Kilroy was here: The First Step Towards Explainability of Neural Networks in Profiled Side-channel Analysis

Daan van der Valk[1], Stjepan Picek[1], and Shivam Bhasin[2]

[1] Delft University of Technology, Delft, The Netherlands
daan@dvandervalk.nl, s.picek@tudelft.nl
[2] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore, sbhasin@ntu.edu.sg

Abstract. While several works have explored the application of deep learning for efficient profiled side-channel analysis, explainability or in other words what neural networks learn remains a rather untouched topic. As a first step, this paper explores the Singular Vector Canonical Correlation Analysis (SVCCA) tool to interpret what neural networks learn while training on different side-channel datasets, by concentrating on deep layers of the network. Information from SVCCA can help, to an extent, with several practical problems in a profiled side-channel analysis like portability issue and criteria to choose a number of layers/neurons to fight portability, provide insight on the correct size of training dataset and detect deceptive conditions like over-specialization of networks.

1 Introduction

Profiled side-channel analysis (SCA) represents the worst-case security analysis by considering the most powerful side-channel attacker with access to an open (since the keys are chosen/known by the attacker) clone device. In recent years, machine learning techniques, and especially deep learning techniques became a standard choice for profiled attacks as they allow very good performance where even targets protected with countermeasures can be broken [2,9]. As such, the progress from the first paper considering convolutional neural networks in 2016 [12] is tremendous. Besides "only" improving the performance of our attacks, we should also aim to understand and explain the machine learning process and models it produces. This problem is commonly known as the *explainability* problem in machine learning.

Unfortunately, explainability is a difficult problem. It is a central problem in a large part of machine learning research and yet, it is far from solved [6]. One aspect of explainability is the representation learning where one tries to understand why a certain representation of a problem (i.e., how the problem is represented in the layers of a deep learning algorithm) is better than some other representation. By understanding this, we can select a good representation that makes the subsequent learning problem easier. More precisely, the supervised

learning with feed-forward neural networks performs a type of representation learning [6]. The last layer gives information about the classes while every hidden layer should ideally find a representation that will make the classification process easier.

While this problem is interesting, it is also very difficult to define the representation of a neuron. Naively, one could define a table of all possible input/output mappings for a neuron (and then do this for the whole network). The problem is then that such tables would be huge and impractical to make conclusions. Consequently, we must consider techniques that allow us to capture relevant information while not requiring too much information. There are only a handful of works that are (relatively) successful in devising tools for investigating the internal representations as discussed in Section 2.3. *To the best of our knowledge, there are no results in representation learning for the domain of side-channel analysis.*

In the side-channel domain, we have distinctive challenges due to countermeasures and portability (settings where an attacker has no access to measurements from the device under attack to conduct a training but only to measurements from a similar or clone device). Therefore, it is important to consider different kinds of data, e.g., protected vs. unprotected implementations, device A vs. device B, etc. So, instead of just focusing on explaining the classifiers' predictions, it is also useful to compare the representations learned by different models.

In this paper, we use the Singular Vector Canonical Correlation Analysis (SVCCA) [24] technique to inspect internal representations learned by two popular types of neural networks: multilayer perceptron and convolutional neural networks. While usually research works concentrate on what can be done with the information at the input (such as feature selection, see, e.g., [20]) or the information at the output of a neural network (accuracy, success rate, guessing entropy [25]), we take a different path and ask what useful information can be obtained only from the middle (hidden layers) in the neural network. Consequently, in this paper, we never consider the information we can obtain from input or output. We analyze several SCA datasets and we show that indeed, different datasets have different internal representations. We see that changing the leakage model or adding/removing countermeasures can result in significantly different internal representations. Additionally, with such a tool, we can better understand the dynamics of the learning process, which can help to design more appropriate architectures. We then concentrate on the portability where we compare internal representations when the clone and attack devices are different and have different keys. Our results show we can gain insights about the internal representations of different SCA datasets. Such knowledge can then be used to design better neural network architectures, e.g., how to select the number of neurons in a layer, the number of layers, and the training dataset size. Finally, we show the hidden layers learn about labels despite never being explicitly provided with that information.

2 Background

2.1 Multilayer Perceptron and Convolutional Neural Networks

The multilayer perceptron (MLP) algorithm is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. An MLP consists of multiple layers (one input layer, one output layer, and at least one hidden layer) of nodes in a directed graph, where each layer is fully connected to the next one and training of the network is done with the backpropagation algorithm.

Convolutional neural networks (CNNs) were first designed for 2-dimensional convolutions as inspired by the biological processes of animals' visual cortex [11] and primarily used for image classification. CNNs are similar to ordinary neural networks (e.g., feed-forward networks like multilayer perceptron): they consist of several layers and each layer is made up of neurons. CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling performs a down-sampling operation along the spatial dimensions. The fully-connected layer computes either the hidden activations or the class scores.

2.2 Comparison of Neural Networks and SVCCA Methodology

Raghu et al. proposed Singular Vector Canonical Correlation Analysis to compare two layers in a network, based on the neurons' activation outputs [24]. By doing so, they were able to compare the learned representations from two neural networks in a way that is invariant to affine transformation (thus, allowing comparison between different layers and networks) and fast to compute.

SVCCA uses the following definitions:

Definition 1. *A **neuron** i is defined by the output it generates over a dataset $X = x_1, \ldots, x_N$. The ith neuron of layer l is represented by $\mathbf{z}_i^l = (z_i^l(x_1), \ldots, z_i^l(x_N))$. Here, $z_i^l(x_j)$ indicates the output (a single number) of the neuron for data sample x_j. Thus, a neuron is a vector in $\mathbb{R}^N$.*

Such an output is also called an activation vector: it stores the neuron's outcome after the activation function is applied, for all N data samples that are fed as inputs to the neural network. For convolutional layers, we treat every output of the as a separate neuron. This means c_j, the number of outputs in a layer, is defined as

$$c_j = \text{input size} \cdot \text{kernel width} \cdot \text{number of channels}, \tag{1}$$

for a convolutional layer j. For a fully connected layer, c_j is simply the number of neurons in that layer.

