with random desynchronisation added [4, 75]. In comparison to the earlier results on the ASCAD_fixed dataset, we now see that the additional trace desynchronisation countermeasure impacts models with some loss functions more than others. In the earlier experiments, we saw that the CER loss performed best. The categorical cross-entropy now performs best for both the ID and HW leakage, showing that it is more resilient to countermeasures. In fact, it is the only loss with which the attack against the ASCAD_desync50 dataset was successful. All MLP and CNN models trained with other loss functions fail to reach a GE of 1 with less than 3000 traces.

## 4.4 Discussion

With these experiments we have, for the first time in the SCA domain, performed a broad comparison of different loss functions in various deep learning-based SCA scenarios. Overall, these results reveal interesting characteristics of the behaviour of the different loss functions. In general, we see that the CER loss performs best in most of the experiments. Besides working well with the HW leakage, in many of the scenarios with the ID leakage, optimised models with CER loss also outperform models with other loss functions in our experiments. While Zhang et al. already demonstrate that CER loss might work on balanced data, our experiments confirm this for datasets often used in research. The other novel loss function proposed specifically for deep learning-based SCA, ranking loss, fared less well in our experiments. Besides being much slower to train than models with other functions, it only performed best in a single scenario. In all the other scenarios, CER loss or categorical cross-entropy are a better choice.

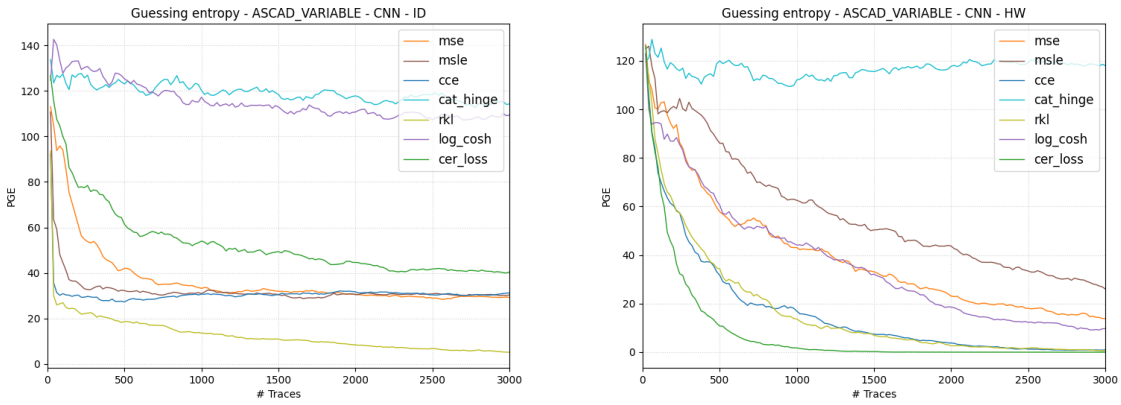
Furthermore, our work also shows that the categorical cross-entropy, often used by default in related works, is still a solid choice. It shows to be more robust to different hyperparameter choices than the two novel functions, performing well with almost any type of combination of hyperparameters within the hyperparameter search space we defined. In terms of guessing entropy and success rate, models with categorical cross-entropy are also often only second to the performance of CER loss. Besides that, our results show that it is faster to train and needs less complex models. In our experiments, it showed no obvious weaknesses.

The other loss functions we considered did in general not show promising results. While used before in other works, MSE and related loss functions such as MSLE and log cosh are almost always outperformed by the categorical cross-entropy and CER loss when an attack can be performed successfully. Besides that, MSE also does not have any significant benefits in terms of training time or model complexity, or robustness against countermeasures.



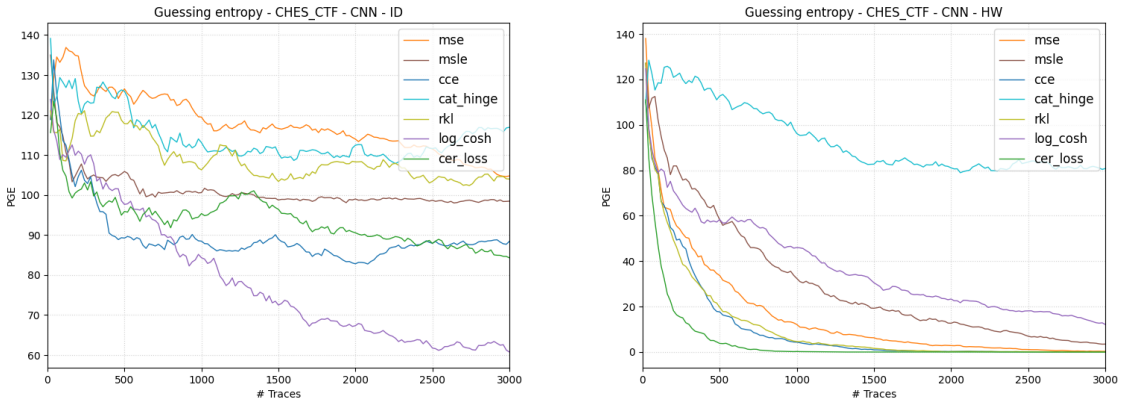

(a) ASCAD_fixed, ID leakage

(b) ASCAD_fixed, HW leakage




(c) ASCAD_variable, ID leakage

(d) ASCAD_variable, HW leakage

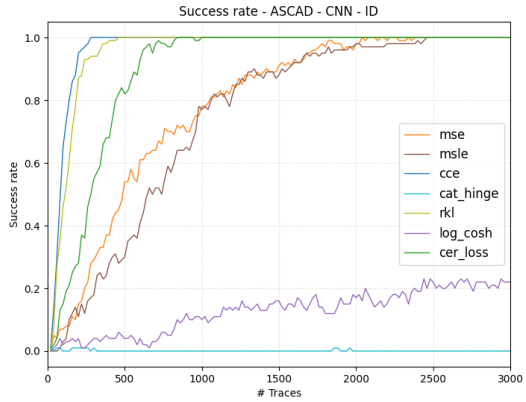



(e) CHES_CTF, ID leakage

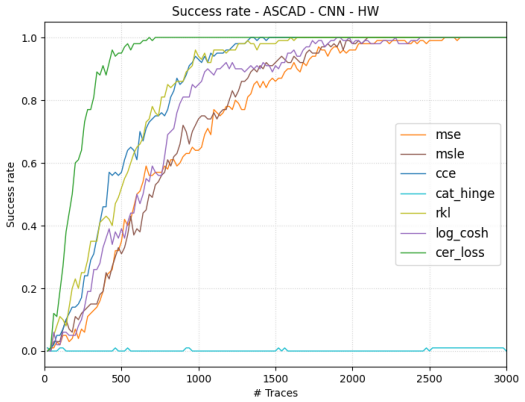
(f) CHES_CTF, HW leakage

Figure 4.19: Guessing entropy of the $CNN_{methodology}$ models on the ASCAD_fixed, -variable and CHES_CTF datasets.

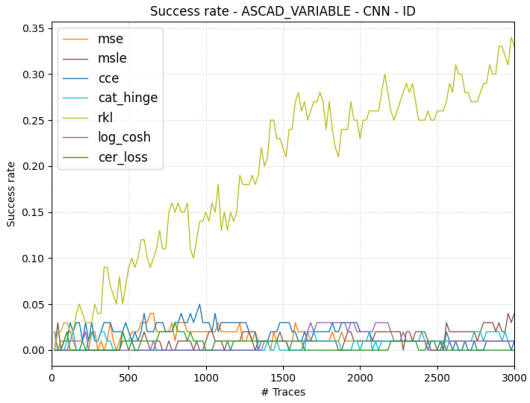
In our results, we saw no consistent differences between the behaviour of loss functions on MLPs or CNNs. In general, loss functions that performed well did so on both architecture types. If we look at the other hyperparameters involved, listed in Appendix B, we do see differences between the loss functions. Functions that perform well, such as categorical cross-entropy or CER loss, often do so with smaller models. The best performing models trained with the functions that, overall, do not perform well, are relatively complex. They require more neurons and dense layers, or more filters.



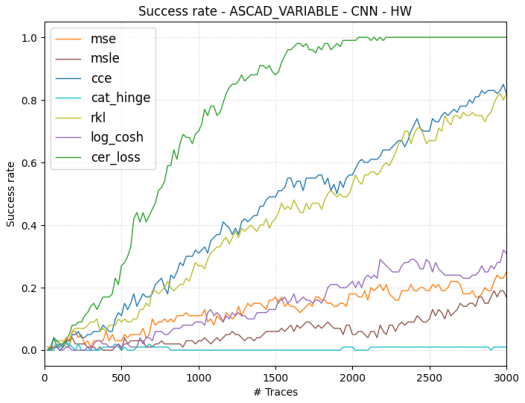

(a) ASCAD_fixed, ID leakage



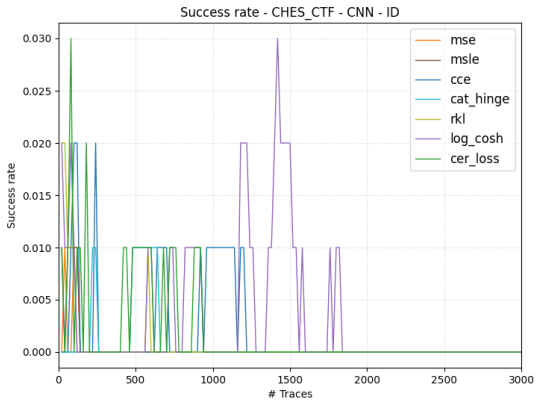

(b) ASCAD_fixed, HW leakage



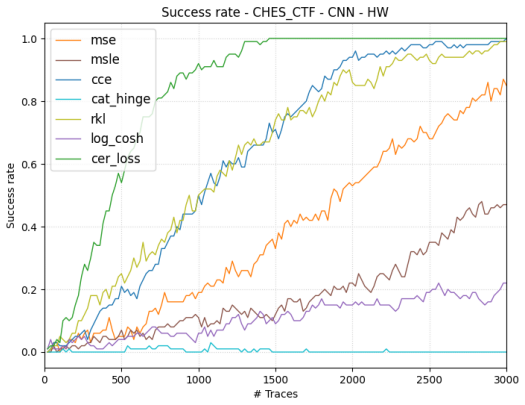

(c) ASCAD_variable, ID leakage




(d) ASCAD_variable, HW leakage




(e) CHES_CTF, ID leakage

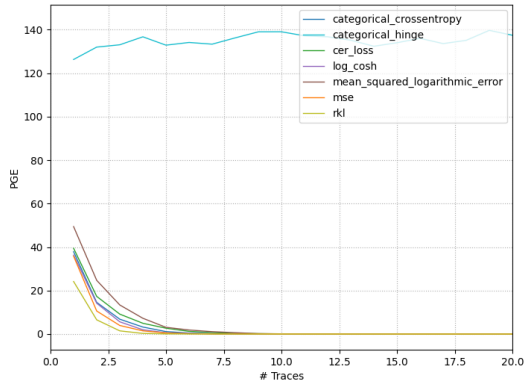



(f) CHES_CTF, HW leakage

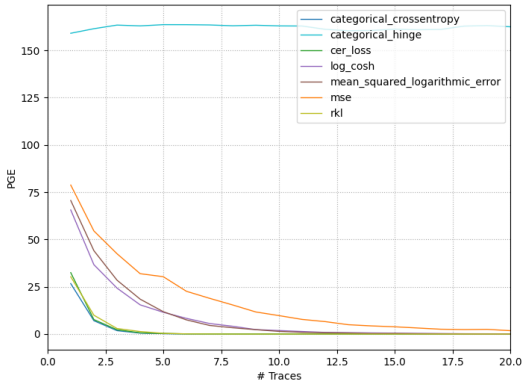
Figure 4.20: Success rate of the $CNN_{methodology}$ models on the ASCAD_fixed, -variable and CHES_CTF datasets.

Despite the more complex models, we do not see large differences in training times between the different functions. The only function that is significantly slower to train than others is the ranking loss. Especially when the ID leakage is considered and the amount of classes is high, the training time is increased by up to a factor of 10, as is visible in Figure 4.5. The CER loss is also slower when a larger $N$ is chosen. But as we have shown in Chapter 4, a larger $N$ is not required for better performing models.

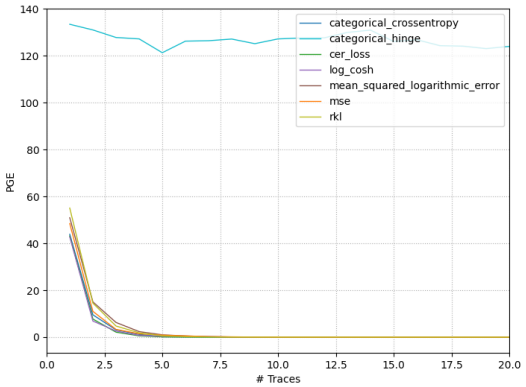
The goal of our work is to improve the tools that researchers have when performing SCA with deep



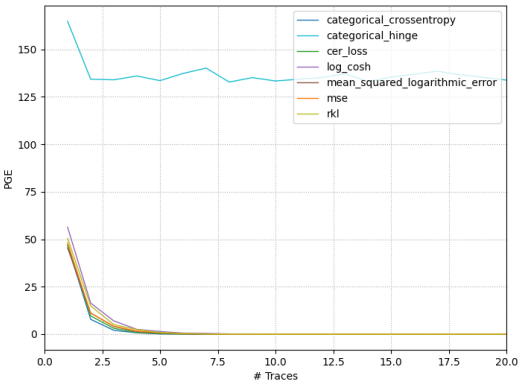

(a) ASCAD_plain, MLP models, ID leakage




(b) ASCAD_plain, CNN models, ID leakage




(c) ASCAD_plain, MLP models, HW leakage




(d) ASCAD_plain, CNN models, HW leakage

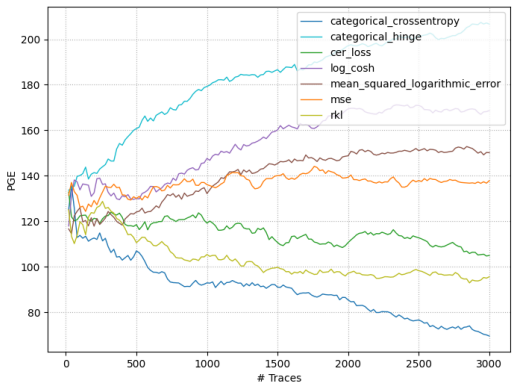
Figure 4.21: Guessing entropy of the optimised models on the ASCAD_plain dataset.

learning. To that end, we have created an overview of strengths and weaknesses for each loss function as seen in our experiments. These are visible in Table 4.7.

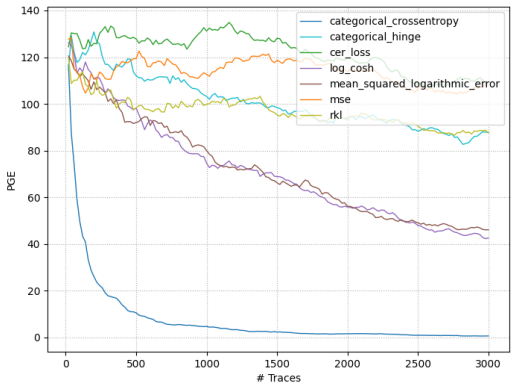
To summarise the result of the experiments in this chapter, we have tested each of the loss functions in a total of 44 different scenarios. Comparing the performance in terms of guessing entropy, models with the CER loss function performed best in 20 of those scenarios, the categorical cross-entropy in 8 and the ranking loss in 4. Other functions like MSE, MSLE or log cosh loss only performed best in scenarios where all models failed to perform a successful attack, e.g. reach a guessing entropy of 1 with less than 3000 traces. Our experiments also show that the categorical hinge loss is not very suitable for application in the SCA domain.

The models in the scenarios where other hyperparameters are optimised most often lead to a successful attack. When we compare the $\overline{N}_{T_{GE}}$ for all those models with CER loss and categorical cross-entropy, we see that 58 models with CER loss reached a guessing entropy of 1, with a median $\overline{N}_{T_{GE}}$ of 580. For categorical cross-entropy, 39 models were successful with a median $\overline{N}_{T_{GE}}$ of 1460. To test the significance of the difference in $\overline{N}_{T_{GE}}$ between the two functions, we perform a Mann-Whitney-U test [41]. We use $n_1 = 58, n_2 = 39$ and $\alpha = 0.05$. The calculated statistic $U = 246 < U_{crit} \approx 864$ with $p < 0.00001$ confirms that difference in $\overline{N}_{T_{GE}}$ is indeed significant. The probability that a model with CER loss has a lower $\overline{N}_{T_{GE}}$ than a model with categorical cross-entropy is $0.89$.

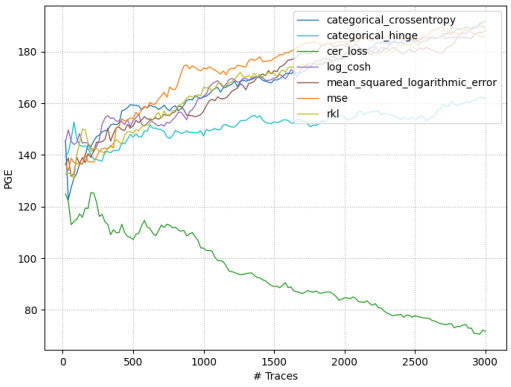
The other novel function, ranking loss, does not seem to increase the performance overall. Considering again all the optimised models, 33 models with ranking loss reach a $\overline{N}_{T_{GE}}$ of 1. The median $\overline{N}_{T_{GE}}$ of these models is 1620 in comparison to 1460 for the models with categorical cross-entropy. We again test the significance of the difference between these results with the Mann-Whitney-U test. There are



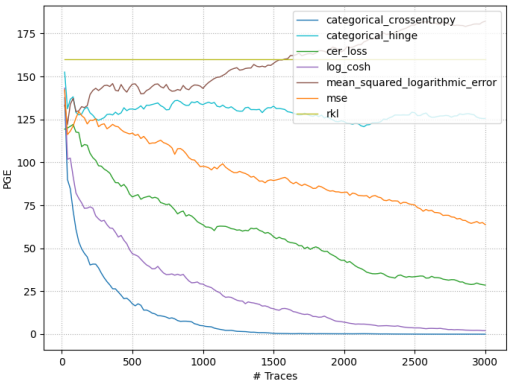

(a) ASCAD_desync50, MLP models, ID leakage




(b) ASCAD_desync50, CNN models, ID leakage




(c) ASCAD_desync50, MLP models, HW leakage




(d) ASCAD_desync50, CNN models, HW leakage

Figure 4.22: Guessing entropy of the optimised models on the ASCAD_desync50 dataset.

33 models with categorical entropy which were successful in attacking the considered datasets, and 58 with CER loss. We use $n_1 = 33$, $n_2 = 58$ and $\alpha = 0.05$. We calculate the statistic $U = 667.5 > U_{crit} \approx 470$ and $p = 0.79$. Since, $0.79 > 0.05$, we cannot say that the difference in performance between the ranking loss and categorical cross-entropy is significant. One explanation for this difference in performance might be the variation in the used architectures. The performance we see with the architecture and leakage model used by Zaid et al. in Figure 4.19a is similar to the performance presented in their paper [74]. However, in scenarios with other architectures and leakage models, the performance in comparison with the categorical cross-entropy shows no improvement. Besides that, using ranking loss also severely impacts the training time and introduces a new hyperparameter $\alpha$, which has to be optimised for each profiling dataset.

We can therefore conclude that the CER loss is, in most cases, the best choice for the loss function when using deep learning for SCA. The categorical cross-entropy is still a solid choice while ranking loss, or other loss functions, should only be considered in very specific cases.

Table 4.7: Strengths and weaknesses for each of the loss functions, based on the results of our experiments.

| **Loss function** | **Strengths** | **Weaknesses** |
|---|---|---|
| Categorical cross-entropy | • Good performance<br>• Robust to different architectures<br>• Robust against different countermeasures<br>• Training time | |
| Categorical hinge | • Training time | • Rarely leads to a successful attack |
| CER loss | • Best performance<br>• Specifically good against HW leakage model | • Less robust to different architectures |
| Log cosh | • Training time | • Performance is mediocre |
| MSE | • Training time | • Performance is mediocre |
| MSLE | • Training time | • Requires complex models<br>• Performance is not good |
| Ranking loss | • Performance in some specific scenarios | • Training time<br>• Not robust to different architectures<br>• Introduces new tunable parameter $\alpha$ |

# Chapter 5

# Multi-loss Functions

As we have seen in the previous chapter, the choice of loss function influences the performance of deep learning models when used for SCA. In this chapter, we will apply state-of-the-art loss functions from other domains to SCA. We will use hyperparameter optimisation via random search to find the best model for each of these functions and compare them to the categorical cross-entropy and the novel loss functions proposed for SCA.

