Journal of Cryptographic Engineering (2023) 13:311–324
https://doi.org/10.1007/s13389-023-00320-6

REGULAR PAPER

# 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[2] · Stjepan Picek[2]

Received: 9 October 2021 / Accepted: 23 April 2023 / Published online: 28 May 2023
© The Author(s) 2023

**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 to reach strong attack performance, which can be far from trivial. Besides many options stemming from the machine learning domain, recent years also brought neural network elements specially designed for side-channel analysis. The loss function, which calculates the error or loss between the actual and desired output, is one of the most important neural network elements. The resulting loss values guide the weights update 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 regarding their effectiveness in side-channel attacks. This work provides a detailed study of the efficiency of different loss functions in the SCA context. We evaluate five loss functions commonly used in machine learning and three loss functions specifically designed for SCA. Our results show that an SCA-specific loss function (called CER) performs very well and outperforms other loss functions in most evaluated settings. Still, categorical cross-entropy represents a good option, especially considering the variety of 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 model based on the leakage of this device, the key recovery of the attack device requires only a few measurements. Today some of the most powerful representatives of profiling attacks come from the deep learning domain [4, 19, 31]. Literature indicates that such attacks can break targets equipped with countermeasures but require a careful hyperparameter tuning to counter such protection mechanisms [12]. Unfortunately, due to the complexity of the deep learning architectures, finding the best hyperparameter combination is a challenging task.

Loss functions, one of the tunable hyperparameters, play 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 value 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 [10, 33], which is also recognized in the SCA domain.

### 1.1 Related works

In recent years, deep learning has become more popular in the context of SCA thanks to its flexibility and strong attack performance in different attack settings [4, 12, 16, 17, 23, 28,

✉ Stjepan Picek
stjepan.picek@ru.nl

Maikel Kerkhof
maikelkerkhof@gmail.com

Lichao Wu
l.wu-4@tudelft.nl

Guilherme Perin
guilherme.perin@ru.nl

1 Cyber Security Research Group, Delft University of Technology, Mekelweg 2, Delft, The Netherlands

2 Digital Security Group, Radboud University, Postbus 9010, Nijmegen, The Netherlands

36]. Many of these works focus on optimizing the network architectures to increase the model's attack capabilities. For instance, Zaid et al. [36] proposed a methodology to find good-performing architectures for SCA. Kim et al. [12] also researched different architectural choices and the influence of noise. More recently, different frameworks have been proposed to automate the network tuning [24, 32]. However, all of these works have in common that no considerations about the used loss function are made. When they first explored using deep learning techniques for SCA [16], the authors mentioned that categorical cross-entropy or the mean squared error is commonly used loss function. Later work on deep learning for SCA seems to exclusively use either categorical cross-entropy [3, 19, 36] or mean squared error [18, 28].

More recently, three novel loss functions specifically for usage in the context of SCA have been proposed: ranking loss (RKL) [35], cross-entropy ratio (CER) [37], and focal loss ratio [11]. The detailed discussion can be found in Sect. 2.3. These papers compare the newly proposed loss functions to the categorical cross-entropy. Unfortunately, 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 error, or hinge loss and these novel SCA-based loss functions on different architectures, leakage models, and datasets.

### 1.2 Motivation and contributions

To verify the generality of various proposed loss functions, there is a strong demand for a systematic evaluation of different loss functions in diverse attack settings. In this work, we aim to fill in that gap. More precisely, we systematically compare commonly used loss functions and novel SCA-specific loss functions on four publicly available datasets and with two commonly used leakage models. We evaluate the performance from various perspectives: attack performance (guessing entropy), neural network types and sizes, and the required training time. Our results show the outstanding performance of recently proposed FLR and CER loss functions, especially when using the Hamming Weight leakage model [11, 37], which can represent a challenging scenario due to class imbalance and the lack of reliability of machine learning metrics [22]. The performance of ranking loss, another recently proposed loss function, is strongly connected with specific neural network architectures: it may perform the best in some specific settings, but in most cases, FLR and CER are better options. Finally, a common choice in deep learning-based SCA, categorical cross-entropy, is confirmed as a competitive option due to its low calculation overhead and the generality with different neural network architectures.

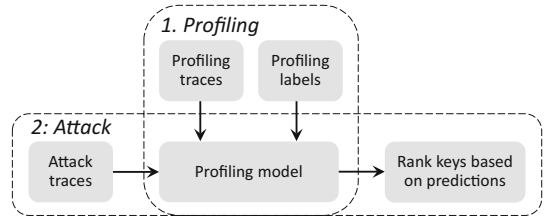



**Fig. 1** Profiling side-channel attack

## 2 Background

### 2.1 Profiling side-channel analysis

For profiling side-channel analysis, an attacker is assumed to have a clone device identical (or at least similar) to the device to be attacked. The attacker uses $O$ leakages measurements from the profiling device (with known label information, i.e., secret variables to be attacked) to build a profiling model. Then, the attacker uses $Q$ measurements from the device to be attacked to infer the secret information. A demonstration of the profiling attack is depicted in Fig. 1. Depending on the profiling technique, one builds different types of profiling models: a template for the template attack and machine learning models. This paper focuses on machine learning models.

### 2.2 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. Supervised learning happens in two phases: training and testing. 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 $k_i$. We divide the dataset into disjoint subsets where the training set has $M$ traces, the validation set has $V$ traces (thus, $O = M + V$), and the attack set has $Q$ traces.

1. Profiling phase: the goal 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. Attack phase: 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 cumulative sum $S(k)$ 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 of the class $v$ derived from the key $k$ and input $d_i$ through a cryptographic function $CF$ and a leakage model $l$.

In SCA, an adversary aims to reveal the secret key $k^*$; standard performance evaluation metrics are the success rate (SR) and the guessing entropy (GE) [27]. This work uses the guessing entropy metric to estimate the attack performance. More specifically, given $Q$ 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: $g_1$ is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate.

## 2.3 Loss functions

In this section, we discuss loss functions that will be used in our experiments. We use $\hat{\mathbf{y}}$ to denote the predicted vector and $\mathbf{y}$ to denote the ground truth vector. Finally, we use $y_i$ to denote the $i$th true value and $\hat{y}_i$ to denote the corresponding predicted value.

