No (Good) Loss no Gain: Systematic Evaluation of Loss functions in Deep Learning-based Side-channel Analysis

Maikel Kerkhof[1], Lichao Wu[1], Guilherme Perin[1], and Stjepan Picek[1]

Delft University of Technology, The Netherlands

Abstract. Deep learning is a powerful direction for profiling side-channel analysis as it can break targets protected with countermeasures even with a relatively small number of attack traces. Still, it is necessary to conduct hyperparameter tuning for strong attack performance, which can be far from trivial. Besides a plethora of options stemming from the machine learning domain, recent years also brought neural network elements specially designed for side-channel analysis.
An important hyperparameter is the loss function, which calculates the error or loss between the actual and desired output. The resulting loss is used to update the weights associated with the connections between the neurons or filters of the deep learning neural network. Unfortunately, despite being a highly relevant hyperparameter, there are no systematic comparisons among different loss functions. This work provides a detailed study on the performance of different loss functions in the SCA context. We evaluate five loss functions commonly used in machine learning and two loss functions proposed for SCA. Our results show that one of the SCA-specific loss functions (called CER) performs very well and outperforms other loss functions in most evaluated settings. Finally, our results show that categorical cross-entropy represents a good option for most settings, especially if there is a requirement to work well with different neural network architectures.

Keywords: Side-channel analysis, Deep Learning, Loss function, Evaluation

1 Introduction

Side-channel analysis (SCAs) represents a powerful type of implementation attack on cryptographic algorithms. A usual division of side-channel analysis is into direct attacks and two-stage (profiling) attacks. Profiling attacks assume an "open" device (or a copy of it). By building the templates based on the leakage of this device, the key recovery of the attack device requires only a few measurements. Today, the most powerful representatives of profiling attacks come from the deep learning domain [3, 17, 28]. Literature indicates that such attacks can break targets protected with countermeasures, but to reach that level of performance, they also require a careful hyperparameter tuning [9]. Unfortunately, due to the complexity of the deep learning techniques, finding the best

hyperparameter combination is a challenging task. Additionally, since the devices are commonly equipped with countermeasures and noise, it makes sense to customize the neural networks to counter such difficulties.

Loss function, one of the tunable hyperparameters, plays a central role in training a deep learning model. They are used to calculate the error or loss between the actual and desired output. The resulting loss is propagated back to learn, i.e., update the weights associated with the connections between the neurons or filters of the deep learning network. The choice of the loss function can influence the performance of the resulting deep learning model [8, 29], which is also recognized in the SCA domain. Indeed, the custom loss functions proposed in the recent works [31,33] report good attack performance on the chosen datasets. However, one should note that a loss function that works for one specific attack setting is not necessary to work on another. In other words, the attack efficiency is influenced not only by a loss function but by many other factors such as the number of traces, model architecture, and weight initialization. Unfortunately, the generality of the proposed loss functions is not explored in these papers.

In this work, we aim to fill in that gap. More precisely, we systematically compare commonly used loss functions and novel application-specific loss functions in the context of side-channel analysis. We evaluate the attack performance (guessing entropy), the number of trainable parameters, and the required training time. We evaluate different loss functions on two publicly available datasets and with two commonly used leakage models. Our results show that the recently proposed CER loss performs very well and manages to outperform other loss functions. The especially strong performance can be observed for the Hamming weight leakage model, which can represent a difficult scenario due to class imbalance and the lack of reliability of machine learning metrics [20]. Ranking loss, another recently proposed loss function, performs much worse and seems to work with specific neural network architectures only. Finally, a common choice in deep learning-based SCA, categorical cross-entropy, can be indeed confirmed as a strong option, especially if the training time or good performance with different neural network architectures is required.

2 Background

This section provides an introduction to profiling side-channel attacks. Afterward, we discuss various loss functions and the datasets we use in our experiments.

2.1 Deep Learning-based Side-channel Analysis

Supervised machine learning aims to learn a function f mapping an input to the output based on examples of input-output pairs. The function f is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^n$, where n denotes the number of trainable parameters. Supervised learning happens in two phases: training and test. This corresponds to profiling SCA phases, commonly denoted as profiling and attack phases.

A dataset is defined as a collection of side-channel traces (measurements) $\mathbf{T}$, where each trace $\mathbf{t}_i$ is associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key $\mathbf{k}_i$. We divide the dataset into disjoint subsets where the training set has M traces, the validation set has V traces, and the attack set has Q traces.

1. The goal of the training phase is to learn $\boldsymbol{\theta}$ (vector of parameters) minimizing the empirical risk represented by a loss function L on a dataset T of size M (i.e., on the profiling (training) set).
2. In the attack phase (also known as testing or inference), the goal is to make predictions about the classes

$$y(x_1, k^*), \ldots, y(x_Q, k^*),$$

where k^* represents the secret (unknown) key on the device under the attack. The outcome of predicting with a model f on the attack set is a two-dimensional matrix P with dimensions equal to $Q \times c$ (where c denotes the number of classes). The probability $S(k)$ for any key candidate k is then used as a maximum log-likelihood distinguisher:

$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{1}$$

The value $\mathbf{p}_{i,v}$ is the probability that for a key k and input d_i, we obtain the class v (with $\sum_v^c \mathbf{p}_{i,v} = 1, \forall i$). The class v is derived from the key and input through a cryptographic function CF and a leakage model l.

In SCA, an adversary aims at revealing the secret key k^*. For this, standard performance metrics are the success rate (SR) and the guessing entropy (GE) of a side-channel attack [25]. In this work, we use the guessing entropy metric to estimate the attack performance. Given Q amount of traces in the attack phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability. Thus, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate.

This work considers two commonly used deep learning models: multilayer perceptron (MLP) and convolutional neural networks (CNNs). Then, the function f is a deep neural network with the *Softmax* output layer. We encode classes in one-hot encoding, where each class is represented as a vector of c values that has zero on all the places, except one place, denoting the membership of that class.

2.2 Loss Functions

Both the MLPs and CNNs in this work are used in the supervised learning setting. The loss, calculated by a loss function, indicates the difference between the outputs of the model (predicted label) and the actual labels that belong to the input. The output of the loss function is used to update the weights in the network, finally reduce the deviation between predicted and real labels.

Mean Squared Error One of the simplest examples of a loss function is the mean squared error (MSE) [24]. 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:

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

where n denotes the number of tested samples. The MSE loss function and its variations have been used to solve regression problems (thus, the output of the function f is continuous) [24]. The loss is calculated evenly for each sample, regardless of which class a sample belongs to. By minimizing the loss, we minimize the distance between output labels and the true labels. MSE is also usable for classification problems and has been used in the context of SCA as discussed before [8].

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

$$msle(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2. \tag{3}$$

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

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 [27]:

$$log\_cosh(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\log(\cosh(\hat{y}_i - y_i)). \tag{4}$$

Classification Losses For classification, the de facto standard loss function 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 [6,10,30] and is also commonly used in SCA [2,9,13]. 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. Minimizing the cross-entropy, which represents the difference between the distribution modeled by the deep learning model and the true distribution of the classes, should therefore improve the predictions of the neural network:

$$cce(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j}\log(\hat{y_{i,j}}), \tag{5}$$

where c denotes the number of classes. 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 correct classes and wrong classes with the highest predicted probability:

$$cat\_hinge(\mathbf{y}, \hat{\mathbf{y}}) = max(1 - y_i * \hat{y_i}, 0). \tag{6}$$

Custom SCA Losses More recently, two SCA-specific loss functions have been proposed. One of them is the ranking loss (RKL) function proposed by Zaid et al. [31]. The ranking loss 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:

$$rkl(\mathbf{s}) = \sum_{\substack{k \in \mathcal{K} \\ k \neq k^*}} \left(\log_2 \left(1 + e^{-\alpha(s(k^*) - s(k))} \right) \right), \tag{7}$$

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, α 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 [31] on Github [1].