## 5.1 Motivation

From our experiments so far, we can see that novel loss functions specifically developed for use in SCA do outperform more classical loss functions in certain scenarios. In other domains in which deep learning is applied, several works have also introduced new loss functions that improve the performance in that context [3, 19, 20, 36]. These functions are created to deal with certain characteristics of the targeted datasets, such as a class imbalance or a low amount of samples per class. First of all, we will consider focal loss [36]. Focal loss introduces two extra parameters: $\alpha$ and $\gamma$, as shown in the definition of focal loss in Equation 5.1.

$$L_{focal}(y,p) = -\alpha(1-p)^{\gamma} CE(y,p) \tag{5.1}$$

where $p$ is the predicted probability, $y$ is the true label and $CE$ is the categorical cross-entropy. The $\alpha$ parameter is a vector of weights used to balance the influence of samples from each of the classes. The $\gamma$ parameter is used to give easy examples, i.e. correctly classified with a high probability, less influence on the loss in comparison to hard examples. When $p$ gets larger, so when a sample is correctly classified with a larger probability, the contribution to the loss gets significantly smaller due to the $\gamma$. In the context of SCA, we also deal with imbalanced datasets when we are dealing with the HW leakage model. We will therefore use the focal loss in our experiments, denoted as $L_{focal}$.

As described in Chapter 3, multi-loss functions, which are combinations of two or more loss functions, have shown to increase the performance or robustness of the resulting deep learning models in other domains [3, 19, 20, 57]. The first of these functions we consider was introduced by Barz and Denzler. They use the cosine similarity and a combination of the cosine similarity and categorical cross-entropy as loss functions. They show that the increase in performance is largest when there are less than 200 samples per class. The datasets we have considered so far have between 135 and 230 samples per class when the ID leakage model is considered. To see if such a combination of losses can improve the performance in the context of SCA, we will also use this combination of losses in our experiments. We will denote this function as $L_{CosCross}$. The definition of this function is given in Equation 5.2.

$$L_{CosCross}(y,p) = 0.1 * CE(y,p) + COS(y,p) \tag{5.2}$$

where $p$ is the predicted probability, $y$ is the true label, $COS$ is the cosine similarity and $CE$ is the categorical cross-entropy.

Finally, Hajiabadi et al. use a combination of three loss functions for text classification and breast cancer prediction [19, 20]. According to Hajiabadi et al., these functions have different properties, such as increasing the margin between the correct class and other classes and robustness against outliers in the data. They identified three loss functions that have such specific characteristics. First, they use multi-class hinge loss, which is identical to the categorical hinge loss described in Subsection 2.2.2. Hinge losses also result in a (small) loss for correctly classified examples, thereby increasing the margin between the correct class and other classes. The second loss function they use is correntropy [68]. Correntropy bounds the loss between 0 and 1 for each sample, thereby making it more robust to outliers. The parameter $\sigma$, as seen in the definition of correntropy in Equation 5.3, is introduced to tune how quickly the loss goes to 1.

$$L_{corr}(y,p) = 1 - \exp(-\frac{||y-p||^2}{\sigma^2}) \tag{5.3}$$

where $p$ is the predicted probability and $y$ is the true label. The third function used is the categorical cross-entropy, as introduced in Subsection 2.2.2. [20] use a linear combination of these three functions, and use a training process to assign each of the functions a weight. For our experiments, we will use the weights they have found to work well [19]. The final combined loss function is shown in Equation 5.4.

$$L_{CorrCrossHinge}(y,p) = 0.29 * L_{corr}(y,p) + 0.36 * CE(y,p) + 0.34 * L_{hinge}(y,p) \tag{5.4}$$

where $L_{corr}$ is the correntropy function, $CE$ is the categorical cross-entropy and $L_{hinge}$ is the categorical hinge loss. We will denote this function by $L_{CorrCrossHinge}$.

## 5.2 Experiment Setup

In this section, we describe the setup of the experiments we have performed with these new loss functions. The experiments have been performed on the same hardware as described in Section 4.2.

For the first experiments with these new functions, we will use a similar approach as to the hyperparameter optimisation experiment described in Subsection 4.2.1. We will use the same three datasets, namely ASCAD_fixed, ASCAD_variable and CHES_CTF. Similarly, we attack both the ID and HW leakage models using CNNs and MLPs. For each of the new loss functions, we again perform hyperparameter optimisation via random search by generating 100 models with parameters randomly chosen from the ranges in Table 4.1 and Table 4.2. We select the best performing models per function in each scenario and compare them to the categorical cross-entropy, ranking loss and CER loss.

Two of the functions, $L_{focal}$ and $L_{CorrCrossHinge}$, introduce new hyperparameters. For these experiments, we will use the proposed values for each of these parameters[19, 36]. For $L_{focal}$, we will use $\alpha = 0.25$ and $\gamma = 2.0$. For $L_{CorrCrossHinge}$, we will use the weights for each function as shown in Equation 5.4.

Finally, we also look closer at the focal loss. In their work, Lin et al. tested several values for $\alpha$ and $\gamma$. They also propose setting $\alpha$ to the inverse class frequencies or optimising the parameters via a random search. Cui et al. propose a strategy for tuning class weights to be used in combination with for example focal loss [14]. For this experiment, we use the different strategies for setting the $L_{focal}$ parameters and compare them in terms of guessing entropy on the same scenarios as defined in Subsection 4.2.1.

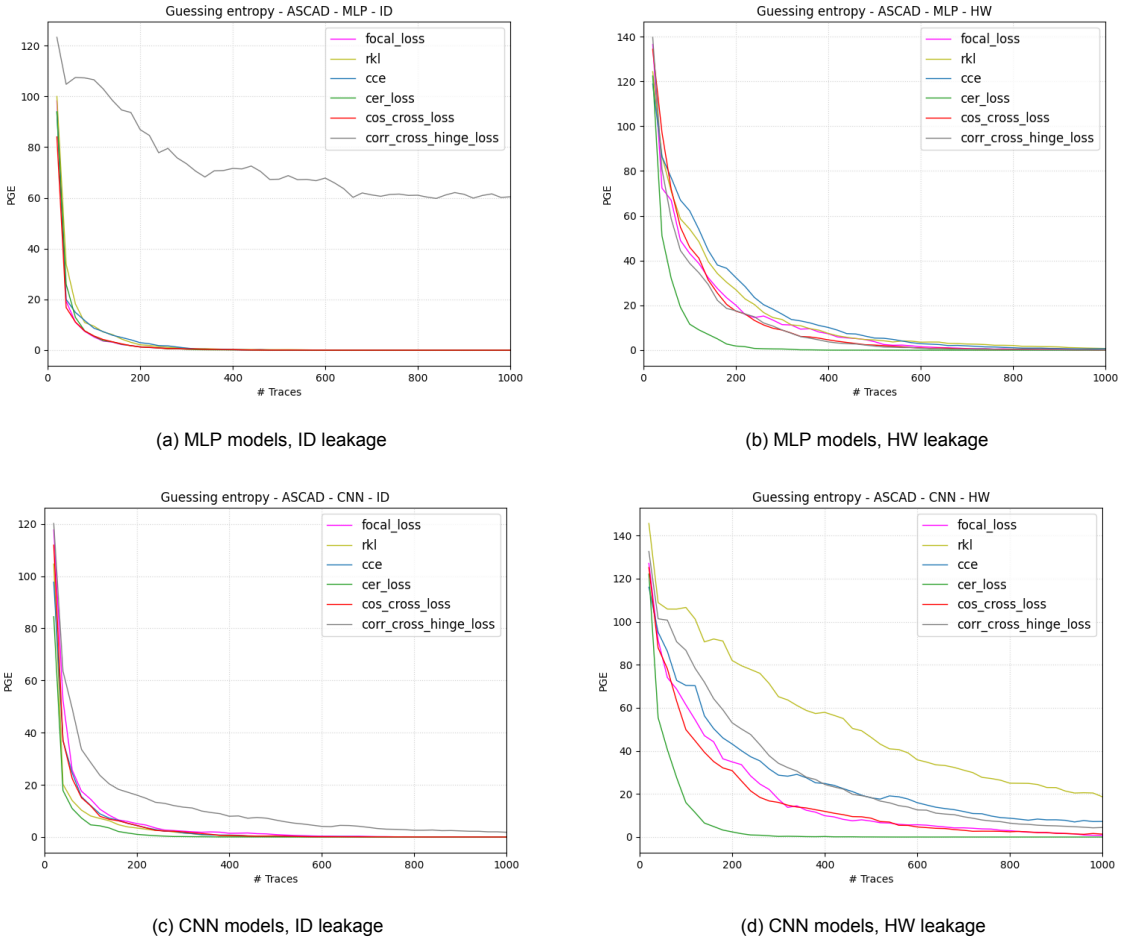
## 5.3 Results

In this section, we discuss the results of the experiments with $L_{focal}$ and the multi-loss functions $L_{CosCross}$ and $L_{CorrCrossHinge}$. After comparing the results of these functions on the three datasets, we will discuss the results of the experiments with $L_{focal}$ and different parameter settings.

### 5.3.1 Focal & Multi-loss Functions

#### ASCAD_fixed

First, we look at the guessing entropy and success rates on the ASCAD_fixed dataset, shown in Figure 5.1 and Figure 5.2.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

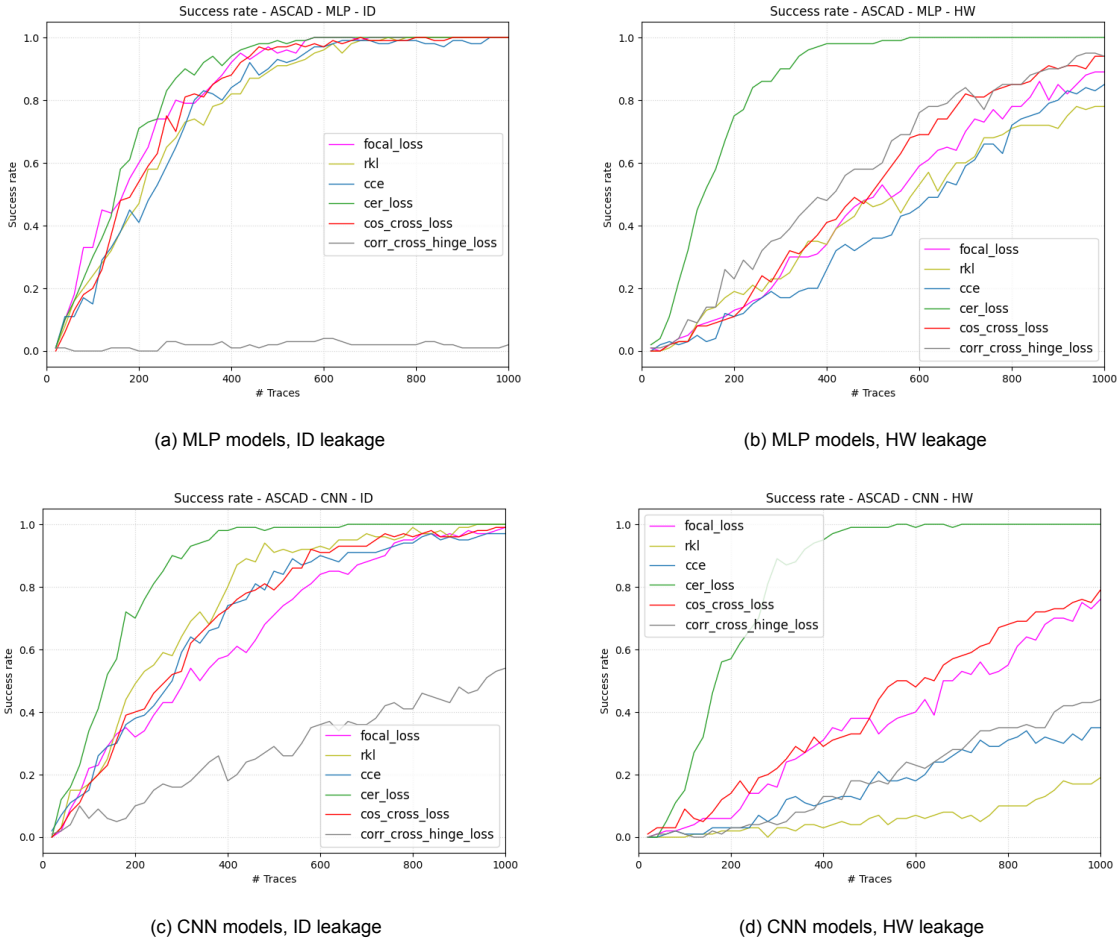
Figure 5.1: Guessing entropy of the optimised models on the ASCAD_fixed dataset.

Figure 5.1 shows that focal loss and multi-loss functions can be successfully applied to deep learning in SCA. In all the scenarios on the ASCAD_fixed dataset, the models with $L_{focal}$ and $L_{CosCross}$ outperform the categorical cross-entropy and ranking loss. Similarly, Table 5.1 shows the median $\overline{N}_{T_{GE}}$ for the functions in each of the scenarios. It is clear that using $L_{CosCross}$ and in particular $L_{focal}$ to train deep learning models leads to successful attacks. Both functions need fewer traces to successfully retrieve the correct key than models trained with categorical cross-entropy.

Table 5.1: Median $\overline{N}_{T_{GE}}$ on the ASCAD_fixed dataset. The lowest $\overline{N}_{T_{GE}}$ for each scenario is marked blue.

| | $L_{focal}$ | RKL | CCE | CER loss | $L_{CosCross}$ | $L_{CorrCrossHinge}$ |
|---|---|---|---|---|---|---|
| MLP ID | 580 | 900 | 860 | 570 | 610 | >2000 |
| MLP HW | 1480 | 1620 | 1560 | 560 | 1560 | 1450 |
| CNN ID | 1250 | 1760 | 1360 | 600 | 1060 | >2000 |
| CNN HW | 1840 | >2000 | >2000 | 540 | 1920 | >2000 |

The results for the other multi-loss function, $L_{CorrCrossHinge}$, are less consistent. While models




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 5.2: Success rates of the optimised models on the ASCAD_fixed dataset.

trained with this function still converge to a GE of 1 in most of the scenarios, it only performs well when the HW leakage is considered. In that case, it performs similarly to the categorical cross-entropy or slightly worse, which does not make it a good alternative.

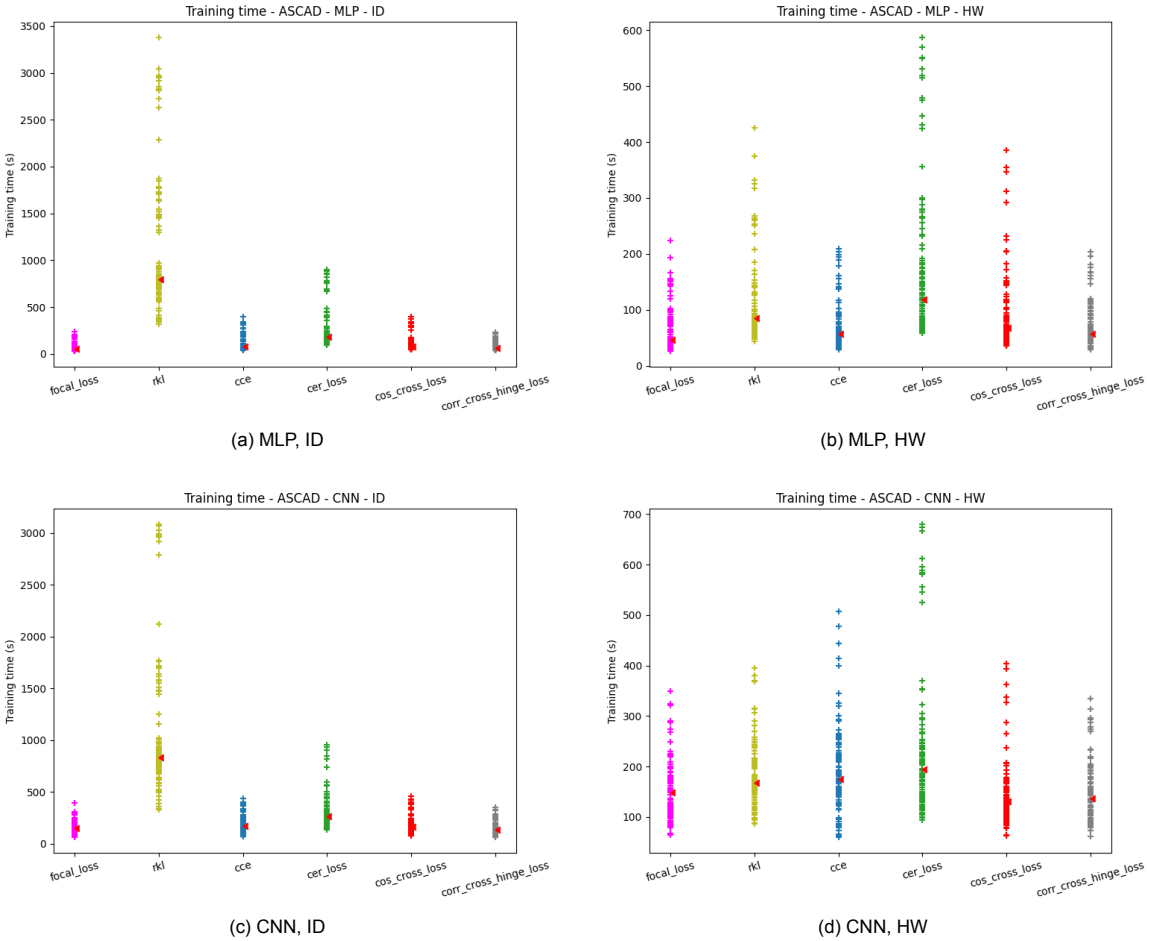
Finally, we also look at the training times with each of the losses. Since we have trained 100 random models with each of the functions, we can use those models to get an idea of how the loss functions impact the training time. Figure 5.3 shows the training times for each of the scenarios. The training times show that the multi-loss functions and focal loss are still as fast as categorical cross-entropy, and faster than the CER loss with $N = 10$ and the ranking loss.

Overall, the best performing function in terms of GE and SR for the ASCAD_fixed dataset is still the CER loss. The focal loss and $L_{CosCross}$ functions do show promising results. They do perform better than the categorical cross-entropy and ranking loss, and do not result in an increase in training time.

### ASCAD_variable

Next, we consider the results on the ASCAD_variable dataset. Figure 5.4 shows the guessing entropy of the optimised models with the different functions and Figure 5.5 shows the success rates.

The results on the ASCAD_variable dataset show a different picture in comparison with the ASCAD_fixed dataset. When the ID leakage model is used in the attack, the focal loss performs similar to the CER loss. After 3000 traces, focal loss reaches a median GE of 1.67 and CER loss 1.68 when MLPs are used. For CNNs, the GE when using 3000 traces is 4.00 and 3.13 respectively. They both easily outperform the categorical cross-entropy, which reaches a GE of 2.71 (MLP) and 17.59 (CNN). When the HW leakage is used, CER loss still outperforms the focal loss. However, considering the




(a) MLP, ID

(b) MLP, HW

(c) CNN, ID

(d) CNN, HW

Figure 5.3: Training times for 100 random models with each of the loss functions. Each mark is one of the 100 random models. The median training time for each function is marked with a red arrow.

increase in required training time, the focal loss is the better choice when the ID leakage is considered.

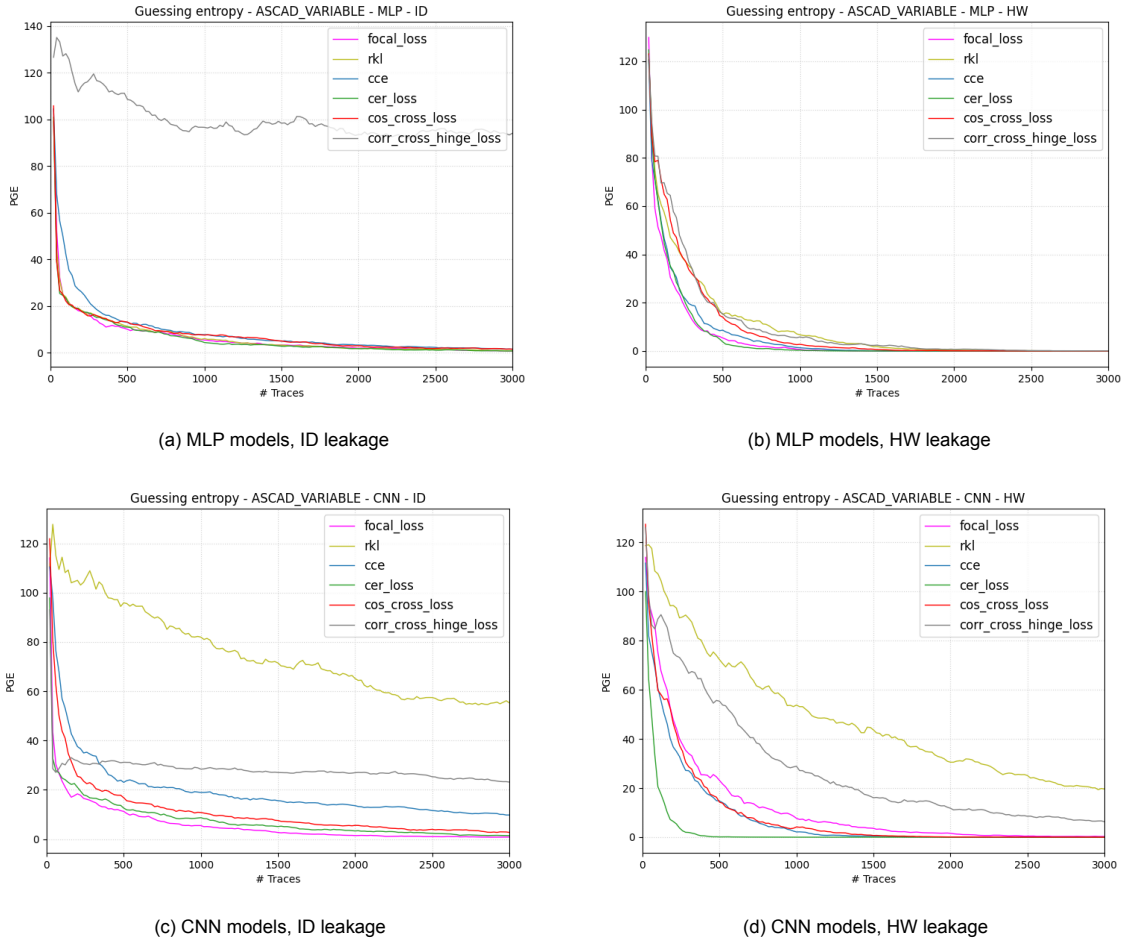
The multi-loss functions perform similarly to our experiments on the ASCAD_fixed dataset. The $L_{CosCross}$ function performs slightly better than the categorical cross-entropy with the ID leakage, but slightly worse when the HW leakage is used. The $L_{CorrCrossHinge}$ function does seem to be able to deliver converging models, but they do not perform very well.

