Non-Profiled Side Channel Attack based on Deep Learning using Picture Trace

Jong-Yeon Park[1], Dong-Guk Han[2], Dirmanto Jap[3], Shivam Bhasin[3], and Yoo-Seung Won[3]

[1] Samsung Electronics, South Korea, jonyeon.park@samsung.com
[2] Department of Financial Information Security, Kookmin University, South Korea, christa@kookmin.ac.kr
[3] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore, {djap,sbhasin,yooseung.won}@ntu.edu.sg

Abstract. A unique method to convert side channel traces to utilize deep learning schemes is proposed. The proposed method can be considered "Picturization" from raw data. We convert raw-trace data based on float or byte data into picture-formatted trace that has information based on the data position because many classical machine learning schemes are based on picture recognition of samples that cannot be easily distinguished using a machine, such as MNIST-type datasets. Our research is the first step toward applying deep learning to side channel analysis using picture-formatted data. The proposed method has some limitations, *e.g.*, overfitting and increased data capacity; however, the learning accuracy is better than previous side channel analysis results based on deep learning. Intuitively, the complex structure of neural networks increases confusion for machine learning because the weight and bias are affected by each other's values when using raw traces. However, using "Picturization" to provide inherent point weights eliminates this confusion naturally. To demonstrate why "Picturization" is more suitable for deep learning, we compare how input and hidden layers interact for each normal and suggested form. Based on this fact, the suggested scheme has the potential to improve side channel analysis. As one potential technique, we employ a binarized neural network (BNN) learning method that relies on a BNN's natural properties to improve variables. As a result, side channel analysis can take advantage of a high-efficiency FPGA/ASIC-based accelerator for BNN implementation. In addition, we provide an innovative criterion for attack success or failure based on statistical confidence level rather than determination of a correct key using a ranking system.

Keywords: Non-profiled side channel attack · Deep learning · Multi-layer perceptron · Convolutional neural network · Binaizred neural network

1 Introduction

Machine learning has seen great application in different use cases. Popular techniques like MLP (Multi-Layer Perceptron) [Bis95] and CNN (Convolutional Neural Network) [LB95, ON15] are widely utilized, but not limited to, in solving classification problems of variable complexity. Security evaluation with techniques like side-channel analysis [CDP17, HGDM+11, KPH+19, LBM15, LPB+15, MDP20, MPP16, WP19, ZBH+20] has also seen rapid adoption of MLP and CNN. The key idea here is to map the measured samples in a side-channel trace to input of a neural network while mapping the output labels to sensitive intermediate values. The network tries to learn the trace to sensitive value mapping in the training phase to classify attack traces with unknown labels.

The research on application of deep learning has seen some interesting advancement since its introduction by Maghrebi *et al.* [MPP16], where it was MLP was shown to break popular masking countermeasures. Cagli *et al.* [CDP17] demonstrated the use of shift invariance property of CNN to counter jitter based countermeasure, further improved by data augmentation techniques. Picek *et al.* [PHJ+19] showed how the commonly used Hamming weight model leads to imbalanced training datasets and proposed techniques to overcome it. In [KPH+19], authors propose the use of added Gaussian noise as a regularization technique to improve attack efficiency for different datasets while keeping the same VGG-like network. Zaid *et al.*. [ZBH+20] further improved the attack efficiency by optimizing the network for each dataset individually. Masure *et al.* [MDP20] a comprehensive study of deep learning techniques was proposed with formal links to well established side-channel metrics. Machine learning schemes such as autoencoders are shown to be used to reduce the noise level in side-channel traces [WP19]. While all these techniques operate in a profiled or supervised setting, Timon [Tim19] proposed an attack technique in the non-profiled setting, using training accuracy of the correct key as a distingiuisher against wrong key.

While all the noted previous work have shown interesting results, they processed side-channel trace as a one-dimensional stream of data. Kim *et al.* [KPH+19] adapted a VGG-like network which performed well in audio application to side-channel applications, owing to similarity in data type. However, it is well known that MLP and CNN are well suited for image classification. Research and optimizations of MLP and CNN have been demonstrated on image datasets like MNIST, CIFAR-10, SVHN *etc.* Thus, it is natural that these networks perform best on images.

In this work, we explore an alternate avenue. Instead of optimizing the network for side-channel applications, we transform the datasets for image classification problem. This is precisely achieved by representing the side-channel trace as a pixelated image. We propose *trace-transform* concept to completely utilize the principle of machine learning schemes. Further, we propose the use of binarized neural networks (BNN) [HMD+16] in side-channel application. BNN are minimalist neural networks where key parameters like weights, bias and activation use binary values giving a performance advantage over complex CNN. The advantages of BNN were shown over MNIST database. We apply *trace-transform* concept to represent the trace an image for application to BNN networks.

1.1 Motivation

The previously conducted studies on side-channel analysis have only focused on how to apply a machine learning scheme without modifying any of its properties. As previously stated, we transform the raw trace to conform to MNIST style by using all properties of the machine learning scheme. More precisely, is the present study is inspired by a previous study [LCB]. As shown in Figure 1, we can recognize "7" as a handwritten number.

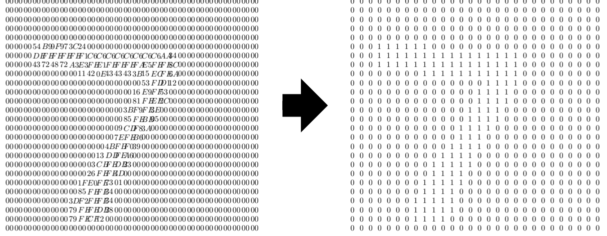

Figure 1: 7 value represented in the MNIST dataset

In fact, the '7' is represented using a hexadecimal value. While a single point in

MNIST data is represented as a hexadecimal value, a float value is used to represent the concentrated form of the figure. Without the concentration, the value "7" is represented as shown on the right-hand side of Figure 1. Then, we describe why we remove the concentration. Above all, this MNIST style is significantly helpful in learning the data in terms of side-channel analysis. We expect that the best performance can be achieved if the trace is converted to MNIST style, because machine learning schemes, such as MLP, CNN, and BNN, have been advanced based on MNIST-style datasets.

1.2 Contributions

In this paper, we introduce an innovative side-channel analysis scheme to apply the properties of machine learning schemes in non-profiling scenarios. We induce the imagification for a trace to enhance the recognition, to fully utilize the properties of machine learning schemes. More precisely, the data form is based on position, also known as picture-formatted trace, which is more human-readable than float or byte values, *i.e.*, n-dimensional vector trace. Compared to previous results, this suggestion allows the significantly reduced trace to retrieve the correct key. In addition, we provide an example to describe why our suggestion outperforms the classical data through intuitive ways. Similar to [Tim19], without information about a countermeasure in given power traces, we demonstrate that our proposed method can retrieve the correct key as well as outperform the previous results. In addition, in non-profiling scenarios, we apply an innovative metric based on statistical confidence level to distinguish between the success and failure of an attack. All experimental results are supported by the ASCAD database [PSB+18] and the ChipWhisperer-Lite board [CWw]. Thus, our proposed method outperforms an n-dimensional vector trace in terms of first- and second-order deep learning (DL) attacks. Moreover, in second-order DL attacks on ASCAD, only picture-formatted traces can be retrieved from the correct key. Furthermore, to validate the potential of our suggestion, we apply a BNN to picture-formatted traces. The result demonstrates that this process can also retrieve the correct key.

2 Preliminaries

DL is a particular type of deep neural network-based machine learning that produces exceptional results. It has also been applied to various fields, such as image and speech recognition. In this section, we briefly describe how to apply DL techniques, such as MLP and BNN, to side-channel analysis.

2.1 MLP

The general objective of MLP is to classify some input vector $\mathbf{x} \in \mathbb{R}^D$ based on its labels $l(\mathbf{x}) \in \mathcal{L} = \{l_1, l_2, \cdots, l_{|\mathcal{L}|}\}$, where D is the dimension of the input data to be categorized and $\mathcal{L}$ is the set of classification labels. The goal of an NN is to produce a function $\mathbf{NN}$: $\mathbb{R}^D \to \mathbb{R}^{|\mathcal{L}|}$ that takes $\mathbf{x} \in \mathbb{R}^D$ as the input vector and output $\mathbf{y}$ of the scores as the output. In other words, the final goal is to create the score vector $l(\mathbf{x})$ based on $\mathbf{NN}(\mathbf{x})$, updating the internal properties. In general, an NN comprises input, output, and hidden layers. In terms of side channel analysis, the input vector can be represented as points of a trace, and the output is compared to hypothetical intermediate variable, such as a Hamming weight model [BCO04], to learn the expected result. Moreover, this hypothetical variable is usually encoded using a one-hot encoding scheme.