The other custom loss function is the cross-entropy ratio (CER) loss proposed by Zhang et al. [33]. The authors introduced the CER as a metric to estimate the performance of a deep learning model in the context of SCA. They also showed that their metric could be used as a loss function directly by using a shuffled set of labels:

$$cer(\mathbf{y}, \hat{\mathbf{y}}) = \frac{CE(\mathbf{y}, \hat{\mathbf{y}})}{\frac{1}{n} \sum_{i=1}^{n} CE(\mathbf{y}_{r_i}, \hat{\mathbf{y}})}, \tag{8}$$

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. Here, the variable N denotes the amount of shuffled sets to use. [33] 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. In our first experiments, to balance between computational complexity and the potential increase in accuracy, we use $N = 10$.

2.3 Datasets

The first dataset considered is the ASCAD dataset introduced by Benadjilla et al. [2]. The ASCAD dataset is generated by taking measurements from an ATMega8515 running masked AES-128 and is proposed as a benchmark dataset

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

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 denote this dataset as ASCAD_fixed. The dataset is provided on the ASCAD GitHub repository [2].

The second ASCAD dataset uses variable keys. The ASCAD_variable dataset consists of 200 000 profiling and 100 000 attack traces, each consisting of 1 400 features. Different from the ASCAD_fixed dataset, the keys used in the profiling set are variable. The ASCAD_variable dataset is available on the ASCAD GitHub repository [3].

3 Related Works

In recent years, the usage of deep learning became more popular in the context of SCA [3, 9, 13, 15, 21, 23, 26, 32]. Many of these works focus on improving certain aspects of the used MLP or CNN architectures. Those improvements aim to increase the model's attack performance by decreasing the number of traces required to reach a guessing entropy of 1 for the correct key. However, all of these 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 [13], the authors 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 [2, 17, 32] or mean squared error [16, 26]. Indeed, in [27], the authors show that minimizing the categorical cross-entropy loss is equivalent to increasing the Perceived Information (PI) [22], a metric commonly used in the context of SCA. Perin and Picek conducted a related analysis where they explored the influence of different optimizers for deep learning-based SCA [18].

More recently, two novel loss functions specifically for usage in the context of SCA have been proposed. Zaid et al. [31] propose ranking loss (RKL), a loss function that uses a pairwise comparison between the possible different key hypotheses to maximize the models' success rate. Zhang et al. [33] 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 [33].

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 our knowledge, no broad comparison has been made between commonly used loss functions such as categorical cross-entropy, mean squared

[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`

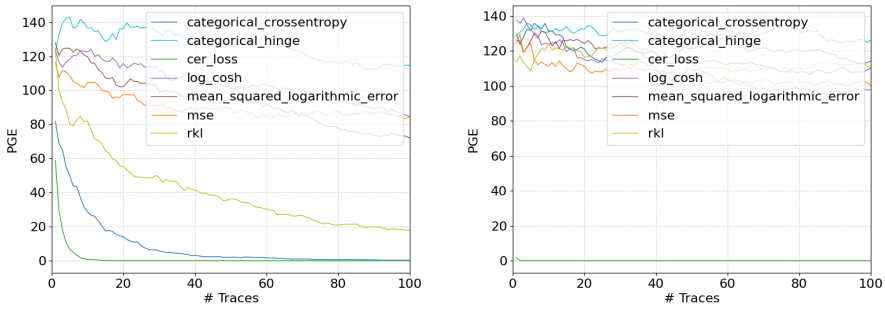
error, or hinge loss and these novel SCA-based loss functions on different architectures, leakage models, and datasets. This work systematically compares commonly used loss functions and the novel loss functions, namely CER and RKL.

4 Experiment Setup

In this section, we describe the setup of the experiments we performed. The experiments were performed with a single CPU and an NVIDIA GTX 1080 Ti graphics processing unit (GPU) with 11 Gigabytes of GPU memory and 3 584 GPU cores.

Several different hyperparameters influence the training process of a deep learning model. 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. As mentioned, by picking a single model with certain training hyperparameters, we could end up with certain hyperparameters that influence one loss function more than the others. A demonstration is shown in Figure 1. 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 1a), CER loss performs the best, followed by CCE and RKL. However, when the learning rate is increased to 0.001 (Figure 1b), the loss functions that lead to a converged GE (CCE and RKL) are not functional anymore. At the same time, for the CER loss, the performance is even increased. Indeed, Figure 1 indicate the influence of the hyperparameter on the performance of the loss function. Benchmarking with a single attack model and the fixed attack setting can not represent the generality of a loss function. Knowing this, we implemented the following testing scenarios for a fair loss function comparison:

1. To reduce the effect of certain combinations of the loss function and other hyperparameters, we use a median model to make our comparison. More precisely, we train 100 random models for each scenario and evaluate these models with the guessing entropy metric. Then, the median-performing model (in terms of guessing entropy) is selected as a representative model to benchmark different loss functions.
2. The optimized training hyperparameters for a specific loss function might lead to well-performing models that may not work for the median-performing model. Therefore, we also perform hyperparameter optimization via a random search for each considered loss function.
3. Recent papers proposed different MLP and CNN architectures that perform well on the considered dataset and leakage model with a specific loss function (CCE)s [2, 32]. The performance of these models with other loss functions is unknown. These architectures might work well only with the categorical cross-entropy, provide good performing models regardless of the chosen loss functions, or perform even better with other loss functions. We, therefore,



(a) Learning rate 0.00001

(b) Learning rate 0.001

Fig. 1: All models are trained with the same hyperparameters, except the learning rate. The learning rate influences the performance of models for some loss functions more than others. In the scenario with the learning rate set to 0.001, the performance of the CER loss is increased while the other losses fail to result in a model converging to a GE of 1.

conduct experiments on these state-of-the-art architectures with different functions and compare their performance.

To perform a broad comparison between the different loss functions, we define 12 different scenarios to make the comparison. Each of these scenarios combines a dataset, a leakage model, and an architecture type.

Architecture Types We consider two architecture types: multilayer 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 works [2, 13, 32].

For both the median model generation and the hyperparameter optimization, various model hyperparameters should be explored. In their work, [18] specified a search space for 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. We use the same search space as a basis for this experiment. [2] perform several experiments with a different number 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 optimizer. To balance computational cost and performance, we train each model for 200 epochs. In terms of optimizers, both the Adam and RMSProp optimizers perform well [2, 18]. In addition to the hyperparameter ranges proposed in [18], we add both optimizers as an option. Naturally, the range of learning rates is broadened. The possible values for each hyperparameter for the MLP models are given in Table 1.

Table 1: Hyperparameter space for multilayer perceptrons.

Hyperparameter	**Min**	**Max**	**Step size**
Dense layers	2	8	1
Neurons per layer	100	1 000	100
Learning rate	0.000001	0.001	0.00001
Batch size	100	1 000	100
	Options		
Activation function	[ReLu, SELU, ELU, Tanh]		
Optimizer	[Adam, RMSProp]		

For the CNN hyperparameters, the search space is again based on the work of [18] with the same changes to the learning rate, the number of epochs, and optimizers as for the MLPs. Additionally, a batch normalization layer, as introduced by [7], is applied after the input layer and after each convolutional block, as is done in earlier work to improve the performance of CNNs [2, 3, 18]. The possible values for each hyperparameter for the MLP models are given in Table 2.