In terms of the number of trainable parameters, there is no consistent difference between the loss functions. Table 5.2 shows the number of trainable parameters for each scenario and function.

Table 5.2: Number of trainable parameters of the optimised models on the ASCAD_variable dataset. The lowest number of trainable parameters for each scenario is marked blue, the highest number orange.

| | $L_{focal}$ | RKL | CCE | CER loss | $L_{CosCross}$ | $L_{CorrCrossHinge}$ |
|---|---|---|---|---|---|---|
| MLP ID | 3,924,256 | 371,856 | 1,830,756 | 1,966,656 | 2,797,456 | 572,856 |
| MLP HW | 1,366,009 | 1,477,709 | 402,609 | 2,079,909 | 603,609 | 1,045,209 |
| CNN ID | 3,365,584 | 4,591,236 | 3,869,540 | 1,602,260 | 3,869,524 | 158,308 |
| CNN HW | 81,501 | 3,634,445 | 3,844,221 | 937,253 | 3,6470,17 | 7,562,253 |

Similar to the ASCAD_fixed dataset, we can conclude that the focal loss is the preferred loss function when targeting the ID leakage model. The performance in GE is similar or slightly better than with CER loss, however, training times are much lower. Only when using CNNs to attack the HW leakage does




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 5.4: Guessing entropy of the optimised models on the ASCAD_variable dataset.

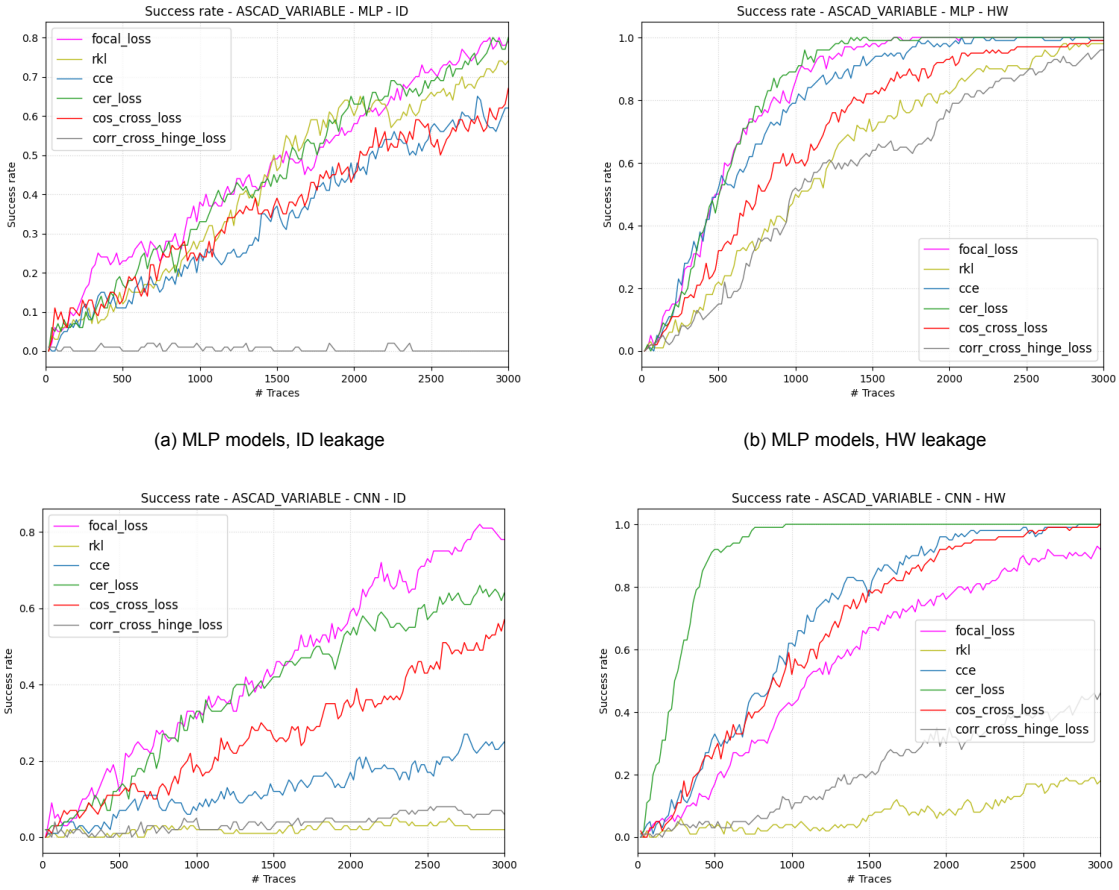
the CER loss outperform the focal loss, needing approximately 1200 traces less to launch a successful attack. In such a scenario, where fewer traces are available, the increase in training time could be a good trade-off.

### CHES_CTF

Finally, we will look at the performance of the functions on the CHES_CTF dataset. Figure 5.6 shows the guessing entropy in the different scenarios.

The results for the CHES_CTF dataset are again less consistent than those of the experiments with the ASCAD datasets. There is no single function that performs best in each of the scenarios. Interestingly, the $L_{CosCross}$ function does increase the previous best performance in the MLP/ID and CNN/HW scenarios. However, in the case of the CNN/HW scenario, the difference is marginal. In the MLP/ID scenario, the small increase is also still not enough to perform a successful key recovery with 3000 traces.

Overall, we can conclude that from the new tested functions, the focal loss has the most potential for use in the SCA context. When considering the ID leakage model, it performs similarly or better than our previous best function, the CER loss, and outperforms the regularly used categorical cross-entropy. It does so without the increase in training time that comes with using CER loss. When the HW leakage is considered, only the CER loss still outperforms the focal loss, but again at the cost of an increased training time. To summarise our findings for these new functions, we provide an addition to Table 4.7 for these new loss functions in Table 5.3




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

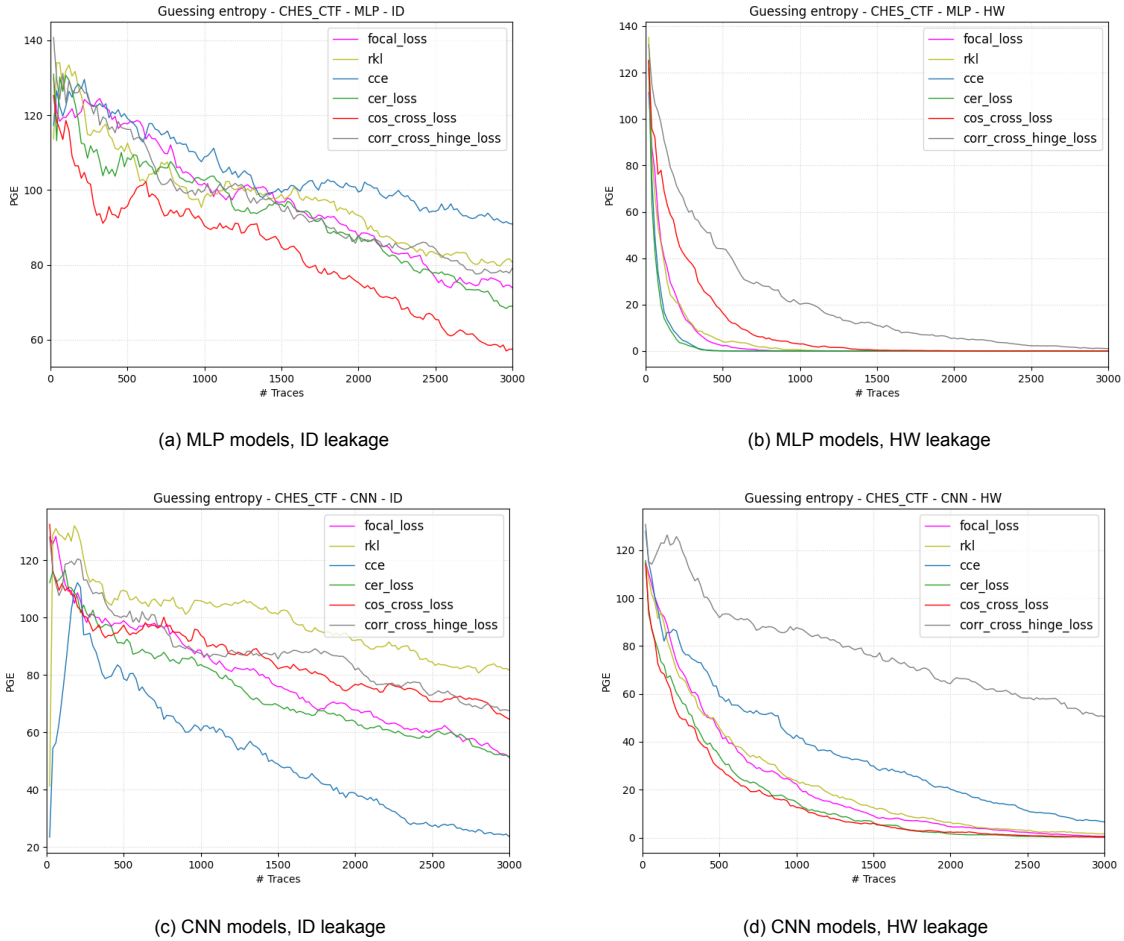
Figure 5.5: Success rates of the optimised models on the ASCAD_variable dataset.

Table 5.3: Strengths and weaknesses for each of the loss functions, based on the results of our experiments.

| Loss function | Strengths | Weaknesses |
|---|---|---|
| $L_{focal}$ | • Good performance, sometimes better than previous best<br>• Robust to different architectures | • Introduces new parameters |
| $L_{CorrCrossHinge}$ | | • Performance not very good<br>• Introduces new parameters |
| $L_{CosCross}$ | • Good performance | • Less robust to different architectures<br>• Introduces new parameters |

### 5.3.2 Tuning Focal Loss

In the previous section, we showed that we can successfully apply the focal loss, $L_{focal}$, to SCA scenarios. For those experiments, the parameters $\alpha$ and $\gamma$ were set to the values proposed in the original




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 5.6: Guessing entropy of the optimised models on the CHES_CTF dataset.

focal loss paper [36]. We have seen that the focal loss performs very well when the ID leakage is considered, but is still slightly worse than CER loss when the HW leakage model is used. In this section, we will look at the results of our experiments with two different strategies to set these parameters, to further improve the performance of the focal loss.

First, we optimise the choice of $\alpha$ and $\gamma$ via random search, as we do with the other hyperparameters. We choose these values from a similar range as Lin et al., as shown in Table 5.4. In our results, models which used this strategy for setting $\alpha$ and $\gamma$ are denoted as focal_optimised.

Table 5.4: Possible values for $\alpha$ and $\gamma$.

| | |
|---|---|
| $\alpha$ | [0.1, 0.25, 0.5, 0.75, 0.9] |
| $\gamma$ | [0, 0.5, 1.0, 2.0, 5.0] |

Next, we also use a strategy called class balancing [14]. With class balancing, the weights for each class, so the $\alpha$ parameter, are based on the number of samples in each class. For each class, the corresponding weight is calculated as shown in Equation 5.3.2.

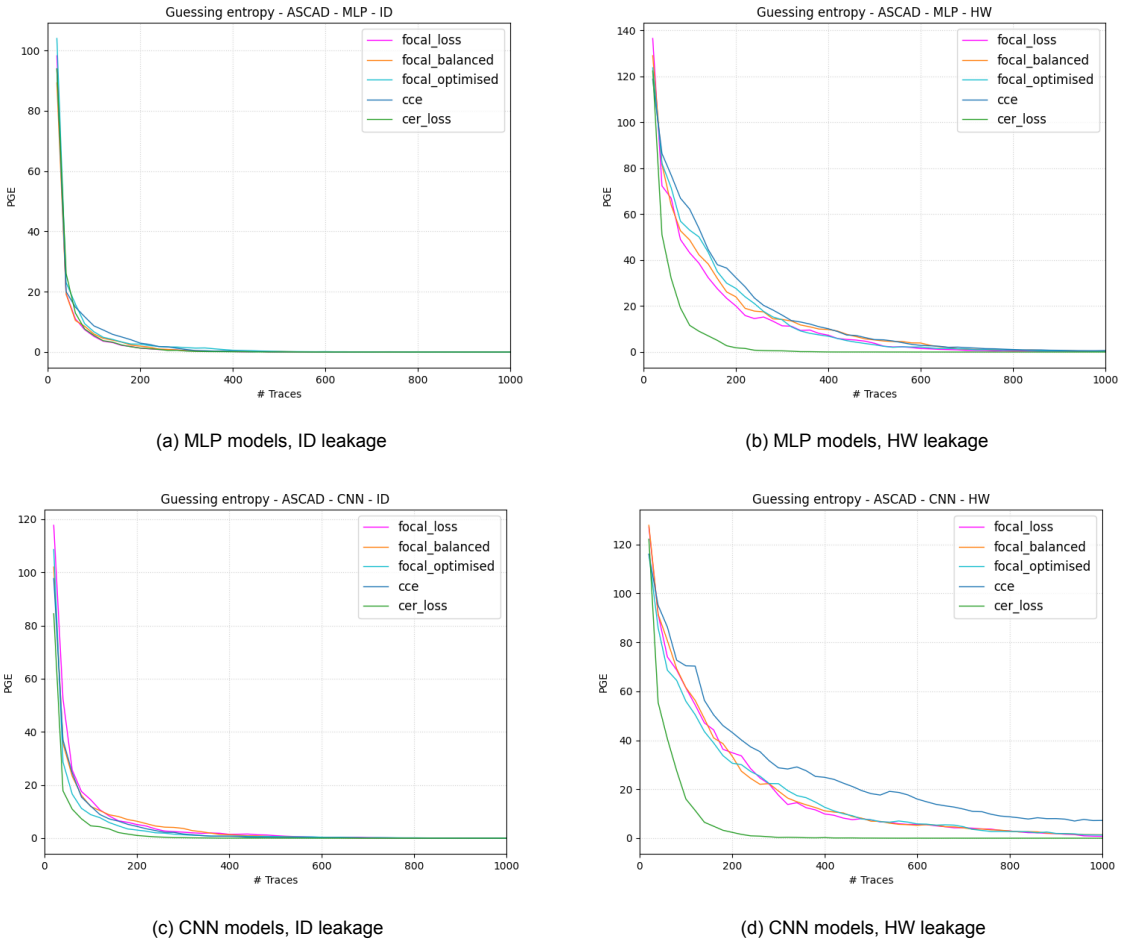
$$\alpha_i = \frac{1-\beta}{1-\beta^{n_y}} \tag{5.5}$$

where $\alpha_i$ is the weight for class $i$, $n_y$ is the number of samples in the considered class in the profiling set and $\beta$ is a new parameter. For this experiment, we used $\beta = 0.999$, since Cui et al. show that this value works well with different datasets. In our results, models trained with these settings are shown as focal_balanced.

For all the following experiments, we use 50,000 profiling traces for the ASCAD datasets and 45,000 traces for training on the CHES_CTF dataset. In the attacking phase, we use up to 2000 traces for the ASCAD_fixed dataset and up to 3000 traces for the ASCAD_variable and CHES_CTF datasets.

### ASCAD_fixed

First, we look at the performance on the ASCAD_fixed dataset. Figure 5.7 shows the guessing entropy of the three different focal losses, the CER loss and the categorical cross-entropy.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

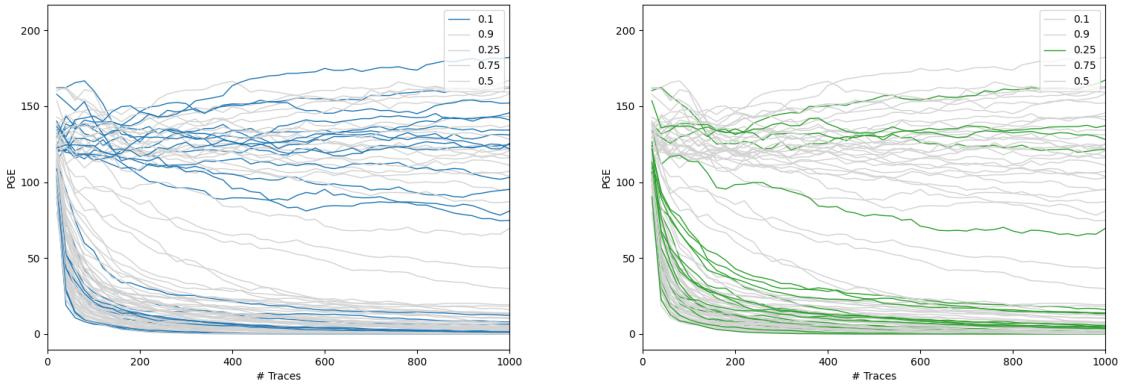
(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 5.7: Guessing entropy of the optimised models on the ASCAD_fixed dataset.

Here we see that optimising $\alpha$ and $\gamma$ for each scenario only slightly improves the performance over the default values of $\alpha = 0.25$ and $\gamma = 2.0$. If we look at performance for different $\alpha$ and $\gamma$ values, models with the default values perform consistently well. However, models with different values are less consistent but some outperform the ones with the default values. This effect is visible in Figure 5.8. For the MLP models, $\gamma$ tends to be smaller ($0$ or $0.5$), while the best performing CNN models often have a larger value ($2$ or $5$). The class balancing strategy for setting $\alpha$ does result in similarly performing models as with the default $\alpha = 0.25$ value. The performance of the focal_balanced loss is, just like the other focal losses, good on the ASCAD_fixed dataset.

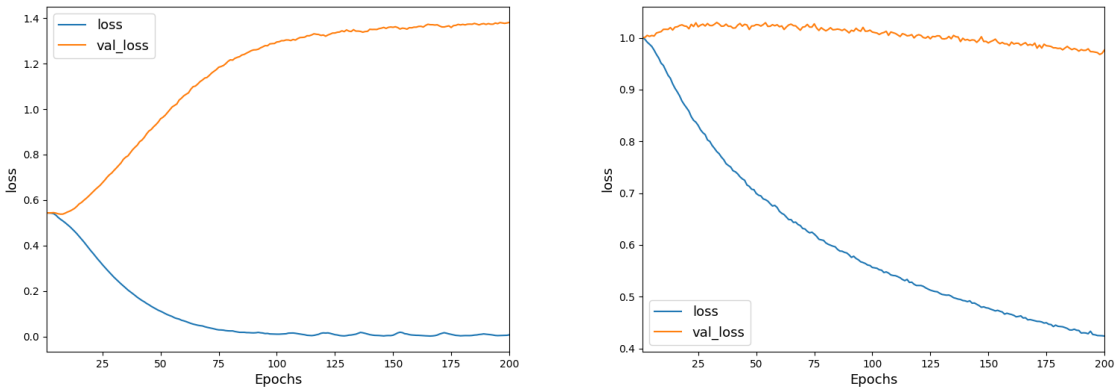
One other difference between the CER loss and the focal losses seems to be the speed of convergence of the loss towards $0$. For these experiments, similar to the previous experiments, we used 200 epochs for training. Figure 5.9 shows the progression of the loss per epoch during training for a similar model with each of the loss functions. For the focal loss, the default settings of $\alpha = 0.25$ and $\gamma = 2.0$ are used. Here we see that the focal loss reaches a loss of almost $0$ after approximately 100 epochs while the validation loss keeps increasing. The CER loss continues to decrease up to 200 epochs,




(a) $\alpha = 0.1$ (b) $\alpha = 0.25$

Figure 5.8: Guessing entropy for MLP models on the ASCAD_fixed dataset from 100 random models with focal loss and different values of $\alpha$. Here we see that models with the default $\alpha = 0.25$ perform consistently well, while models with $\alpha = 0.1$ are less consistent, but some of them are performing very good.

with the validation loss also decreasing. This could be an indication that the model with focal loss is over-fitting and that training with fewer epochs could result in better performance when using focal loss, while further training could increase the performance of with the CER loss.