Definition 2. *A **layer** j is defined as the subspace spanned by its neurons, i.e., a subspace in $\mathbb{R}^N \times \mathbb{R}^{c_j}$. It is constructed as a $N \times c_j$ matrix, where c_j is the number of outputs in layer j where layer j is $l_j = \mathbf{z}_1^{l_j}, \ldots, \mathbf{z}_{c_j}^{l_j} .. \; c_j$*

Based on this definition, the SVCCA algorithm compares two layers l_1 and l_2. It operates on two matrices, each having an entry per neuron per data sample (trace). The layers can have a different number of neurons but there should be an equal number of samples N used to compare the layers. After the layers have been trained and their outputs have been stored as l_1 and l_2, the SVCCA procedure for layer comparison works as follows:

1. Singular Value (SV) decomposition of both layers separately. For both layers l_1 and l_2, their singular value (SV) decompositions are computed and outputted as $l_1' \subset l_1$ and $l_2' \subset l_2$. This transformation represents the same data in another form: matrices l_1' and l_2' will still have N rows (one row per data sample), but contain $L_1' \leq c_1$ and $L_2' \leq c_2$ columns, respectively. With this transformation, a preset percentage of the variance is explained. After this step, two reduced subspaces $l_1' \subset l_1$ and $l_2' \subset l_2$ are used as inputs for the next step.
2. Canonical Correlation Analysis (CCA) computes the linear transformations on l_1', l_2' to maximize correlation, which results in an ordered set of SVCCA components. These operations can be defined as matrices W_X and W_Y to operate on l_1' and l_2', respectively. The outputted subspaces $\tilde{l}_1 = W_X l_1'$ and $\tilde{l}_2 = W_Y l_2'$ are maximally correlated. Consequently, the algorithm returns the following outputs:
   - CCA components: the number of components is $min(L_1', L_2')$, the smallest dimension of the SVD-reduced layers. For each component, there is:
     - The value of the CCA component for both of the networks, for each trace in the comparison dataset. $o_m^i(x_j)$ denotes the value of the ith component for model m for data sample x_j;
     - the correlations $corrs = \rho_1, \ldots, \rho_{min(L_1', L_2')}$, which indicate how well each component correlates between both layers.
   - To express the output of SVCCA in a single metric, the SVCCA similarity $\overline{\rho}$ represents how well the representations of two layers are aligned with each other:

$$\overline{\rho} = \frac{1}{\min(m_1, m_2)} \sum_i \rho_i. \tag{2}$$

Note that the first step of the algorithm, Singular Value Decomposition, is the backbone of Principal Component Analysis (PCA), which is commonly used in machine learning for data reduction but also in profiled side-channel analysis. The SVCCA steps are depicted in Figure 1. The produced correlations *corrs* and the average correlation $\overline{\rho}$ will be used as a metric to evaluate common knowledge. *Common knowledge describes the similarity between the layers' representations. If two layers have common knowledge, that means their layer representations are similar and vice versa.* SVCCA can be used on any layer regardless of its position in the neural network architecture. Still, it does not make sense to consider the input layer (as there is nothing done yet) and the output layer (as this is what is commonly evaluated through various metrics).

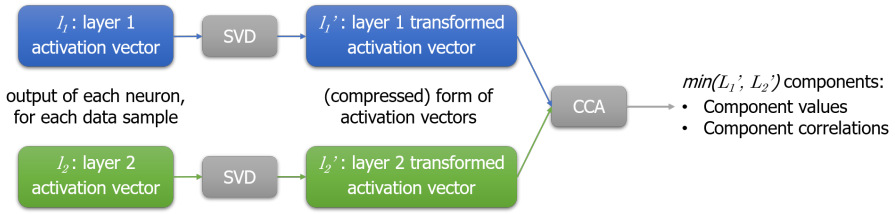


Fig. 1: Overview of the operations in SVCCA. As input, it takes the activation vectors from both layers, containing each neuron's output for all samples in the dataset. The singular vector decomposition (SVD) is applied to both layers individually, resulting in a (reduced) matrix per layer. These matrices are compared using Canonical Correlation Analysis (CCA), which is a linear transformation that maximizes the correlation between both layers. Both the correlations (from high to low correlation) and the values of the components, per sample in the dataset, are outputted.

2.3 Related Work

Templates [4] were proposed as the first profiled side-channel analysis and widely used over the years. It is shown that templates are optimal from the information-theoretic point of view if the assumed leakage model is correct and the adversary has access to a sufficient number of traces [4]. In practice, the number of traces is limited and often perturbed by noise or countermeasures. In such practical settings with limited profiling traces and added noise, machine learning algorithms can perform better than templates [22]. Maghrebi et al. [12] first started with a comparison of deep neural networks (DNN) with classical machine learning and templates and its application to break masking countermeasure using convolution neural networks. DNN was further shown to break jitter based countermeasures [2]. Later, Kim et al. [9] used Gaussian noise-based regularization to break protected implementation in as low as 3 traces. Several other applications of DNN were proposed including but not limited to non-profiled deep learning attack [27] and attack on public key cryptography [3].

Some works have also explored other aspects of DNN for side-channel analysis rather than just attack performance. The conflict of the standard metrics used in side-channel (success rate, guessing entropy) and machine learning (loss, accuracy) is studied in [21]. Picek et al. considered how to limit the number of traces for the profiling phase to better evaluate the attack performance [19]. Deep learning model generalization and understanding exploiting the class probabilities provided by the output layer was proposed in [16]. Several works have looked into the interpretation of the model learned by a DNN after training to extract the interesting features by evaluation of input activation gradient [13], occlusion techniques [8], layer-wise backpropagation [17], and sensitivity analysis [27] as a metric. The performance of machine learning techniques can be also cast as the robustness problem where one explores how different perturbations influence the performance of machine learning techniques [18]. *Despite these at-*

tempts, there is still little known about the inner working of neural networks in the domain of side-channel analysis.

Considering general research in the deep learning domain, in recent years, efforts have been made to better explain predictions of black-box techniques, typically focusing on insight in prediction decisions. This includes rule extraction, visual representations, feature importance, sensitivity analysis, and activation maximization [7]. In 2017, Raghu et al. proposed Singular Vector Canonical Correlation Analysis to compare two layers in a network [24]. The authors used this technique to measure the intrinsic dimensionality of layers, to explore learning dynamics throughout training, to show where class-specific information in networks is formed, and to suggest new training regimes. In the follow-up research, Morcos et al. further developed and tested the SVCCA methodology [15]. There, the authors use projection weighted CCA to understand neural networks and their internal representations. This technique is based on SVCCA but enables further differentiation between signal and noise.

An alternative approach for the layer-wise comparison was proposed by Yu and Chen [28]. Similar to SVCCA, it is based on neuron outputs. These activations are computed for real data and many other similar examples where noise is added using PCA. Still, this curvature-based approach has some downsides in comparison to SVCCA: for each data sample, more than 10 000 artificial samples are generated and used – providing both a conceptual disadvantage of relying on augmented data, high memory and run time complexity. Additionally, as it relies on complex mathematical concepts (Riemannian manifolds and curvatures), interpreting the outcomes is difficult.

3 Establishing a Baseline

Here, we ask the question of whether SVCCA can provide useful information about internal representations of neural network models as observed in various SCA settings.

3.1 DPAcontest v4 Dataset

DPAcontest v4 (DPAv4) provides measurements of a masked AES software implementation [26]. As the masking is found to leak first-order information [14], the mask can be considered as known and dataset as unprotected one. DPAv4 is a software implementation with the most leaking operation the processing of the S-box operation. Accordingly, the leakage model equals:

$$Y(k^*) = \texttt{Sbox}[P_i \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{3}$$

where P_i is a plaintext byte where we choose to attack the first byte, i.e., $i = 1$. The measurements consist of 3 000 features around the S-box part of the algorithm execution and in total, there is 100 000 traces available. This dataset is

available at http://www.dpacontest.org/v4/. We denote this dataset as "DPAv4 (unmasked)".

Additionally, we can also ignore the information about the mask and consider the dataset as being protected. Then, Eq. (3) changes to:

$$Y(k^*) = \texttt{Sbox}[PT_1 \oplus k^*]. \tag{4}$$

Here, we simply classify by considering the output of the first S-box. In the rest of this paper, we denote this setting as "DPAv4 (ignoring masks)".