2.1.1 Construction of MLP

In this section, we provide an example of a simple MLP. The number of inputs, also known as points, is four, and as shown in Figure 21, the example MLP comprises two hidden layers with three neurons and an output layer with two neurons. In addition, a bias neuron is included in all layers, except for the output layer. All arrows in Figure 21 represent weights, as illustrated in Figure 2. Typically, prior to performing DL, the values of weights and the bias neurons are initialized from a normal distribution using the Xavier scheme [GB10]. To obtain the expected result from the output layer, some operations are performed to determine the output of each neuron. This procedure is called forward propagation. However, learning has not even begun, because the weights have not been changed to obtain the expected result. Optimal weights and bias values required to obtain the expected result are tuned via backward propagation. This is done by comparing the expected result and the result of the output layer. The expected result is sometimes encoded using one-hot encoding. If one-hot encoding is applied to a single-bit-Hamming weight model, the output layer should comprise two neurons. If the Hamming weight value is 1, then the expected result is encoded to $[1, 0]$ or $[0, 1]$. More precisely, by comparing $[y_1, y_2]$ and the one-hot encoded value, backward propagation is performed to update all weights.

2.1.2 Forward Propagation

Forward propagation can be calculated as shown in Figure 2. A single neuron in all hidden layers and the output layer is computed by simple multiplication, addition, and an activation function. In the figure, $x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + b$ is operated prior to calculating the activation function. Then, the activation function f_{act} such as sigmoid, tanH, ReLU, softmax, and Swish, is computed to calculate forward propagation. Excluding the output layer, ReLU is sometimes adopted for all hidden layers.

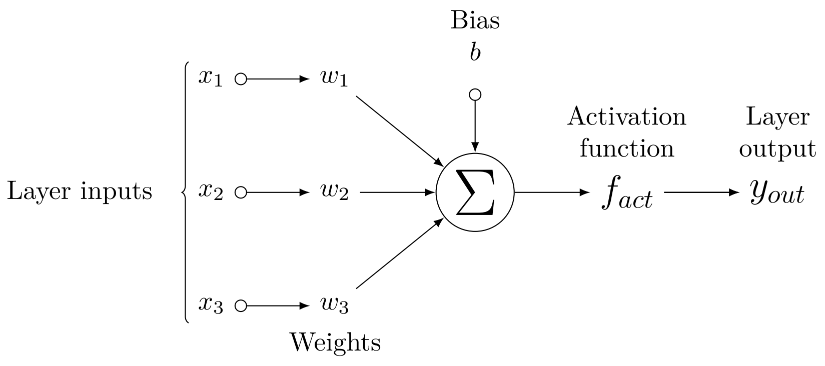

Figure 2: Calculation for a single neuron in hidden and output layers

2.1.3 Backward Propagation

The core of MLP is backward propagation because each weight can be updated to learn the expected result. Backward propagation is performed by comparing the expected result $l(\mathbf{x})$ and the output of MLP $\mathbf{NN}(\mathbf{x})$. Here, an error function, such as Euclidean distance, can be used to learn the expected result. Normally, an error function $E : \mathbb{R}^D \to \mathbb{R}$ can be defined, *e.g.*, as the Euclidean distance between the MLP output and the one-hot encoded label, as follows:

$$E(\mathbf{x}) = \sum_{i=1}^{|\mathcal{L}|} (l(\mathbf{x})[i] - \mathbf{NN}(\mathbf{x})[i])^2 \tag{1}$$

where $l(\mathbf{x})[i]$ and $\mathbf{NN}(\mathbf{x})[i]$ indicate the label value and the output neuron value, respectively. The error function value represents the gap between the expected result and the MLP output. In other words, backward propagation narrows the gap. To reflect the error for all training data $\mathbf{X} = (\mathbf{x}_i)_{1 \le i \le T}$, a loss function is defined as follows.

$$\mathcal{L} = \frac{1}{T} \sum_{i=1}^{T} E(\mathbf{x}_i) \tag{2}$$

The weights can be updated using a gradient descent technique [GBC16], which is applied to the loss function $\mathcal{L}$. Because loss function variables are based on the weights, the weights are trainable. Here, this concept is denoted as $\mathcal{L}_{\mathsf{w}}$. In other words, based on the gradient descent technique, the weights in the loss function can be updated with $\nabla \mathcal{L}_{\mathsf{w}}$. Utilizing the t-th result, $\mathcal{L}_{\mathsf{w}(t)}$, $(t+1)$-th weights can be learned as follows:

$$\mathsf{w}(t+1) = \mathsf{w}(t) - \alpha \nabla \mathcal{L}_{\mathsf{w}(t)} \tag{3}$$

where α denotes the learning rate. The total quantity for training T is consumed to learn w. However, even though the T quantity is used, the T quantity can be reused to learn w, which is referred to as an epoch.

For side channel analysis, DL is applied as follows.

- **Input layer** The points of a trace correspond to the number of input values.
- **Output layer** The adversary sets the expected result. If the target intermediate variable for attacks is the Hamming weight of a single bit of the S-box output, then the output layer has two neurons where one-hot encoding is applied.
- **Hidden layer** Excluding the input and output layers, DL can be applied to the intermediate layers in order to learn a more accurate result than a single-layer NN. This hyperparameter significantly depends on a rule of thumb, *i.e.*, the adversary selects the parameter based on trial and error.
- **Learning rate** A ratio utilizing the previous training result is used to update the weight, between the range of 0.0 and 1.0.
- **Activation function** To activate each neuron, activation functions such as sigmoid, tanH, ReLU, and softmax can be applied. Empirically in many studies, ReLU is applied to hidden layers and softmax is used for the output layer.
- **Initialization** Initialization involves setting the initial values of weights and biases. The initial values can be set as random variables in a Gaussian distribution or additional techniques can be used.

2.1.4 Additional Functions

Various additional functions are used to avoid some obstacles, such as overfitting and inefficient use of memory, in machine learning schemes. We describe the two techniques that are used in this study.

- **Batch size** The total training data are divided into batches or sets, in order to avoid overfitting by updating the weights based on batch size.

- **Dropout** The key idea is to randomly drop units from the NN during training. That is, only some connections are updated. Here, the range is between 0.0 and 1.0. For example, 1.0 means no drop.

2.2 BNN

In this section, we introduce the application of a BNN. BNN variables are suitable to our picture-formatted scheme [HMD+16]. Using a BNN can reduce the memory and computational requirements of a deep NN. Here, the core idea is binarization of all possible components, such as activations and weights. After converting the input to $+1$ and -1, we can utilize the binarization of weights and activations.

2.2.1 Binarization of Weights

When calculating initialization and forward propagation, applying binarization is not problematic. The primary issue with binarization is back-propagation. Courbariaux *et al.* [HMD+16] found very simple solutions for maintaining the real values of trained weights.

$$W_{\mathcal{B}} = \mathrm{sign}(W_{\mathcal{R}}) \tag{4}$$

After updating the weights using gradient descent, they are normally real values ($W_{\mathcal{R}}$). For binarization, we can use the sign function resulting in a tensor with values of $+1$ and -1. Then, in order to use forward propagation, $W_{\mathcal{B}}$ can be obtained from the binarization of $W_{\mathcal{R}}$.

2.2.2 Binarization of Activations

To perform backward propagation, we need to make the activation function learnable. Normal backward propagation cannot be applied to a BNN because the output of the activation function is -1 or $+1$. [HMD+16] suggested using gradient descent considering binarization, as follows.

$$\frac{\partial L}{\partial a_{\mathcal{R}}} = \frac{\partial L}{\partial a_{\mathcal{B}}} * 1_{|a_{\mathcal{R}}|\leq 1} \tag{5}$$

Here, $a_{\mathcal{B}}$ is the binarized output of the activation function and $a_{\mathcal{R}}$ is the real value input to the activation function. $1_{|a_{\mathcal{R}}|\leq 1}$ evaluates to 1 if $|a_{\mathcal{R}}| \leq 1$; otherwise 0. This zero drops out the gradient if the input of the activation function is greater than absolute value 1. We utilize an open-source BNN that is publicly available at `https://github.com/uranusx86/BinaryNet-on-tensorflow`.

3 Picture Trace and Analysis

In this section, we introduce a preprocessing method for power traces and a basic attack principle. In general power trace processes, if an event occurs at one time, there is a voltage value that has a single floating point value. This is a one-dimensional vector. For example,

$$v(trace) = 2.34$$

A collected power trace is a sequence of these voltages. If there is n-time point, one can express n-dimension (*i.e.*, n-time) as

$$v(trace) = (2.34, 1.36, 2.50, 2.97, ...)$$

This means that the *n*-dimensional vector has a rational number on n-time. The floating with the *n*-dimensional vector is described as follows (Figure 3), and is a normal power trace, where the x-axis indicates time and the y-axis indicates voltage.

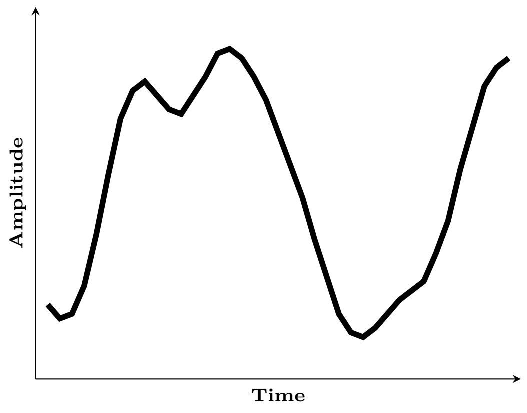

Figure 3: Trace for side channel analysis

In Figure 3, the voltage on the time axis expresses “degree”, and the power trace can be visualized for human recognition. People recognize power flow through upper markings, which represent a high degree of voltage, and lower markings, which represent a low degree of voltage. However, a real voltage flow is an n-dimensional vector sequence. This is not a picture that is expressed as a picture.