(a) Focal loss (b) CER loss

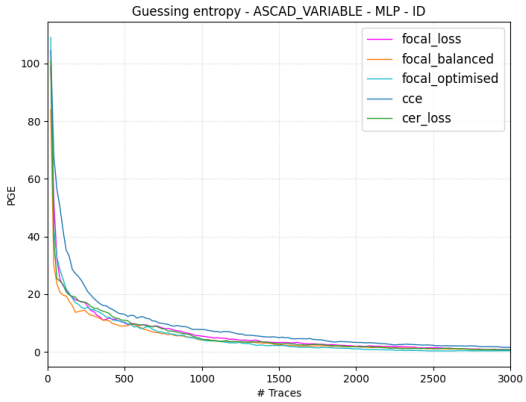
Figure 5.9: Loss value during training of the same model with CER loss and focal loss.

## ASCAD_variable

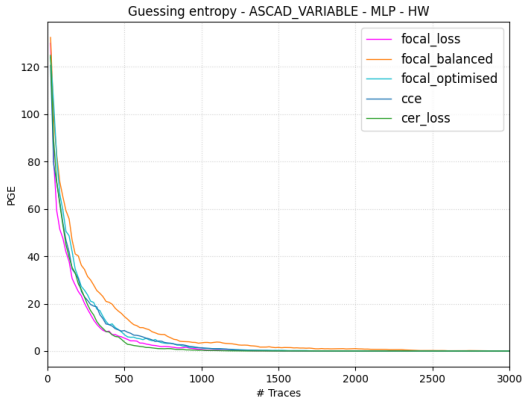
Next, we look at the results of the different focal loss settings on the ASCAD_variable dataset. Figure 5.10 shows the guessing entropy of the models in the different scenarios.

On the ASCAD_variable dataset, there is again little to no improvement with the different parameter tuning strategies in comparison with the default values. Interestingly, the class balanced focal loss outperforms the other functions slightly in the scenarios where ID leakage is used. The median GE when using 3000 traces for the attack is 1.46 (MLP) and 2.23 (CNN), which is lower than that of CER loss (1.69 and 3.13) and the focal loss with default parameter values (1.67 and 4.0). Although the difference is not large, it indicates that optimising the parameters via random search or class balancing can increase the performance.

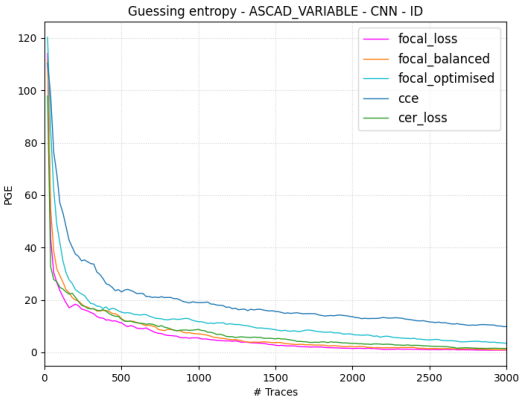
The difference in training time between the different parameter tuning strategies is negligible. Figure 5.11 shows the training time for the 100 models generated with each of the parameter tuning strate-



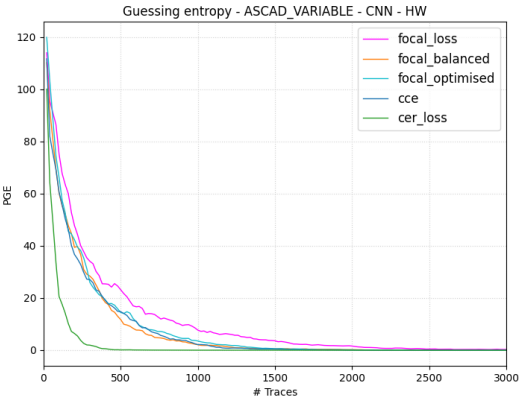

(a) MLP models, ID leakage




(b) MLP models, HW leakage



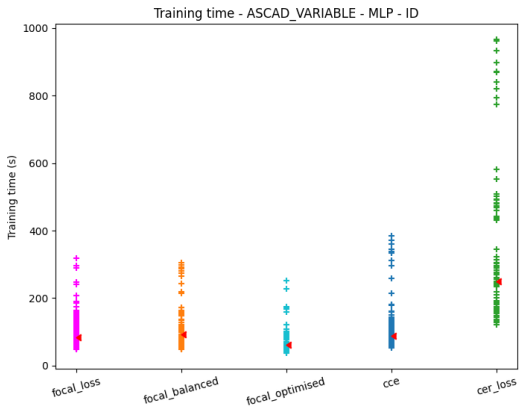

(c) CNN models, ID leakage



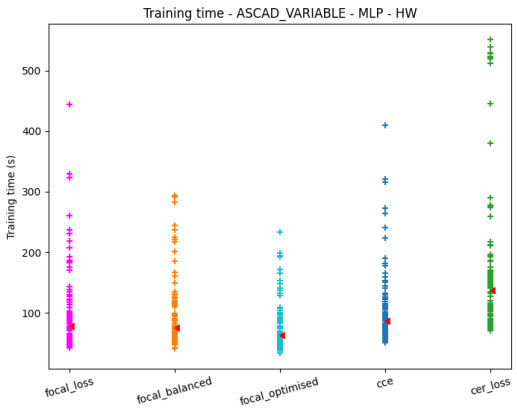

(d) CNN models, HW leakage

Figure 5.10: Guessing entropy of the optimised models on the ASCAD_variable dataset.

gies. There is no notable difference between the focal losses, multi-loss functions and the other loss functions except the ranking loss. This difference in training time is consistent across the datasets and scenarios.




(a) ID leakage model

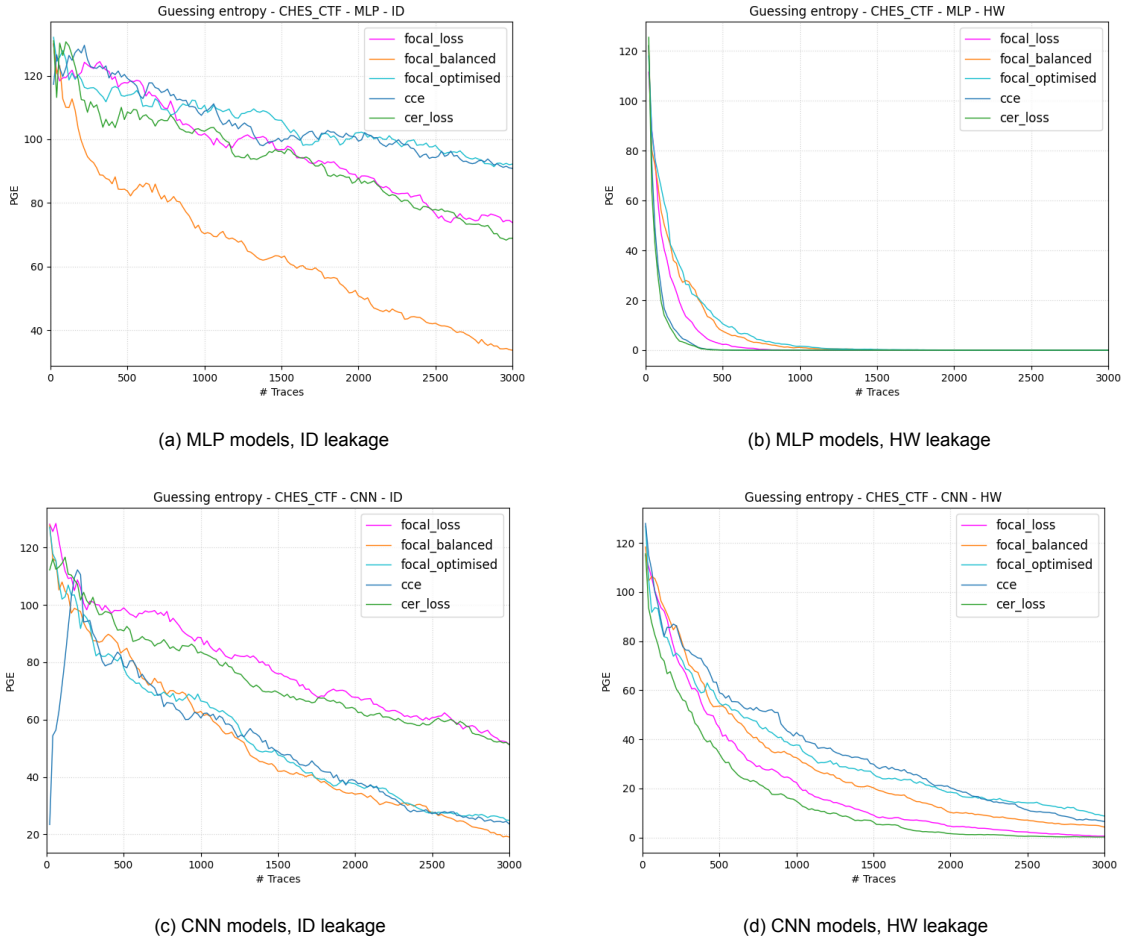



(b) HW leakage model

Figure 5.11: Training times of 100 models with each of the loss functions, trained on the ASCAD_variable dataset. The median training time per loss function is marked with a red mark.

### CHES_CTF

Finally, we look at the performance of the different focal losses on the CHES_CTF dataset. The performance in guessing entropy is shown in Figure 5.12. In comparison with the focal loss with default parameters, the gain in performance is quite significant on this dataset. For the MLP models with the ID leakage as a target, the class balanced focal loss improves the performance quite a bit in comparison with the default focal loss and every other loss. The median guessing entropy after 3000 traces is approximately 34, in comparison to 69 for CER loss and 74 for the default focal loss. Although this is an improvement, the success rate still only reaches 0.12 after 3000 traces, meaning it is still no successful attack with the considered amount of attack traces.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 5.12: Guessing entropy of the optimised models on the CHES_CTF dataset.

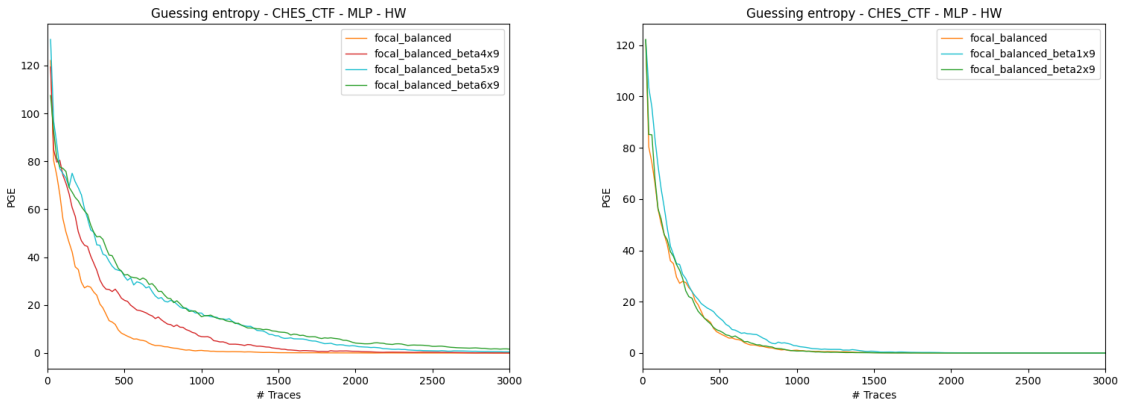
When the ID leakage and CNN models are considered, both the class balanced and optimising strategies improve the performance in comparison with the focal loss with default parameters. Again, the class balanced strategy performs best. Such improvements over the default values are not visible when using the HW leakage, which is interesting, since class re-balancing was proposed as a solution to class imbalances [14]. The cause of this is likely to be the choice of $\beta$ described in Subsection 5.3.2. Due to the small class imbalance in the dataset when the ID leakage is considered, the $\beta = 0.999$ results in small and almost constant $\alpha_i$ values for each of the classes. As shown in Figure 5.8, models with constant and small values of $\alpha$ can perform very well. When the HW leakage is considered, the difference between the class frequencies is larger. With $\beta = 0.999$, the smallest $\alpha_i$, for the most frequent class, is $0.001$, while the largest $\alpha_i$ is only $0.007$. The difference between these weights is not enough to solve the imbalance between the classes. Using a larger $\beta$ value does remove the imbalance, because as $\beta$ goes to $1$, the weights for each class approach the inverse class frequency. For the HW leakage model on the CHES_CTF dataset, this leads to $\alpha_i \approx 0.006$ for the smallest class

and $\alpha_i \approx 0.00008$ for the largest class. The weights and effective contribution to the loss of each trace for different values of $\beta$ are shown in Table 5.5. By choosing such a larger $\beta$, the traces of the minority

Table 5.5: $\alpha$ and the effective contribution of each of the traces to the loss. When using a smaller value for $\beta$, the imbalance between the classes remains. Using a larger $\beta = 0.999999$ makes every trace approximately have the same contribution to the loss.

| **Traces per class** | $\beta = 0.999$ | | $\beta = 0.999999$ | |
|---|---|---|---|---|
| | $\alpha$ | Effective | $\alpha$ | Effective |
| 168 | 0.0065 | 1.092 | 0.00595 | 0.996 |
| 1353 | 0.0013 | 1.759 | 0.00074 | 1.000 |
| 4958 | 0.0010 | 4.958 | 0.00020 | 1.002 |
| 9779 | 0.0010 | 9.779 | 0.00010 | 1.007 |
| 12286 | 0.0010 | 12.286 | 0.00008 | 1.005 |
| 9849 | 0.0010 | 9.849 | 0.00010 | 1.005 |
| 4986 | 0.0010 | 4.986 | 0.00020 | 1.002 |
| 1471 | 0.0013 | 1.912 | 0.00068 | 1.000 |
| 150 | 0.0072 | 1.080 | 0.00667 | 1.000 |

classes have a very large impact on the loss in comparison to the traces from the majority class, while these traces might in general not be more important. The imbalance between the classes is caused by transforming the original labels to their Hamming weight, the original dataset does not contain a large imbalance between classes. So while using a larger $\beta$ value does solve the class imbalance caused by using the HW leakage, it might not lead to a more successful attack. To verify this, we have tested the MLP model with the HW leakage model and class balanced focal loss with different $\beta$ values. Figure 5.13 shows the performance with several different $\beta$ values. It is clear that increasing $\beta$ does not increase the performance of the model, but only decreases it. Choosing a smaller value, leaving the class imbalance largely intact, also does not increase the performance. The value chosen based on the work of [14], $\beta = 0.999$, works best when using a class balanced focal loss for SCA.




(a) (b)

Figure 5.13: Class balanced focal loss applied to the CHES_CTF MLP HW leakage scenario. Several different values of $\beta$ are used. Class balanced focal loss with $\beta = 0.999$ is denoted by focal_balanced, beta1x9 denotes $\beta = 0.9$, beta2x9 for $\beta = 0.99$, etc.

## 5.4 Discussion

In this section, we have shown for the first time that multi-loss functions can be used to successfully perform SCA. In several works introducing new loss functions in other domains, the used datasets show

similarities with datasets used in the SCA domain. By using combinations of functions that proved to be effective in these other applications of deep learning, we can perform successful attacks against datasets commonly used for SCA. Although the performance with these functions was not always better than with the commonly used categorical cross-entropy or novel CER loss, it shows that combinations of loss functions have the potential to be successful in the context of SCA. In this work, we have only considered two multi-loss functions that had already been used in other applications. Future work could for example look into using different combinations of functions, or optimising for example the weights for each function.

Furthermore, we have for the first time applied a focal loss to SCA. Our first experiments show that without optimising the parameters of the focal loss, we can already successfully use it in the context of SCA. Models trained with focal loss perform consistently good and almost always better than the categorical cross-entropy. They do so without the increase in training time introduced by using the CER loss. This shows that in most cases, the focal loss should be preferred over the categorical cross-entropy. The CER loss offers a further increase in performance when using the HW leakage, but is slower, and should therefore only be used when training time is not an issue.

We have also looked into tuning the $\alpha$ and $\gamma$ parameters introduced by the focal loss. In Subsection 5.3.2, we show that different strategies can be used to set these parameters. When optimising each of these values together with other hyperparameters, the performance of models with focal loss can be improved in most scenarios. However, for this work, we have only looked at a limited range of values for $\alpha$ and $\gamma$. By increasing the search space for these values, better combinations might be found at the cost of increasing the time to optimise them.

Finally, we have also looked at a strategy called class balancing to set $\alpha$. This strategy improves the performance even further when using the ID leakage model, but a new parameter $\beta$ is introduced. We have shown that increasing $\beta$, to effectively solve the class imbalance caused by the HW leakage model, does not increase the performance of the resulting model. Choosing a lower value for $\beta$, thereby leaving some imbalance between the classes, works better. In that case, a similar increase in performance is seen for the scenarios with the HW leakage.

To conclude, we have successfully applied multi-loss functions and focal loss to deep learning for SCA. Models trained with focal loss almost always perform better than those with categorical cross-entropy, without an increase in training time. Further performance improvements can be gained by tuning the focal loss parameters via optimisation or class balancing. The focal loss should therefore be preferred over the categorical cross-entropy and CER loss when using deep learning for SCA and targeting the ID leakage. When the HW leakage is considered, the CER loss remains the best choice.

# Chapter 6

# Focal Loss Ratio

In this chapter, we discuss the proposal of a new loss function for SCA. We propose the focal loss ratio based on the lessons learned from our previous experiments. To verify the potential of the new function, we perform experiments on the same scenarios as before, comparing our new loss function to the previously best functions.

## 6.1 Building Blocks

In the previous chapters, we have seen various loss functions with different strengths and weaknesses. In all considered scenarios, either the CER loss or focal loss outperformed the default choice of categorical cross-entropy as a loss function. However, none of these two functions performs best in every scenario. The CER loss performs best in almost all scenarios with the HW leakage model, while in some scenarios with the ID leakage model, it is outperformed by the focal loss. In this section, we look further into what makes these functions successful and how we can use those characteristics to construct a new loss function. The goal is to create a single new loss function that performs best in all the considered SCA scenarios, without increasing the required training time or the complexity of the models.

### 6.1.1 Ratio Loss

The cross-entropy ratio loss, or CER loss, is introduced by Zhang et al. to improve the performance of deep learning models in the SCA domain. They start with a metric, the cross-entropy ratio, which purpose is to give an idea of the performance of a deep learning model. Classical machine learning metrics like accuracy do not give an accurate representation of the performance of a model when SCA is considered against protected targets [60, 77]. On the other hand, launching practical attacks and averaging the key rank to estimate the guessing entropy is a computationally costly process. They show the relation between their new metric, the cross-entropy ratio, and guessing entropy and success rate. A key part of this is the estimation of the cross-entropy ratio. Zhang et al. define this estimation as shown in Equation 6.1.

$$\widehat{CER} = \frac{H_{N_P}(S_P, M_\theta)}{H_{N_P}(S_P^r, M_\theta)} \tag{6.1}$$

where $H_{N_P}$ is a function that implements the cross-entropy, $S_P$ is the set of profiling traces, $S_P^r$ is the profiling set with random labels and $M_\theta$ is the deep learning model. This estimation is used directly as a loss function, using the average of $n$ sets of traces with shuffled labels as the denominator.

Something similar was also done by Zhu et al.. They use the same principle, using the ratio between the correct label and incorrect labels together with the cross-entropy as a loss. The idea behind this is that with categorical cross-entropy, or negative log-likelihood, only the probability for the correct class (true label $y = 1$) contributes to the loss. For all other classes with $y = 0$, the contribution to the loss is $0$. By using the other classes in the calculation of the loss, either by taking the ratio before calculating

the cross-entropy or by using the cross-entropy on a set with shuffled labels, the performance of a model can be increased. We have seen in Chapter 4 and Chapter 5 that this is indeed the case in the context of SCA, especially when the HW leakage model is used.

For other loss functions we have considered in this work, this is not the case. For example, the mean squared error (MSE) between a vector with true labels, $y = [0, 0, 1]$ and predicted labels $\hat{y_1} = [0.1, 0.1, 0.8]$ or $\hat{y_2} = [0, 0.2, 0.8]$ is different: $MSE(y, \hat{y_1}) = 0.020$ and $MSE(y, y_2) = \hat{0}.027$. However, the categorical cross-entropy for both predictions would be the same. Therefore, using the categorical cross-entropy in such a way seems like a logical basis for our own loss function.

### 6.1.2 Focal Loss

Besides the categorical cross-entropy as a loss itself, the categorical cross-entropy is also used in the focal loss. As we recall, the definition of the focal loss used in this work is shown in Equation 6.2.

$$L_{focal}(y, p) = -\alpha(1-p)^{\gamma} CE(y, p) \tag{6.2}$$

where $CE$ is the categorical cross-entropy function. $\alpha$ is a vector of weights for each of the classes and $\gamma$ is the parameter that increases the loss for correctly classified examples that are hard, i.e. the probability is further from $1.0$, and decreases the contribution of easy correctly classified examples, i.e. with a probability close to $1.0$.