3.2 Comparison Datasets

To get a better picture of what SVCCA outcomes mean, we compare models for DPAv4 with several other datasets. These include a dataset from another field, a set of generated "side-channel" measurements, and random data.

CIFAR-10 Dataset We use the CIFAR-10 dataset[3] as a reference problem from the computer vision domain [10]. It consists of 50 000 training and 10 000 test images. Each image consists of 32 x 32 pixels in 3 channels (colors); for this comparison, we "flatten" those to obtain 3 076 features. CIFAR-10 is an interesting dataset for our comparison, as:

- It is from a completely different domain, so no patterns are expected to overlap between DPAv4 and CIFAR-10.
- A neural network can be built with a very similar architecture: it has 3 076 features and 10 classes. This is quite close to the 3 000 features around the S-box computation of DPAv4, where we can select 9 classes (HW model).
- It was also used in the SVCCA paper [24] as a baseline.

Generated Traces To further compare with similar data, we generated a random dataset that is similar to the DPAv4 dataset and has common assumptions about side-channel measurements:

- There are 3 000 features: 2 900 are drawn completely random, from the standard normal distribution. The other 100 are semi-random:
  - For all classes, a class mean is computed for each of the 100 semi-random features.
  - For these 100 features, for some sample i, feature j is drawn: $x_{i,j} = 0.5 \cdot N(0,1) + 0.5 \cdot \mu_{k,j}$ where k is the class of sample i and $\mu_{k,j}$ indicates this class k's mean for feature j.
- The columns are shuffled randomly.

Note, although artificial, this dataset follows the Gaussian noise distribution.

[3] The CIFAR-10 dataset is available at `https://www.cs.toronto.edu/~kriz/cifar.html`.

Random "Outputs" Dataset Finally, instead of computing activation vectors from some neural network layer, a matrix of the same size is randomly generated. As this has no relation with deep learning representations, we expect to see no common knowledge with other datasets. All entries are randomly drawn from the standard normal distribution ($\sim N(0, 1)$).

3.3 Experimental Setup

To test the SVCCA methodology, we compare small MLP instances with (nearly) the same architecture. We consider only a very simple MLP with a single hidden layer, consisting of 100 neurons with the ReLU activation function. The next layer is the output layer, having either 9 (DPAv4 HW, Generated), 10 (CIFAR-10), or 256 (DPAv4 intermediate value) classes/neurons. The output layer uses the Softmax activation function. The models are trained minimizing the categorical cross-entropy loss, using the Adam optimizer, and the training runs for 50 epochs. All models are trained on 25 000 measurements.

Note that all networks have 3 000 inputs, except for those trained on CIFAR-10, which have 3 076. To adjust for this small mismatch, the comparison data is either padded with zeros at the end (when there are not enough features) or cropped at the end (when there are too many features). The models generally performed well after training, with close to 100% accuracy for the DPAv4 and Generated dataset and roughly 50% accuracy for the CIFAR-10 dataset. The comparison given here is conducted between the only hidden layer of each model. For each of the scenarios, an MLP is randomly initialized and trained. We list all considered settings in Table 1. It compares a model trained on the first 25 000 DPAv4 traces, with labels as described by the unmasked leakage model (Eq. (3), in the HW leakage model) with itself and several other models.

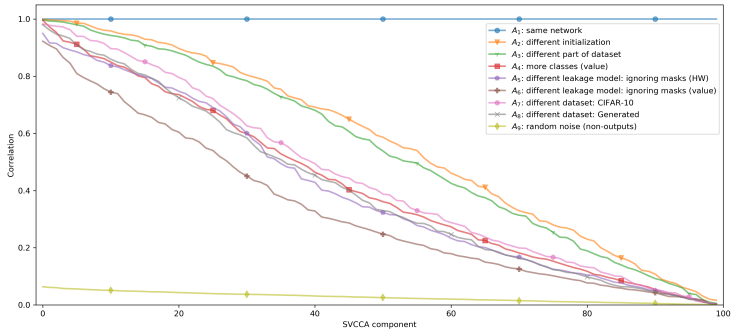


Fig. 2: Results for baseline experiment A, the SVCCA comparison between a model trained on DPAv4 (HW) and several other models. A detailed description of these settings is listed in Table 1.

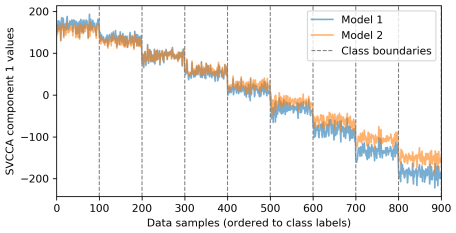
Network 1	Network 2			Scenario
	Dataset	Indexes	Model	
DPAv4 (unmasked), HW model, trained on indexes 0–25 000	DPAv4 (unmasked)	0–25 000	HW	A_1 Same network
		0–25 000	HW	A_2 Different initialization
		25 000–50 000	HW	A_3 Different part of dataset
		0–25 000	value	A_4 More classes (value model)
	DPAv4 (ignoring masks)	0–25 000	HW	A_5 Different leakage model: ignoring masks
		0–25 000	value	A_6 Different leakage model: ignoring masks
	CIFAR-10	0–25 000	–	A_7 Different dataset: CIFAR-10
	Generated	0–25 000	–	A_8 Different dataset: generated traces
	"Random" activation vectors			A_9 Comparison with random data

Table 1: Baseline experiment A: a single network trained on the DPAv4 dataset in the HW model compared with itself and several other models. The comparisons are based on the networks' hidden layer outputs, using all 100 000 traces in the DPAv4 dataset. "Random" activation vectors mean there is no network outputs, but "activation vector" are randomly drawn from the standard normal distribution.

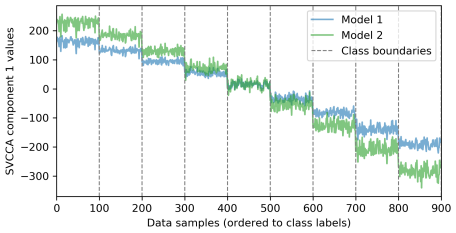
The resulting SVCCA correlations are shown in Figure 2. For each comparison, the entire DPAv4 dataset is used to generate the models' activation vectors. The blue line indicates the comparison with the same network's outputs (A_1). As expected, it shows a perfect correlation ($\rho_i = 1$ for all i). Next, we give a comparison with an MLP trained on the same data, but with a different random initialization of the model's weights before training (A_2, orange line, $\overline{\rho} = 0.5646$). The green line (A_3) compares with yet another network, trained on the different 25 000 traces from DPAv4, showing a slightly lower correlation ($\overline{\rho} = 0.5404$).