### 2.3.1 Mean squared error

One of the simplest loss functions is the mean squared error (MSE) [26]. The MSE is calculated by taking the mean of the pairwise squared differences between the elements of the predicted vector $\hat{\mathbf{y}}$ and the vector $\mathbf{y}$ with the true values in one-hot format:

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

where $M$ denotes the number of training samples. MSE and its variations have been used to solve regression problems (thus, the function's output $f$ is continuous) [26]. The loss is calculated evenly for each sample, regardless of which class a sample belongs to. By minimizing the loss, we reduce the distance between the predicted and true labels. MSE is also usable for classification problems [10] and has been used in the context of SCA as discussed before [29].

One variation of the MSE is the mean squared logarithmic error (MSLE) [1]. 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. Compared with its counterpart, MSLE is less sensitive to outliers in the data.

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

Finally, we consider the logarithm of the hyperbolic cosine (log cosh) as a loss function. Log cosh loss, like MSLE, is also robust to outliers [30] when compared with MSE:

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

### 2.3.2 Classification losses

The de facto standard loss function for classification tasks 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 [8, 13, 34] and is also commonly used in SCA [3, 12, 16, 17]. Cross-entropy is a measure of the difference between two distributions. 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:

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

where $c$ denotes the number of classes.

Another loss function used for classification is the (categorical) hinge loss [5]. The hinge loss increases the margin between the predicted probability for correct classes and wrong classes with the highest predicted probability:

$$cat_hinge(\mathbf{y}, \hat{\mathbf{y}}) = max(1 - y_i \hat{y}_i, 0). \tag{6}$$

### 2.3.3 Custom SCA losses

More recently, three SCA-specific loss functions have been proposed. One is the ranking loss (RKL) function proposed by Zaid et al. [35]. 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:

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

where $\mathbf{s}$ is the vector with scores for each key hypothesis generated by processing the training samples by the model, $\mathcal{K}$ 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 $\mathcal{K}$. Finally, $\alpha$ is a parameter that needs to be set dependent on different attack settings. The implementation of the ranking loss function is provided by [35] on GitHub.[1] Note that $\alpha$ is a critical parameter for RKL. We optimize this hyperparameter based on a random search.

The other custom loss function is the cross-entropy ratio (CER) loss proposed by Zhang et al. [37]. The authors introduced the CER as an SCA metric to estimate the performance of a deep learning model in the context of SCA. Meanwhile, this metric could be used as a loss function directly by using a shuffled set of labels:

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

where $cce$ is the categorical cross-entropy. $\mathbf{y}_{r_i}$ denotes the vector with the true probabilities, 1 for the correct class and 0 for all others, for each class but randomly shuffled (shuffled labels). The variable $N$ denotes the number of shuffled sets to use. The authors do not provide a value for $N$, but state that increasing $N$ should increase the accuracy of the metric. In our first experiments, to balance computational complexity and the potential increase in accuracy, we use $N = 10$.

Recently, the focal loss ratio (FLR) loss function [11] was proposed, and it combines the benefits from focal loss function [14], categorical cross-entropy, and CER. FLR is computed as:

$$FLR(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\alpha(1-\hat{\mathbf{y}})^{\gamma} cce(\mathbf{y}, \hat{\mathbf{y}})}{\frac{1}{N}\sum_{i=1}^{N} \alpha(1-\hat{\mathbf{y}})^{\gamma} cce(\mathbf{y}_{\mathbf{s_i}}, \hat{\mathbf{y}})}, \tag{9}$$

Aligned with CER, $\mathbf{y}$ and $\mathbf{y_s}$ denote the true and shuffled labels, respectively; $cce$ is the categorical cross-entropy, and $N$ is the number of negative samples to use. $\alpha$ and $\gamma$ are introduced to weigh the classes and emphasize hard samples for both the numerator and denominator, respectively. In this paper, we set $\alpha$, $\gamma$, and $N$ to 0.25, 2, and 3, respectively, following the original paper on selecting these three hyperparameters [11].

## 2.4 Datasets

### 2.4.1 ASCAD fixed key (`ASCADf`)

The ASCAD dataset is generated by taking measurements from an ATMega8515 running masked AES-128 and is proposed as a benchmark dataset for SCA [3]. The dataset consists of 50,000 profiling traces and 10,000 attack traces, each trace consisting of 700 features. In this paper, we set the number of profiling traces to 50,000. In the attacking phase, we use up to 2000 traces. The profiling and attacking sets both use the same fixed key. We denote this dataset as `ASCADf`. We attack the third key byte as that is the first masked byte. Both Hamming weight (HW) and Identity (ID) leakage models are considered in this paper. The dataset is provided on the ASCAD GitHub repository.[2]

### 2.4.2 ASCAD random key (`ASCADr`)

The `ASCADr` dataset consists of 200,000 profiling and 100,000 attack traces, each consisting of 1400 features. Unlike the `ASCADf` dataset, the keys used in the profiling set are variable. In this paper, we set the number of profiling traces to 50,000. In the attacking phase, 3000 traces are used. We attack the third key byte as that is the first masked byte. We will consider attacks in the Hamming weight (HW) and Identity (ID) leakage models. The `ASCADf` dataset is available in the ASCAD GitHub repository.[3]

### 2.4.3 CHES CTF 2018 (`CHES_CTF`)

This dataset was released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES).[4] The target implementation is masked AES-128 encryption executing on a 32-bit STM microcontroller. In our experiments, we use 45 000 traces for the training set, which contain a *fixed key*. The validation and test sets consist of 5000 traces each, where we used 3000 traces for the attack phase. We considered a pre-processed dataset version where each trace consists of 2 200 features. Unlike the ASCAD dataset, the key used in the training and validation set differs from the key for the test set. We attack the first key byte and we consider the Hamming weight and Identity leakage models.

### 2.4.4 DPAv4.2

The `DPAv4.2` dataset contains side-channel measurements obtained from a masked AES-128 software implementation.[5] The countermeasure is based on RSM (*Rotation* `S-box` *Masking*). The original `DPAv4.2` contains 80,000 traces subdivided into 16 groups of 5000 traces. Each group is defined with a separate but fixed key. Each measurement has 1,704,046 samples. In this work, we conduct our side-channel

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

[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://chesctf.riscure.com/2018/news.