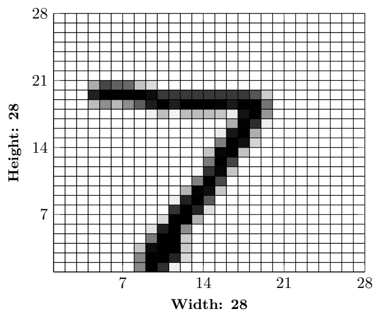

Figure 4: "7" represented in MNIST database with concentration

We perform side-channel analysis with picture-formatted power traces. There are many methods for converting an n-dimensional vector sequence into a picture. We get hints from the MNIST database, which is a simple approach of creating a picture from power traces. Figure 4 shows a handwritten number “7”. We can perceive the value through some values in a 28×28 area. Here, there are visually unnecessary areas that are treated as the value “0”, which means NULL. We seek to visualize a power trace using this method. Figure 5 shows a visualized power trace, which is a fundamental shape that we desire to change from a vector sequence to a picture. Here, the meaningful line is 1, and 0 represents nothing (like a foundation that helps us perceive the relative location of point “1”). We call it a picture trace. The steps for obtaining a picture trace are described as follows.

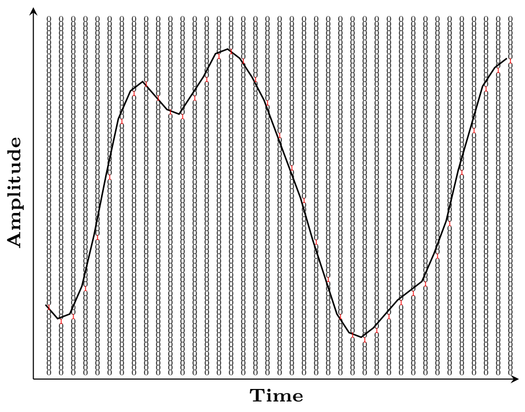

Figure 5: Picture-formatted trace for side channel analysis

3.1 Picture Encoding

3.1.1 Resolution

Resolution determines how a picture trace is represented. To obtain all information from the original power trace, we know the original trace's resolution when the trace is collected from the collecting device, *e.g.*, an oscilloscope. Otherwise, we can conduct a brute-force search to find the resolution. Figure 6 shows how we determine the resolution when converting a power trace to a picture trace. Here, the original trace is an n-dimensional vector space whose size is equal to the number of points; however, the size of a picture trace is expressed as (number of points $\times$ resolution). Therefore, a picture trace has a greater dimension.

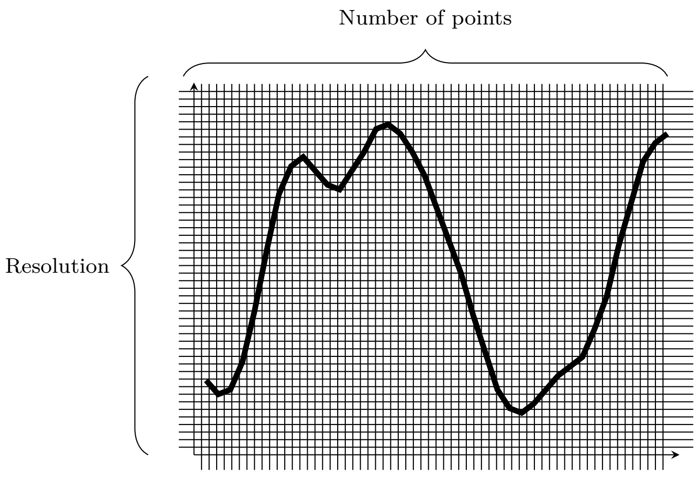

Figure 6: Determination of resolution for a trace

Note that we can select a small resolution, which results in a small input size. This facilitate faster learning and reduces memory consumption. However, some information from the original trace might be missed, which can distort the real information in sensitive data. In what follows, we discuss how to control such content.

3.1.2 Drawing

The drawing is the main step in generating a picture trace from an original trace. To draw a picture, we must determine the position of a point and place a dot in fixed space. In other words, we should determine the position of all points within the defined resolution. The related position of a point is determined using the upper bound, lower bound, and the chosen resolution. First, the gap, *i.e.*, the difference between the lower and upper bounds, needs to be computed. Note that each gap in each trace differs; however, the transformed traces must be identical to realize the same analysis standard and facilitate simple implementation.

$$\mathsf{G}ap = upperbound - lowerbound \tag{6}$$

Here, the upper bound is the maximum voltage value and the lower bound is the minimum value of all sample power traces. After computing the boundary, the related location is computed. The resolution is denoted as n, and the voltage value at a specific time is denoted as v; thus, the related location is computed as $t = \varphi(v)$, which is an intuitive concept.

$$\varphi : R \to Z_n \tag{7}$$

$$t = \varphi(v) = \left\lfloor \frac{v - lowerbound}{Gap} \right\rfloor \times n \le n \tag{8}$$

Finally, we can generate an n(resolution)-th vector with related location t. The method for expressing the picture follows MNIST; however, there is no depth in our power trace. Therefore, "1" represents the related location, and "0" means NULL. Assuming that the related location is t, the n-th vector is computed as follows.

$$\delta : Z_n \ \to \ (Z_2)^n$$

$$\delta(t) = (a_0, a_1, \ldots, a_{n-1})$$

$$a_i = \begin{cases} 0 & (i \neq t) \\ 1 & (i = t) \end{cases}$$

The final picture of each trace is a set of vectors $\delta(t)$. The picture is $\{\delta_1(t_1), \delta_2(t_2), \ldots, \delta_m(t_m)\}$.

3.1.3 Reducing

The reducing step involves the elimination of unnecessary parts of the picture. The newly drawn picture-shaped trace has unchanged "1" to "0" or "0" to "1". This area has no effect on power analysis. In our experimental results, most of the unchanged sections of the pictures are "0", *i.e.*, generally the foundation. Note that we cannot know an unchanged section from a single trace, and thus, filter out the unchanged portions. The filter is created by stacking all picture traces and selecting vectors with a value of "1".

Figure 7 shows the method used to generate the filter. The filter eliminates unchanged "0" values; thus, we obtain only meaningful locations (Figure 8). The output picture on the right side of Figure 9 is inside the filter boundary.

In our results, filtering the picture traces reduces the input size of the ASCAD example by approximately 90%. Reducing the input size enables efficient learning time and reduces the number of training epochs.

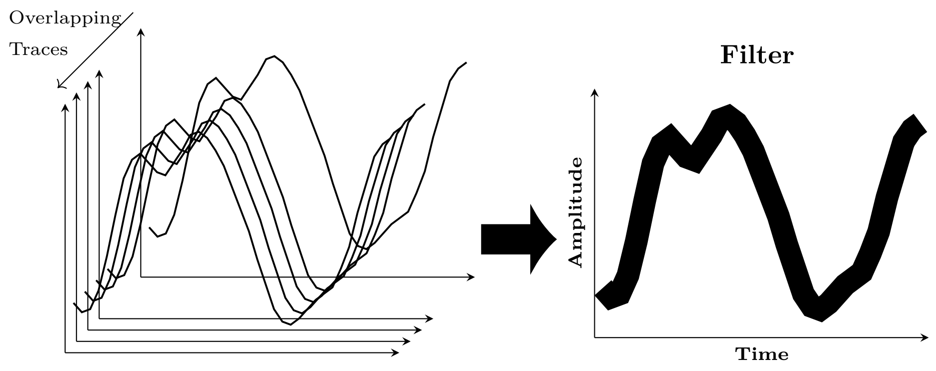

Figure 7: Overlapping traces for generating the filter

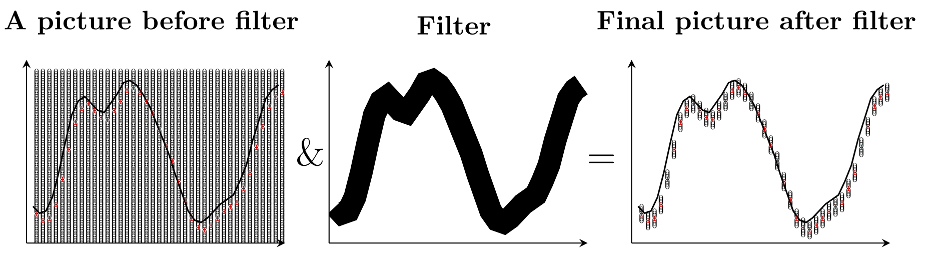

Figure 8: Generating final figure trace based on the filter

3.2 Correct Key determination

For correct key determination, the previous non-profiling attack published in CHES 2019 **??** selected the best training speed, compared to other incorrect keys. However, this method assumes that DL with a correct key will result in a more efficient way of providing labels. This means that weights and bias with higher training accuracy can compute correct labels for any power trace. However, a highly accurate training trace does not guarantee finding a correct key (Section 4.3), because DL can fall into the overfitting problem. In some results, incorrect keys have a faster learning speed, even though techniques to remove overfitting are adjusted in the experiments. Thus, we must develop other standards to determine correct keys. The reason why overfitting is difficult to eliminate is discussed in Section 3.3.2. We use a validation set for the final determination of correct or incorrect keys. Note that this validation set is not used for the learning phase. In fact, we require additional power traces to determine a correct key. In addition, the validation set must be uniformly distributed for each label. For example, single-bit labeling has 50% “1” and 50% “0”. If labeling is not distributed uniformly, some cases may induce incorrect learning; for example, selecting all “1” values has high accuracy, which is statically meaningful. Here, the validation set has n power traces with exactly 50% "0" and 50% "1". Bernoulli trials $X_1, X_2, ..., X_n$ are independent. All Bernoulli trials with success probability 1. Then, their sum is distributed according to a binomial distribution with n and $\frac{1}{2}$