A slight modification of the original network's problem is the switch from the HW leakage model to the intermediate value model (A_4, $\overline{\rho} = 0.4162$). Based on the same data, a network can also learn the original S-box output (i.e., Eq. (4)) in the HW (A_5, $\overline{\rho} = 0.3992$) or the intermediate value model (A_6, $\overline{\rho} = 0.3287$). A_5 and A_6 are interesting datasets as they depict the significant impact of adding a masking countermeasure on common knowledge (and internal representation of models). Although these DPAv4-related models still show some similarity with the original model, we observe a stronger similarity between the original model and some unrelated models. In particular, we see a slightly higher correlation with the CIFAR-10 model (A_7, $\overline{\rho} = 0.4429$) when compared to the models with/without masking. Roughly on the same levels as the others, we see the similarity with the model trained on generated data (A_8, $\overline{\rho} = 0.4031$). Finally, as expected, we see almost no correlation with a completely random vector drawn from the standard normal distribution (A_9, $\overline{\rho} = 0.0271$).

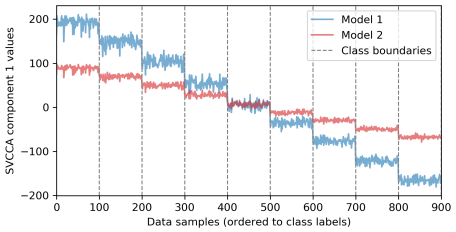
Label-based Inspection To further investigate SVCCA, Figure 3 shows the values of the first component for each of the scenarios in Figure 2 – except the first one, as identical models result in identical outcomes. While the model 1 remains fixed in Figure 3, the blue lines indicate model 1's SVCCA-transformed output, which depends on the model that it is compared to. As SVCCA finds



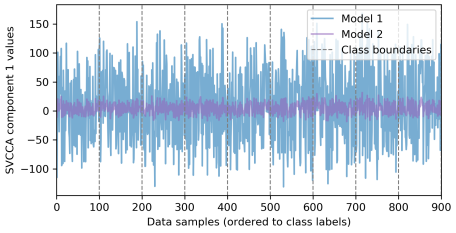
(a) A_2: Different initialization.



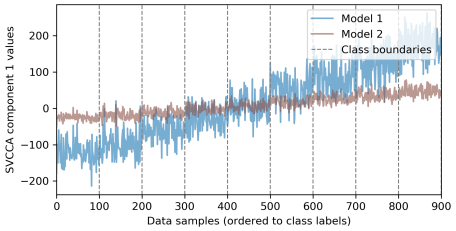
(b) A_3: Different part of dataset.



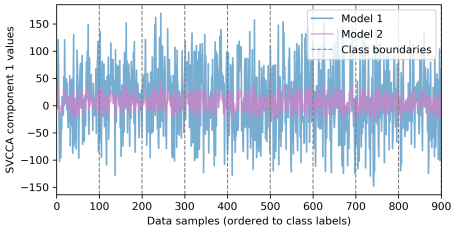
(c) A_4: More classes (value model).



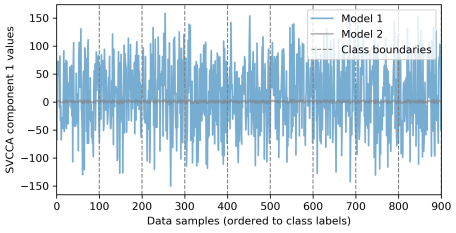
(d) A_5: Different leakage model: ignoring masks (HW model).



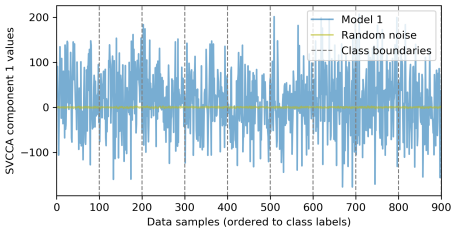
(e) A_6: Different leakage model: ignoring masks (value model).



(f) A_7: Different dataset: CIFAR-10 images.



(g) A_8: Different dataset: generated traces.



(h) A_9: Random noise, instead of a network's outputs.

Fig. 3: First SVCCA component for comparisons as described in Table 1 and shown in Figure 2. Along the x-axis, data samples are sorted according to class label; for each class, 100 samples are randomly selected. The y-axis indicates the value of the first SVCCA component for these samples. When training in the same leakage model (regardless of being HW or intermediate value), a strong correlation between the class label and SVCCA component is observed.

the best linear mapping to align the two models' outputs, each graph shows different values for model 1. Along the x-axis, 100 samples are randomly selected for all 9 classes in the HW model. The y-axis shows the first SVCCA component value per data sample. Notice that SVCCA is purely based on the activation vectors, in this case of the hidden layer. SVCCA is not based on class labels, but interestingly enough, we can see a relationship between the class labels and the first component value for some scenarios. Besides considering the first component, one can also use other SVCCA components but the most correlated information is of course contained in the first one.

Next, we require a definition for the relation between an SVCCA component and the class label. For this, the following correlation metric suffices:

Definition 3. *The class-correlation of some experiment E for two models ($m \in [1, 2]$) for an ordered list of class labels Y is $\rho^{E}_{Y,\,m}$ equals the Pearson correlation between the first SVCCA component, based on experiment E, and the class labels Y, for one of the two compared models m.*

Here, Y represents the labels of DPAv4 where the known mask is removed (Eq. (3)) in the HW leakage model. We observe that when different MLPs are trained on exactly the same data, there is a very high correlation with the class label for both networks (Figure 3a, $\rho^{A_2}_{Y,\,1} = -0.9824$, $\rho^{A_2}_{Y,\,2} = -0.9827$). This means that the strongest pattern (i.e., the first component) that SVCCA finds among the hidden layer's outputs, are highly correlated with the class labels learned by the networks. When training on similar, but not identical data, we see a comparable situation (Figure 3b, $\rho^{A_3}_{Y,\,1} = -0.9826$, $\rho^{A_3}_{Y,\,2} = -0.9820$).

In the next scenario, we compare with a model trained with identical data, but taking intermediate values, instead of HW as labels. Notice that these 256 classes are "encapsulated" in the 9 HW classes. For example, value "42" always maps to the Hamming weight "3". Figure 3c shows the result: again, we see a high correlation between the first component values for both models, and the HW class label $\rho^{A_4}_{Y,\,1} = -0.9895$, $\rho^{A_4}_{Y,\,2} = -0.9885$. In other scenarios, we see lower correlations between the first SVCCA component and labels. When comparing with a model trained for DPAv4 HW labels while ignoring the mask (Eq. (4)), we see no significant relation between the component values and the class labels (Figure 3d, $\rho^{A_5}_{Y,\,1} = 0.0371$, $\rho^{A_5}_{Y,\,2} = 0.0356$). Clearly, the most similar patterns in these layers say nothing meaningful about the samples' classes. Similar lack of correlation is observed in Figures 3d until 3h.

For models trained on similar data and the same leakage model, we see their internal representation is extremely similar. This does not depend on the choice of the leakage model. Although SVCCA is not provided with class labels, the underlying patterns it finds show that the MLPs have an extremely similar internal representation, aiming for high class separability.

Based on the obtained results, we observe the following:

- Changing the parts of the dataset used in training has a similar effect as changing the initialization values. Both changes have little impact on internal representation, which indicates one should not be too worried about the influence of such changes (when compared to some other possible changes).