[5] http://www.dpacontest.org/v4/42_doc.php.

analyses on the interval of second-order leakages from key byte 12, which has the highest SNR from its secret shares. The attacked interval starts at sample 305 000 and finishes at sample 315,000. Finally, we apply a resampling process with a resampling window of 10 and step of 5 to the concatenated intervals, resulting in 2000 samples per measurement. We attack the dataset in both the Hamming weight and Identity leakage models.

### 2.5 Leakage model

Two leakage models are considered in this paper:

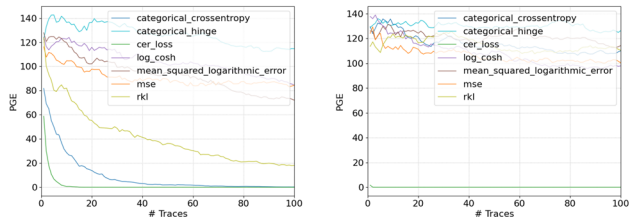
1. Hamming weight (HW): the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. When considering the AES cipher (8-bit S-box[6]), this leakage model results in nine classes.
2. Identity (ID): the attacker considers the leakage in the form of an intermediate value of the cipher. When considering the AES cipher (8-bit S-box), this leakage model results in 256 classes.

## 3 Experimental setup

Several different hyperparameters, such as the number of layers and neurons per layer, the activation function each neuron uses, and the loss function, influence the training process of a deep learning model. 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 Fig. 2. 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 (Fig. 2a), CER loss performs the best, followed by CCE and RKL. However, when the learning rate is increased to 0.001 (Fig. 2b), CCE and RKL, the loss functions that lead to a converged GE, are not functional anymore. At the same time, for the CER loss, the performance is even increased. Thus, benchmarking with a single attack model and the fixed attack setting cannot represent the generality of a loss function. Knowing this, we consider the following scenarios for a fair loss function comparison:

*Hyperparameter optimization* We perform hyperparameter optimization via a random search for each considered loss function for best-performing models. More specifically, we perform the following steps to select and evaluate the best model:

[6] Or any cipher with 8-bit S-box.




(a) Learning rate 0.00001 (b) Learning rate 0.001

**Fig. 2** 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. The example is given for the `ASCADf` dataset in the ID leakage model

1. Generate, train, and test 100 random models for each loss function.
2. Select the best-performing model in terms of guessing entropy.
3. Train and test the selected 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 and training time.

We decided to showcase the results for the median model since (1) it avoids outlier behavior, whether being a well-performing model or a model that cannot converge at all and (2) there is no guarantee that any of the tested models would behave exactly as the averaged model.

*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 [3, 16, 36]. For each dataset, leakage model, and loss function combination, we deploy a hyperparameter search with two searching ranges, resulting in different network sizes. Following this, we can investigate the training efficiency of loss functions with different learning capabilities of models. Table 1 provides the hyperparameter ranges for small MLP models. For larger MLP models, we provide the hyperparameter ranges in Table 2 following [20] to balance between good performance and a broad range of possible values per hyperparameter. We train each model for 200 epochs. Knowing that the number of epochs highly impacts the performance of a model, we introduce an early-stopping mechanism to terminate the training process when the model starts overfitting. Guessing entropy is used to monitor the model's performance during training, following the approach presented in [25]. In terms of optimizers, since the Adam and RMSProp optimizers perform well [3, 20], both

**Table 1** Hyperparameter space for multilayer perceptrons–small models

| Hyperparameter | Options |
|---|---|
| Dense layers | 2, 3, 4, 5, 6 |
| Neurons per layer | 10, 20, 30, 40, 50 |
| Learning rate | 0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001 |
| Batch size | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Activation function | ReLu, SELU, ELU, Tanh |
| Optimizer | Adam, RMSProp |
| Total search space | 18,000 |

**Table 2** Hyperparameter space for multilayer perceptrons–large models

| Hyperparameter | Options |
|---|---|
| Dense layers | 2, 3, 4, 5, 6, 7, 8 |
| Neurons per layer | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Learning rate | 0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001 |
| Batch size | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Activation function | ReLu, SELU, ELU, Tanh |
| Optimizer | Adam, RMSProp |
| Total search space | 50,400 |

optimizers are considered as an option and, naturally, the range of learning rates is broadened.

For the CNN hyperparameters, as shown in Tables 3 and 4, we again define two search ranges. Additionally, a batch normalization layer, as introduced by [9], is applied after the input layer and after each convolutional block, as done in earlier work to improve the performance of CNNs [3, 4, 20]. Note that the size of the architectures is not directly correlated with the search space size. Even for a small architecture, there can be many possibilities to build it.

*Attack scenarios* To perform a broad comparison between the different loss functions, we define 32 different attack scenarios to make the comparison for each dataset. Each of these scenarios combines a dataset, a leakage model, an architecture type, and a network search range.

*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 Sect. 2.3. It is worth noting that when performing the random search with RKL, a fixed $\alpha$ (e.g., the value used in the original paper) could restrict to specific attack settings. Therefore, we included $\alpha$ as a hyperparameter to be searched and optimized during the network searching. The possible values are 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

**Table 3** Hyperparameter space for convolutional neural networks–small models

| Hyperparameter | Options |
|---|---|
| Dense layers | 1, 2 |
| Neurons per layer | 10, 20, 30, 40, 50 |
| Convolutional layers | 1, 2, 3, 4 |
| Convolutional filters | 4, 8, 12 |
| Kernel size | 10, 20, 30, 40 |
| Strides | 5, 10, 15, 20 |
| Pooling size | 2, 3, 4, 5 |
| Pooling stride | 2, 3, 4, 5 |
| Pooling type | Max pooling, average pooling |
| Learning rate | 0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001 |
| Batch size | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Activation function | ReLu, SELU, ELU, Tanh |
| Optimizer | Adam, RMSProp |
| Total search space | 44,236,800 |