As we have seen in Chapter 5, such a focal loss performs very well in the context of SCA. Our experiments show that the focal loss with $\alpha$ set via the class re-balancing strategy [14] or a low constant value of $\alpha$ outperforms the categorical cross-entropy, especially on the more challenging ASCAD_variable and CHES_CTF datasets when the ID leakage is considered.

## 6.2 Focal Loss Ratio

The characteristics that make the CER loss and focal loss perform so well are thus identified as using a ratio between the categorical cross-entropy and the categorical cross-entropy on a profiling set with shuffled labels, and adding $\alpha$ and $\gamma$ to balance the classes and emphasise hard-to-classify examples. The ratio loss performs really well when the HW leakage model is considered, while the emphasis on hard training examples improves the performance when the ID leakage is considered. We propose to combine those two aspects into a single new loss function, the focal loss ratio (FLR). The purpose of this new function is to have a single loss function that is suitable for every SCA scenario and performs equally or better than the CER loss and focal loss. We define the FLR function as shown in Equation 6.3.

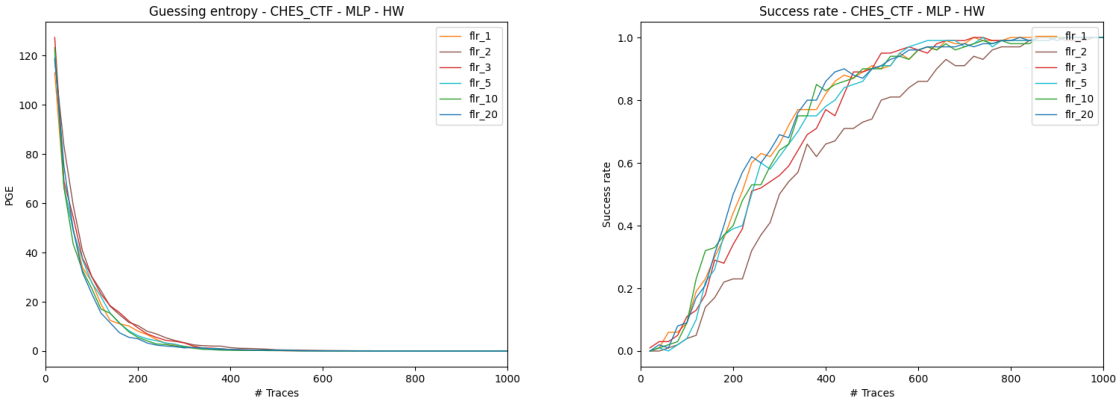
$$FLR(y, p) = \frac{-\alpha(1-p)^{\gamma} CE(y, p)}{\frac{1}{n}\sum_{i=1}^{n} -\alpha(1-p)^{\gamma} CE(y_s, p)} \tag{6.3}$$

where $y$ are the true labels, $y_s$ are the shuffled labels, $CE$ is the categorical cross-entropy and $n$ is the number of shuffled sets of labels to use.

For the CER loss function, the value of $n$ proved to have no noticeable influence on the performance of the resulting model, as shown in Figure 4.6. In Figure 6.1, we show the performance of the same model on the CHES_CTF dataset using FLR and different values of $n$. While the difference in $\overline{N}_{T_{GE}}$ is small, the median $\overline{N}_{T_{GE}}$ is lowest for the models with $n = 3$. The same small difference is also visible when tested on the other datasets. Since the impact on training time of using $n = 3$ in comparison to $n = 1$ is negligible, we will use $n = 3$ for our experiments with FLR.

## 6.3 Experiment Setup

In this section, we discuss the setup of the experiments we perform with the focal loss ratio function. We consider the same combinations of datasets, leakage models and architecture types as described




(a) Guessing entropy

(b) Success rate

Figure 6.1: Guessing entropy and success rate for FLR with different values of $n$.

in Subsection 4.2.1. We also use a similar approach, generating 100 random models with hyperparameters from the ranges defined in Subsection 4.2.1 and picking the best set of hyperparameters based on the guessing entropy. For the comparison, we train 10 models with those hyperparameters and select the median model. We compare our function against the CER loss, ranking loss, categorical cross-entropy and focal loss. We look at the guessing entropy and success rate, training time and complexity of the models.

For the FLR function, we use three different strategies to set the $\alpha$ and $\gamma$ parameters. For the first strategy, we use the values given by Lin et al., namely $\alpha = 0.25$ and $\gamma = 2.0$. Models with these settings are denoted as FLR in the plots. The second strategy is the class balancing approach introduced by Cui et al., denoted by FLR_balanced. The final strategy is optimising both $\alpha$ and $\gamma$ via random search. Models with that strategy are denoted by FLR_optimised. For this strategy, we select the parameters randomly for each of the 100 random models from the ranges defined in Table 5.4.

Besides that, we also test the new function in the context of countermeasures. Similar to Subsection 4.2.3, we compare the performance of models trained with FLR on an unprotected implementation of AES with those with masking and a random desynchronisation countermeasure. For this experiment, we use the ASCAD_fixed, ASCAD_plain and ASCAD_desync50 datasets.

## 6.4 Results

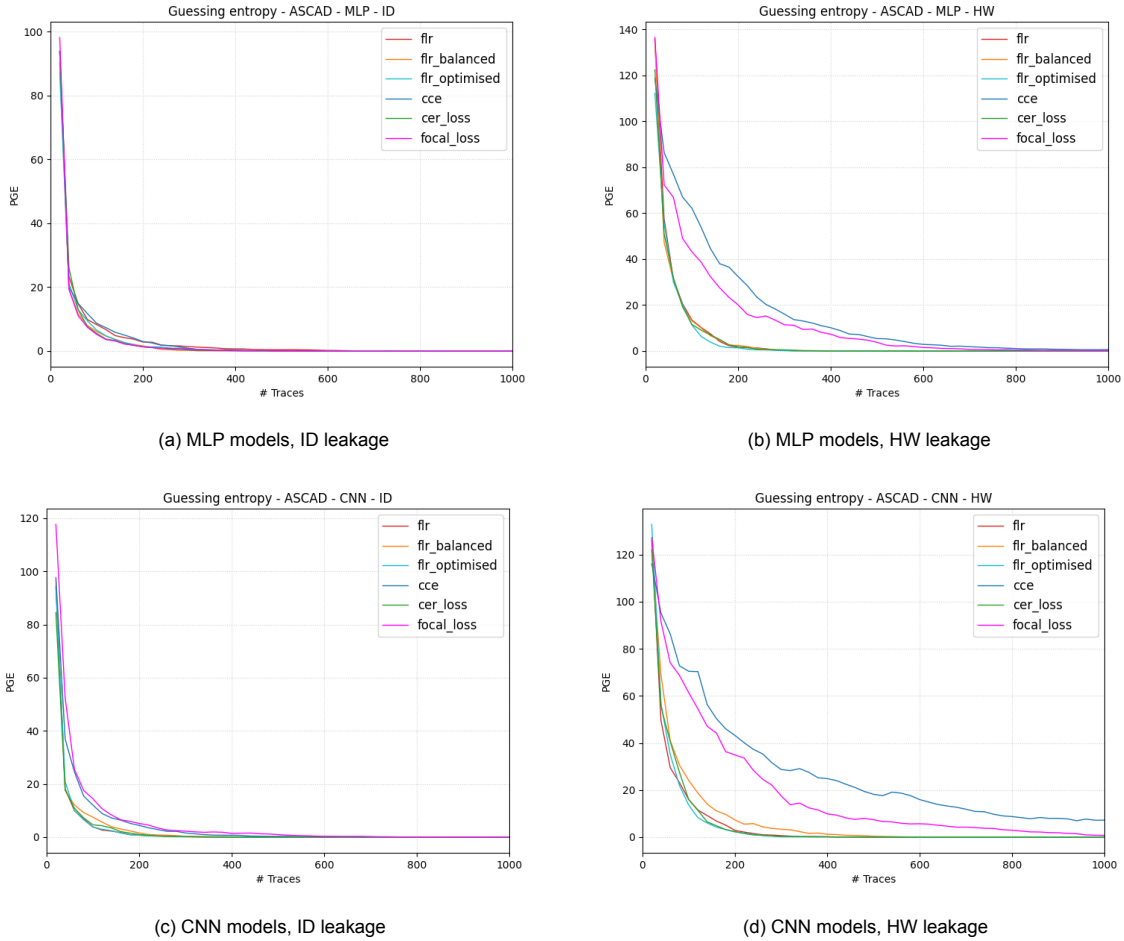
In this section, we discuss the results of the experiments with our newly proposed function, the focal loss ratio. First, we will consider the ASCAD_fixed dataset, then the ASCAD_variable dataset and CHES_CTF dataset. Finally, we will discuss the results on the datasets with and without countermeasures.

For our experiments on the ASCAD datasets, 50,000 profiling traces are used. For the CHES_CTF, we use 45,000 profiling traces. In the attacking phase, we use up to 2000 traces for the ASCAD_fixed dataset and up to 3000 traces for the ASCAD_variable and CHES_CTF datasets. In some of the plots, the x-axis is reduced to increase the visibility of the differences between the loss functions.

### 6.4.1 ASCAD_fixed

We start with the results of the models with different functions on the ASCAD_fixed dataset. Figure 6.2 shows the guessing entropy in the different scenarios while Figure 6.3 shows the success rates.

From the guessing entropy and success rate plots, it is straight away clear that models trained with FLR loss outperform the categorical cross-entropy and focal loss in each of the scenarios with the ASCAD_fixed dataset. When the HW leakage model is considered, the FLR models need 1000 to 1500 traces less than models with categorical cross-entropy or focal loss to reach a GE of 1.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

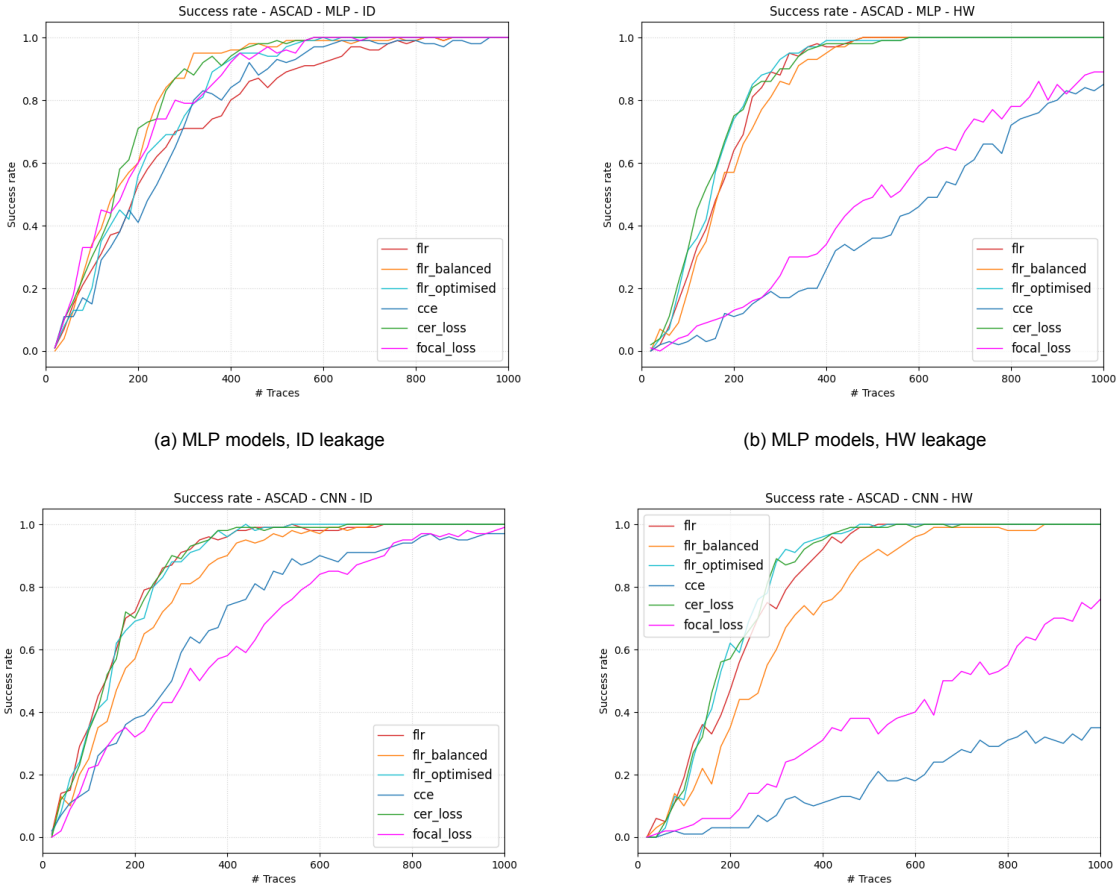
Figure 6.2: Guessing entropy of the optimised models on the ASCAD_fixed dataset.

The difference in performance with the CER loss function is smaller. In Table 6.1, we show the median $\overline{N}_{T_{GE}}$ for each of the functions and scenarios. Here we see that in three out of four of the scenarios, models trained with FLR outperform the CER loss models. In the fourth scenario, the CNN model with HW leakage, the models with FLR still perform almost as good as the CER loss. All three strategies for setting $\alpha$ and $\gamma$ work well and lead to successful attacks with relatively few traces. However, which strategy works best differs per scenario.

Table 6.1: Median $\overline{N}_{T_{GE}}$ on the ASCAD_fixed dataset. The lowest $\overline{N}_{T_{GE}}$ for each scenario is marked blue.

| | $L_{focal}$ | CCE | CER loss | FLR | FLR_balanced | FLR_optimised |
|---|---|---|---|---|---|---|
| MLP ID | 580 | 860 | 570 | 810 | 540 | 680 |
| MLP HW | 1480 | 1560 | 560 | 460 | 570 | 510 |
| CNN ID | 1250 | 1360 | 600 | 610 | 850 | 550 |
| CNN HW | 1840 | >2000 | 540 | 570 | 790 | 560 |

There is no significant difference in the training time between the compared loss functions except for the CER loss. Figure 6.4 shows the training times for the 100 random models with each of the loss functions. The slowdown for the CER loss is caused by the choice for $N = 10$ as discussed in Chapter 4. Choosing a lower $N$ would not have impacted the performance and would have brought down the training time. We also look at the complexity of models. Table 6.2 shows the number of trainable parameters for each of the loss functions and scenarios. Here we see that in general, the models trained with FLR are relatively small. In fact, we see that in the scenarios where FLR performs




(a) MLP models, ID leakage

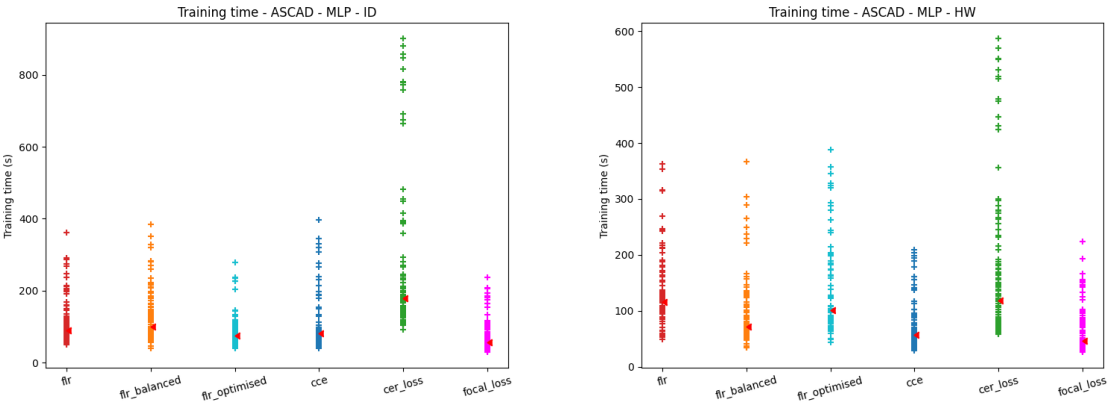
(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.3: Success rate of the optimised models on the ASCAD_fixed dataset.

well, it also uses small or the smallest models. This is an advantage, as it shows that it can outperform other loss functions while still using relatively small models.




(a) ID leakage

(b) HW leakage

Figure 6.4: Training time of 100 random models per loss function on the ASCAD_fixed dataset.

From our results on the ASCAD_fixed dataset, we can conclude that the FLR is a suitable loss

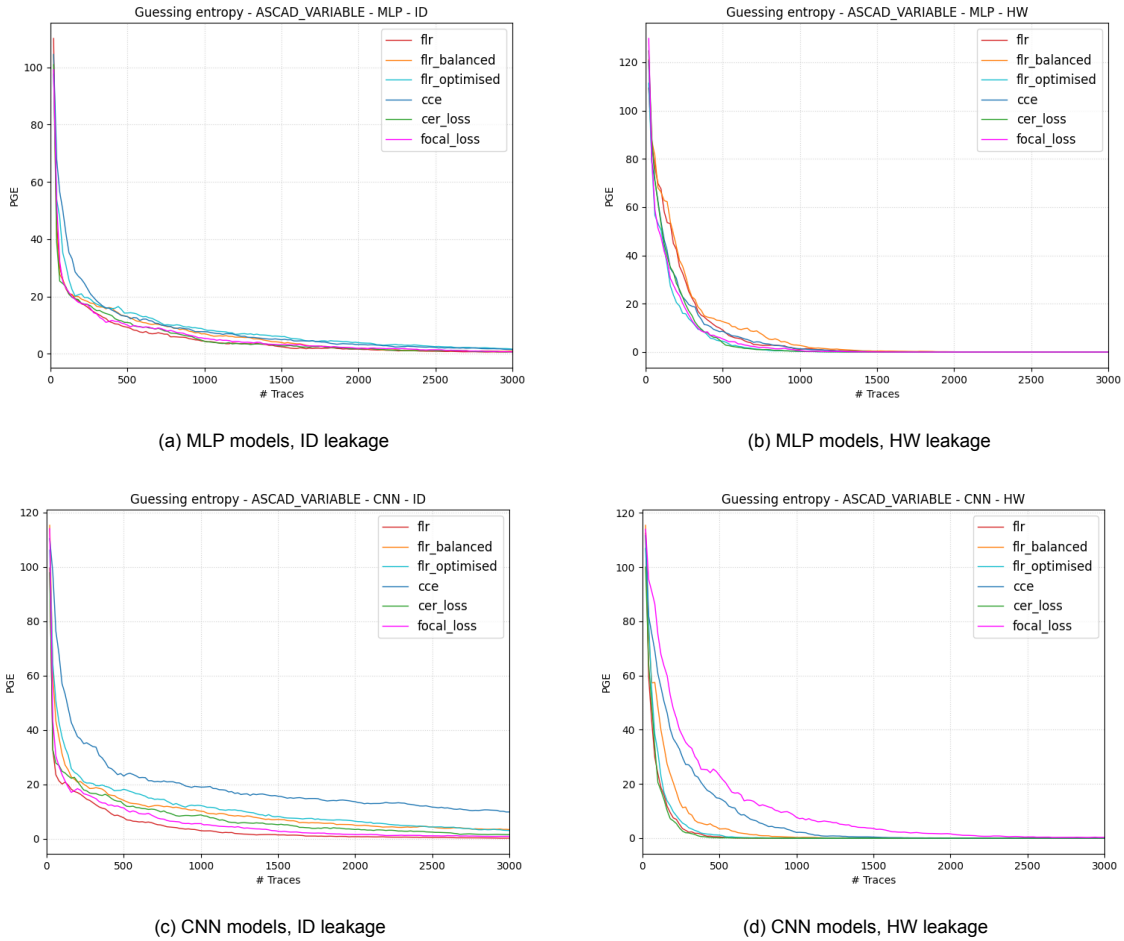
function to use for deep learning SCA. It outperforms the previously best loss function in most cases without negatively impacting training time or model complexity.

Table 6.2: The number of trainable parameters of the optimised models on the ASCAD_fixed dataset. The lowest number of trainable parameters for each scenario is marked blue, the highest is marked orange.

| | $L_{focal}$ | CCE | CER loss | FLR | FLR_balanced | FLR_optimised |
|---|---|---|---|---|---|---|
| MLP ID | 146,456 | 116,156 | 116,156 | 2,688,256 | 106,056 | 377,656 |
| MLP HW | 754,809 | 302,809 | 483,909 | 222,409 | 845,109 | 605,509 |
| CNN ID | 1,233,640 | 53,244 | 356,424 | 93,772 | 73,432 | 93,772 |
| CNN HW | 1,068,209 | 1,124,565 | 598,853 | 977,853 | 954,429 | 101,777 |

### 6.4.2 ASCAD_variable

Next, we look at the results on the ASCAD_variable dataset. We start with the guessing entropy and success rates in Figure 6.5 and Figure 6.6.