– When comparing networks trained with the HW leakage model to those on the intermediate value model, the inner representation can be similar.
– The effect of having or not having a masking countermeasure influences the internal representation of a model significantly.
– Certain correlation is to be expected even when comparing very different datasets (as seen for DPAv4 and CIFAR-10).
– Simply looking at the correlation values can be misleading as the datasets that are closer from the domain perspective (DPAv4 with and without masks) can differ more than datasets that are completely non-related (DPAv4 and CIFAR-10).
– Although SVCCA is independent of class labels, its components can be highly correlated with the labels.

To conclude, we can use SVCCA to compare internal representations of different models. Unfortunately, SVCCA is not a reliable measure for comparing arbitrary datasets as we can see correlation differences but we cannot estimate how significant is that difference in practice.

4 Portability

In the previous section, we concluded that SVCCA is not a suitable method to compare neural networks trained on entirely different datasets. In realistic scenarios in SCA, we use two different devices for profiling and attacking (commonly known as portability), where those devices are similar, which in turn means that the acquired datasets should be similar. Recently, several works explored the portability issue for deep learning attacks and concluded it represents a problem for their performance [1, 5].

4.1 Datasets and Experimental Setup

We use data from several devices running AES-128 in software. The target device is an 8-bit AVR microcontroller running at $16MHz$. The devices are not protected with any countermeasures and we attack the first S-box of round 1. For each copy of the device, there are 50 000 traces where each trace has 600 features. We use three datasets with the following relationship among them:
– Datasets 1 and 2 are taken from different devices but have the same key.
– Datasets 1 and 3 are taken from different devices and use different keys.
– Datasets 2 and 3 are taken from the same device but have different keys.

To allow a meaningful comparison, we use the neural network architectures as proposed in [1]:
– MLP: a small multilayer perceptron with three hidden layers, having 50, 25, and 50 neurons. The input layer consists of 50 features, which are selected based on the Pearson correlation.
– MLP2: a multilayer perceptron with four hidden layers, having 500 neurons each. For this architecture, all 600 features are used.

– CNN: a convolutional neural network with one convolutional block and two fully connected layers. The convolutional layer has a filter size of 64 and kernel 11. We use the average pooling layer with pooling size 2 and stride 2. The fully connected layers have 128 neurons each. Again, we use all 600 features.

All algorithms aim to optimize the categorical cross-entropy, with a batch size of 256, and $RMSProp$ optimizer. For multilayer perceptrons, we train for 50 epochs and use a learning rate of 0.001, while for CNN we train for 125 epochs and use a learning rate of 0.0001. These hyperparameter values are based on [23]. Following the scenarios from [1], we train those networks with either 10 000 or 40 000 training examples.

4.2 Results

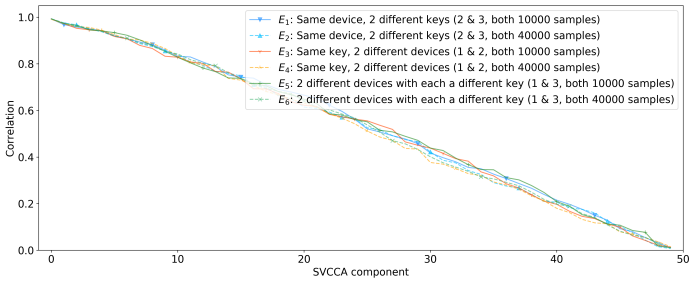
Figure 4 shows a comparison of several MLP models in the intermediate value leakage model for each hidden layer. Comparing networks trained on different datasets (1 vs. 2, 1 vs. 3, and 2 vs. 3) seems to result in a homogeneous amount of common knowledge (i.e., the internal representation is very similar despite changing the devices/keys). Also, the training set size seems to not influence the correlation between networks. This may be explained by MLP's small architecture: the networks roughly learn the same function, which approximates the training data but do not overfit. Finally, we see that all neurons are involved in every layer, which indicates those neurons indeed carry the information relevant for the internal representation. Notice a faster drop in the correlation value for the hidden layer 3, which indicates one could use fewer neurons in that layer without limiting the internal representation. We omitted the results for the HW model as they produce very similar results.

For the MLP2 architecture, we show results for the HW model in Figure 5 and for the intermediate value model in Figure 6. When considering the HW model, in the first layer we see that the correlation is much lower when using a larger training set size. This may indicate that with a larger network like MLP2, the neurons fit much more precisely around the training data.

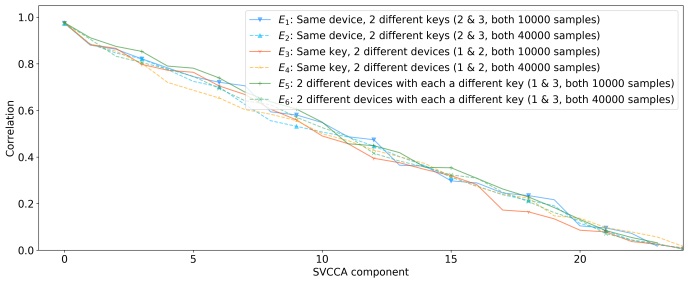
Although [1] reports better performance for these larger networks, their specialization leads to a divergence from models learning other (large) datasets. This also suggests that when in portability settings, one could benefit from using smaller training set sizes as those will result in less specialization. This is also following observations made by Bhasin et al. [1]. *Here, by specialization, we consider the phenomenon where a part of the network learns the feature representation for a specific dataset.* We formalize the notion of over-specialization in the context of portability.

Definition 4. *Over-specialization is an effect where a neural network (or a part of it) learns to generalize only for a specific dataset and is not able to generalize for other datasets as seen in portability.*

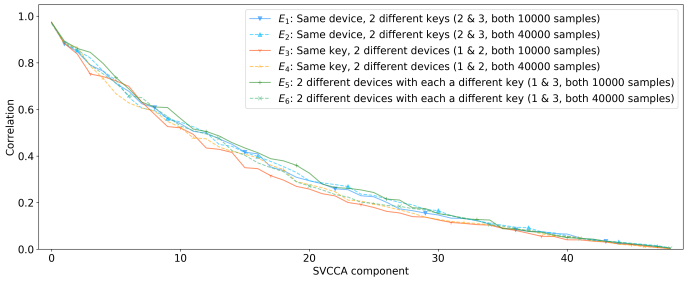
Note, if a neural network overfits, it also over-specializes, but the converse is not necessarily true. Indeed, one can easily have a neural network that general-



(a) MLP correlation, hidden layer 1.



(b) MLP correlation, hidden layer 2.

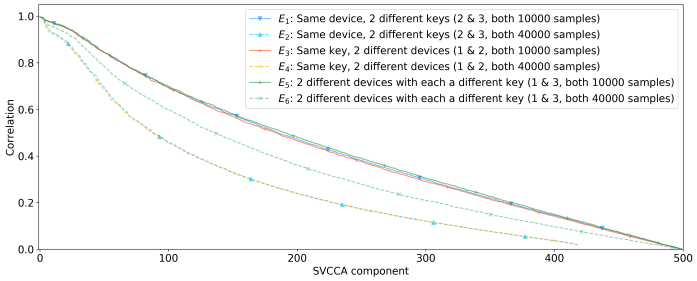


(c) MLP correlation, hidden layer 3.