**Table 4** Hyperparameter space for convolutional neural networks–large models

| Hyperparameter | Options |
|---|---|
| Dense layers | 2, 3 |
| Neurons per layer | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Convolutional layers | 1, 2 |
| Convolutional filters | 4, 8, 12, 16, 32 |
| Kernel size | 10, 12, 14, 16, 18, 20 |
| Strides | 5, 10, 15, 20 |
| Pooling size | 2, 3, 4, 5 |
| Pooling stride | 2, 3, 4, 5 |
| Pooling type | Max pooling, Average pooling |
| Learning rate | 0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001 |
| Batch size | 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| Activation function | ReLu, SELU, ELU, Tanh |
| Optimizer | Adam, RMSProp |
| Total search space | 55,296,000 |

*Pre-processing* The pre-selected window of features is used for both datasets, and no further selection of points of interest is made. Interestingly, we notice that directly applying the raw features or following earlier work that scales the SCA

features to values between 0 and 1 [15, 35] could lead to the loss value equal to a $NaN$ during training, indicating the possibility of triggering the exploding gradients problem: the gradients during the backpropagation are getting too large or small. In the worst case, it may cause the learning process to fail [21]. To verify this, Fig. 3 shows the largest and smallest gradient of the input layer for each epoch during the training process of one of these models. The cause of the exploding gradient problem could come from combining several hyperparameters, e.g., the activation function, the loss function, and the 0–1 normalization used during pre-processing. For instance, normalizing all feature values to values between 0 and 1 removes any negative values from the profiling traces. The output of the activation function, such as ELU [shown in Eq. (10)], 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 causes the gradients to get too large, leading to a poorly performing model or even failed training.

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0. \end{cases} \tag{10}$$

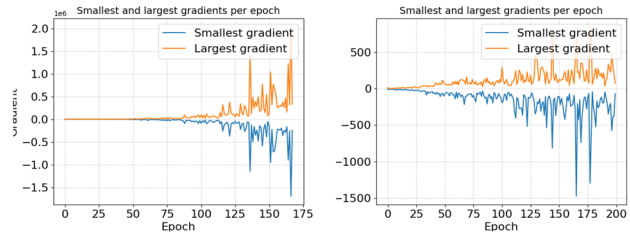
There are two methods to solve this problem: (1) normalizing the leakage traces by Z-score normalization (standardization); (2) clipping the gradients when they get too large or too small. We followed the first method as the pre-processing method for all experiments. Although the attack performance could be similar to when 0–1 normalization is used, the possibility of triggering the exploding gradient problem can be dramatically reduced.

## 4 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 models optimized via random search. 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 3584 GPU cores.

### 4.1 ASCADf

We first consider the performance of the different loss functions on the ASCADf dataset. Figure 4 shows the guessing entropy over 100 attacks for each of the scenario's best-performing models generated with small and large search spaces, respectively. The required number of attack traces



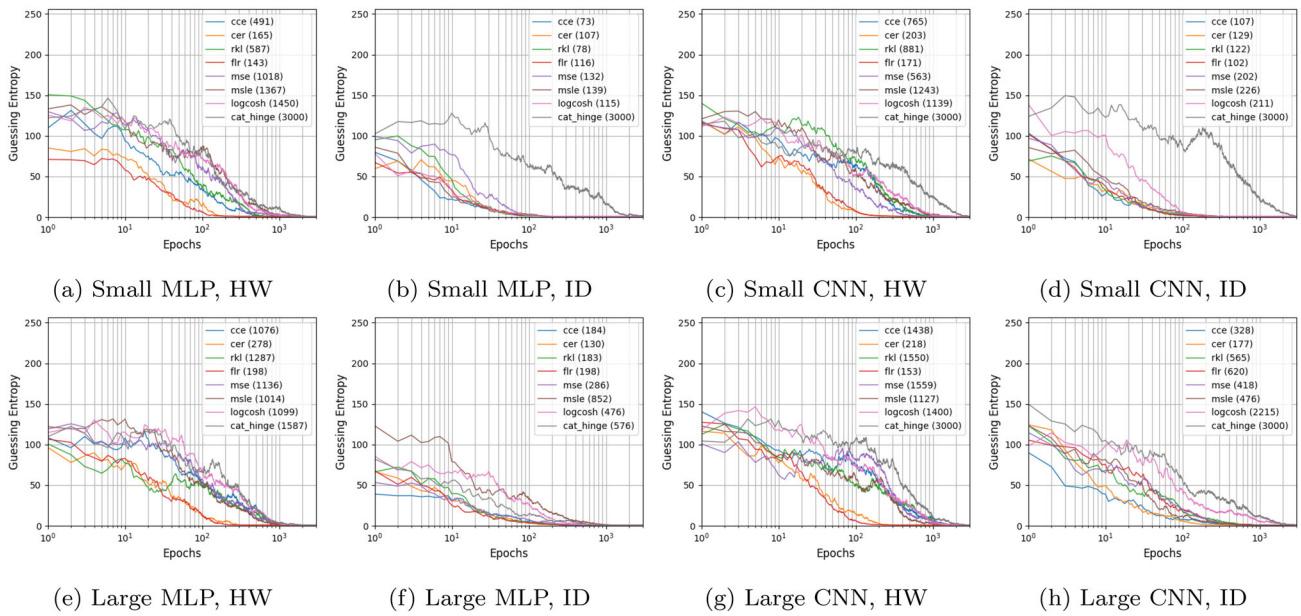

(a) 0-1 normalization (b) Z-score normalization

**Fig. 3** 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 when Z-score normalization is applied

to reach guessing entropy zero is listed in the legend of the figure.

First, we notice that the ASCADf dataset is vulnerable to SCA with various attack settings. Most of the loss functions lead to a model that can retrieve the correct key in less than 1000 traces with best-performing MLP and CNN architectures. Besides, even a simple hyperparameter optimization approach via random search results in an improved attack performance. For instance, CER models reach a GE of 1 in less than 300 traces, which is comparable to the state-of-the-art attack performances [31, 35]. Besides, one can observe that using a small or big network architecture may influence the attack efficiency but does not significantly change the performance rank of the different loss functions.