$$\sum_{k=1}^{n} X_k \sim \mathcal{B}(n,\ \frac{1}{2})$$

If n is sufficiently large, $\mathcal{B}(n,p)$ is given a normal distribution

$$\mathcal{N}(np,\ np(p-1))$$

For example, if n equals 3000, the binomial distribution is considered a normal distribution. Therefore, the distribution follows$\mathcal{N}(1500, 750)$, and we can determine whether the selected key for the error possibility is correct, using a probability density function, where the lower cumulative distribution $\mathcal{P}\left(x, 1500, \sqrt{750}\right)$ is computed as$\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{x-1500}{\sigma}\right)^2}$. If 1600 passes in 3000 quires of key= 0x33, $\mathcal{P}\left(1600,\ 1500, \sqrt{750}\right) = 0.999869$. Here, the desired correct key is 0x33, and error rate is $1 - 0.999869 = 0.000131$.

3.3 Characteristics of Picture Trace

3.3.1 Easy learning

A notable difference in picture traces is that the alteration of voltage for each trace changes the vector's location. For example, as shown in Figure 9, there are only two power traces, and three voltage values at three times. The left side of Figure 9 shows normal traces, *i.e.*, a three-dimensional vector sequence. The first trace's third value is 0.1, and the second trace's third value is 0.2. The right side of Figure 9 shows the picture trace. Here, the input vectors of the picture trace are $[0, 1, 0, 0, 0, 1, 1, 0, 0]$ in the first trace and $[0, 1, 0, 0, 0, 1, 0, 1, 0]$ in the second trace. Alteration of these two traces is a location change expressed as $[1, 0]$ to $[0, 1]$. The normal trace's alteration is 0.1 to 0.2.

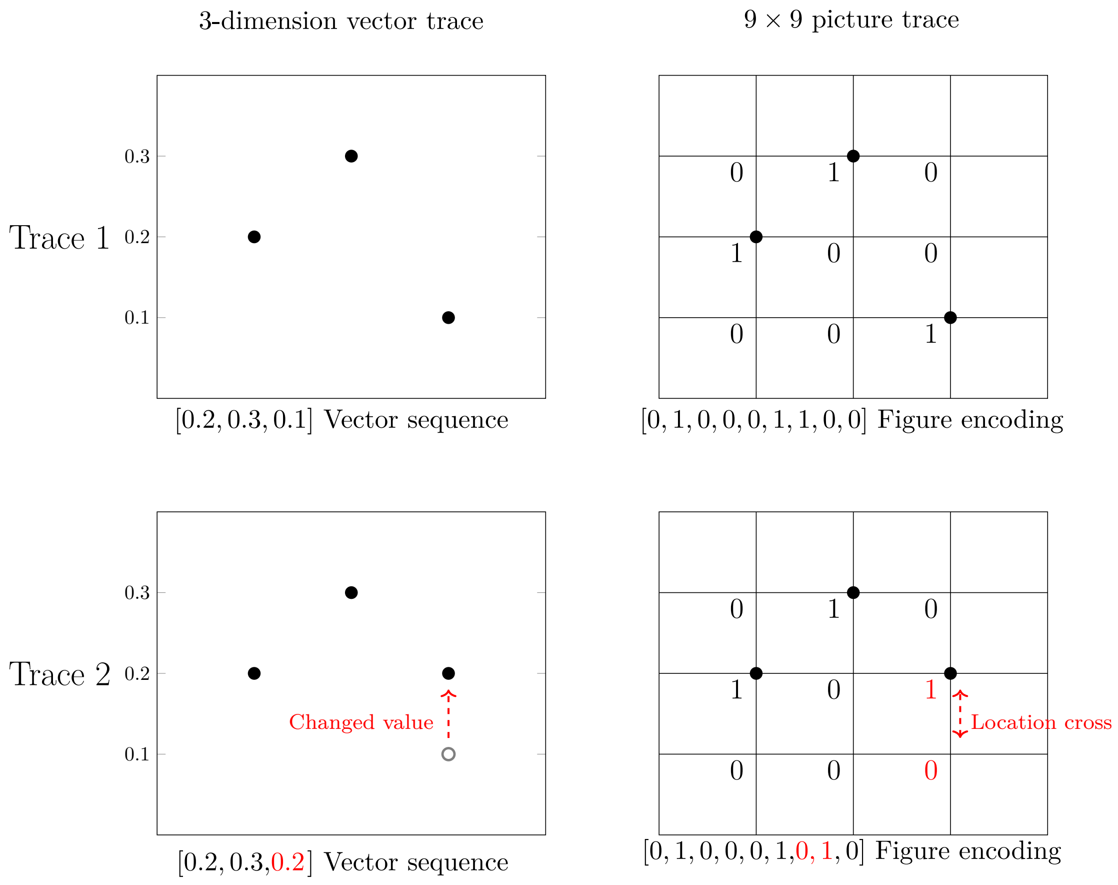
3-dimension vector trace
9 × 9 picture trace
Trace 1
0.3
0.2
0.1
0 1 0
1 0 0
0 0 1
[0.2, 0.3, 0.1] Vector sequence
[0, 1, 0, 0, 0, 1, 1, 0, 0] Figure encoding
Trace 2
0.3
0.2
0.1
Changed value
0 1 0
1 0 1
0 0 0
Location cross
[0.2, 0.3, 0.2] Vector sequence
[0, 1, 0, 0, 0, 1, 0, 1, 0] Figure encoding

Figure 9: Example of easy learning

According to this characteristic, it is easier to learn for NN fitting with weight and bias. Figure 10 shows an intuitive appearance of learning using normal traces. For example, in the last layer of the NN with softmax and one-hot encoding, the voltage values of the two traces are 1 or 2. The last step of the network selects a location with a greater value of P_1 or P_2 after softmax. Note that softmax is only used to normalize the final value from 0 to 1; thus, an upper neuron is selected when $x_iw_1 + b_1$ is greater than $x_iw_2 + b_2$; otherwise, a lower neuron is selected. If the learning process is performed effectively, the NN computes a different selection that depends on input 1 or 2. Let us consider that input 1 makes the network select the upper neuron and input 2 makes the network select the lower neuron (Figure 10). For the desired result, $w_1 + b_1 > w_2 + b_2$ if the input is 1 and $2w_1 + b_1 < 2w_2 + b_2$ if the input is 2. Note that these inequalities share the same variables, with opposite inequality signs.

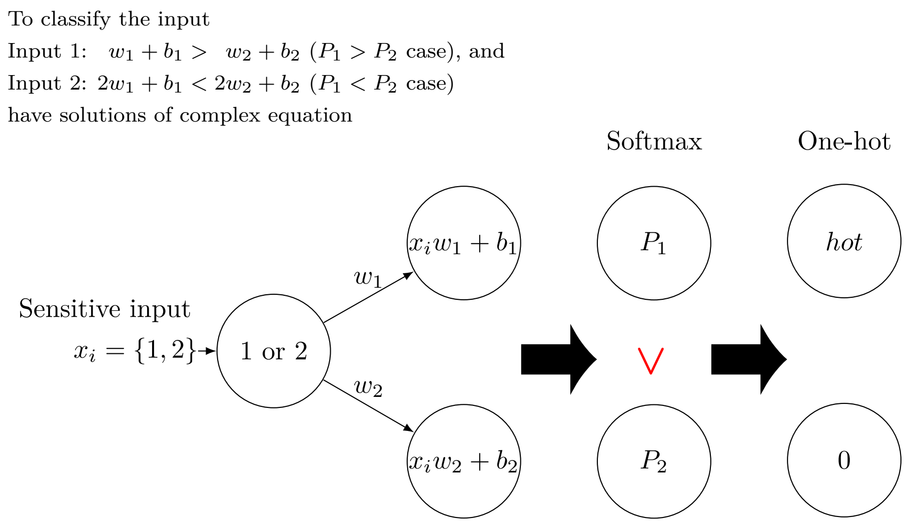
To classify the input
Input 1: $w_1 + b_1 > w_2 + b_2$ ($P_1 > P_2$ case), and
Input 2: $2w_1 + b_1 < 2w_2 + b_2$ ($P_1 < P_2$ case)
have solutions of complex equation
Softmax
One-hot
Sensitive input
$x_i = \{1, 2\}$
1 or 2
w_1
w_2
$x_iw_1 + b_1$
$x_iw_2 + b_2$
P_1
$\vee$
P_2
hot
0

Figure 10: Intuitive appearance of final layer using normal traces

Here, we find the solution. If we set $b_2 - b_1 = t$, we obtain the following:

$$w_1 > w_2 + t \tag{9}$$

$$w_1 < w_2 + t/2 \tag{10}$$

Figure 11 shows the area that is sufficient for those inequalities, where the x-axis indicates w_2 and the y-axis indicates w_1. The blue area indicates the values of weights that solve the simultaneous inequality. According to Figure 11, we can identify two conditions. The first condition is that t must have a negative sign, which means that b_1 must be greater than b_2; otherwise, the simultaneous inequality does not have solutions. The second condition is that the area satisfying the simultaneous inequality can be considerably narrow relative to variable t. Even though there are solutions, the machine will likely struggle to learn to find the desired solution without requiring many epochs.