Table 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	1 000	100
Learning rate	0.000001	0.001	0.00001
Batch size	100	1 000	100
	Options		
Activation function	[ReLu, SELU, ELU, Tanh]		
Optimizer	[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. In almost all recent works on deep learning for SCA, the categorical cross-entropy and mean squared error (MSE) loss functions are used. In this paper, 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 [31], and cross-entropy ratio (CER) [33], two recently proposed novel loss functions specifically for the SCA domain are considered as well.

Pre-processing The pre-selected window of features is used for both datasets, and no further selection of points of interest is made. Earlier work suggests that scaling SCA features to values between 0 and 1 works well [12, 31]. Therefore, a similar method is applied in this work, and for every experiment we perform, the features are normalized to values between 0 and 1. This is done by using the MinMaxScaler [4] from the scikit-learn Python module [5].

Random Architectures First, we have to find a median model to test each loss function. 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 summarize, for each of the scenarios, we perform the following steps:

1. Generate, train, and test 100 random models for each loss function.
2. Select the median model in terms of guessing entropy.
3. Train and test the median model 10 times to compensate for the effect of random weight initialization.
4. From those ten models, select the median model per loss function based on guessing entropy.
5. Compare the attack performance of each loss function in terms of guessing entropy, training time, and the number of trainable parameters.

The second set of experiments optimized the training hyperparameter via a random search for each loss function. When comparing with the previous test scenario, the only difference comes from selecting the best-performing model instead of the median-performing model. Similarly, the best models for each loss function are then benchmarked on guessing entropy, training time, and the number of trainable parameters.

Note that we train ten new models in both phases with the found median or best-performing model (step 3). Indeed, the models with the same hyperparameters sometimes perform differently due to randomness in, for example, the random initialization of the weights, outliers might occur in terms of performance, or the training might fail. We, therefore, choose the median of those ten 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 the architectures optimized for each loss function in the second phase.

For all of the experiments, the same attack settings are used. The number of profiling traces used is 50 000 for both datasets. In the attacking phase, we use up to 2 000 traces for the ASCAD_fixed dataset and up to 3 000 traces for the ASCAD_variable dataset.

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`

[5] `https://github.com/scikit-learn/scikit-learn`

State-of-the-art Architectures in Profiling SCA Different state-of-the-art deep learning models proposed in recent papers only consider the categorical cross-entropy loss function [2, 32]. Here, we will use the MLP_{best} architecture proposed by [2], and the CNN architecture used in [31], which we will denote $CNN_{methodology}$. Although there are some remarks in terms of the designing of the $CNN_{methodology}$ model [28], 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 models are trained and tested on the
ASCAD_fixed and ASCAD_variable datasets with each loss function. Although the architecture is optimized for the ASCAD dataset with the fixed key, we also test the performance on the ASCAD_variable dataset for consistency. Like before, we train the same model ten times for each of the datasets, leakage models, and loss functions and take the median performing model in terms of guessing entropy. As specified before, the evaluation metrics are guessing entropy, training time, and the number of trainable parameters.

5 Experimental Results

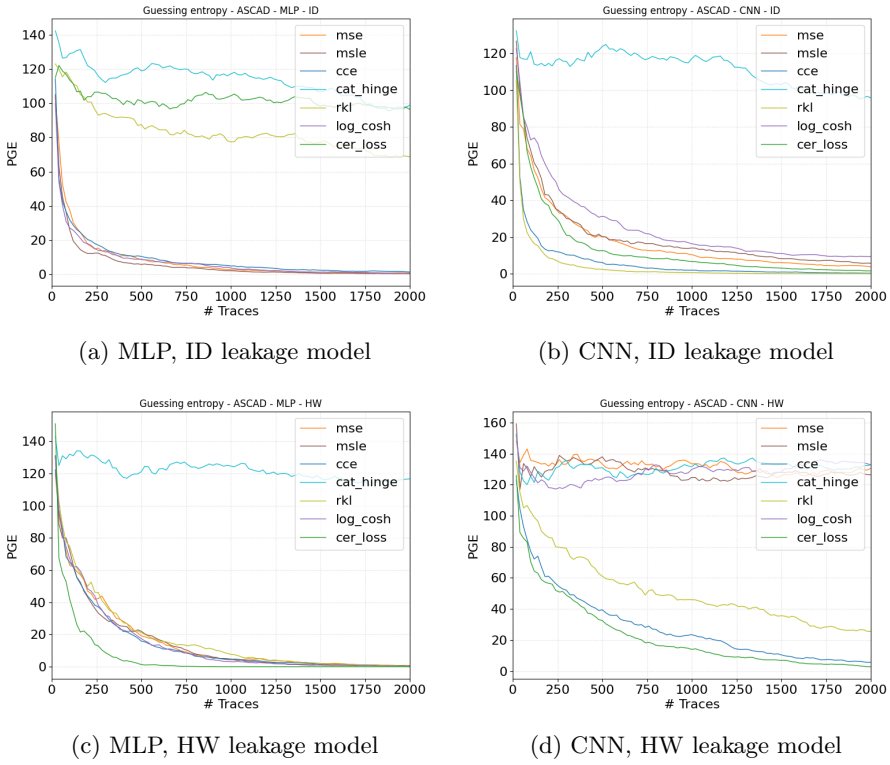
In this section, we discuss the results for each of the experiments above. We will look at the performance of the loss functions on median models and models optimized via random search. The performance of the state-of-the-art architectures with different loss functions is evaluated as well.

5.1 Median Model and Hyperparameter Optimization

ASCAD_fixed We first consider the performance of the different loss functions on the ASCAD_fixed dataset. Figure 2 shows the guessing entropy over 100 attacks for each of the scenario's median models. Figure 3 shows the guessing entropy for each of the optimized models.

First, we notice that the ASCAD_fixed dataset is vulnerable to SCA with various attack settings. Most of the loss functions lead to a model which can retrieve the correct key in less than 2 000 traces with median-performing MLP architecture. When we look at the optimized models, we see that even a simple parameter optimization approach via random search results in an improved attack performance. For instance, CER models reach a GE of 1 in less than 500 traces, which is comparable to the state-of-the-art attack performances [28, 31].

When looking at the performance of each loss function, as expected, 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 shows the 100 models generated for the hyperparameter optimization experiment with a CNN architecture and the ID leakage model for both the categorical cross-entropy and MSE.



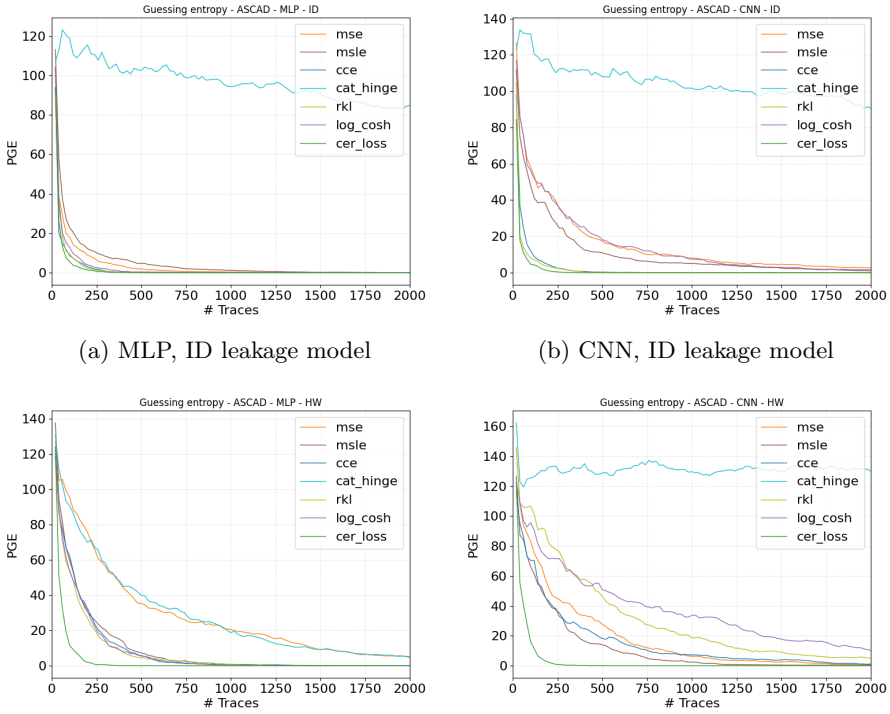
(a) MLP, ID leakage model

(b) CNN, ID leakage model

(c) MLP, HW leakage model

(d) CNN, HW leakage model

Fig. 2: GE of the median MLP and CNN models on the ASCAD_fixed dataset.