From the results, although not the best, the commonly used categorical cross-entropy does perform very consistently in these scenarios. On the other hand, CER and FLR are almost always the top-two loss functions thanks to the introduction of negative samples in the loss functions. In terms of CER, it is worth noting that Zhang et al. [37] only test this loss function on the HW leakage model, i.e., imbalanced data. Our results show that the CER loss is also very suitable for the ID leakage model. For FLR, the emphasis on the hard samples makes it even outperforms CER in most evaluated cases. Therefore, we can conclude that when the ASCADf dataset is considered, the best choice of loss functions is the FLR loss. Surprisingly, ranking loss (RKL) performs less consistently in the tested scenarios. Zaid et al. [35] compared the RKL function to categorical cross-entropy and CER loss, stating that the RKL outperforms both those functions when the hyperparameter $\alpha$ is properly tuned for each class. However, they only compared a single CNN architecture and considered the ID leakage model [35]. Our results confirm that RKL works better with the ID leakage model than the HW leakage model with the ASCADf dataset. However, except in two test scenarios where RKL performs the best (Fig. 4d), it always performs worse than CER and FLR loss functions.




(a) Small MLP, HW (b) Small MLP, ID (c) Small CNN, HW (d) Small CNN, ID

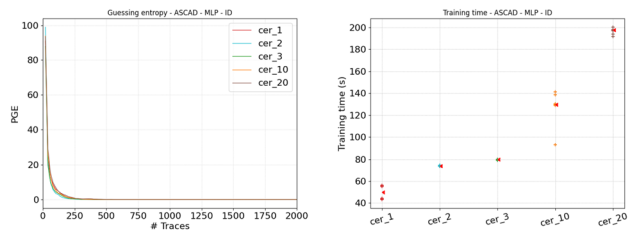
(e) Large MLP, HW (f) Large MLP, ID (g) Large CNN, HW (h) Large CNN, ID

**Fig. 4** GE of the best MLP and CNN models on the ASCADf dataset

Another remark that has to be made when discussing these results is the required training time. When all other hyperparameters are the same, FLR, RKL, and CER loss functions are significantly slower (training speed time) when 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 for the FLR and CER loss is also due to how those loss functions are constructed.

For instance, CER 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. However, as shown in Fig. 5, different values of $N$ give approximately the same $\overline{N}_{T_{GE}}$, where $\overline{N}_{T_{GE}}$ represents the number of attack traces required to reach guessing entropy of zero. Moreover, 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 an increase in performance in GE. A similar argument holds for FLR as it uses a small value of $N$, which has a negligible influence on the training time while providing excellent performance [11].

The worst-performing loss function is the categorical hinge loss. 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. Furthermore, if we look at the definition of the categorical hinge loss as described in Sect. 2.3, 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 many wrong classes (255) and only one correct class with the ID leakage model. Due to the random initialization of the weights, the loss coming from the wrongly classified traces will stay at 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.




(a) Guessing entropy (b) Training time

**Fig. 5** Guessing entropy and training time for the optimized model with CER loss using different values of $N$

Overall, when considering the ASCADf dataset, if an adversary uses a random search for hyperparameter tuning, considering the FLR and CER loss functions increase the model's capability in retrieving the secret key thanks to their resilience to various attack settings. Indeed, they significantly outperform models with categorical cross-entropy, ranking loss, and other loss functions.

### 4.2 ASCADr

Next, we look at the results on the ASCADr dataset. Figure 6 shows the GE performance on the best-performing models with different search ranges.

For the experiments performed on the ASCADr dataset, we see results comparable to those on the ASCADf dataset. In most scenarios, the models trained with FLR and CER loss perform the best, followed closely by those trained with categorical cross-entropy. RKL performs worse than CCE in most cases. Finally, categorical hinge loss performs the worst in the majority of cases. Surprisingly, CCE performs the best when working with an ID leakage model (e.g., Fig. 6b, d), indicating its excellent performance when working with smaller network size output nodes. With the ID leakage model or larger network sizes, more complicated loss functions, such as FLR and CER, outperform CCE in most cases.

### 4.3 CHES_CTF

The results of the CHES_CTF dataset are shown in Fig. 7. Similar to ASCADf and ASCADf datasets, the difference in the network size could vary the required number of attack traces to reach guessing entropy zero. However, it has a limited impact on the benchmark of different loss functions. Compared with previously evaluated datasets, the CHES_CTF dataset requires more attack traces to break the target. Specifically, the HW leakage models lead to more powerful attacks than its counterpart, which aligns with the results from literature [24, 32]. When looking at each loss function, although all of the loss functions lead to GE convergence, FLR and CER remain the top-two candidates in breaking the target when using the HW leakage model. When moving to the ID leakage model, the results are more difficult to interpret due to their random performance. Still, CER and FLR remain the top candidates that lead to the fastest GE convergence. CCE and RKL loss functions are in the middle regarding GE performance. Finally, categorical hinge loss performs the worst among all considered loss functions.

### 4.4 DPAV4.2

In general, DPAV4.2 is the simplest dataset to break compared to the other dataset evaluated before. The results with a large search range are shown in Fig. 8. Many loss functions can break the target within 100 attack traces. Nevertheless, one can observe the performance variation between different loss functions. For instance, while most of the loss function breaks the target with less than 40 traces when using MLP and HW leakage model (Fig. 8b, e), categorical hinge loss requires ten times more attack traces, which is consistent with the observation from the other datasets. On the other hand, the minor performance variation of CCE, CER, RKL, and FLR is due to the variation in the neural network architecture. Still, when using CNN with the HW leakage model, FLR loss outperforms other loss functions.

## 5 Discussion

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 and allow us to answer the following questions:

*How does the choice of loss function impact the SCA performance in terms of guessing entropy for different datasets and leakage models?* We generally see that the FLR and CER loss perform the best in most of the experiments. Specifically, while Zhang et al. [37] already demonstrated that CER loss might work well 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 slower to train than models with other functions, it performed best in isolated scenarios. In all the other scenarios, CER and FLR remain the best choices.

Furthermore, our work also shows that categorical cross-entropy, often used by default in related works, is a solid choice. It performs well with almost any type of combination of hyperparameters within the hyperparameter search space we defined. Regarding guessing entropy, models with categorical cross-entropy are often third in the performance (after CER and FLR loss). Besides that, our results show that it is faster to train and needs less complex models. To conclude, it showed no obvious weaknesses.