Here, we assume that the input is (1,0) and (0,1) (rather than 1 and 2), which is a picture trace example (Figure 12). Note that this case requires four weights, *i.e.*, w_{11}, w_{12}, w_{21}, and w_{22}. As in the previous example, $w_{11} + b_1 > w_{12} + b_2$ when the input is $(1, 0)$, $w_{21} + b_1 < w_{22} + b_2$ when the input is $(0, 1)$. Unlike normal traces, each neuron has its own weight value, and it is relatively easy to find a solution, *e.g.*, $w_{11} > w_{21}$ and $w_{12} < w_{22}$. Here, we set $t = b_2 - b_1$. Then, we obtain the following:

$$w_{11} > w_{21} + t \tag{11}$$

$$w_{21} < w_{22} + t \tag{12}$$

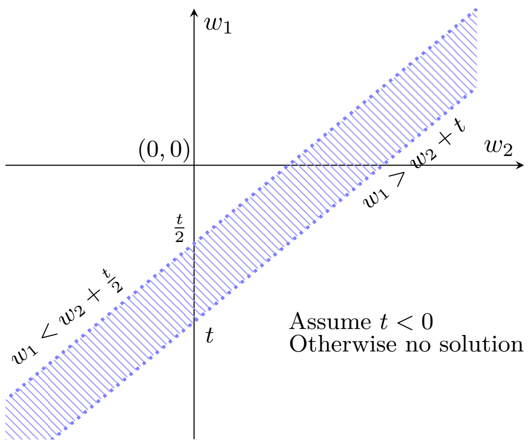

Figure 11: Area of solution of final layer using normal traces

There are five variables on two simultaneous inequalities, *e.g.*, $w_{11}, w_{21}, w_{22}, w_{12}$ and t. Regardless of the values of the four variables, the value of the remaining variable allows simultaneous inequality to work by itself.

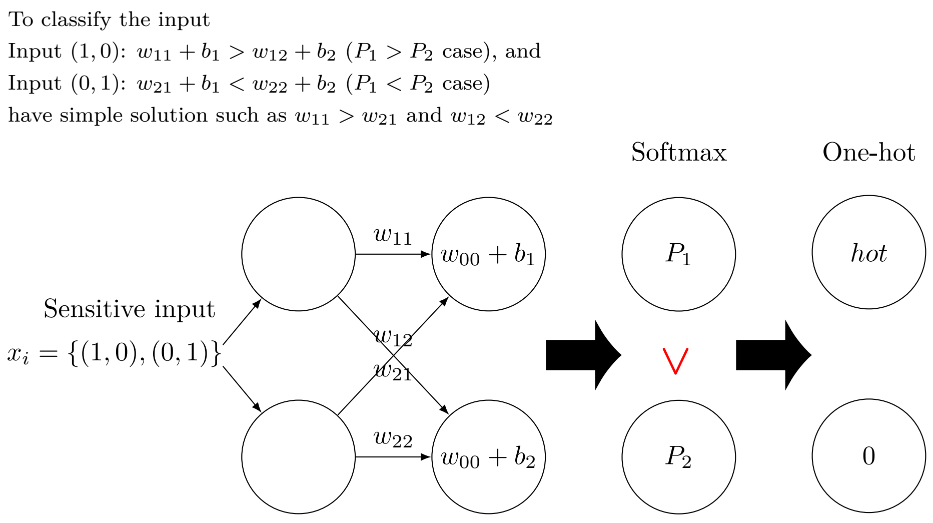
To classify the input
Input $(1,0)$: $w_{11} + b_1 > w_{12} + b_2$ ($P_1 > P_2$ case), and
Input $(0,1)$: $w_{21} + b_1 < w_{22} + b_2$ ($P_1 < P_2$ case)
have simple solution such as $w_{11} > w_{21}$ and $w_{12} < w_{22}$

Figure 12: Intuitive appearance of final layer using figure traces

In the intermediate layers, there is no choice function, *i.e.*, only the weight and bias influence the inputs to the next layer. Figure 14 shows a model where the input affects multiple neurons. This is a toy example, with one neuron in the first layer and two neurons in the second layer with ReLU. This example intuitively demonstrates huge increases in the neuron size. We concentrate on only "one neuron to one neuron," whose output is n_1 (ignore n_2 in here). Here, we assume that the input is 0.5 or 0.6. Then, the output of n_1 is $0.5w_1 + b_1$, $0.6w_1 + b_1$, or 0, as computed by ReLU activation.To improve learning, we seek a huge difference between $0.6w_1 + b_1$ and $0.5w_1 + b_1$. The difference between the two values is normally $0.1w_1$ in our example; however, due to ReLU, there cannot be a difference between $0.6w_1 + b_1$ and $0.5w_1 + b_1$. If both are negative, then both are “0.” In contrast, if either of $0.6w_1 + b_1$ and $0.5w_1 + b_1$ is negative, a few of the values of w and b make the equation sufficient; see the bottom-right of Figure 13. When we consider all cases, the maximum probability is 0.25, `i.e.`, one of the four areas. Therefore, the probability has a value of at most 0.25 in the learning system when we only consider normal traces. In other words, to learn the traces, the weight and bias are stochastically affected by each other's value.

$$0.5w_1 + b_1 < 0 \tag{13}$$

$$0.6w_1 + b_1 > 0 \tag{14}$$

denotes,

$$b_1 < -0.5w_1 \tag{15}$$

$$b_1 > -0.6w_1 \tag{16}$$

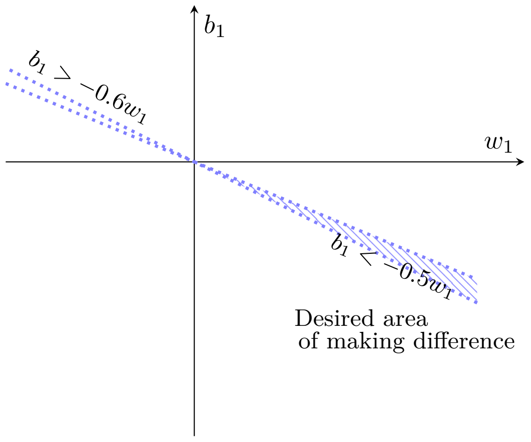

Figure 13: Intuitive appearance of an intermediate layer using normal traces

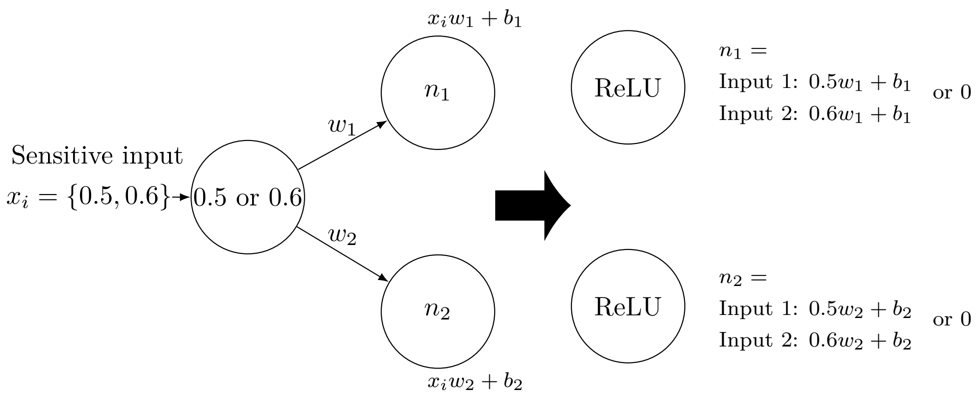

Figure 14: Intuitive appearance of hidden layer using normal traces

Figure 15 shows the intermediate layer of a picture trace. Similar to the last layer example shown in Figure 12, here, (1,0) or (0,1) matches different weightsw_{11}, w_{12}, w_{21}, and w_{22} independently, which affects the next layer's neurons more than the normal inputs shown in Figure 14.