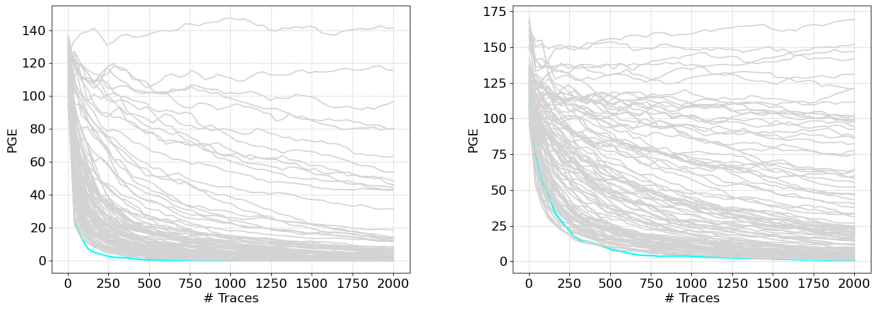
(a) MLP, ID leakage model

(b) CNN, ID leakage model

(c) MLP, HW leakage model

(d) CNN, HW leakage model

Fig. 3: GE of the optimized MLP and CNN models on the ASCAD_fixed dataset.



(a) Models trained with CCE

(b) Models trained with MSE

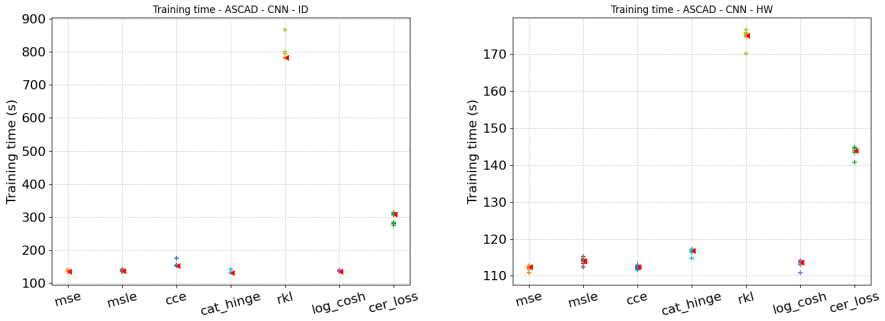
Fig. 4: The 100 random CNN models generated for hyperparameter optimization with categorical cross-entropy and MSE as loss functions on the ASCAD_fixed dataset and ID leakage model. Almost all models trained with the categorical cross-entropy perform well and converge towards a GE of 1 relatively fast, while there is more variation in the models trained with MSE. The blue line indicates the model with the lowest $\overline{N}_{T_{GE}}$.

On the other hand, looking at CER loss optimized for SCA, we also see some interesting behavior. First, CER loss outperforms every other function in all but two scenarios. In fact, it is only nonfunctional on the scenarios with a median model and the ID leakage model Figure 3a. [33] introduced the CER loss function to improve the performance of deep learning models on imbalanced SCA data, i.e., when the HW leakage model is considered. Our results confirm that this loss function performs well in those scenarios. Our results outperform the MLP models of the original CER paper and performing much better than the other loss functions. Furthermore, [33] showed 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 optimized, as shown in Figure 3a and Figure 3b, the CER loss is also very suitable for the ID leakage model. Again, the models trained with CER loss outperform the models trained with categorical cross-entropy. Therefore, we can 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 optimized, it significantly reduces the number of traces needed to perform a successful attack compared to the categorical cross-entropy and other loss functions.

The second novel loss function, ranking loss (RKL), performs less consistently. [31] compared the RKL function to categorical cross-entropy and CER loss, stating that the RKL outperforms both those functions. However, they only compare a single CNN architecture and only consider the ID leakage model [31].

Our CNN median and optimized model results for the ID leakage model show 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.

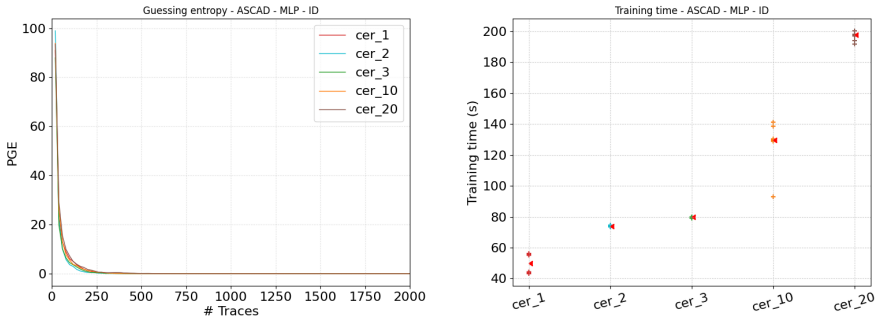
Another remark that has to be made when discussing these results is the required training time. Figure 5 shows the training times for the median model with each of the loss functions on both the HW and ID leakage models. When all other hyperparameters are the same, the RKL and CER loss functions significantly reduce the training speed compared with other loss 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 model is considered, where the output consists of nine classes, and an even larger impact when the ID leakage model is used, where there are 256 output classes. The increased training time in case of the CER loss is also due to how the function is constructed. CER loss, as explained in subsection 2.2. More precisely, it 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 slower training than other loss functions. Since [33] do not analyze the impact of different values for N, we chose $N = 10$ for these experiments. However, as shown in Figure 6, 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 compared 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.



(a) ID leakage model

(b) HW leakage model

Fig. 5: For each of the loss functions, ten models were trained with the hyperparameters of the median model. For each loss function, the ten models with the same hyperparameters are plotted. The red triangles mark the median training times.



(a) Guessing entropy (b) Training time

Fig. 6: Guessing entropy and training time for the optimized model with CER loss using different values of N.

The only function that does not often lead to a converging model is the categorical hinge loss. Only in two scenarios, namely the optimized MLP and CNN models with the HW leakage model (Figure 3c and Figure 3d), 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 number of classes when the HW leakage model is considered. The median models for these scenarios all have a learning rate between 0.0001 and 0.0007, while the optimized 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, the negative part of the loss is calculated based on the wrong class with the highest probability, i.e., the biggest classification error. Unfortunately, there are too many wrong classes (255) and only one correct class with the ID leakage model. Due to random initialization of the weights, the loss coming from wrongly classified traces will stay approximately 1 at the start of training, and the main contribution to the change of the loss has to come from a correctly classified example. This will not occur often enough since, with the ID leakage model, there are 256 classes. With the HW leakage model, there are only nine classes to consider. Even with random guessing, a correct classification will happen more often, impacting the loss and allowing the model to learn. Consequently, due to the difference in the number of classes between the ID and HW leakage models and a low learning rate, the categorical hinge loss works better with the HW leakage model.