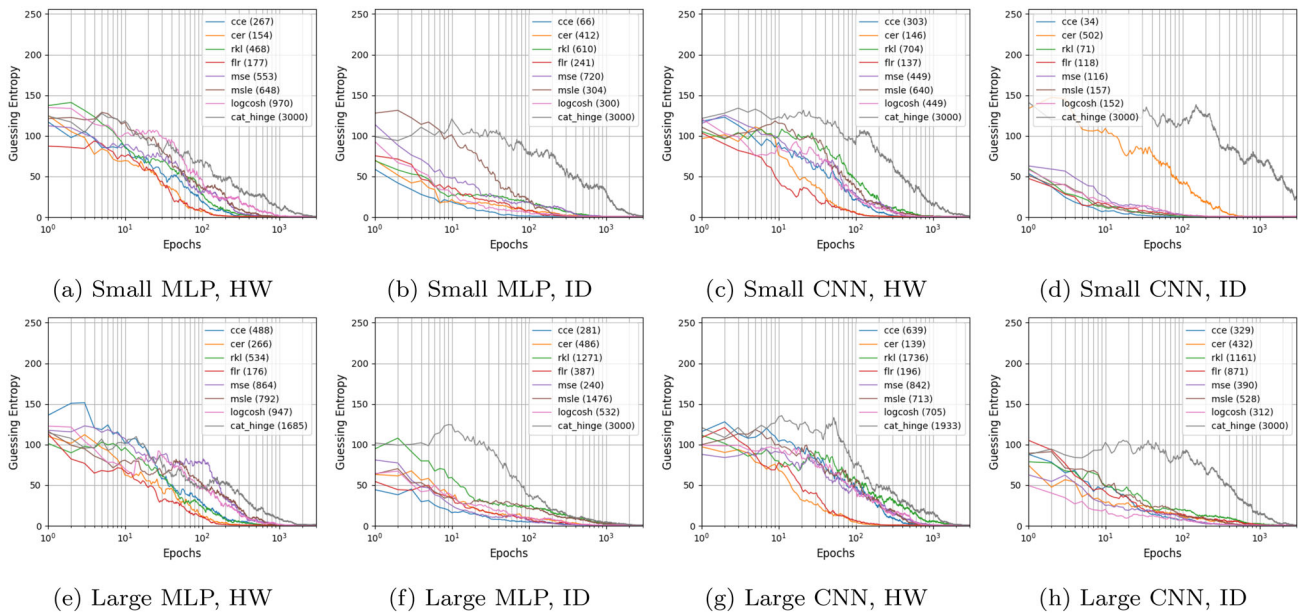
The other loss functions we considered did, in general, not show promising results. While used before in related works, MSE and related loss functions such as MSLE and log cosh are almost always outperformed by the categorical cross-entropy, CER, and FLR loss when an attack can be performed successfully.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

*What is the influence of loss functions when different architectures like multilayer perceptrons and convolutional neural networks are considered?* 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 and any network size.

*How does the choice of loss function impact the training time?* In terms of the training time, we do not see large differences between the different loss functions despite the more complex models. 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

**Table 5** Strengths and weaknesses for each of the loss functions, based on the results of our experiments

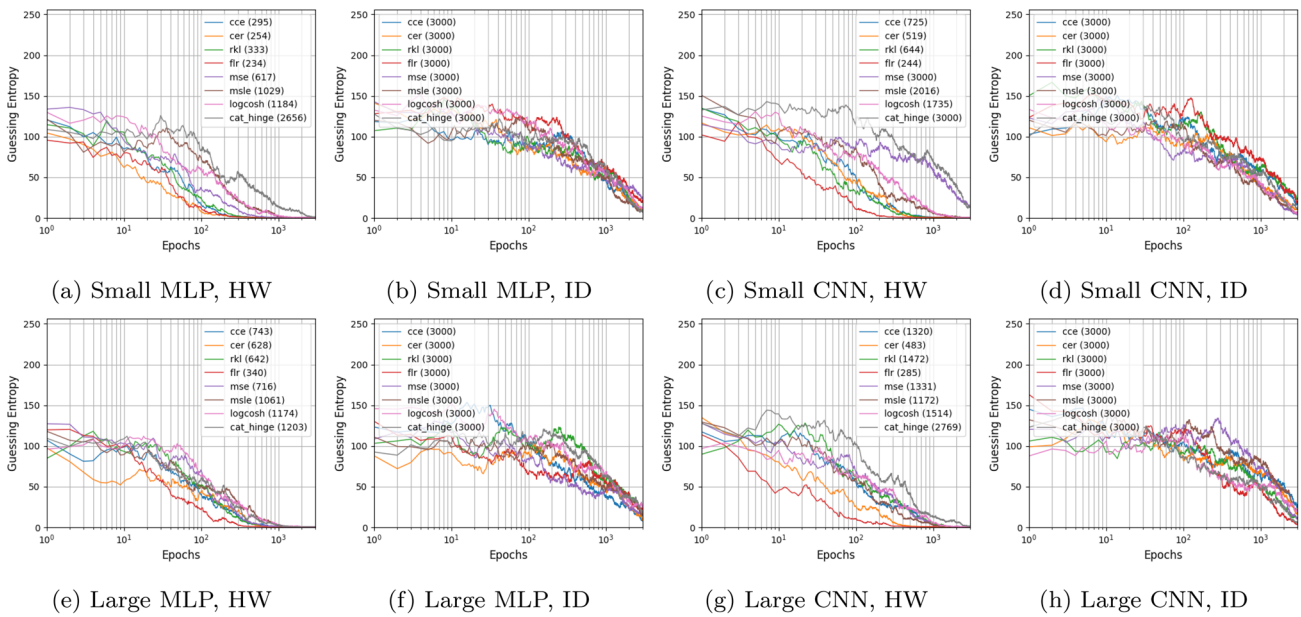
| Loss function | Strengths | Weaknesses |
|---|---|---|
| CCE | –Robust to different architectures | –Not the best performance compared with SCA-optimized loss functions |
| | –Works well with small architectures and output nodes (HW) | |
| | –Low computation overhead | |
| Categorical hinge | –Low computation overhead | –Worst attack performance |
| Log Cosh | –Low computation overhead | –Mediocre performance |
| MSE | –Low computation overhead | –Mediocre performance |
| MSLE | –Low computation overhead | –Mediocre performance |
| Cross-entropy ratio (CER) | –Second best performance | –New tunable parameter $N$, which may slow down the training speed when set to a large number |
| | –Consistent with SCA performance metrics | |
| Focal loss ratio (FLR) | –Best performance | –New tunable parameter $\alpha$, $\gamma$, but even fixed values can conduct good results |
| | –Consistent with SCA performance metrics | –New tunable parameter $N$, which may slow down the training speed when set to a large number |
| Ranking loss (RKL) | –Perform well in some specific scenarios | –Slow training due to the pairwise comparison between the rank of the correct key and other keys |
| | –Consistent with SCA performance metrics | –Not robust to various test cases |
| | | –New tunable parameter $\alpha$, needs to be carefully tuned for different attack settings |