To realize better learning, we seek a huge difference between $w_{11} + b_1$ and $w_{21} + b_1$. Irrespective of whether or not $w_{11} + b_1 < 0$, $w_{21} + b_1$ can theoretically have any positive

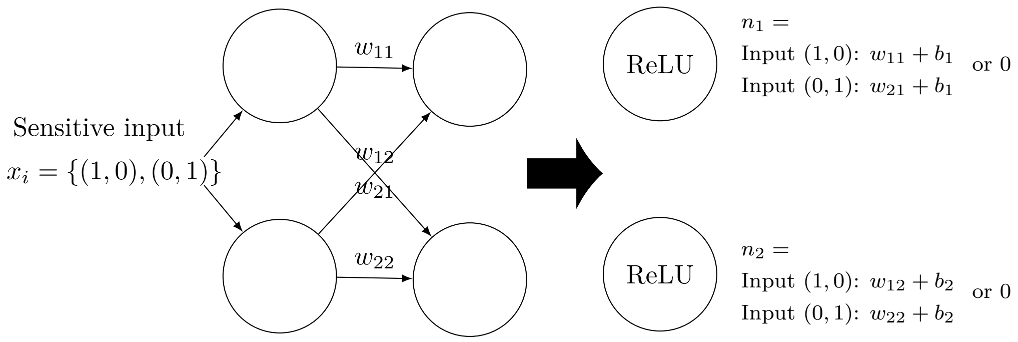
Sensitive input
$x_i = \{(1,0),(0,1)\}$
w_{11}
w_{12}
w_{21}
w_{22}
ReLU
$n_1 =$
Input $(1,0)$: $w_{11} + b_1$
Input $(0,1)$: $w_{21} + b_1$
or 0
ReLU
$n_2 =$
Input $(1,0)$: $w_{12} + b_2$
Input $(0,1)$: $w_{22} + b_2$
or 0

Figure 15: Intuitive appearance of hidden layer using figure traces

value. In fact, it depends on the scale of b1's value; however, it affects the same addition on each part. Thus, we ignore it in our theoretical analysis here. Therefore, the difference in the result of the next neurons is theoretically 0 to infinity, owing to each weight value. In a side channel attack with Hamming weight, a single-bit-difference Hamming weight has a significant impact according to the inputs.

3.3.2 Overfitting

According to different attack models, most of the voltage information of a power trace for a side-channel attack is simple noise that is not used to find the correct key. Here, large inputs do not correspond to significant amounts of information, and larger inputs may induce additional noise. With a normal trace, additional noise slows the learning process because noise interferes with finding a solution. Thus, we consider the following simple mapping rule.

Bigger noise = learning slow

However, picture trace makes learning easy with many weights, biases, and input sizes. We find simultaneous equations with many variables using only a single simple equation. There are many possible solutions; however, only one solution is correct. In this case, deep learning produces a network with weights and biases, but this is a wrong answer. In addition, there are significant noise levels in power traces for a side channel analysis in some attack models. Thus, we present another simple mapping rule.

Bigger noise
+ High learning ability
+ a lot of inputs and follwing weight and bias
= overfitting

Therefore, DL with a picture trace is more susceptible to overfitting problems, because of the following reasons.

1) Considerable noise (some of the attack models in side channel analysis)
2) Several inputs, followed by weights and biases
3) High learning ability

Overfitting is a characteristic of analysis with picture traces. Many studies have investigated reducing overfitting, *e.g.*, using sufficient traces, L1 and L2, regularization, and dropout or early stop. In this study, we only adjust "early stop" and determine the correct key using the validation set, because we do not require a perfect network to find the key, like image-net. Here, the purpose of DL is to find a key in only this environment; therefore, related research will be the focus of future work.

In a BNN, the outputs of the activation function are -1 or +1; therefore, the NN has a structural limitation relative to finding the correct answer with variables. This limitation allows us to expect to remove overfitting, *e.g.*, dropout, which randomly ignores some neurons. Because BNN distorts the output variables as -1 or +1, it cannot be a perfect learning model like NN with drop-out or regularization techniques.

4 Experimental Results

In this section, we describe our experiments and present the results. The experiments were conducted using ChipWhisperer lite [CWw] and the ASCAD database [PSB+18]. The resolution for the picture traces was set to the maximum, and a quarter of the maximum resolution. Reducing the resolution may result in information distortion. However, it makes learning faster, and to some extent, reduces overfitting caused by reduction in the number of weights related to inputs to the first layers. The only difference between the first-order attack and higher-order attacks is the label. If one uses power traces that have masked sensitive data, the label is generated by the pure data. If the masking value is known, the label can be set with unmasked values. Thus, our first-order attack uses unmasked labels with known masking values. In addition, the second-order attack uses exactly the same power traces and sets labels the S-box output without masking. Assuming a second-order attack, the machine can solve difficult problems.

In addition, we describe how MLP and BNN schemes are used to perform side channel analysis. An MLP scheme is normally handled on machine learning attack for side channel analysis. We employ a BNN in side channel analysis because its scheme fits the proposed approach. To the best of our knowledge, this is the first time that a BNN has been employed in side channel analysis.

4.1 Attack on Chipwisperer Lite in MLP/BNN attack

The basic specifications of ChipWhisperer lite are described in Appendix C. The target operation is AES SubByte and the target data are the S-box output (fourth-order S-box). The data are randomized by exclusive-or, with random 8-bit masking. First, we check the correlation power analysis (CPA) [BCO04] on the power traces, such that it is impossible to find the key from the first-order attack. The CPA results are shown in Figure 16. There are two correlation coefficient lines on 100 points. The blue line indicates the analysis of the S-box output with known masking, which is the first-order CPA attack on the S-box output, and the pink line indicates the CPA with only masking value. We make simple sample power traces that only involve a fourth SubByte operation and masking value of over 100 points. Each correlation coefficient is formed up to $0.8 \sim 0.9$. One can easily expect that the first-order analysis is well conducted with a DL attack using the known mask values.

Figure 17 shows the attack results with MLP and BNN, where the left graph shows the first-order attack and the right graph shows the second-order attack. In both graphs, Figure-α means a deep learning attack with picture trace using resolution α. Normal means a DL attack with normal traces, where 256 is the maximum resolution as per ChipWhisperer specifications and 64 is an example of reducing the resolution to one-quarter of 256. There are two red lines: the lower line indicates minimum accuracy for a

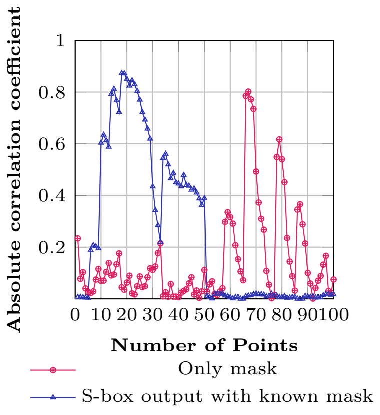

Figure 16: Result of correlation coefficient of CW

confidence level of 99.9% and the upper line indicates 99.99% confidence level with 3, 000 validation traces. If the accuracy is higher than the upper line, the key is correct at 99.99%.

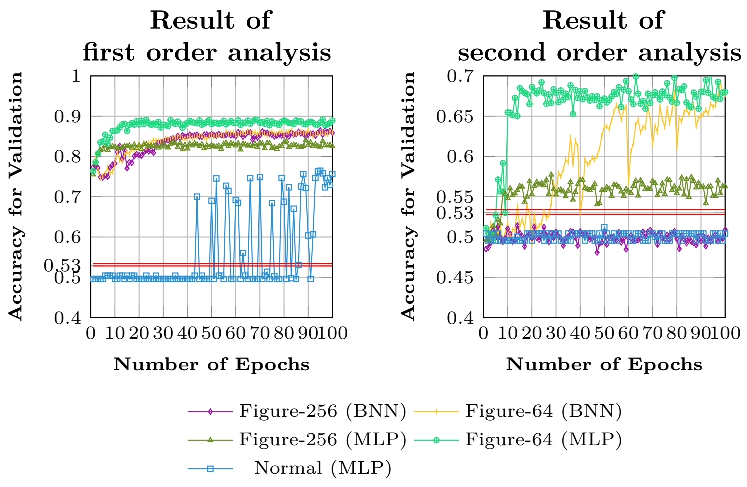

Figure 17: Result of first and second order MLP/BNN attacks on CW board

The hyperparameter settings are listed in Appendix B. According to Figure 17, the result with picture traces has higher accuracy than that with normal traces regardless of the resolution and attack orders. The result differs with different variables, such as hyperparameters, number of layers, or number of neurons. Therefore, we cannot easily say that the picture traces show better results. For epoch 200, the accuracy of normal traces is 0.9.

The results of the second-order attack are remarkable. The attack accuracy with picture traces is higher than the accuracy obtained with normal traces. Moreover, lower resolution produces a better result, which is also the case in the first-order attack. This is due to the small input size and reduced overfitting. We can know that a lower resolution reduces noise. The attack with normal traces fails at extremely high epochs, for example, epoch 10, 000. This is an unexpected result because the accuracy of the first-order attack is 0.9 on training epoch 200. As a result, a DL attack with picture traces performs better than that with normal traces in the ChipWhisperer environment. As mentioned previously, we consider a BNN attack because our solution is well-suited to a BNN. Compared to MLP and

CNN schemes, a BNN can reduce the hardware components and improve the performance because all weights and activations can be binarized. More precisely, the BNN outperforms MLP and CNN schemes because most of the 32-bit floating-point multiple accumulations are replaced by 1-bit XNOR-count operations. Our proposed picture-formatted form well fits the BNN scheme without any modification.