(a) MLP models, ID leakage

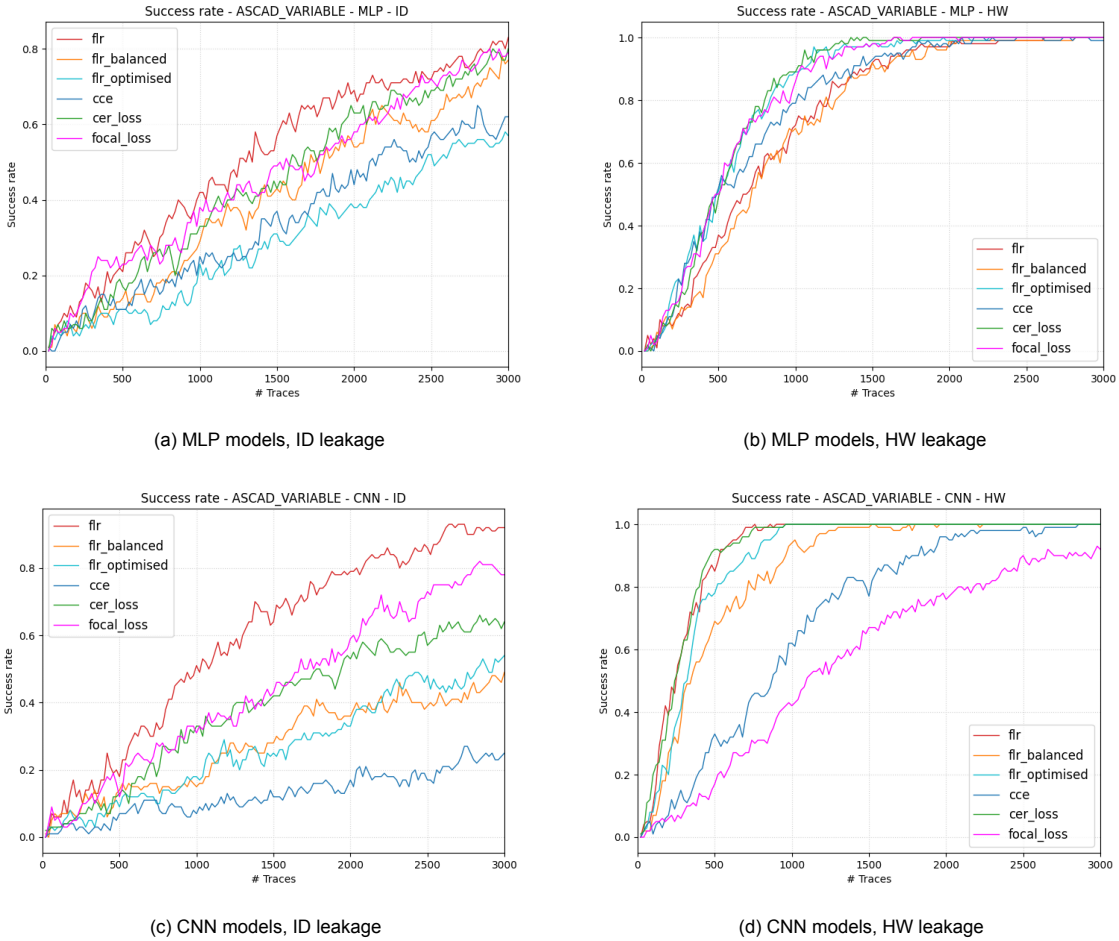
(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.5: Guessing entropy of the optimised models on the ASCAD_variable dataset.

The ASCAD_variable dataset is a more difficult dataset to attack than the ASCAD dataset with fixed keys. For the ID leakage, neither the MLP nor CNN models reach a GE of 1 with less than 3000 traces. If we look at the median GE when using 3000 traces for the attack, the difference between the CER loss and FLR is small. The CER loss reaches a GE of 1.7 with an MLP model and 3.13 with the CNN model, while the models with FLR reach 2.11 and 1.18. When the HW leakage is considered, the key is




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.6: Success rate of the optimised models on the ASCAD_variable dataset.

retrieved successfully by models with any loss function. Again, when MLPs are considered, CER loss outperforms models with FLR: $\overline{N}_{T_{GE}} = 1340$ versus $\overline{N}_{T_{GE}} = 1800$. And similarly, CER loss performs slightly worse when CNNs are considered: $\overline{N}_{T_{GE}} = 950$ versus $\overline{N}_{T_{GE}} = 800$. If we look at the success rates in Figure 6.6, we do see that the FLR model reaches a higher SR slightly faster than the other loss functions in both the ID leakage model scenarios. For the HW leakage model scenarios, the FLR and CER loss perform approximately equal.

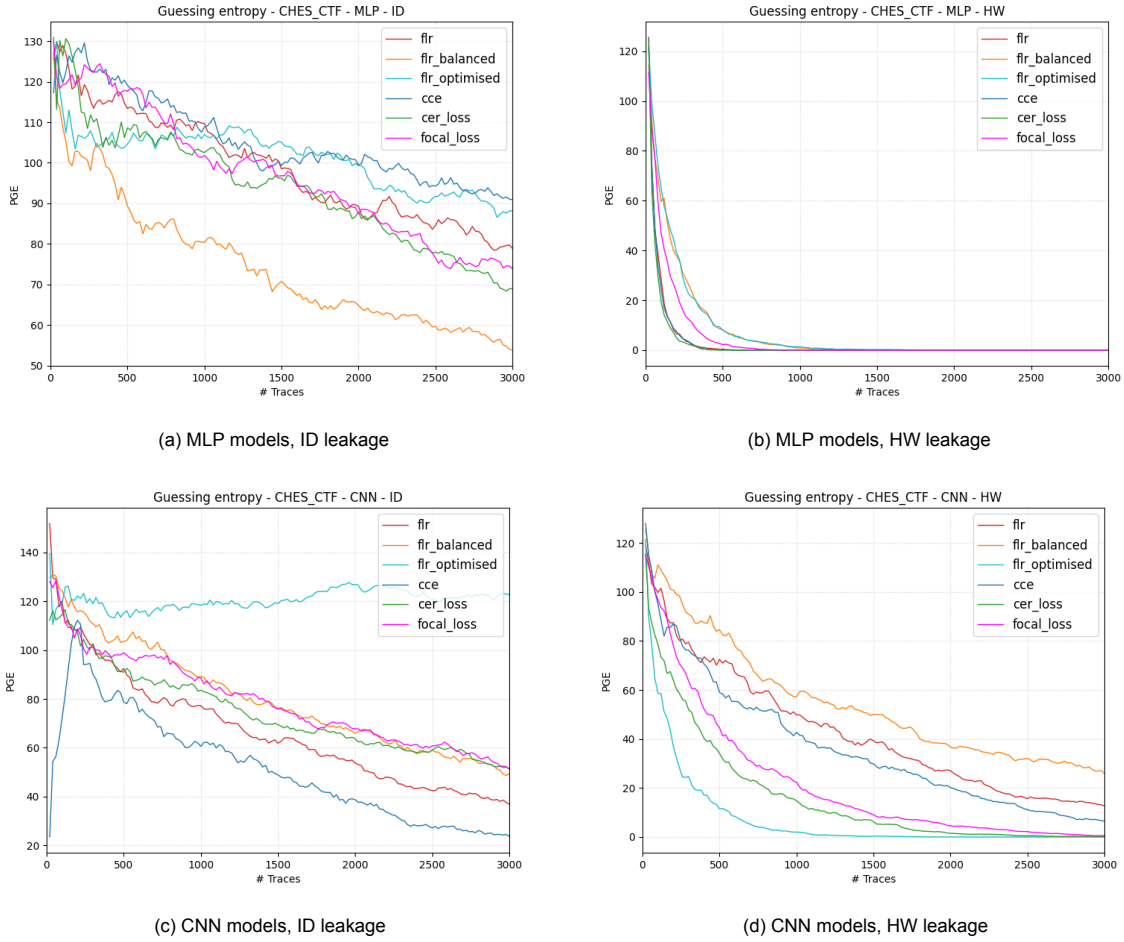
Finally, we also look at the number of trainable parameters. Table 6.3 shows the number of parameters per loss function and scenario. We again see that the FLR models are relatively small in comparison to models trained with other functions.

Table 6.3: Number of trainable parameters of the optimised models on the ASCAD_variable dataset. The lowest number of trainable parameters for each scenario is marked blue, the highest orange.

| | $L_{focal}$ | CCE | CER loss | FLR | FLR_balanced | FLR_optimised |
|---|---|---|---|---|---|---|
| MLP ID | 3,924,256 | 1,830,756 | 1,966,656 | 371,856 | 2,607,456 | 1,715,656 |
| MLP HW | 1,366,009 | 402,609 | 2,079,909 | 161,209 | 171,309 | 402,609 |
| CNN ID | 3,365,584 | 3,869,540 | 1,602,260 | 4,499,440 | 1,082,748 | 240,392 |
| CNN HW | 81,501 | 3,844,221 | 937,253 | 772,977 | 595,241 | 1,079,589 |

### 6.4.3 CHES_CTF

In this subsection, we discuss the results on the CHES_CTF dataset. Figure 6.7 shows the guessing entropy in the different scenarios and Figure 6.8 shows the success rates.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

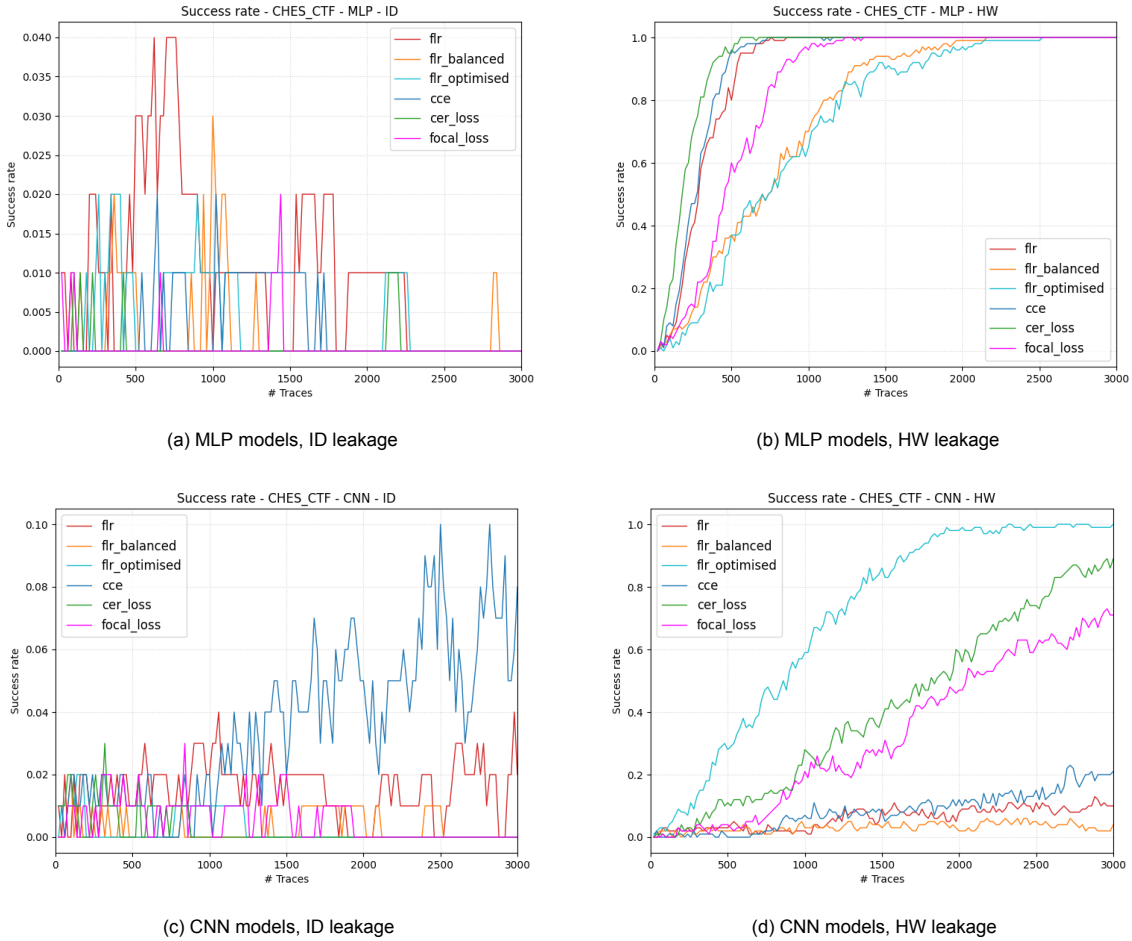
(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.7: Guessing entropy of the optimised models on the CHES_CTF dataset.

Similar to the class balanced focal loss in Chapter 5, we also see a large improvement with the MLP models and ID leakage model when using the balanced FLR. Such an improvement is also visible by some of the CNN models with FLR, however, these models turned out to be less consistent in terms of performance. This caused the median of the 10 CNN models with balanced FLR to still perform mediocrely. Something similar is also visible for the FLR with optimised parameters, where the performance turned out not to be very consistent. Overall, also visible in the success rates, we are still not able to successfully retrieve the correct key with less than 3000 traces. When the HW leakage model is considered, we again see a large increase in the performance when a CNN is used. In fact, the models with FLR and FLR with optimised parameters were the only models which successfully retrieved the correct key with a CNN model. The median out of 10 models with FLR and FLR_optimised were successful with a $\overline{N}_{T_{GE}}$ of $2740$ and $2000$ respectively. When MLPs are used, there is no significant increase and the performance is approximately equal to the CER loss.

Overall, we can conclude that the focal loss ratio is also well suited as a loss function for usage with a more difficult dataset such as the CHES_CTF dataset. The performance is approximately equal and often better than the CER loss and outperforms typically used loss functions such as the categorical cross-entropy. It does so without increasing the training time or model complexity.




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.8: Success rate of the optimised models on the CHES_CTF dataset.

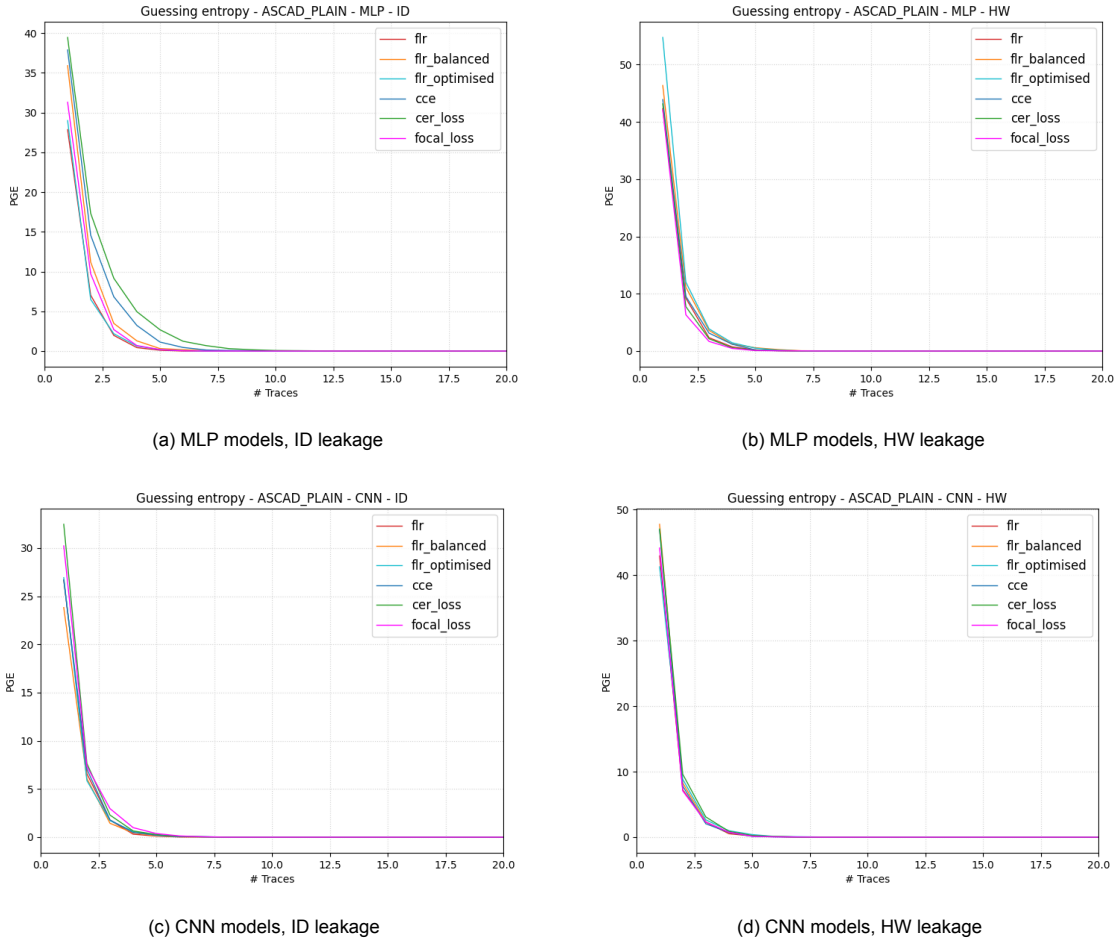
### 6.4.4 Countermeasures

Finally, we look at the difference in the performance of the different functions when faced with unprotected implementations and countermeasures. Figure 6.9 shows the guessing entropy on the ASCAD_plain dataset.

As is visible from the GE plots, the unprotected dataset is easily broken. Every model in every scenario retrieves the correct key byte trivially with less than 10 traces. The models trained with the FLR perform slightly better, within every scenario a median $\overline{N}_{T_{GE}}$ of $6$ or $7$, while the other loss functions often need $8$ or more traces.

If we look at the impact that adding masks has by comparing these results with the models in Figure 6.2, we see the largest increase $\overline{N}_{T_{GE}}$ for the models with focal loss or categorical cross-entropy. This shows that the FLR and CER loss are slightly more resilient to a countermeasure such as masking.

The random desynchronisation countermeasure decreases the performance of every model drastically. Figure 6.10 shows the GE of the models on the ASCAD_desync50 dataset. Similar to before, the MLPs perform poorly and none of the loss functions results in a successful model. When we look at the performance of the CNNs, we see that when the ID leakage model is used, the FLR and CER loss functions are not very successful. None of the models with these functions leads to a successful attack, only the model trained with the categorical cross-entropy is successful. It seems that the combination of desynchronisation and the number of classes considered when using the ID leakage model does not work well for the losses where a ratio is used. The effect is not visible when the HW leakage model is considered. In that case, the FLR functions outperform every other function, needing less than half of the amount of traces in comparison with the categorical cross-entropy ($1250$ versus $2660$). The other




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.9: Guessing entropy of the optimised models on the ASCAD_plain dataset.

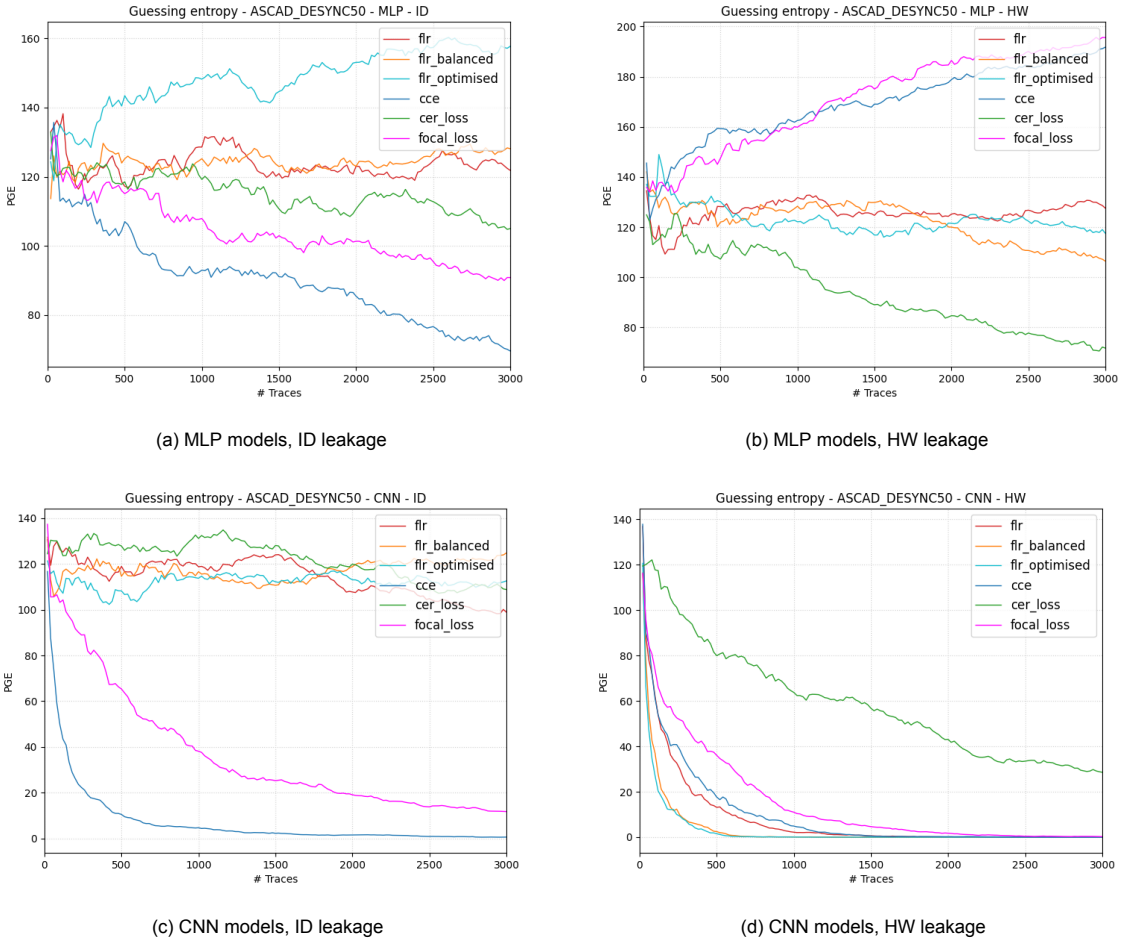
two functions, CER loss and focal loss, do not reach a GE of 1 when using 3000 traces or less to attack.