(a) Small MLP, HW (b) Small MLP, ID (c) Small CNN, HW (d) Small CNN, ID

(e) Large MLP, HW (f) Large MLP, ID (g) Large CNN, HW (h) Large CNN, ID

**Fig. 6** GE of the best MLP and CNN models on the ASCADr dataset




(a) Small MLP, HW (b) Small MLP, ID (c) Small CNN, HW (d) Small CNN, ID

(e) Large MLP, HW (f) Large MLP, ID (g) Large CNN, HW (h) Large CNN, ID

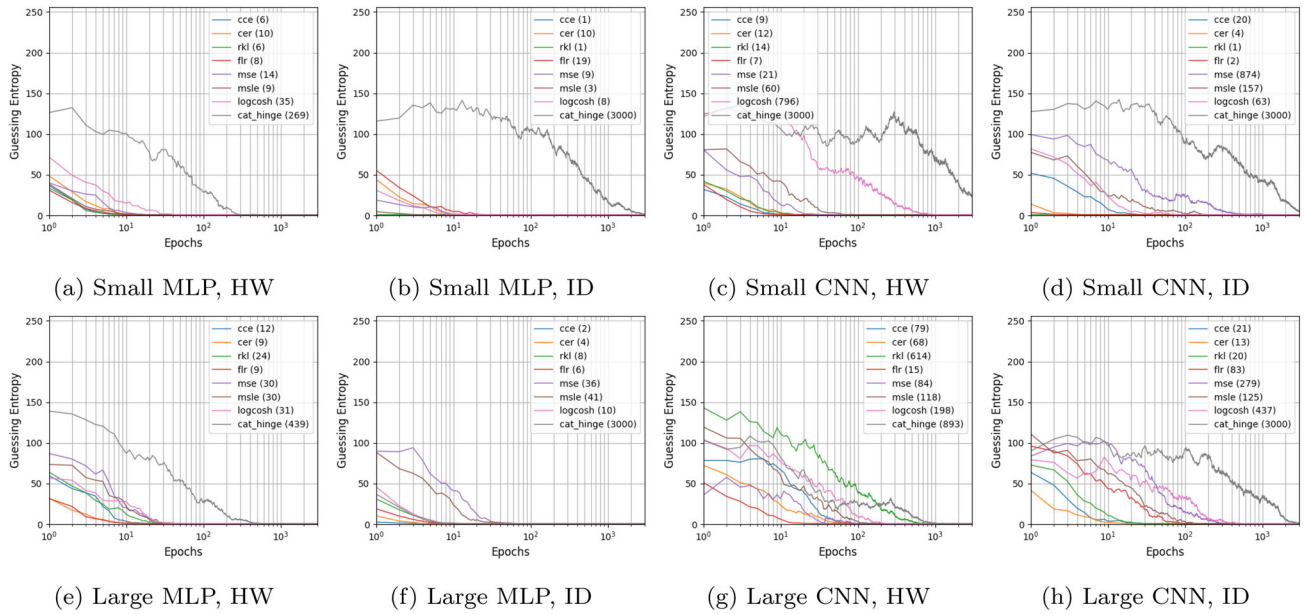
**Fig. 7** GE of the best MLP and CNN models on the CHES_CTF dataset

classes is high, the training time is increased by up to a factor of ten. The CER and FLR loss functions are 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.

## 6 Conclusions and future work

This work investigates several loss functions commonly used in the machine learning domain and compares them with three recently proposed SCA-specific loss functions. We analyze four datasets and two leakage models, considering guessing entropy, neural network size, and training size. Our results show that the FLR and CER loss are, 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,




(a) Small MLP, HW (b) Small MLP, ID (c) Small CNN, HW (d) Small CNN, ID

(e) Large MLP, HW (f) Large MLP, ID (g) Large CNN, HW (h) Large CNN, ID

**Fig. 8** GE of the best MLP and CNN models on the DPAv4.2 dataset

while ranking loss, or other loss functions, should be used more carefully, as they only work the best in specific attack settings. We emphasize that the goal of this paper is not to select a single best loss function to be used in every scenario. Rather, we aimed to:

1. Systematically evaluate several loss functions and compare their performance in a fair setting. Indeed, related works do not provide a detailed comparison of multiple loss functions, making it necessary to compare results over different research works and settings.
2. Provide a choice of a few loss functions that perform well in different settings to be included in future works as part of the hyperparameter tuning process. In this direction, we indeed recognize two loss functions (FLR and CER) that we recommend being used.

Since our experiments confirm the outstanding performance of some custom SCA loss functions, this opens interesting future research directions for better SCA-optimized loss functions than are generalized for different attack scenarios. We notice that in other domains in which deep learning is applied, several works have also introduced new loss functions that improve the performance in that context [2, 6, 7, 14]. These functions are created to deal with certain characteristics of the targeted datasets, such as a class imbalance or a low number of measurements per class. It remains an open question of how more complex loss functions would perform in the SCA context. We consider especially interesting whether it would be possible to design a loss function that can deal with the desynchronization countermeasure. While this seems far from trivial, possibly borrowing certain concepts from the shift invariance and shift equivariance of convolutional neural networks would be interesting. 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. Finally, we consider the setting where the number of features is not too high (i.e., preprocessed interval). It would be interesting to re-evaluate the performance of various loss functions when using raw features.