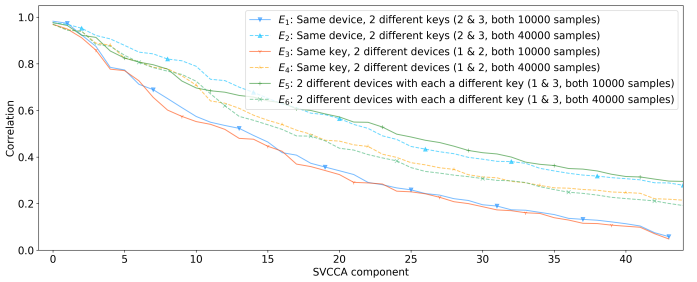
Fig. 4: Correlation results for MLP in the intermediate value model for every hidden layer.

izes well for the unseen data from the same dataset (device/key), but will not generalize to another device/key setting.

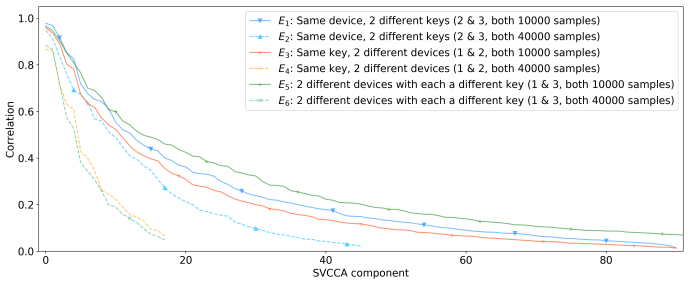
The results for hidden layers 2 and 3 show that the learned representations can differ significantly, thus signifying the portability can represent a problem for deep learning. Interestingly, we also see that the number of SVCCA components is much lower than the number of neurons, which means we do not require so many neurons in these layers to capture the internal representation of data. For hidden layer 3, we see an even larger influence of over-specialization if we use more training examples. The fourth hidden layer also requires a much smaller number of neurons as this is the last layer before the output layer where there



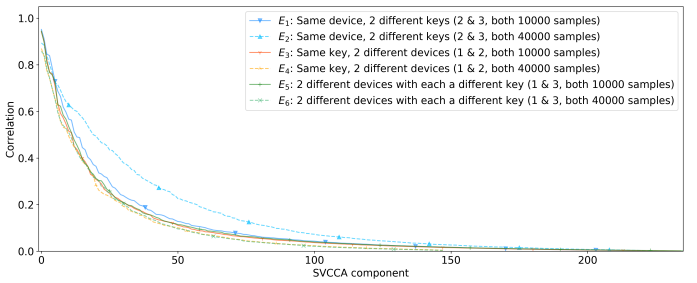
(a) MLP2 correlation, hidden layer 1.



(b) MLP2 correlation, hidden layer 2.



(c) MLP2 correlation, hidden layer 3.



(d) MLP2 correlation, hidden layer 4.

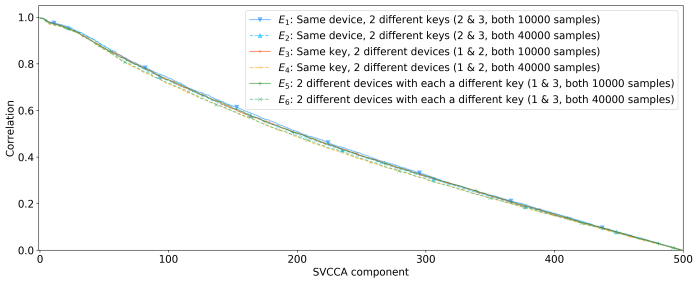
Fig. 5: Correlation results for MLP2 in the Hamming weight model for every hidden layer.

are have 9 classes (thus, having a smaller number of neurons also makes sense). Finally, we see the largest part of the specialization is happening in the middle layers.

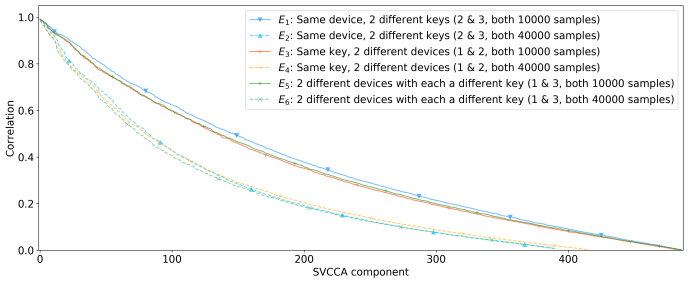
When considering the intermediate value model (Figure 6), we see the findings are somewhat similar to those for the HW leakage model. In the first hidden layer, the internal representations are very similar and using all neurons. This indicates that the internal representation in the first hidden layer still did not manage to pinpoint on finer differences in the datasets. Already in the second hidden layer, we see a significant drop in correlation for datasets using 40 000 in the training phase. This confirms that having more measurements can lead to over-specialization, which can result in worse performance in portability settings. The last hidden layer shows a relatively stable behavior but with quite a fast drop in the correlation. Consequently, some of the settings that had a bad correlation in the previous layer managed to improve their internal representation but it is still quite low for most of the SVCCA components, which indicates potential problems for the classification process. Interestingly, for hidden layers 2 to 4, we see several scenarios where we do not need 500 neurons. We do not see significant differences (for certain scenarios) between layers 2, 3, and 4, which means there is no added benefit of having those layers. Consequently, it could be beneficial to explore smaller architectures here. Finally, we observe that the specialization occurs in the middle layers, similar to the HW scenario. Naturally, the effect is smaller here as we use more classes so to specialize, we also need more training examples.

To test this hypothesis, we now aim to design a smaller multilayer perceptron architecture with better performance. We conducted a grid search with 0–5 hidden layers, having 100/200/300/400/500 neurons per layer, for learning rates 10^{-lr} with $lr = 2, 3, 4, 5$. The model is trained on 40 000 samples and uses all 600 features. Based on the validation accuracy of 5 000 samples, we find that the best performance is obtained with a multilayer perceptron with 2 hidden layers, having 300 and 100 neurons. We note that this architecture was optimized for dataset 1 where we used Adam optimizer with a learning rate of 0.001. We see such an architecture is following the information provided by SVCCA (cf. Figure 6 as there, the correlation values indicated we need fewer neurons and layers. Naturally, as we change the number of neurons in the first hidden layer (300 instead of 500), it becomes difficult to compare with previous results for the second hidden layer as the architecture is now changed.