Overall, we can conclude that the FLR loss is robust to a first-order masking countermeasure. As expected, there is an increase in the required amount of traces for a successful attack, but this increase is smaller when compared with other loss functions. When random desynchronisation is added, the losses with a ratio perform poorly when the ID leakage model is used. When the HW leakage model is considered, models with FLR outperform the other loss functions by a large margin.

## 6.5 Discussion

In this section, we have proposed a new way to construct loss functions for deep learning SCA. We have identified aspects of several loss functions performing well in the context of different deep learning-based SCA scenarios. By using those characteristics, we constructed a new loss function for deep learning SCA called the focal loss ratio (FLR). By testing this new function on various combinations of datasets, leakage models and architectures, we have validated the usability of the loss function in the context of SCA. We have shown that models using FLR work with different parameter optimisation strategies and that FLR outperforms the CER loss and other loss functions like the categorical cross-entropy in most of the considered scenarios.

The only downside to the usage of FLR as a loss function is the introduction of the $\alpha$ and $\gamma$ parameters, similar to the focal loss. In our experiments, we have used three different strategies. One with the fixed default values of $\alpha = 0.25$ and $\gamma = 2.0$, one with these parameters optimised together




(a) MLP models, ID leakage

(b) MLP models, HW leakage

(c) CNN models, ID leakage

(d) CNN models, HW leakage

Figure 6.10: Guessing entropy of the optimised models on the ASCAD_desync50 dataset.

with the other hyperparameters and finally one strategy in which the values of $\alpha$ are determined by the frequency of each class. Throughout the experiments, there was not a single strategy that worked best in every scenario. However, in almost all cases, the best performing FLR variants are the ones with the fixed $\alpha$ values for every class. In some of the scenarios with the ID leakage model, the class balanced strategy improves the performance. But as discussed in Subsection 5.3.2, using class balancing with the ID leakage model results in almost constant, low values of $\alpha$. This leads us to conclude that the best strategy is the variant where the $\alpha$ and $\gamma$ parameters are optimised and where $\alpha$ is the same for each class. To set the values, optimisation via random search or other strategies can be performed. In combination with an increased range of the possible values, e.g. the addition of lower $\alpha$ values, FLR_optimised outperforms the other variants. To test this hypothesis, we have performed another set of experiments on the 12 scenarios. For these experiments, we have increased the search space for $\alpha$ by adding smaller values. The search space for $\alpha$ then becomes $0.005, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75$ and $0.9$. For each scenario, we use FLR as loss function and optimise the other hyperparameters via random search. The results of these experiments are listed in Table 6.4, Table 6.5 and Table 6.6.

These results confirm our hypothesis. The scenarios in which the class balanced FLR was previously best, still perform the best with a fixed low $\alpha$ for every class. In for example the ASCAD_fixed, MLP and ID leakage model scenario, the best performing model uses $\alpha = 0.005$. While it did not perform as well as the CER loss or FLR_balanced in this case, it did perform better than the other strategies. When using the HW leakage model on the ASCAD_variable and CHES_CTF datasets, we also see results similar to the previous experiments. In most cases, we are able to outperform the CER loss with the FLR loss. In the case of the CHES_CTF dataset using CNNs, the FLR loss is the only

Table 6.4: Median $\overline{N}_{T_{GE}}$ on the ASCAD_fixed dataset. The lowest $\overline{N}_{T_{GE}}$ for each scenario is marked blue.

| | $L_{focal}$ | CCE | CER loss | FLR |
|---|---|---|---|---|
| MLP ID | 580 | 860 | 570 | 640 |
| MLP HW | 1480 | 1560 | 560 | 490 |
| CNN ID | 1250 | 1360 | 600 | 520 |
| CNN HW | 1840 | >2000 | 540 | 500 |

Table 6.5: Median $\overline{N}_{T_{GE}}$ on the ASCAD_variable dataset. The lowest $\overline{N}_{T_{GE}}$ for each scenario is marked blue.

| | $L_{focal}$ | CCE | CER loss | FLR |
|---|---|---|---|---|
| MLP ID | >3000 | >3000 | >3000 | >3000 |
| MLP HW | 1940 | 2600 | 1340 | 1340 |
| CNN ID | >3000 | >3000 | >3000 | >3000 |
| CNN HW | >3000 | 2840 | 950 | 800 |

Table 6.6: Median $\overline{N}_{T_{GE}}$ on the CHES_CTF dataset. The lowest $\overline{N}_{T_{GE}}$ for each scenario is marked blue.

| | $L_{focal}$ | CCE | CER loss | FLR |
|---|---|---|---|---|
| MLP ID | >3000 | >3000 | >3000 | >3000 |
| MLP HW | 1220 | 630 | 480 | 1080 |
| CNN ID | >3000 | >3000 | >3000 | >3000 |
| CNN HW | >3000 | >3000 | >3000 | 2070 |

loss resulting in a successful attack with less than 3000 traces. When we consider the ID leakage, we are still not able to perform a successful attack with 3000 traces. The results are similar to the best results of the FLR_optimised and FLR_balanced mentioned in Section 6.4. The benefit, however, is that a single strategy can be used this time.

Furthermore, we have also compared the robustness against various countermeasures of each of the loss functions. There we saw that the combination of desynchronisation and the ID leakage works poorly for CER loss and FLR equally. The CNN models with FLR targeting the HW leakage model worked better though, outperforming all other functions by a margin.

To conclude, we have introduced a new loss function for deep learning in SCA called the focal loss ratio. We have shown that this new function outperforms all other considered loss functions in almost all cases when the right strategy to set the $\alpha$ and $\gamma$ parameters is used. We can therefore say that we have successfully introduced a new loss function for usage in the SCA domain, which performs equally to or better than commonly used functions like the categorical cross-entropy and recently proposed novel loss functions such as the CER loss.

# Chapter 7

# Conclusion

This chapter concludes our work on the analysis of loss functions for deep learning applied to side-channel analysis. In Chapter 4, we have provided a broad comparison between different loss functions in the context of SCA. We have compared commonly used loss functions against novel loss functions proposed specifically for SCA and provided an overview of strengths and weaknesses for each of these functions. In Chapter 5, we have applied focal loss and multi-loss functions to SCA scenarios and have shown that they can successfully be used in such a context. Finally, in Chapter 6, we have introduced a novel loss function called the focal loss ratio, based on our analysis of the different loss functions in earlier chapters. Via experiments, we have shown that this new function improves the performance of deep learning models in SCA scenarios in comparison to the previous best functions.

In this section, we first summarise the answers to our research questions. Next, we provide an overview of the scientific contributions of this work. Finally, we discuss the limitations of our work and give our thoughts about future work that could be done in this area.

## 7.1 Research Questions

### Research question 1

**How do commonly used loss functions compare to novel, application-specific loss functions when deep learning is applied to side-channel attacks?**
In Chapter 4, we have compared several commonly used loss functions to two novel loss functions specifically introduced for usage in the SCA domain. We have shown that from the commonly used loss functions, the categorical crossentropy, which is often used by default in the SCA domain, is indeed the best choice. It is more robust to different architectures and performs better than for example MSE or the log cosh loss. However, the recently introduced CER loss is by far the best performing loss function in almost all SCA scenarios. The categorical crossentropy only performs better when a dataset with a desynchronisation countermeasure is considered. The other novel loss function we have considered, the ranking loss, performs less consistent. When used with the $CNN_{methodology}$ architecture and ID leakage model, with which it was introduced by Zaid et al., it does perform very well. However, it does not significantly outperform the categorical crossentropy overall and is almost always outperformed by the CER loss. Besides that, it significantly increases the training time. By performing such a broad comparison, we have shown that application-specific loss functions have the potential to outperform typically used loss functions in the context of SCA. To help feature researchers working on the topic, we have provided an overview of the strengths and weaknesses of the various loss functions.

### Research question 2

**How do novel loss functions from other fields and multi-loss functions perform when applied to deep learning for side-channel analysis?**
In Chapter 5, we have identified several loss functions from other domains that have the potential to work well in the context of SCA. The datasets of the applications they are used for show similarities with commonly used SCA datasets. These characteristics are for example a large class imbalance or

a low amount of samples per class. The considered loss functions are the focal loss and two multi-loss functions, $L_{CosCross}$ and $L_{CorrCrossHinge}$. With our experiments we have, for the first time, shown the potential of multi-loss functions in the context of SCA. The $L_{CorrCrossHinge}$ function often resulted in a model that converged to a GE of 1. However, it did not perform better than previously tested functions and did not show any additional benefits. The $L_{CosCross}$ performed better, outperforming the categorical cross-entropy on the ASCAD_fixed dataset and performing similarly to it on the other datasets. It also has downsides, however, such as the introduction of new parameters and the fact that it is less robust to different architectures. Finally, we also for the first time used a focal loss in the context of SCA. Our experiments with the focal loss show that it is very suitable for usage in the SCA domain. Especially when the ID leakage is used, the focal loss performs better than the categorical cross-entropy and CER loss. We have tried three different strategies for setting the focal loss parameters. Of these three strategies, using class balanced weights or optimised values for $\alpha$ resulted in the best performing models. To conclude, we have shown that novel (multi-)loss functions from other domains can be used successfully for SCA and lead to well-performing models.

#### Research question 3

**Can we construct a loss function that improves the performance of deep learning models when used for side-channel analysis?**
Yes, we have shown in Chapter 6 that we can construct a loss function that improves the performance in almost all SCA scenarios. We do so by identifying certain characteristics from other, well-performing loss functions. We show that this method works by identifying characteristics from the focal and CER loss, two functions that showed good performance in our earlier experiments. Based on this, we introduced a new loss function, the focal loss ratio, which combines these characteristics into a single function. By performing various experiments with this function, we have shown that it can outperform the previously best function in most SCA scenarios.

## 7.2 Summary of Scientific Contributions

The scientific contributions in this work can be summarised as follows:

- We have provided a broad analysis of the performance of various loss functions in the context of SCA. We have compared two novel loss functions with commonly used loss functions on various datasets, leakage models and architectures and have created an overview of the strengths and weaknesses of each function based on these results.
- We have applied a focal loss and two multi-loss functions to various SCA scenarios. We have shown that these functions have the potential to perform well in the context of SCA.
- We have demonstrated that it is possible to construct loss functions based on other well-performing loss functions. We do so by identifying characteristics that make certain losses perform well in the context of SCA and combining those aspects.
- We have, based on our experiments, identified why certain loss functions performed well and used that to create a new loss function, namely the focal ratio loss. We have shown that such a function can outperform the previously best-known loss function in the context of SCA.

## 7.3 Limitations

During the course of this project we have identified several limitations of our work. The first limitation is the explainability of the behaviour of different loss functions. In some cases, we were able to find a reason for very good or poor performance of a certain function. For example for the poor performance of the categorical hinge loss on the ID leakage, or the occasional bad performance on random architectures of the CER and ranking loss. In other cases, the works in which the functions were proposed already give an explanation for why they work well. However, in general, we were not always able to find the cause of certain behaviour.

Furthermore, while we have tried to provide some kind of 'neutral' architecture to test the loss functions on with the introduction of a median model, this method is not a guarantee that the resulting architecture does not influence the performance of the resulting model. In other words, it is hard to directly compare the loss functions without the impact other hyperparameters might have.

Besides that, we also use a fixed amount of epochs of $200$ for all our experiments. This number is based on earlier work [4], but in other experiments, lower numbers of epochs have been used because a higher number of epochs such as 200 might cause over-fitting with the used optimisers [46].

Finally, while we tested the functions in many different SCA scenarios, it is in no way an exhaustive comparison or an attempt to find the absolute best performing model. We provide an insight into the performance of these loss functions on commonly used SCA datasets and try a broad selection of different architectures, but models and functions that perform even better in these scenarios might certainly exist.

## 7.4 Future work

Our contributions and limitations provide interesting directions for possible future work. First of all, since our aim was a fair comparison between the functions, we used the same parameter ranges and optimisation strategy for each of the functions. We have not tried to get the absolute best performing model for each of the loss functions and scenarios. Future work could look into finding the best architecture for the loss functions with the best performance and see how those would compare against other state-of-the-art solutions. To find such architectures, other optimisation techniques could be used instead of random search. Possible approaches could be the technique proposed by Wu et al. or using reinforcement learning to find suitable architectures, such as done by Rijsdijk et al. [54, 70].

Furthermore, we have identified two multi-loss functions introduced for other domains, from which the datasets show similarities to SCA datasets. In our case, these similarities are the low amount of samples per class related to the ID leakage model, and the class imbalance caused by the HW leakage model. However, more multi-loss functions and novel functions for other domains exist. For example, in the face recognition domain, multiple loss functions have been proposed that add some sort of margin to existing loss functions [16, 66]. In our work, we have seen that such improvements to the categorical cross-entropy (or SoftMax loss) can work well for SCA, proven by the performance of the focal loss. An interesting direction to look into might be the application or adaptation of such functions to SCA.

In the final stages of the work on this thesis, a new loss function specifically for deep learning SCA called ensembling loss was proposed [76]. The purpose of the ensembling loss is to increase the diversity between the models in an ensemble of models. Zaid et al. demonstrate the performance of their new loss function and compare it to the ranking loss and categorical crossentropy. Future work could for example look into how other loss functions, like CER or FLR, perform in such a scenario with ensembles.

Finally, we have constructed a new loss function based on existing functions that perform well on SCA datasets. Based on the strengths and weaknesses of each loss function we list in Chapter 4, other loss functions could be constructed similarly. However, another direction to look into might be the construction of a new loss function based on SCA related metrics. The CER loss also is based on the similarly named cross-entropy ratio metric introduced in the same paper. Constructing a loss function in such a way makes sense since the loss that has to be reduced is related directly to a measure of the performance we are interested in. An example could be to use the guessing entropy directly. Since the guessing entropy is commonly estimated by performing a number of attacks, this would be impractical to incorporate in a loss. However, a feasible way might be using the fast guessing entropy estimation method as proposed by Zhang et al. in a loss function [78].

# Bibliography


[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side—Channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

[2] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide. In *Constructive Side-Channel Analysis and Secure Design*, pages 249–264, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-29912-4.

[3] Björn Barz and Joachim Denzler. Deep Learning on Small Datasets without Pre-Training using Cosine Loss. Technical report, 2020.

[4] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database-Long Paper. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020. doi: 10.1007/s13389-019-00220-8.

[5] Bogdanov Andrey, Khovratovich Dmitry, and Rechberger Christian. Biclique Cryptanalysis of the Full AES. In *Advances in Cryptology – ASIACRYPT 2011*, pages 344–371, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25385-0.

[6] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29, 2004. ISBN 978-3-540-28632-5.

[7] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures-Profiling Attacks without Pre-Processing. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 45–68, 2017. doi: 10.1007/978-3-319-66787-4{_}3.

[8] Cedric Archambeau, Eric Peeters, Francois-Xavier Standaert, and Jean-Jacques Quisquater. Template Attacks in Principal Subspaces. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 1–14, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46561-4.

[9] Sujita Chaudhary, Austin O'Brien, and Shengjie Xu. Automated Post-Breach Penetration Testing through Reinforcement Learning. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–2, 2020. doi: 10.1109/CNS48642.2020.9162301.

[10] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 13-17-August-2016, pages 785–794. Association for Computing Machinery, 8 2016. ISBN 9781450342322. doi: 10.1145/2939672.2939785.

[11] François Chollet. Keras, 2015. URL https://github.com/fchollet/keras.

[12] Omar Choudary and Markus G. Kuhn. Efficient Template Attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 253–270, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08302-5.

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

[14] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J Belongie. Class-Balanced Loss Based on Effective Number of Samples. *CoRR*, abs/1901.05555, 2019. URL http://arxiv.org/abs/1901.05555.

[15] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[16] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. 1 2018. URL http://arxiv.org/abs/1801.07698.

[17] Genkin Danieland, Pachmanov Levand, Pipman Itamar, and Tromer Eran. Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 207–228, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48324-4.

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. Technical report, 2011.

[19] Hamideh Hajiabadi, Vahide Babaiyan, Davood Zabihzadeh, and Moein Hajiabadi. Combination of loss functions for robust breast cancer prediction. *Computers and Electrical Engineering*, 84, 6 2020. ISSN 00457906. doi: 10.1016/j.compeleceng.2020.106624.

[20] Hamideh Hajiabadi, Diego Molla-Aliod, Reza Monsefi, and Hadi Sadoghi Yazdi. Combination of loss functions for deep text classification. *International Journal of Machine Learning and Cybernetics*, 11(4):751–761, 4 2020. ISSN 1868808X. doi: 10.1007/s13042-019-00982-x.

[21] William Grant Hatcher and Wei Yu. A Survey of Deep Learning: Platforms, Applications and Emerging Research Trends. *IEEE Access*, 6:24411–24432, 4 2018. ISSN 21693536. doi: 10.1109/ACCESS.2018.2830661.

[22] Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the Selection of Initialization and Activation Function for Deep Neural Networks, 2018.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015.

[24] 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):293–302, 12 2011. ISSN 21908508. doi: 10.1007/s13389-011-0023-x.

[25] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2 2015. URL http://arxiv.org/abs/1502.03167.

[26] Katarzyna Janocha and Wojciech Marian Czarnecki. On Loss Functions for Deep Neural Networks in Classification. Technical report, 2017. URL https://arxiv.org/abs/1702.05659.

[27] 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 ISSN 2569-2925*, 2019 (3):148–179, 2019. doi: 10.13154/tches.v2019.i3.148-179.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017.

[29] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-Normalizing Neural Networks. Technical report, 2017.

[30] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.

[31] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2.

[32] Nataliia Kussul, Mykola Lavreniuk, Sergii Skakun, and Andrey Shelestov. Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data. *IEEE Geoscience and Remote Sensing Letters*, PP:1–5, 7 2017. doi: 10.1109/LGRS.2017.2681128.

[33] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. In *Smart Card Research and Advanced Applications - 12th International Conference*, pages 61–75, Berlin, 11 2013. doi: 10.1007/978-3-319-08302-5{_}5.

[34] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 844–848, 2014. doi: 10.1109/ICARCV.2014.7064414.

[35] Ming Liang and Xiaolin Hu. Recurrent Convolutional Neural Network for Object Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, 6 2015.

[36] Tsung-Yi Lin, Priya Goyal, Ross B Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. *CoRR*, abs/1708.02002, 2017. URL http://arxiv.org/abs/1708.02002.

[37] Xingjun Ma, Hanxun Huang, Yisen Wang, Simone Romano, Sarah Erfani, and James Bailey. Normalized Loss Functions for Deep Learning with Noisy Labels. Technical report, 2020.

[38] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit Card Fraud Detection Using Bayesian and Neural Networks. In *In: Maciunas RJ, editor. Interactive image-guided neurosurgery. American Association Neurological Surgeons*, pages 261–270, 1993.

[39] Houssem Maghrebi. Deep Learning based Side Channel Attacks in Practice. *IACR Cryptol. ePrint Arch.*, 2019:578, 2019.

[40] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking Cryptographic Implementations Using Deep Learning Techniques. Technical report, 2016. URL https://eprint.iacr.org/2016/921.

[41] Henry B. Mann and Donald R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947. doi: 10.1214/aoms/1177730491. URL https://doi.org/10.1214/aoms/1177730491.

[42] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, page 44. USENIX Association, 11 2016. ISBN 9781931971331. URL https://www.tensorflow.org/.

[43] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A Comprehensive Study of Deep Learning for Side-Channel Analysis, 2019.

[44] Thorben Moos, Felix Wegener, and Amir Moradi. DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):552–598, 7 2021. doi: 10.46586/tches.v2021.i3.552-598. URL https://tches.iacr.org/index.php/TCHES/article/view/8986.

[45] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. Technical report, 2010.

[46] Guilherme Perin and Stjepan Picek. On the Influence of Optimizers in Deep Learning-based Side-channel Analysis. Technical report, 2020.

[47] Guilherme Perin, Łukasz Chmielewski, and Stjepan Picek. Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis. Technical report, 2020.

[48] George Philipp, Dawn Song, and Jaime G Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. *CoRR*, abs/1712.05577, 2017. URL http://arxiv.org/abs/1712.05577.