**Acknowledgements** The datasets analyzed during the current study are freely available under the links given in the paper. The last author (Stjepan Picek) is on the editorial board of the journal to which this work is submitted. This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References


1. Acharya, M.S., Armaan, A., Antony, A.S.: A comparison of regression models for prediction of graduate admissions. In: 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), pp. 1–5. IEEE (2019)
2. Barz, B., Denzler, J.: Deep learning on small datasets without pre-training using cosine loss. Tech. rep. (2020)
3. 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. J. Cryptogr. Eng. **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8
4. 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
5. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. **2**, 265–292 (2001)
6. Hajiabadi, H., Babaiyan, V., Zabihzadeh, D., Hajiabadi, M.: Combination of loss functions for robust breast cancer prediction. Comput. Electr. Eng. (2020). https://doi.org/10.1016/j.compeleceng.2020.106624
7. Hajiabadi, H., Molla-Aliod, D., Monsefi, R., Yazdi, H.S.: Combination of loss functions for deep text classification. Int. J. Mach. Learn. Cybern. **11**(4), 751–761 (2020). https://doi.org/10.1007/s13042-019-00982-x
8. Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, Deep Residual Learning for Image Recognition, CoRR, abs/1512.03385, (2015). http://arxiv.org/abs/1512.03385, arXiv:5120.3385. https://dblp.org/rec/journals/corr/HeZRS15
9. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift (2015). http://arxiv.org/abs/1502.03167
10. Janocha, K., Czarnecki, W.M.: On loss functions for deep neural networks in classification. Tech. rep. (2017). https://arxiv.org/abs/1702.05659
11. Kerkhof, M., Wu, L., Perin, G., Picek, S.: Focus is key to success: a focal loss function for deep learning-based side-channel analysis. In: Balasch, J., O'Flynn , C. (eds.) Constructive Side-Channel Analysis and Secure Design—13th International Workshop, COSADE 2022, Leuven, Belgium, April 11–12, 2022, Proceedings, Lecture Notes in Computer Science, vol. 13211, pp. 29–48. Springer (2022). https://doi.org/10.1007/978-3-030-99766-3_2
12. 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 Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 148–179 (2019). https://doi.org/10.13154/tches.v2019.i3.148-179. (ISSN 2569-2925)
13. Kussul, N., Lavreniuk, M., Skakun, S., Shelestov, A.: Deep learning classification of land cover and crop types using remote sensing data. IEEE Geosci. Remote Sens. Lett. (2017). https://doi.org/10.1109/LGRS.2017.2681128
14. Lin, T.Y., Goyal, P., Girshick, R.B., He, K., Dollár, P.: Focal loss for dense object detection (2017). arXiv:1708.02002
15. Maghrebi, H.: Deep learning based side channel attacks in practice. IACR Cryptol. ePrint Arch. **2019**, 578 (2019)
16. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 3–26. Springer (2016)
17. Masure, L., Dumas, C., Prouff, E.: A Comprehensive Study of Deep Learning for Side-Channel Analysis (2019). IACR Transactions on Cryptographic Hardware and Embedded Systems, **2020**(1):348–375 (2019). https://tches.iacr.org/index.php/TCHES/article/view/8402 https://doi.org/10.13154/tches.v2020.i1.348-375
18. Moos, T., Wegener, F., Moradi, A.: Dl-la: Deep learning leakage assessment: a modern roadmap for SCA evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 552–598 (2021). https://doi.org/10.46586/tches.v2021.i3.552-598. https://tches.iacr.org/index.php/TCHES/article/view/8986
19. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 337–364 (2020). https://doi.org/10.13154/tches.v2020.i4.337-364. https://tches.iacr.org/index.php/TCHES/article/view/8686
20. 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
21. Philipp, G., Song, D., Carbonell, J.G.: The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions (2017). arXiv:1712.05577
22. 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 Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 209–237 (2018). https://doi.org/10.13154/tches.v2019.i1.209-237. https://tches.iacr.org/index.php/TCHES/article/view/7339
23. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 157–176. Springer (2018)
24. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 677–707 (2021). https://doi.org/10.46586/tches.v2021.i3.677-707
25. Robissout, D., Zaid, G., Colombier, B., Bossuet, L., Habrard, A.: Online performance evaluation of deep learning networks for profiled side-channel analysis. In: Bertoni, G.M., Regazzoni, F. (eds.) Constructive Side-Channel Analysis and Secure Design, pp. 200–218. Springer, Cham (2021)
26. Sammut, C., Webb, G.I. (eds.): Mean Squared Error, pp. 653–653. Springer, Boston (2010). https://doi.org/10.1007/978-0-387-30164-8_528
27. 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 (2009)
28. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(2), 107–131 (2019). https://doi.org/10.13154/tches.v2019.i2.107-131. https://tches.iacr.org/index.php/TCHES/article/view/7387
29. van der Valk, D., Picek, S.: Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Paper 2019/570 (2019). https://eprint.iacr.org/2019/570
30. Wang, Q., Ma, Y., Zhao, K., Tian, Y.: A comprehensive survey of loss functions in machine learning. Ann. Data Sci. (2020). https://doi.org/10.1007/s40745-020-00253-5
31. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 147–168 (2020). https://doi.org/10.13154/tches.v2020.i3.147-168. https://github.com/KULeuven-COSIC/TCHES20V3_CNN_SCA

32. Wu, L., Perin, G., Picek, S.: I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Transactions on Emerging Topics in Computing. Cryptology ePrint Archive (2022) p. 1–12, https://doi.org/10.1109/TETC.2022.3218372
33. Srivastava, Y., Murali, V., Dubey, S.R.: A performance evaluation of loss functions for deep face recognition. In: Venkatesh, B.R., Prasann, M., Namboodiri, V.P. (eds.) Computer Vision, Pattern Recognition, Image Processing, and Graphics, pp. 322–332. Springer, Singapore (2020)
34. Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., Bao, X.: Byte-level malware classification based on Markov images and deep learning. Comput. Secur. **92**, 101740 (2020)
35. Zaid, G., Bossuet, L., Dassance, F., Habrard, A., Venelli, A.: Ranking loss: maximizing the success rate in deep learning side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 25–55 (2020). https://doi.org/10.46586/tches.v2021.i1.25-55. https://tches.iacr.org/index.php/TCHES/article/view/8726
36. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 1–36 (2019). https://doi.org/10.13154/tches.v2020.i1.1-36. https://tches.iacr.org/index.php/TCHES/article/view/8391
37. 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. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 73–96 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8583. https://doi.org/10.13154/tches.v2020.i3.73-96

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.