The BNN results are quite similar to the MLP results. In terms of first-order analysis, the BNN outperforms the normal trace attacks. Even though Figure-256 (Figure 17) in the second-order BNN analysis cannot find the correct key, the Figure-64 BNN result reveals the correct key. Thus, the BNN results are not better than the MLP results. However, because the learning cost of BNN is significantly less than that of MLP, BNN still has some merit.

4.2 Attack on ASCAD database in MLP/BNN attacks

We use the ASCAD.h5 file, which is well-aligned. To compare other attack performances using ASCAD, we use the same NN settings as those used for CHES 2019 [Tim19], but with $7,000$ traces, which is approximately $\frac{1}{3}$ the number of traces used for CHES 2019. The results of the first- and second-order attacks are shown in Figure 18. The encoding resolution is 113, and 29 is quarter of 113 for the same environment with ChipWhisperer. Because we do not know the power trace information, we consider all voltage values in ASCAD.h5. There are 113 different voltage values in integer form from -47 to 66.

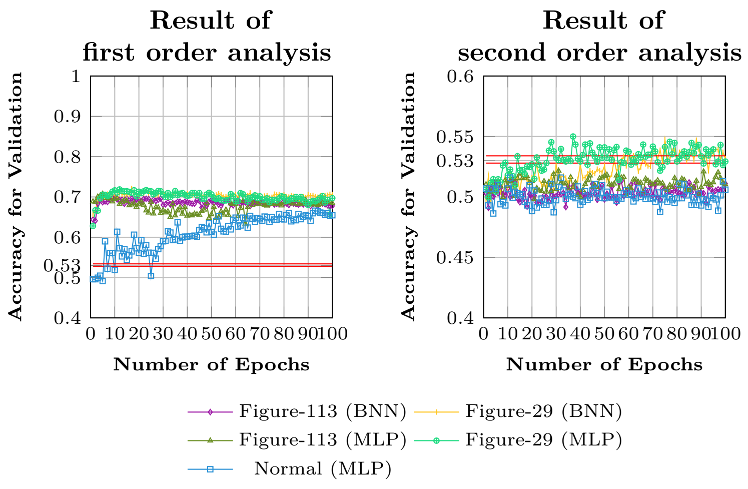

Figure 18: Result of first- and second-order MLP/BNN attacks on ASCAD

In the first-order attack, the minimum epoch with meaningful accuracy is lower with the picture traces than normal traces for both Figure-113 and Figure-29. Epoch 1 is sufficient for obtaining the correct key. This means that the picture traces demonstrate better performance in the first-order attack. Note that the red line in Figure 18 has the same meaning as that of the ChipWhisperer attack environment.

The attack with resolution 29 of picture traces is the best result in the second-order attack. With normal trace in our environmental setting (Appendix B), the attack succeeds in more than the trace epoch $1,000$. Because the minimum number of training epochs in the first-order attack is only five, the non-picture trace is less able to solve relatively difficult problems, such as higher-order attacks. However, picture encoding provides greater learning ability and organizes the inputs and neurons to solve difficult problems.

In addition, the validation accuracy of a BNN attack is quite similar to that of an MLP attack. Even though more neurons and layers are required (Appendix B), the BNN is expected to outperform the MLP because the BNN scheme uses more suitable hardware components [HMD+16, TD19].

4.3 Differential analysis and DDLA [Tim19]

Rather than employing absolute criteria, the accuracy of 256 candidates can be compared and a correct key that has the highest accuracy can be selected. Figure 19 shows accuracy values with $3,000$ validation sets of all key candidates in ASCAD. As a result, some accuracy values for wrong keys are higher than the red line (99.99%); however, this does not occur frequently. Indeed, this method can distinguish the correct key from incorrect keys. The expected attack time is twice that required when using absolute criteria with the red line. In addition, even if one can determine a correct key that has the best accuracy, it can still be a wrong key. This is because, without applying statistical confidence-level tests, the best accuracy does not necessarily mean higher accuracy.

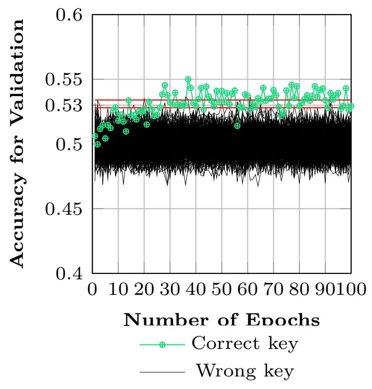

Figure 19: Result of second-order analysis for Figure-29 with all key candidates (Validation of $3,000$ traces)

The differential deep learning analysis (DDLA) [Tim19] results for picture traces on ASCAD are shown in Figure 20. Because of overfitting, training accuracy values are very high for all key candidates. Interestingly, the correct key tends to demonstrate high training accuracy for each epoch, but does not always demonstrate the highest accuracy. In addition, the results are not statistically significant; therefore, it is not possible to select a single correct key. This may depend on the attacker's heuristic result. Therefore, training accuracy is not a trustworthy criterion; a validation set must be applied for correct key determination.

The key point of DDLA is the difference in the learning aspect between the correct key and wrong keys. According to the BNN result (right side of Figure 20), there is no overfitting of wrong keys because the training accuracy values are not greater than 0.525 for wrong keys. In our experimental results, we could not remove overfitting by any overfitting elimination method, such as L1 regularization, L2 regularization, or drop-out. The BNN approach can eliminate overfitting if one does not have sufficient power trace to separate out a validation set from the collected power traces. BNN is a good solution to fully use power traces and validate the accuracy of key candidates with a training set only, such as DDLA.

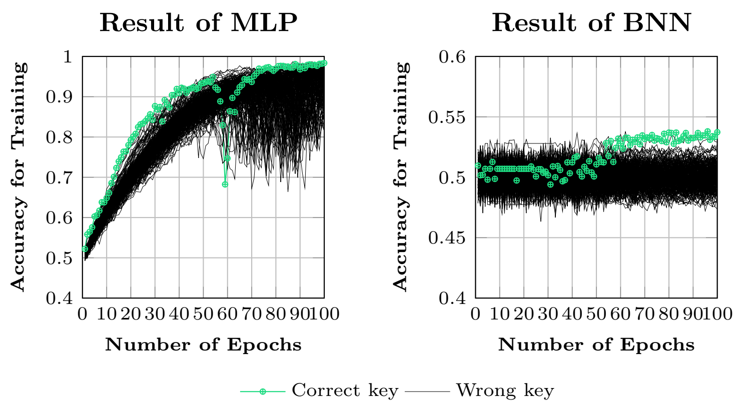

Figure 20: Result of second-order analysis for Figure-29 with all key candidates (Training of $7,000$ traces)

5 Conclusion

Many researchers have investigated the application of DL to side-channel analysis. MLP, CNNs, and autoencoders can be applied to enhance a side channel analysis. However, no study has attempted to transform the side channel information to fit a machine learning scheme. This paper proposes converting the side channel information based on an n-th dimension vector into picture form. Experimental results indicate that, compared to previous schemes using ASCAD and ChipWhisperer board, the validation accuracy is significantly higher and the number of learning epochs required to obtain the secret key is reduced. In addition, our picture format enables the retrieval of the correct key in second-order DL analysis; however, previous suggestions cannot recover the correct key in our criteria based on statistical confidence.

Moreover, our conversion scheme has the potential to enhance side channel analysis. In the future, the following potential applications will be investigated.

- **Using additional dimension to create depth** Some picture-formatted traces can be overlapped. For example, if plaintext is the same, then some traces can be overlapped by changing the concentration. Therefore, the dot in picture-formatted traces would have a specific value rather than "1". Because picture overlapping does not lose the original information, it differs from the trace integration of normal traces, such as the average.

- **Additional weight to point of interest (PoI)** Similar to the previous potential application, some additional weight on the critical data, represented as a dot, can be provided.

- **Applying additional DL schemes** Because the target is converted to the MNIST dataset style, additional DL schemes, such as BNN, can be applied and may be more effective, compared to previous side channel analysis based on DL. It is also impossible that the normal trace adds extra weight on PoI, because the extra weight distorts the original attack models.

References

[AAB$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay

Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org (2015)

[ANS18a] ANSSI. Ascad database. https://github.com/ANSSI-FR/ASCAD. (2018)

[ANS18b] ANSSI. secaes-atmega8515. https://github.com/ANSSI-FR/secAES-ATmega8515. (2018)

[BCO04] É. Brier, C. Clavier, and F. Olivier. *Correlation Power Analysis with a Leakage Model*. In M. Joye and J.-J. Quisquater (eds) Cryptographic Hardware and Embedded Systems - CHES 2004, LNCS, vol. 3156, pp. 16-29, Springer, Heidelberg (2004)

[Bis95] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Inc., New York, NY, USA, 1995.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[CDP17] E. Cagli, C. Dumans, and E. Prouff. *Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures*. In W. Fischer and N. Homma (eds), Cryptographic Hardware and Embedded Systems - CHES 2017, LNCS, vol. 10529, pp. 45-68. Springer, Heidelberg (2017)

[CWw] ChipWhisperer website. https://newae.com/tools/chipwhisperer/.