Finally, we look at the number of trainable parameters for the scenarios with optimized 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 3 shows the number of trainable parameters that each of the optimized models per loss function has. We see that the optimized models trained with the categorical cross-entropy often have the least trainable parameters compared to the other loss functions. Other functions, like MSE and CER

loss, seem to perform well with smaller models as well, while the categorical hinge and MSLE loss only perform well with larger models.

Table 3: The number of trainable parameters for the optimized models per loss function and scenario. The smallest number of trainable parameters for each scenario is marked blue, the largest orange.

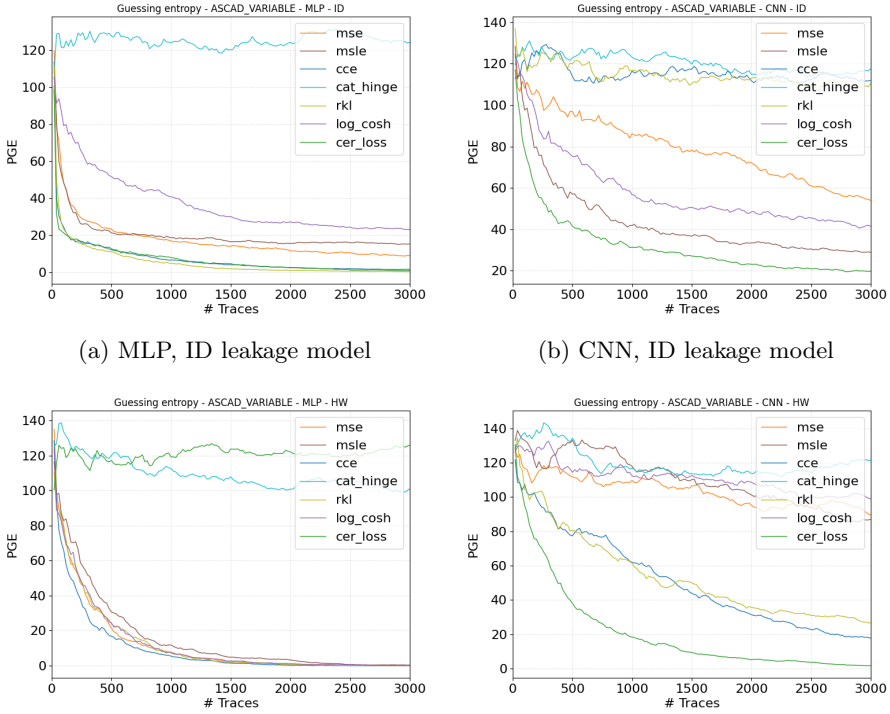
Loss function	**MLP ID**	**MLP HW**	**CNN ID**	**CNN HW**
Categorical cross-entropy	116 156	302 809	53 244	1 124 565
Categorical hinge	1 295 656	3 882 609	1 022 920	2 852 165
CER loss	467 956	483 909	356 424	598 853
Log cosh	126 256	856 009	335 824	1 011 657
MSLE	543 456	1 449 909	5 738 736	1 335 177
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 the CER loss seems to be the best choice of 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 number of trainable parameters and, when using $N = 1$, are still fast during training.

We also conducted experiments with this dataset and added a desynchronization countermeasure (desynchronization equal to 50). Interestingly, both SCA-specific loss functions (RKL and CER) perform poorly (do not converge) for both the HW and ID leakage models, while CCE performs well.

ASCAD_variable Next, we look at the results on the ASCAD dataset with random keys used during the profiling phase. Figure 7 shows the GE performance on the median models. Figure 8 shows the performance for the optimized models.

For the experiments performed on the ASCAD_variable dataset, we see results comparable to those on the ASCAD_fixed dataset. In most scenarios, the models trained with CER loss perform the best, followed closely by those 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 the HW leakage model in Figure 7c, the CER loss model does not converge. Compared to other scenarios with the HW leakage model, 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, RMSprop optimizer, and batch size of 300. The investigation of the training process of models with these hyperparameters and CER loss shows that the loss sometimes becomes a NaN value during training. The underlying cause could be the exploding gradients problem: gradients during the backpropagation are getting too large or small, causing the learning process to fail [19]. To verify this, Figure 9 shows the largest and smallest gradient of the input layer for each of the epochs



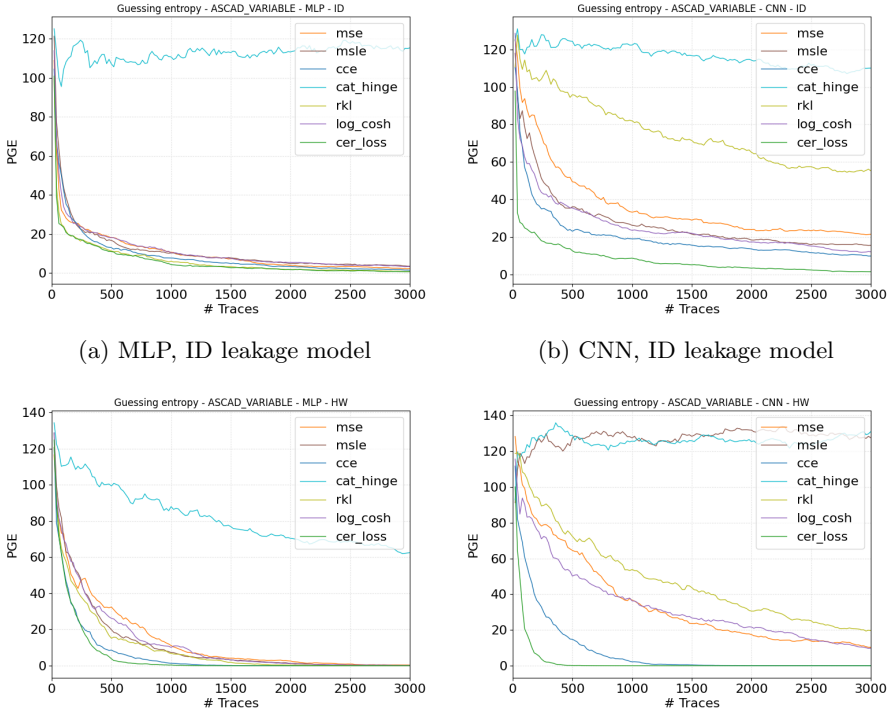
(a) MLP, ID leakage model

(b) CNN, ID leakage model

(c) MLP, HW leakage model

(d) CNN, HW leakage model

Fig. 7: GE of the median MLP and CNN models on the ASCAD_variable dataset.



(a) MLP, ID leakage model

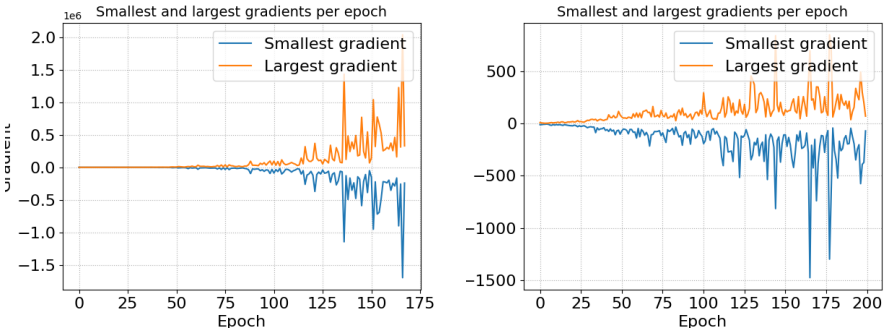
(b) CNN, ID leakage model

(c) MLP, HW leakage model

(d) CNN, HW leakage model

Fig. 8: GE of the optimized MLP and CNN models on the ASCAD_variable dataset.

during the training process of one of these models. The cause of the exploding gradient problem could come from the combination of several hyperparameters, i.e., the ELU activation function, the loss function, and the 0-1 normalization used during pre-processing. Indeed, normalizing all feature values to values between 0 and 1 removes any negative values from the profiling traces. The ELU activation function's output 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. What is more, it also means that, since we normalized to values between 0 and 1, the output of the activation function will always be positive (also holds for the ReLU activation function). This, in combination with the CER loss function, and some cases ranking loss, causes the gradients to get too large, leading to a poorly performing model or even a failed training.