[49] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template Attack vs. Bayes Classifier. Technical report, 2017.

[50] Stjepan Picek, Ioannis Petros Samiotis, Annelie Heuser, Jaehun Kim, Shivam Bhasin, and Axel Legay. On the Performance of Convolutional Neural Networks for Side-channel Analysis. pages 157–176. Springer Verlag, 2018. ISBN 9783030050719. doi: 10.1007/978-3-030-05072-6{_}10. URL https://research.tudelft.nl/en/publications/on-the-performance-of-convolutional-neural-networks-for-side-chan.

[51] Emmanuel Prouff. DPA Attacks and S-Boxes. In *Fast Software Encryption*, pages 424–441, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31669-5.

[52] U Rajendra Acharya, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, Muhammad Adam, Arkadiusz Gertych, and Ru San Tan. A deep convolutional neural network model to classify heartbeats. 2017. doi: 10.1016/j.compbiomed.2017.08.022. URL http://dx.doi.org/10.1016/j.compbiomed.2017.08.022.

[53] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat Charvillon, Dina Kamel, and Denis Flandre. A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6632 of *Lecture Notes in Computer Science*, page 109. Springer, 2011. doi: 10.1007/978-3-642-20465-4{_}8. URL https://www.iacr.org/archive/eurocrypt2011/66320107/66320107.pdf.

[54] 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*, 2021(3):677–707, 7 2021. doi: 10.46586/tches.v2021.i3.677-707. URL https://tches.iacr.org/index.php/TCHES/article/view/8989.

[55] Riscure. CHES CTF 2018, 2018.

[56] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[57] Luisa F. Sánchez-Peralta, Artzai Picón, Juan Antonio Antequera-Barroso, Juan Francisco Ortega-Morán, Francisco M. Sánchez-Margallo, and J. Blas Pagador. Eigenloss: Combined PCA-based loss function for polyp segmentation. *Mathematics*, 8(8), 8 2020. ISSN 22277390. doi: 10.3390/MATH8081316.

[58] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. ISSN 0036-8075. doi: 10.1126/science.aar6404. URL https://science.sciencemag.org/content/362/6419/1140.

[59] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. 9 2014. URL http://arxiv.org/abs/1409.1556.

[60] 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 ISSN 2569-2925*, 2019 (1):209–237, 2019. doi: 10.13154/tches.v2019.i1.209-237. URL https://doi.org/10.13154/tches.v2019.i1.209-237.

[61] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

[62] Adrian Thillard. Countermeasures to side-channel attacks and secure multi-party computation. Technical report, 2016. URL https://tel.archives-ouvertes.fr/tel-01764625.

[63] Benjamin Timon. Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):107–131, 2 2019. doi: 10.13154/tches.v2019.i2.107-131. URL https://tches.iacr.org/index.php/TCHES/article/view/7387.

[64] Chih Fong Tsai, Yu Feng Hsu, Chia Ying Lin, and Wei Yang Lin. Intrusion detection by machine learning: A review, 12 2009. ISSN 09574174.

[65] Juan Vanerio and Pedro Casas. Ensemble-Learning Approaches for Network Security and Anomaly Detection. In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, Big-DAMA '17, pages 1–6, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350549. doi: 10.1145/3098593.3098594. URL https://doi.org/10.1145/3098593.3098594.

[66] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*, 25(7):926–930, 7 2018. ISSN 10709908. doi: 10.1109/LSP.2018.2822810.

[67] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, 2020. ISSN 21985812. doi: 10.1007/s40745-020-00253-5.

[68] Weifeng Liu, P.P. Pokharel, and J.C. Principe. Correntropy: A localized similarity measure. In *IEEE International Conference on Neural Networks - Conference Proceedings*, pages 4919–4924, 2006. ISBN 0780394909. doi: 10.1109/ijcnn.2006.247192.

[69] 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*, 2020(3):147–168, 2020. doi: 10.13154/tches.v2020.i3.147-168. URL https://github.com/KULeuven-COSIC/TCHES20V3_CNN_SCA.

[70] Lichao Wu, Guilherme Perin, and Stjepan Picek. I Choose You: Automated Hyperparameter Tuning for Deep Learning-based Side-channel Analysis. Technical report, 2020.

[71] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. On the Attack Evaluation and the Generalization Ability in Profiling Side-channel Analysis. Technical report, 2020. URL https://eprint.iacr.org/2020/899.pdf.

[72] Yash Srivastava, Vaishnavand Murali, and Shiv Ram Dubey. A Performance Evaluation of Loss Functions for Deep Face Recognition. In Babu R. Venkatesh, Mahadeva Prasann, and Vinay P Namboodiri, editors, *Computer Vision, Pattern Recognition, Image Processing, and Graphics*, pages 322–332, Singapore, 2020. Springer Singapore. ISBN 978-981-15-8697-2.

[73] Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740, 2020. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2020.101740. URL https://www.sciencedirect.com/science/article/pii/S0167404820300262.

[74] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis. Technical report, 2020. URL https://eprint.iacr.org/2020/872https://github.com/gabzai/Ranking-Loss-SCA.

[75] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for Efficient CNN Architectures in Profiling Attacks. 2020. doi: 10.13154/tches.v2020.i1.1-36.

[76] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Efficiency through Diversity in Ensemble Models applied to Side-Channel Attacks: – A Case Study on Public-Key Algorithms –. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3): 60–96, 7 2021. doi: 10.46586/tches.v2021.i3.60-96. URL https://tches.iacr.org/index.php/TCHES/article/view/8968.

[77] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A Novel Evaluation Metric for Deep Learning-Based Side Channel Analysis and Its Extended Application to Imbalanced Data. 2020(3):73–96, 2020. doi: 10.13154/tches.v2020.i3.73-96.

[78] Ziyue Zhang, A Adam Ding, and Yunsi Fei. A Fast and Accurate Guessing Entropy Estimation Algorithm for Full-key Recovery. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):26–48, 3 2020. doi: 10.13154/tches.v2020.i2.26-48. URL https://tches.iacr.org/index.php/TCHES/article/view/8543.

[79] Donglai Zhu, Hengshuai Yao, Bei Jiang, and Peng Yu. Negative Log Likelihood Ratio Loss for Deep Neural Network Classification. 4 2018. URL http://arxiv.org/abs/1804.10690.

# Appendix A

# Hyperparameters Median Models

## A.1 ASCAD

Table A.1: Hyperparameters median MLP models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Dense layers | 6 | 5 |
| Neurons per layer | 900 | 300 |
| Learning rate | 0.0001 | 0.0007 |
| Batch size | 800 | 600 |
| Activation function | ELU | Tanh |
| Optimiser | Adam | Adam |
| # trainable parameters | 648556 | 4693509 |

Table A.2: Hyperparameters median CNN models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Convolutional layers | 1 | 1 |
| Convolutional filters | 8 | 16 |
| Kernel size | 10 | 14 |
| Pooling size | 4 | 3 |
| Pooling stride | 5 | 5 |
| Dense layers | 2 | 2 |
| Neurons per layer | 900 | 100 |
| Learning rate | 0.00046 | 0.00002 |
| Batch size | 600 | 400 |
| Activation function | ELU | SELU |
| Optimiser | RMSProp | RMSprop |
| Pooling type | Max pooling | Average pooling |
| # trainable parameters | 1828013 | 260328 |

## A.2 ASCAD_variable

Table A.3: Hyperparameters median MLP models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Dense layers | 4 | 2 |
| Neurons per layer | 200 | 200 |
| Learning rate | 0.00034 | 0.0006 |
| Batch size | 300 | 200 |
| Activation function | ELU | ELU |
| Optimiser | RMSprop | RMSprop |
| # trainable parameters | 402609 | 371856 |

Table A.4: Hyperparameters median CNN models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Convolutional layers | 1 | 1 |
| Convolutional filters | 20 | 32 |
| Kernel size | 18 | 14 |
| Pooling size | 4 | 4 |
| Pooling stride | 5 | 10 |
| Dense layers | 2 | 2 |
| Neurons per layer | 800 | 200 |
| Learning rate | 0.00026 | 0.00014 |
| Batch size | 500 | 700 |
| Activation function | Tanh | RELU |
| Optimiser | Adam | RMSprop |
| Pooling type | Average pooling | Average pooling |
| # trainable parameters | 5129229 | 988400 |

## A.3 CHES_CTF

Table A.5: Hyperparameters median MLP models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Dense layers | 6 | 8 |
| Neurons per layer | 1000 | 800 |
| Learning rate | 0.00004 | 0.0004 |
| Batch size | 900 | 400 |
| Activation function | Tanh | ELU |
| Optimiser | RMSprop | RMSprop |
| # trainable parameters | 7215009 | 6451456 |

Table A.6: Hyperparameters median CNN models for the ID and HW leakage scenarios.

| Hyperparameter | HW leakage | ID leakage |
|---|---|---|
| Convolutional layers | 2 | 1 |
| Convolutional filters | 8 | 24 |
| Kernel size | 20 | 10 |
| Pooling size | 4 | 5 |
| Pooling stride | 10 | 10 |
| Dense layers | 3 | 3 |
| Neurons per layer | 600 | 1000 |
| Learning rate | 0.00024 | 0.00024 |
| Batch size | 900 | 800 |
| Activation function | SELU | Tanh |
| Optimiser | RMSProp | RMSprop |
| Pooling type | Max pooling | Average pooling |
| # trainable parameters | 834297 | 7539568 |

# Appendix B

# Hyperparameters Optimised Models

## B.1 ASCAD

Table B.1: Hyperparameters optimised MLP models for ID leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 3 | 3 | 3 | 4 | 2 | 8 | 2 |
| Neurons per layer | 100 | 600 | 300 | 100 | 400 | 100 | 400 |
| Learning rate | 0.0004 | 0.00078 | 0.00026 | 0.0008 | 0.00044 | 0.0003 | 0.00046 |
| Batch size | 200 | 1000 | 1000 | 400 | 800 | 300 | 900 |
| Activation function | SELU | ReLU | SELU | tanh | ReLU | tanh | ELU |
| Optimiser | RMSprop | RMSprop | RMSprop | RMSprop | RMSprop | Adam | Adam |

Table B.2: Hyperparameters optimised MLP models for HW leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 5 | 5 | 4 | 3 | 2 | 7 | 3 |
| Neurons per layer | 200 | 900 | 300 | 500 | 900 | 300 | 400 |
| Learning rate | 0.00016 | 0.00008 | 0.00056 | 0.0004 | 0.00026 | 0.00072 | 0.00016 |
| Batch size | 400 | 800 | 500 | 900 | 300 | 800 | 500 |
| Activation function | SELU | SELU | ReLU | ELU | ReLU | ReLU | ReLU |
| Optimiser | RMSprop | RMSprop | Adam | Adam | RMSprop | Adam | RMSprop |

Table B.3: Hyperparameters optimised CNN models for ID leakage

| **Hyperparameter** | **cce** | **cat_hinge** | **cer_loss** | **log_cosh** | **msle** | **mse** | **rkl** |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 2 | 2 | 1 | 1 | 1 | 1 | 1 |
| Convolutional filters | 8 | 32 | 16 | 8 | 32 | 24 | 20 |
| Kernel size | 20 | 12 | 20 | 18 | 32 | 24 | 20 |
| Pooling size | 4 | 3 | 5 | 2 | 5 | 4 | 4 |
| Pooling stride | 10 | 10 | 10 | 10 | 5 | 10 | 10 |
| Pooling type | Average | Average | Average | Max | Average | Max | Average |
| Dense layers | 3 | 3 | 3 | 2 | 2 | 3 | 3 |
| Neurons per layer | 100 | 600 | 200 | 300 | 1000 | 100 | 100 |
| Learning rate | 0.00014 | 0.0008 | 0.00008 | 0.00018 | 0.00022 | 0.00032 | 0.00022 |
| Batch size | 600 | 1000 | 900 | 500 | 600 | 100 | 300 |
| Activation function | SELU | SELU | tanh | ReLU | ReLU | tanh | ELU |
| Optimiser | RMSProp | Adam | RMSprop | Adam | Adam | Adam | RMSprop |

Table B.4: Hyperparameters optimised CNN models for HW leakage

| **Hyperparameter** | **cce** | **cat_hinge** | **cer_loss** | **log_cosh** | **msle** | **mse** | **rkl** |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 2 | 1 | 2 | 2 | 2 | 1 | 2 |
| Convolutional filters | 12 | 12 | 24 | 16 | 8 | 20 | 28 |
| Kernel size | 14 | 10 | 12 | 16 | 10 | 12 | 12 |
| Pooling size | 4 | 2 | 5 | 4 | 5 | 5 | 3 |
| Pooling stride | 5 | 10 | 5 | 5 | 10 | 5 | 5 |
| Pooling type | Average | Average | Average | Average | Average | Average | Average |
| Dense layers | 2 | 3 | 2 | 2 | 3 | 2 | 2 |
| Neurons per layer | 900 | 1000 | 500 | 800 | 800 | 100 | 700 |
| Learning rate | 0.00002 | 0.00008 | 0.00002 | 0.00026 | 0.00094 | 0.00078 | 0.00002 |
| Batch size | 800 | 400 | 500 | 200 | 900 | 200 | 600 |
| Activation function | ELU | SELU | ELU | ELU | ELU | ELU | ReLU |
| Optimiser | Adam | RMSprop | RMSprop | RMSprop | RMSprop | Adam | RMSprop |

## B.2 ASCAD_variable

Table B.5: Hyperparameters optimised MLP models for ID leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 5 | 8 | 2 | 6 | 8 | 7 | 2 |
| Neurons per layer | 500 | 500 | 800 | 1000 | 300 | 400 | 200 |
| Learning rate | 0.00026 | 0.00072 | 0.0004 | 0.0005 | 0.00038 | 0.0005 | 0.00052 |
| Batch size | 900 | 500 | 500 | 400 | 500 | 200 | 300 |
| Activation function | ELU | SELU | SELU | ReLU | ELU | ELU | ELU |
| Optimiser | RMSprop | Adam | RMSprop | Adam | RMSprop | Adam | RMSprop |

Table B.6: Hyperparameters optimised MLP models for HW leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 4 | 5 | 2 | 4 | 4 | 4 | 2 |
| Neurons per layer | 200 | 900 | 900 | 900 | 1000 | 600 | 700 |
| Learning rate | 0.00032 | 0.00006 | 0.00034 | 0.00002 | 0.00014 | 0.00022 | 0.00016 |
| Batch size | 300 | 900 | 700 | 400 | 200 | 100 | 400 |
| Activation function | ReLU | SELU | ReLU | ELU | ELU | ELU | tanh |
| Optimiser | Adam | RMSprop | RMSprop | RMSprop | RMSprop | Adam | Adam |

Table B.7: Hyperparameters optimised CNN models for ID leakage

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 1 | 1 | 1 | 2 | 1 | 1 | 1 |
| Convolutional filters | 8 | 20 | 16 | 16 | 16 | 24 | 20 |
| Kernel size | 20 | 12 | 16 | 12 | 16 | 14 | 16 |
| Pooling size | 5 | 3 | 5 | 5 | 5 | 4 | 5 |
| Pooling stride | 5 | 10 | 5 | 5 | 10 | 10 | 5 |
| Dense layers | 3 | 2 | 3 | 3 | 3 | 3 | 2 |
| Neurons per layer | 900 | 300 | 300 | 200 | 1000 | 400 | 700 |
| Learning rate | 0.00022 | 0.00046 | 0.00016 | 0.0006 | 0.00096 | 0.00054 | 0.0003 |
| Batch size | 1000 | 800 | 700 | 600 | 600 | 500 | 1000 |
| Activation function | ELU | ReLU | SELU | ELU | ReLU | ReLU | SELU |
| Optimiser | RMSProp | Adam | RMSprop | RMSprop | RMSprop | Adam | RMSprop |
| Pooling type | Average | Max | Average | Average | Average | Average | Average |

Table B.8: Hyperparameters optimised CNN models for HW leakage

| **Hyperparameter** | **cce** | **cat_hinge** | **cer_loss** | **log_cosh** | **msle** | **mse** | **rkl** |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 1 | 1 | 2 | 1 | 1 | 1 | 1 |
| Convolutional filters | 24 | 28 | 24 | 16 | 12 | 12 | 32 |
| Kernel size | 10 | 20 | 16 | 14 | 20 | 10 | 20 |
| Pooling size | 5 | 5 | 5 | 4 | 5 | 5 | 5 |
| Pooling stride | 10 | 5 | 5 | 10 | 5 | 10 | 10 |
| Dense layers | 2 | 2 | 2 | 3 | 2 | 2 | 2 |
| Neurons per layer | 900 | 600 | 500 | 100 | 1000 | 100 | 700 |
| Learning rate | 0.00082 | 0.00008 | 0.00002 | 0.00004 | 0.00004 | 0.00048 | 0.00002 |
| Batch size | 700 | 1000 | 200 | 400 | 400 | 500 | 700 |
| Activation function | ELU | SELU | ELU | ELU | ELU | SELU | ELU |
| Optimiser | Adam | RMSprop | RMSprop | RMSprop | RMSprop | RMSprop | RMSprop |
| Pooling type | Average | Average | Average | Average | Average | Max | Average |

## B.3 CHES_CTF

Table B.9: Hyperparameters optimised MLP models for ID leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 2 | 3 | 4 | 8 | 6 | 5 | 4 |
| Neurons per layer | 700 | 300 | 400 | 600 | 300 | 1000 | 700 |
| Learning rate | 0.0002 | 0.00032 | 0.0005 | 0.00036 | 0.00062 | 0.00022 | 0.00006 |
| Batch size | 400 | 1000 | 900 | 800 | 400 | 500 | 700 |
| Activation function | SELU | tanh | tanh | SELU | SELU | tanh | ELU |
| Optimiser | RMSprop | Adam | RMSprop | RMSprop | RMSprop | RMSprop | Adam |

Table B.10: Hyperparameters optimised MLP models for HW leakage.

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Dense layers | 7 | 3 | 6 | 3 | 3 | 8 | 2 |
| Neurons per layer | 200 | 900 | 600 | 200 | 1000 | 200 | 100 |
| Learning rate | 0.0005 | 0.00038 | 0.0003 | 0.00076 | 0.00038 | 0.0005 | 0.00062 |
| Batch size | 200 | 300 | 600 | 700 | 300 | 700 | 200 |
| Activation function | SELU | ReLU | ReLU | ReLU | ReLU | ELU | ELU |
| Optimiser | Adam | RMSprop | Adam | Adam | RMSprop | Adam | Adam |

Table B.11: Hyperparameters optimised CNN models for ID leakage

| Hyperparameter | cce | cat_hinge | cer_loss | log_cosh | msle | mse | rkl |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 2 | 2 | 1 | 1 | 1 | 2 | 1 |
| Convolutional filters | 8 | 16 | 8 | 24 | 8 | 8 | 20 |
| Kernel size | 14 | 12 | 12 | 20 | 12 | 18 | 20 |
| Pooling size | 2 | 2 | 3 | 3 | 4 | 3 | 2 |
| Pooling stride | 5 | 10 | 10 | 10 | 10 | 5 | 10 |
| Dense layers | 2 | 3 | 3 | 2 | 2 | 3 | 3 |
| Neurons per layer | 1000 | 200 | 700 | 400 | 400 | 500 | 1000 |
| Learning rate | 0.00072 | 0.00042 | 0.00008 | 0.00072 | 0.0003 | 0.00054 | 0.00028 |
| Batch size | 100 | 300 | 1000 | 700 | 400 | 200 | 700 |
| Activation function | ELU | SELU | SELU | SELU | tanh | ELU | ELU |
| Optimiser | RMSProp | Adam | RMSprop | Adam | RMSprop | Adam | RMSprop |
| Pooling type | Max | Average | Average | Average | Max | Max | Max |

Table B.12: Hyperparameters optimised CNN models for HW leakage

| **Hyperparameter** | **cce** | **cat_hinge** | **cer_loss** | **log_cosh** | **msle** | **mse** | **rkl** |
|---|---|---|---|---|---|---|---|
| Convolutional layers | 1 | 1 | 1 | 2 | 1 | 2 | 1 |
| Convolutional filters | 20 | 12 | 8 | 8 | 12 | 24 | 8 |
| Kernel size | 14 | 10 | 14 | 10 | 20 | 10 | 14 |
| Pooling size | 3 | 5 | 5 | 2 | 4 | 5 | 5 |
| Pooling stride | 10 | 5 | 10 | 10 | 5 | 5 | 10 |
| Dense layers | 3 | 3 | 3 | 2 | 2 | 3 | 2 |
| Neurons per layer | 600 | 500 | 100 | 1000 | 400 | 700 | 100 |
| Learning rate | 0.00052 | 0.00084 | 0.0002 | 0.00004 | 0.00076 | 0.00038 | 0.00096 |
| Batch size | 500 | 600 | 600 | 100 | 200 | 600 | 800 |
| Activation function | SELU | ELU | ELU | ELU | tanh | tanh | tanh |
| Optimiser | Adam | Adam | Adam | Adam | RMSprop | RMSprop | RMSprop |
| Pooling type | Max | Average | Average | Average | Average | Average | Average |