[DLw] Deep learning website. https://deeplearning.net/tutorial/tutorial.

[GB10] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. Artificial intelligence and statistics (AISTATS), volume 9, pp. 249-256, 2010.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, https://www.deeplearningbook.org. (2016)

[HGDM$^+$11] G. Hospodar, B. Gierlichs, E. D. Mulder, I. Verbauwhede, and J. Vandewalle. *Machine learning in side-channel analysis: a first study*. Journal of Cryptographic Engineering, 1(4):293, Oct 2011. (2011)

[HMD$^+$16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. *Binarized Neural Networks*. In D. D. Lee and M. Sugiyama and U. V. Luxburg and I. Guyon and R. Garnett (eds), Advances in Neural Information Processing Systems 29 - NIPS 2016, pp. 4107-4115. (2016)

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. *Differential Power Analysis*. In M.J. Wiener (ed), Advances in Cryptology – CRYPTO '99, LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)

[KPH$^+$19] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic. *Make Some Noise Unleashing the Power of Convolutional Neural Netoworks for Pofiled Side-channel Analysis*. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(3), pp. 148-179. (2019)

[LB95] Y. Lecun and Y. Bengio. *Convolutional networks for images, speech, and times-series.* MIT Press, 1995.

[LBM15] L. Lerman, G. Bontempi, and O. Markowitch. *A machine learnig approach against a masked AES.* Journal of Cryptographic Engineering, 5(2):123-139, Jun 2015. (2015)

[LCB] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[LPB$^+$15] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert. *Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis).* In S. Mangard and A. Y. Poschmann (eds), Constructive Side-Channel Analysis and Secure Design (COSADE), pp. 20-23, Cham, 2015. (2015)

[MDP20] L. Masure, C. Dumans, and E. Prouff. *A Comprehensive Study of Deep Learning for Side-Channel Analysis.* IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1). (2020)

[MPP16] H. Maghrebi, T. Portigliatti, and E. Prouff. *Breaking cryptographic implementations using deep learning techniques.* In C. Carlet, M. A. Hassan, an V. Saraswat (eds), Security, Privacy, and Applied Cryptography Engineering - SPACE 2016, LNCS, vol. 10076, pp. 3-26, Springer, Heidelberg (2016)

[ON15] K. O'Shea and R. Nash. *An introduction to convolutional neural networks.* 11 2015.

[PHJ$^+$19] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni. *The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations.* IACR Transactions on Cryptographic Hardwareand Embedded Systems, 2019(1), pp. 209-237. (2019)

[PSB$^+$18] E. Prouff, R. Strullu, R. Benadjila, E. Cagli, and C. Dumans. *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database.* Cryptology ePrint Archive, Report 2018/053, https://eprint.iacr.org/2018/053. (2018)

[TD19] T. Simons and D-J Lee, *A Review of Binarized Neural Networks.* Multidisciplinary Digital Publishing Institute (MDPI) Electronics 2019, 8, 661, 1-25. (2019)

[Tim19] B. Timon, *Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis.* IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(2), pp. 107-131. (2019)

[WP19] L. Wu and S. Picek. *Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders.* Cryptology ePrint Archive, Report 2019/1474, https://eprint.iacr.org/2019/1474.pdf. (2019)

[ZBH$^+$20] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli. *Methodology for Efficient CNN Architectures in Profiling Attacks.* IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1). (2020)

A Example of a simple MLP

We provide an example of a simple MLP as follows.

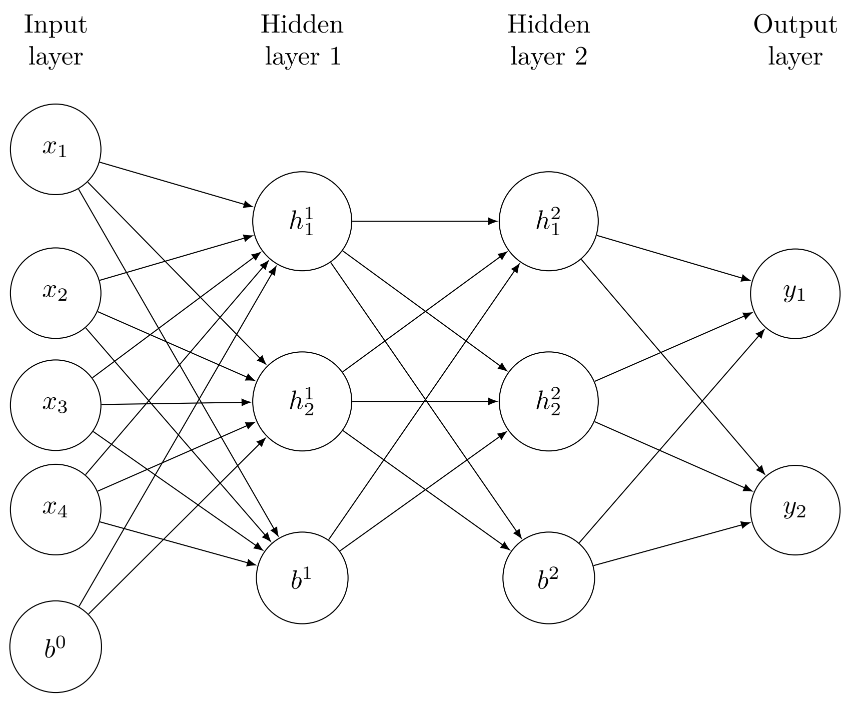

Figure 21: Example of a simple MLP

B DL Hyperparameter for all experimental results

The hyperparameter information about our attacks is provided in this appendix.

Table 1: DL Hyperparameter for ChipWhisperer on first- and second-order MLP attacks

ChipWhisperer	**Normal**	**Figure-64**	**Figure-256**
Input size	100	$1,023$	$3,690$
Hidden layer	4		
Neuron	$30 \times 30 \times 30 \times 30$	$80 \times 80 \times 80 \times 80$	
Label	MSB Hamming weight		
Optimizer	Adam (default setting)		
Dropout	70% per each layer		
Activation function	Relu, softmax, and one-hot encoding		
Learning rate	0.01	0.001	
Batch	N/A	200	
#Traces	Train: $7,000$ / Valid: $3,000$		
Initializing	Xavier initialization		

C Experimental environment on CW board

We use the CW1173 ChipWhisperer-Lite board [CWw] to acquire power traces, whose specifications specifications are as follows:

Table 2: DL Hyperparameter for ChipWhisperer on first- and second-order BNN attacks

ChipWhisperer	**Figure-64**	**Figure-256**
Input size	$1,023$	$3,690$
Hidden layer	4	
Neuron	$3000 \times 30000 \times 3000 \times 2400$	
Label	MSB Hamming weight	
Optimizer	Adam (default setting)	
Dropout	70% per each layer	
Activation function	Relu, softmax, and one-hot encoding	
Learning rate	layer-wise learning rate from 0.01 to 0.01×10^{-4}	
Batch	500	
#Traces	Train: $7,000$ / Valid: $3,000$	
Initializing	Xavier initialization	

Table 3: DL Hyperparameter for ASCAD on first- and second-order MLP attacks

ASCAD DB	**Normal**	**Figure-29**	**Figure-113**
Input size	700	$3,271$	$10,528$
Hidden layer	2		
Neuron	20×10	200×100	
Label	LSB Hamming weight		
Optimizer	Adam (default setting)		
Dropout	N/A	70% per each layer	
Activation function	Relu, softmax, and one-hot encoding		
Learning rate	0.001		
Batch	$1,000$	500	
#Traces	Train: $7,000$ / Valid: $3,000$		
Initializing	Xavier initialization		
Normalization	-1 to 1	N/A	

- ADC frequency: 29.5MHz
- ADC sample clock source: External input $4\times$ multiplier
- System frequency: 96MHz
- USB interface: Custom open-source USB firmware, up to 25MB/s speed.
- Target device: Atmel XMEGA128D4 (on classic device)
- Programming protocols: Atmel PDI (for XMEGA)

Table 4: DL Hyperparameter for ASCAD on first- and second-order BNN attacks

ASCAD DB	**Figure-29**	**Figure-113**
Input size	$3,271$	$10,528$
Hidden layer	4	
Neuron	$2000 \times 500 \times 500 \times 500$	
Label	LSB Hamming weight	
Optimizer	Adam (default setting)	
Dropout	N/A	
Activation function	Relu, softmax, and one-hot encoding	
Learning rate	layer-wise learning rate from 0.01 to 0.01×10^{-4}	
Batch	100	
#Traces	Train: $7,000$ / Valid: $3,000$	
Initializing	Xavier initialization	
Normalization	N/A	

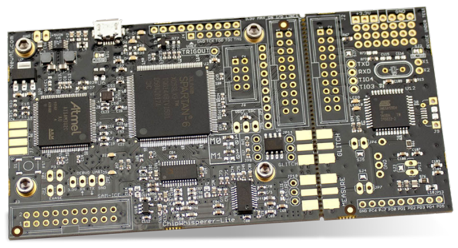

Figure 22: CW1173 ChipWhisperer-Lite board