(a) 0-1 normalization

(b) Z-score normalization

Fig. 9: The largest and smallest gradient of the input layer during training of a model with CER loss in the median MLP with the HW leakage model when different pre-processing is done. The gradients explode to large values with 0-1 normalization, but they do not with applied Z-score normalization.

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{9}$$

Some solutions to mitigate this problem do exist. We could use a different combination of the loss function, activation function, and pre-processing method. Figure 9b shows, for example, the same gradients but with the profiling traces normalized by Z-score normalization (standardization) [28]. The attack performance is similar to when 0-1 normalization is used. Consequently, when using CER loss and RKL with activation functions such as ELU, standardization instead of 0-1 normalization seems preferred. Another possible solution might be clipping the gradients when they get too large or too small. For our experi-

ments, we have kept the pre-processing similar for each experiment as described in Table 4.

Some differences are visible if we compare the number of trainable parameters of the optimized models. Table 4 shows the number of trainable parameters of the optimized 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 optimized categorical cross-entropy and ranking loss MLP models are relatively small, while their CNN counterparts are larger.

Table 4: The number of trainable parameters for the optimized models per loss function and scenario. The smallest number of trainable parameters for each scenario is marked blue, the largest orange.

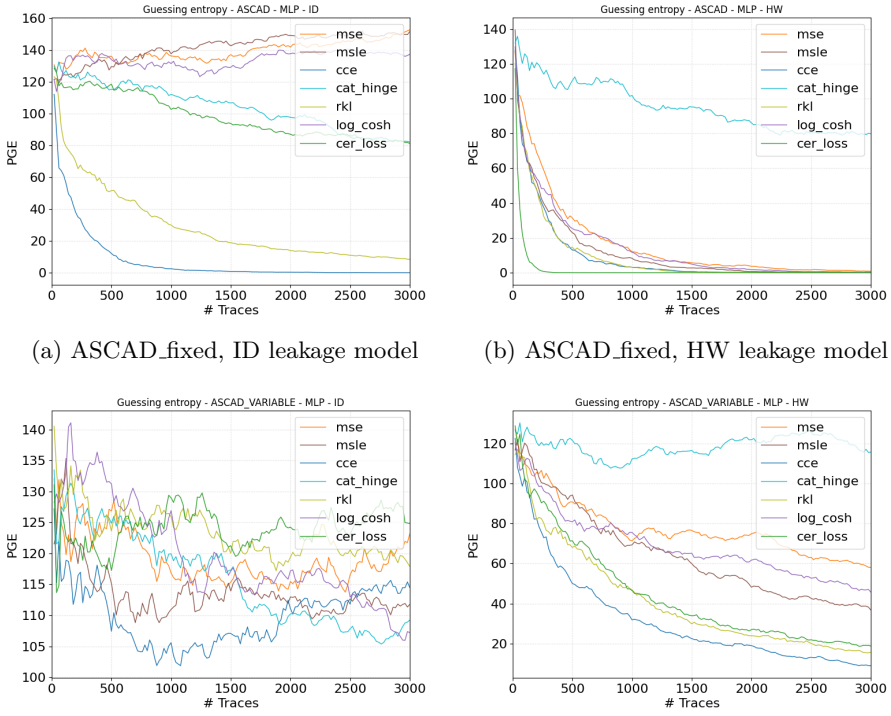
Loss function	**MLP ID**	**MLP HW**	**CNN ID**	**CNN HW**
Categorical cross-entropy	1 830 756	402 609	3 869 540	3 844 221
Categorical hinge	2 582 256	4 512 609	1 007 956	5 071 253
CER loss	1 966 656	2 079 909	1 602 260	937 253
Log cosh	6 662 256	3 701 709	314 616	245 481
MSLE	1 129 456	4 413 009	4 499 560	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

Similar to the ASCAD_fixed dataset, we can conclude that the CER loss is the preferred loss function when the ASCAD_variable dataset is considered. It again outperforms the categorical cross-entropy in terms of guessing entropy 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 optimized CNN models, the categorical cross-entropy has a $\overline{N}_{T_{GE}}$ of 2 520, while for CER loss, this is reduced to 720.

5.2 State-of-the-art Architectures

Next, we present the results of the experiments where two state-of-the-art architectures are considered, as described in section 4. First, we look at the results from the MLP_{best} model, introduced by [2], followed by the results of the $CNN_{methodology}$ model introduced by [32]. Figure 10 shows the performance in guessing entropy for the models.

Observe 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 optimized models from the previous experiments outperform the MLP_{best} in every scenario. The authors managed to successfully attack this model on the ASCAD_fixed dataset and the ID leakage model, but only when they train the model for 400 epochs or more.



(a) ASCAD_fixed, ID leakage model

(b) ASCAD_fixed, HW leakage model

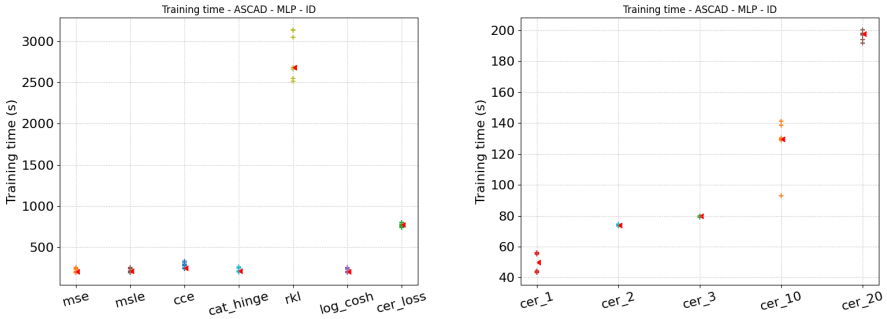
(c) ASCAD_variable, ID leakage model

(d) ASCAD_variable, HW leakage model

Fig. 10: GE of the MLP_{best} models on the ASCAD_fixed and ASCAD_variable datasets.

In the context of the original paper, similar to the scenario in Figure 10a, the categorical cross-entropy performs the best. Our results show that using another loss does not improve the performance presented in their work, where they only use the ID leakage model. When the HW leakage model is considered, the CER loss again performs best but still not better than the previously shown median or optimized models. Therefore, we can conclude that the MLP_best architecture does not perform particularly well, despite the loss function used.