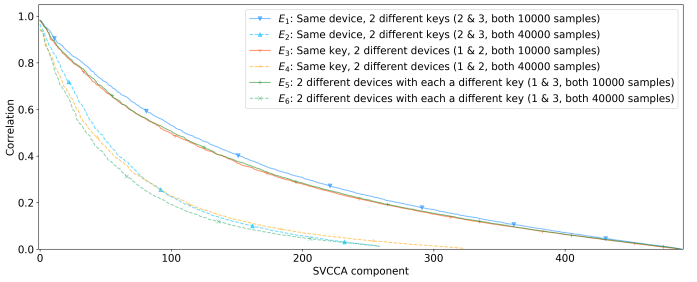
In Figure 7, we depict the results for every hidden layer when correlating the class labels and the first SVCCA component. As before, along the x-axis, data samples are sorted according to the class label. For each class, 100 samples are randomly selected. The y-axis indicates the value of the first SVCCA component for these samples. Since here we consider the intermediate value leakage model, 256 classes are encapsulated in the Hamming weight classes. We can see the values of the first SVCCA component increasing as going toward deeper hidden layers. Additionally, while the values are generally well correlated, we see certain differences, which are especially apparent in the last hidden layer. So, while the



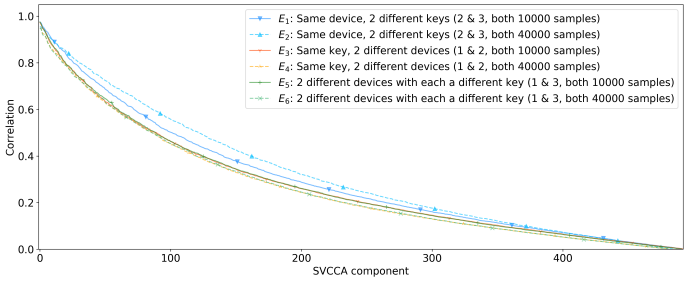
(a) MLP2 correlation, hidden layer 1.



(b) MLP2 correlation, hidden layer 2.



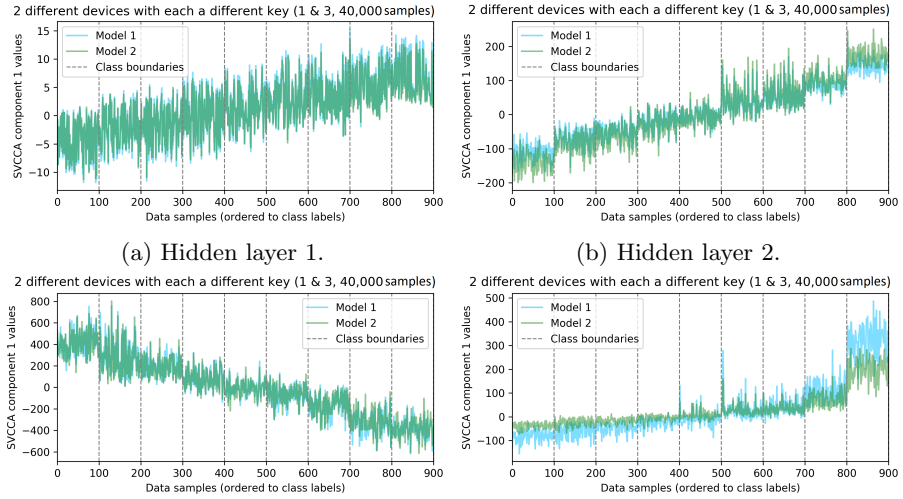
(c) MLP2 correlation, hidden layer 3.



(d) MLP2 correlation, hidden layer 4.

Fig. 6: Correlation results for MLP2 in the intermediate value model for every hidden layer.

layers managed to learn about the labels, one could expect potential issues in the attack performance due to over-specialization with training sets.



(a) Hidden layer 1.

(b) Hidden layer 2.

(c) Hidden layer 3.

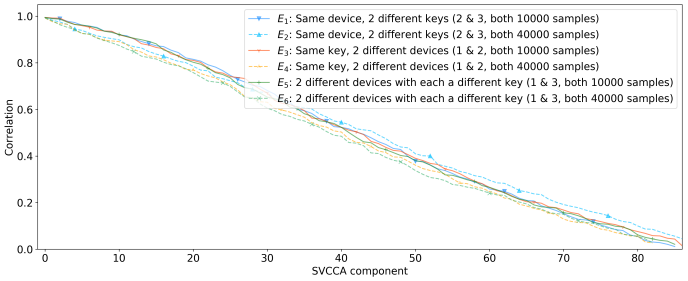
(d) Hidden layer 4.

Fig. 7: Label-based inspection for MLP2 when both devices and keys differ, intermediate value model for every hidden layer.

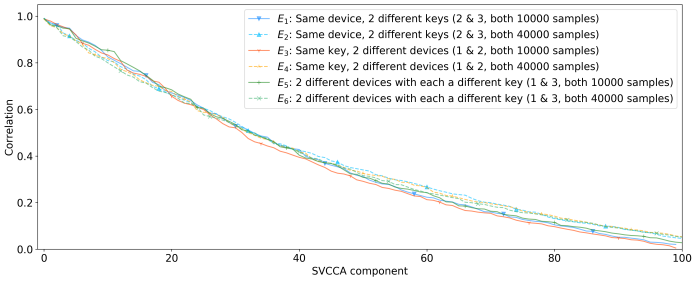
Finally, we investigate the results for the CNN architecture. First, we note that we omit the convolutional layer due to practical limitations. While the number of parameters in convolutional layers is low, they produce massive activation vectors. As an example, the CNNs' first layers outputs are roughly 750 times larger than those of the MLP, taking more than 28 GB to store a single convolution activation vector.

In Figure 8, we depict results for 2 fully connected layers for both HW and intermediate value leakage models. First, when considering the HW model, we see that the internal representations are similar and that we require fewer neurons than used. This means that the portability setting does not produce many issues in the HW model but also that one fully connected layer could suffice.

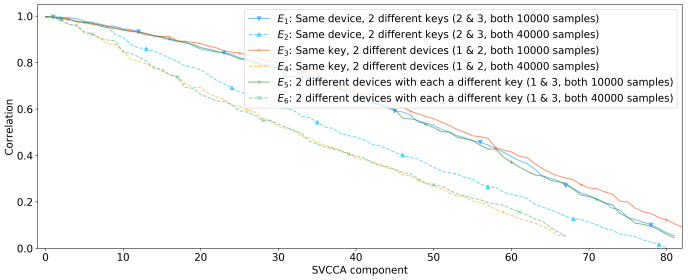
For the intermediate value model, we see the correlation is higher, especially for the smaller training set size. Additionally, the first fully connected layer needs fewer neurons than the second one. Again, we observe the problem reported by Bhasin et al. [1] that having too much training data can cause over-specialization in portability scenarios. As the correlation behaves similarly in both layers (while decreasing faster for the second layer), we can assume that only a single hidden layer would be sufficient. Finally, we can observe that the first fully connected layer tends to specialize more. This is aligned with the results for the MLP2 scenario where we also noticed middle layers to specialize more.



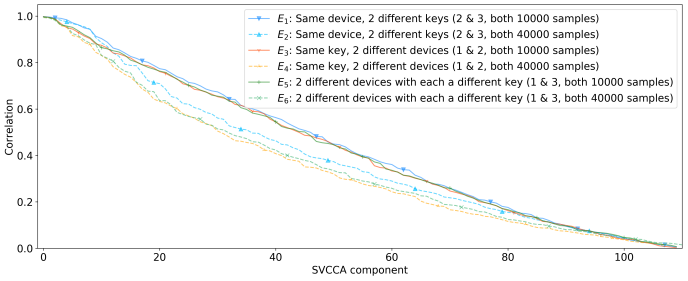
(a) CNN correlation in the Hamming weight model, the first fully connected layer.