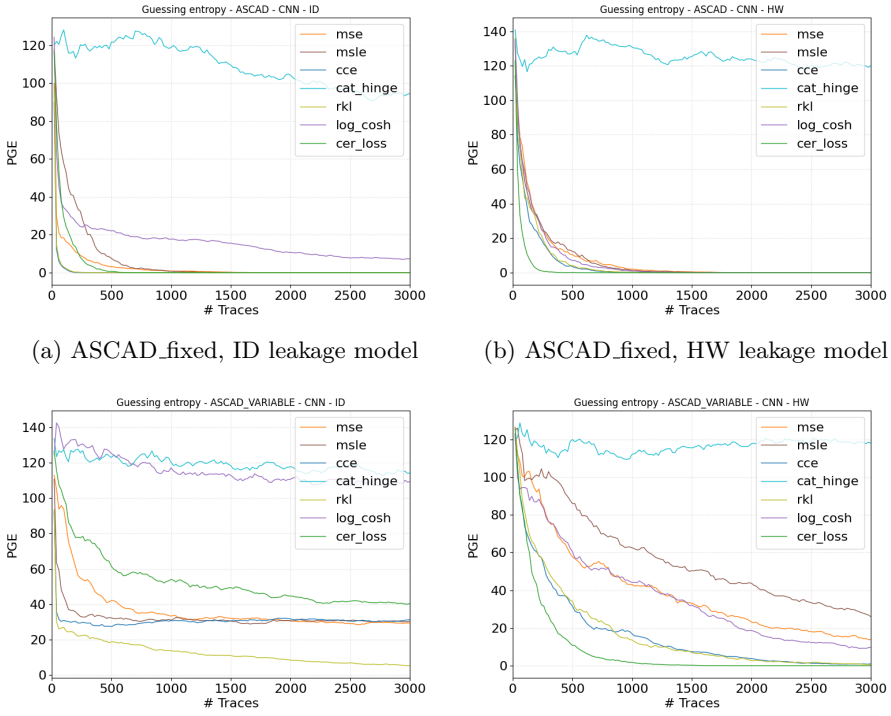
The training times align with what we previously saw when comparing the different loss functions, although it takes significantly longer than the optimized and median models from our earlier experiments. Figure 11 shows, for example, the training times for the models on the ASCAD_fixed dataset with the ID leakage model. Each of the optimized 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, almost all of the median and optimized models have fewer layers (2-4 instead of 5), reducing the complexity and using a larger batch size of 800-1 000. This is also visible in the number of trainable parameters. The MLP_{best} has 352 456 trainable parameters, where most of the optimized models had less than 200 000.



(a) MLP_{best} models (b) Optimized models

Fig. 11: Training times for the MLP_{best} and optimized models on the ASCAD_fixed dataset.

Next, we look at the performance of the $CNN_{methodology}$ model. In their work, [32] introduced a methodology to create small but well-performing CNN architectures. For our experiments, we used their CNN model created for the ASCAD_fixed dataset. Note that this is also the model that was used by [31] to demonstrate the performance of the ranking loss function. Figure 12 shows the results in terms of guessing entropy.



(a) ASCAD_fixed, ID leakage model

(b) ASCAD_fixed, HW leakage model

(c) ASCAD_variable, ID leakage model

(d) ASCAD_variable, HW leakage model

Fig. 12: Guessing entropy of the $CNN_{methodology}$ models on the ASCAD_fixed and ASCAD_variable datasets.

Since the CNN architecture was created specifically for the ASCAD_fixed dataset and the ID leakage model, it is no surprise that the performance is best in that scenario. Both the categorical cross-entropy and ranking loss can perform a successful attack with less than 200 traces. This is comparable to the performance of the model in the original paper [32]. However, in our experiments, the ranking loss does not outperform the categorical cross-entropy. In contrast with the ranking loss paper, the categorical cross-entropy performs slightly better. One possible explanation of this difference with [31] is the way they present their results. More precisely, they take the average $\overline{N}_{T_{GE}}$ over ten converging models and compare those averages. In our experiments, however, we take the median of 10 models and observe that 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 for the other dataset, we see that the model is not very successful when the ID leakage model is used. This is no surprise since the model is optimized for usage on the ASCAD_fixed dataset with the categorical cross-entropy in mind. In the HW leakage model, the models trained with the CER loss outperform the other functions by quite a margin. They retrieve the correct key for the ASCAD_variable with less than 1 500 traces, where the models with other functions are not successful or need more than 3 000 traces.

6 Discussion

With these experiments, we systematically compared different loss functions in various deep learning-based SCA scenarios for the first time in the SCA domain. The results reveal interesting behavior 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 model, optimized models with CER loss outperform models with other loss functions in many scenarios with the ID leakage model. While [33] already demonstrated that CER loss might work on balanced data, our experiments confirm this for datasets often used in the SCA research domain. 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 better choices.

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, 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. To conclude, it showed no obvious weaknesses.

The other loss functions we considered did, in general, not show promising results. While used before in related 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.

In our results, we saw no consistent differences between the behavior of loss functions with MLPs or CNNs. In general, loss functions that performed well did so on both architecture types. If we look at the other hyperparameters, we 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 do not perform well are relatively complex. They require more neurons and dense layers or more filters.

We do not see large differences between the different loss functions despite the more complex models in terms of the training time. The only function that is significantly slower to train than others is the ranking loss. Especially when the ID leakage model is considered, and the number of classes is high, the training time is increased by up to a factor of ten, as visible in Figure 5. The CER loss is also slower when a larger N is chosen. Nevertheless, a larger N is not required for better-performing models.

Our work aims to improve the tools that researchers have when performing SCA with deep learning. To that end, we created an overview of strengths and weaknesses in Table 5 for each loss function as seen in our experiments.

Comparing the guessing entropy performance, models with the CER loss function performed best in 20 tested scenarios, the categorical cross-entropy in eight, and the ranking loss in four. 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 3 000 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 optimized most often lead to a successful attack. For each of the loss functions, ten models with the same optimized hyperparameters were trained per scenario. Since there are eight scenarios (combinations of two datasets, two leakage models, two architecture types), a total of 80 models per loss function were trained. When we compare the $\overline{N}_{T_{GE}}$ for all those models with CER loss and categorical cross-entropy, we see that 54 out of 80 models with CER loss reached a guessing entropy of 1, with a median $\overline{N}_{T_{GE}}$ of 590. For categorical cross-entropy, 37 models were successful with a median $\overline{N}_{T_{GE}}$ of 1 560. To test the significance of the difference in $\overline{N}_{T_{GE}}$ between the two functions, we perform a Mann-Whitney-U test [14]. We use $n_1 = 54, n_2 = 37$, and $\alpha = 0.05$. The calculated statistic $U = 178.5 < U_{crit} \approx 756$ 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 0.91.

Table 5: 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 – Robust to different architectures – Training time	
Categorical hinge	– Training time	– Rarely leads to a successful attack
CER loss	– Best performance – Specifically good against the 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 – Performance is not good
Ranking loss	– Performance in some specific scenarios	– Training time – Not robust to different architectures – Introduces new tunable parameter α

The other novel function, ranking loss, does not seem to increase the performance. Considering all the optimized models again, 24 models with ranking loss reach a $\overline{N}_{T_{GE}}$ of 1. The median $\overline{N}_{T_{GE}}$ of these models is 1 420 in comparison to 1 560 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 37 models with categorical entropy, which successfully attacked the considered datasets and 24 with ranking loss. We use $n_1 = 24$, $n_2 = 37$, and $\alpha = 0.05$. We calculate the statistic $U = 336 > U_{crit} \approx 311$ and $p = 0.23$. Since $0.23 > 0.05$, 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 with the original paper might be the variation in the used architectures. The performance we see with the architecture and leakage model used by [31] in Figure 12a is similar to the performance presented in their paper [31]. However, in scenarios with other architectures and leakage models, the performance compared to the categorical cross-entropy shows no improvement. Besides that, using ranking loss also severely impacts the training time and introduces a new hyperparameter α, which has to be optimized for each profiling dataset.

7 Conclusions and Future Work

This work investigates several loss functions commonly used in the machine learning domain and compares them with two recently proposed SCA-specific loss functions. We analyze two datasets and two leakage models, considering guessing entropy, the number of trainable parameters, and the training size. Our results show 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.

Since our experiments indeed confirm that a custom SCA loss function is the best, this opens interesting future research directions. In other domains in which deep learning is applied, several works have also introduced new loss functions that improve the performance in that context [1, 4, 5, 11]. 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. It remains an open question how would such more complex loss functions perform in the SCA context. Next, it would be interesting to explore what elements of loss functions perform well and how to combine them to construct new loss functions for side-channel analysis.

References

1. Barz, B., Denzler, J.: Deep Learning on Small Datasets without Pre-Training using Cosine Loss. Tech. rep. (2020)
2. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD

Database-Long Paper. Journal of Cryptographic Engineering **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8
3. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures-Profiling Attacks without Pre-Processing. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68 (2017). https://doi.org/10.1007/978-3-319-66787-4_3
4. Hajiabadi, H., Babaiyan, V., Zabihzadeh, D., Hajiabadi, M.: Combination of loss functions for robust breast cancer prediction. Computers and Electrical Engineering **84** (6 2020). https://doi.org/10.1016/j.compeleceng.2020.106624
5. Hajiabadi, H., Molla-Aliod, D., Monsefi, R., Yazdi, H.S.: Combination of loss functions for deep text classification. International Journal of Machine Learning and Cybernetics **11**(4), 751–761 (4 2020). https://doi.org/10.1007/s13042-019-00982-x
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition (2015)
7. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2 2015), http://arxiv.org/abs/1502.03167
8. Janocha, K., Czarnecki, W.M.: On Loss Functions for Deep Neural Networks in Classification. Tech. rep. (2017), https://arxiv.org/abs/1702.05659
9. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make Some Noise Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925 **2019**(3), 148–179 (2019). https://doi.org/10.13154/tches.v2019.i3.148-179
10. Kussul, N., Lavreniuk, M., Skakun, S., Shelestov, A.: Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data. IEEE Geoscience and Remote Sensing Letters **PP**, 1–5 (7 2017). https://doi.org/10.1109/LGRS.2017.2681128
11. Lin, T.Y., Goyal, P., Girshick, R.B., He, K., Dollár, P.: Focal Loss for Dense Object Detection. CoRR **abs/1708.02002** (2017), http://arxiv.org/abs/1708.02002
12. Maghrebi, H.: Deep Learning based Side Channel Attacks in Practice. IACR Cryptol. ePrint Arch. **2019**, 578 (2019)
13. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. Tech. rep. (2016), https://eprint.iacr.org/2016/921
14. Mann, H.B., Whitney, D.R.: 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). https://doi.org/10.1214/aoms/1177730491, https://doi.org/10.1214/aoms/1177730491
15. Masure, L., Dumas, C., Prouff, E.: A Comprehensive Study of Deep Learning for Side-Channel Analysis (2019)
16. Moos, T., Wegener, F., Moradi, A.: Dl-la: Deep learning leakage assessment: A modern roadmap for sca evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 552–598 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.552-598, https://tches.iacr.org/index.php/TCHES/article/view/8986
17. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 337–364 (Aug 2020). https://doi.org/10.13154/tches.v2020.i4.337-364, https://tches.iacr.org/index.php/TCHES/article/view/8686

18. Perin, G., Picek, S.: On the influence of optimizers in deep learning-based side-channel analysis. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12804, pp. 615–636. Springer (2020). https://doi.org/10.1007/978-3-030-81652-0_24, https://doi.org/10.1007/978-3-030-81652-0_24
19. Philipp, G., Song, D., Carbonell, J.G.: The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. CoRR **abs/1712.05577** (2017), http://arxiv.org/abs/1712.05577
20. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339
21. Picek, S., Samiotis, I.P., Heuser, A., Kim, J., Bhasin, S., Legay, A.: On the Performance of Convolutional Neural Networks for Side-channel Analysis. pp. 157–176. Springer Verlag (2018). https://doi.org/10.1007/978-3-030-05072-6_10, https://research.tudelft.nl/en/publications/on-the-performance-of-convolutional-neural-networks-for-side-chan
22. Renauld, M., Standaert, F.X., Charvillon, N.V., Kamel, D., Flandre, D.: 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. Lecture Notes in Computer Science, vol. 6632, p. 109. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_8, https://www.iacr.org/archive/eurocrypt2011/66320107/66320107.pdf
23. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 677–707 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.677-707, https://tches.iacr.org/index.php/TCHES/article/view/8989
24. Sammut, C., Webb, G.I. (eds.): Mean Squared Error, pp. 653–653. Springer US, Boston, MA (2010). https://doi.org/10.1007/978-0-387-30164-8_528, https://doi.org/10.1007/978-0-387-30164-8_528
25. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
26. Timon, B.: 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). https://doi.org/10.13154/tches.v2019.i2.107-131, https://tches.iacr.org/index.php/TCHES/article/view/7387
27. Wang, Q., Ma, Y., Zhao, K., Tian, Y.: A Comprehensive Survey of Loss Functions in Machine Learning. Annals of Data Science (2020). https://doi.org/10.1007/s40745-020-00253-5
28. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a Methodology for Efficient CNN Architectures in Profiling Attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–

168 (2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://github.com/KULeuven-COSIC/TCHES20V3_CNN_SCA
29. Yash Srivastava, Vaishnavand Murali, Shiv Ram Dubey: A Performance Evaluation of Loss Functions for Deep Face Recognition. In: Babu R. Venkatesh, Prasann, M., Namboodiri, V.P. (eds.) Computer Vision, Pattern Recognition, Image Processing, and Graphics. pp. 322–332. Springer Singapore, Singapore (2020)
30. Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X.: Byte-level malware classification based on markov images and deep learning. Computers & Security **92**, 101740 (2020). https://doi.org/https://doi.org/10.1016/j.cose.2020.101740, https://www.sciencedirect.com/science/article/pii/S0167404820300262
31. Zaid, G., Bossuet, L., Dassance, F., Habrard, A., Venelli, A.: Ranking loss: Maximizing the success rate in deep learning side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(1), 25–55 (Dec 2020). https://doi.org/10.46586/tches.v2021.i1.25-55, https://tches.iacr.org/index.php/TCHES/article/view/8726
32. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391
33. Zhang, J., Zheng, M., Nan, J., Hu, H., Yu, N.: A Novel Evaluation Metric for Deep Learning-Based Side Channel Analysis and Its Extended Application to Imbalanced Data **2020**(3), 73–96 (2020). https://doi.org/10.13154/tches.v2020.i3.73-96

A Details of Optimized Architectures

Table 6: Hyperparameters optimized MLP models for the ID leakage model, ASCAD_fixed dataset.

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 7: Hyperparameters optimized MLP models for the HW leakage model, ASCAD_fixed dataset.

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 8: Hyperparameters optimized CNN models for the ID leakage model, ASCAD_fixed dataset.

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 9: Hyperparameters optimized CNN models for the HW leakage model, ASCAD_fixed dataset.

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

Table 10: Hyperparameters optimized MLP models for the ID leakage model, ASCAD_variable dataset.

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 11: Hyperparameters optimized MLP models for the HW leakage model, ASCAD_variable dataset.

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 12: Hyperparameters optimized CNN models for the ID leakage model, ASCAD_variable dataset.

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 13: Hyperparameters optimized CNN models for the HW leakage model, ASCAD_variable dataset.

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

Table 14: Hyperparameters median MLP models for the ID and HW leakage models, ASCAD_fixed dataset.

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 15: Hyperparameters median CNN models for the ID and HW leakage models, ASCAD_fixed dataset.

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

Table 16: Hyperparameters median MLP models for the ID and HW leakage models, ASCAD_variable dataset.

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 17: Hyperparameters median CNN models for the ID and HW leakage models, ASCAD_variable dataset.

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