(b) CNN correlation in the Hamming weight model, the second fully connected layer.



(c) CNN correlation in the intermediate value model, the first fully connected layer.



(d) CNN correlation in the intermediate value model, the second fully connected layer.

Fig. 8: Correlation results for CNN for the fully connected layers.

Based on the obtained results, we make the following observations:

- There is some common knowledge (shared inner representation) across networks that were trained on very similar data. We observe a similar level of common knowledge across the portability scenarios; it does not matter much whether the device, key, or both, are changed.
- When looking from a portability perspective, one should be careful not to train neural networks with too much data (leading to over-specialization of certain hidden layers). The SVCCA correlations decrease when networks are trained with more data, thus allowing to conclude about the needed number of training examples.
- SVCCA indicates that the middle layers (e.g., hidden layers 2 and 3 in MLP2) specialize more than the first and last hidden layers.
- SVCCA can indicate on the required number of layers or neurons.
- SVCCA components can be highly correlated with the class labels in the portability setting.

Finally, our results indicate certain advantages and disadvantages of SVCCA. Most importantly, it is not possible to use SVCCA (as a sole tool) to design a neural network for profiled SCA, but rather it can be used to give insights into the neural network's behavior. The main advantages of SVCCA are:

- The method allows comparing similarity across layers, independently of output shape and context (i.e., type of data processed by the neural network).
- It shows the largest possible correlation when the inputs are linearly transformed.
- It is invariant to affine transformations: no re-scaling or ordering the most important neurons is required.
- The method enables dissecting the similarity for particular samples or classes.
- SVCCA can indicate the required number of layers or neurons.

On the down side, SVCCA has the following shortcomings:

- It is difficult to interpret it: there is no formal relationship with the networks' performance, only the similarity is measured. Also, some context of other SVCCA outcomes (i.e., Figure 2) is required to be able to understand whether the correlation is meaningful.
- The method doesn't find non-linear relations: when changing the learned function (i.e., considering the mask or not), no significant correlation is found.
- SVCCA is computationally intensive when comparing convolutional layers, as the outputs have large dimensions.

5 Conclusions and Future Work

In this paper, we investigate how neural networks internally represent various side-channel data/settings. We use the SVCCA tool and show that there is common knowledge between various datasets. While this tool is far from perfect, it still provides us with a great deal of useful information. As an example, there seems to be more common knowledge between HW or intermediate

value models than when considering datasets with and without countermeasures. This indicates that while we can hope to use the same neural networks for the HW/intermediate value models, the same networks for both protected and unprotected scenarios will have a much more challenging task. Next, we observe how information about the class labels is also captured by SVCCA. Finally, we see how the information about the correlation for SVCCA components can help us in the design of the attacks by selecting the more appropriate number of hidden layers and the number of neurons, as well as the training set size. In future work, we plan to concentrate on SVCCA for convolutional layers as we believe this information would further help in understanding the dynamics of internal representation within the model.

References

1. Bhasin, S., Jap, D., Chattopadhyay, A., Picek, S., Heuser, A., Ranjan Shrivastwa, R.: Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis. CoRR (2019), https://eprint.iacr.org/2019/661.pdf
2. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without preprocessing. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **10529 LNCS**, 45–68 (2017)
3. Carbone, M., Conin, V., Cornélie, M.A., Dassance, F., Dufresne, G., Dumas, C., Prouff, E., Venelli, A.: Deep Learning to Evaluate Secure RSA Implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019, Issue**, 132–161 (2019). https://doi.org/10.13154/tches.v2019.i2.132-161, https://tches.iacr.org/index.php/TCHES/article/view/7388
4. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. LNCS, vol. 2523, pp. 13–28. Springer (aug 2002)
5. Das, D., Golder, A., Danial, J., Ghosh, S., Raychowdhury, A., Sen, S.: X-DeepSCA: Cross-Device Deep Learning Side Channel Attack. In: Proceedings of the 56th Annual Design Automation Conference 2019 on - DAC '19. vol. 1, pp. 1–6. ACM Press, New York, New York, USA (2019)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
7. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A Survey of Methods for Explaining Black Box Models. ACM Computing Surveys **51**(5), 1–42 (aug 2018). https://doi.org/10.1145/3236009, https://dl.acm.org/citation.cfm?id=3236009http://dl.acm.org/citation.cfm?doid=3271482.3236009
8. Hettwer, B., Gehrer, S., Tim, G.: Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery. CoRR pp. 1–17 (2019)
9. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(3), 148–179 (May 2019)
10. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research) http://www.cs.toronto.edu/~kriz/cifar.html
11. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks **3361**(10) (1995)

12. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 10076 LNCS, pp. 3–26 (2016)
13. Masure, L., Dumas, C., Prouff, E.: Gradient visualization for general characterization in profiling attacks. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 145–167. Springer (2019)
14. Moradi, A., Guilley, S., Heuser, A.: Detecting Hidden Leakages. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS. vol. 8479. Springer (jun 2014)
15. Morcos, A.S., Raghu, M., Bengio, S.: Insights on representational similarity in neural networks with canonical correlation. In: 32nd Conference on Neural Information Processing Systems (NIPS 2018). No. Nips, Montréal (2018). https://doi.org/arXiv:1806.05759v3, http://arxiv.org/abs/1806.05759
16. Perin, G.: Deep learning model generalization in side-channel analysis. Cryptology ePrint Archive, Report 2019/978 (2019), https://eprint.iacr.org/2019/978
17. Perin, G., Ege, B., Chmielewski, L.: Neural network model assessment for side-channel analysis. Cryptology ePrint Archive, Report 2019/722 (2019), https://eprint.iacr.org/2019/722
18. Picek, S., Heuser, A., Alippi, C., Regazzoni, F.: When theory meets practice: A framework for robust profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1123 (2018), https://eprint.iacr.org/2018/1123
19. Picek, S., Heuser, A., Guilley, S.: Profiling side-channel analysis in the restricted attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019), https://eprint.iacr.org/2019/168
20. Picek, S., Heuser, A., Jovic, A., Batina, L.: A systematic evaluation of profiling through focused feature selection. IEEE Transactions on Very Large Scale Integration (VLSI) Systems pp. 1–14 (2019). https://doi.org/10.1109/TVLSI.2019.2937365
21. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339
22. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 4095–4102. IEEE (2017)
23. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Report 2018/053 (2018), https://eprint.iacr.org/2018/053
24. Raghu, M., Gilmer, J., Yosinski, J., Sohl-Dickstein, J.: SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 6076–6085. Curran Associates, Inc. (2017)
25. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **5479 LNCS**, 443–461 (2009)
26. TELECOM ParisTech SEN research group: DPA Contest (4th edition) (2011), http://www.dpacontest.org/v4/index.php

27. Timon, B.: Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019, Issu**(2), 107–131 (2019). https://doi.org/10.13154/tches.v2019.i2.107-131, `https://tches.iacr.org/index.php/TCHES/article/view/7387`
28. Yu, T., Long, H., Hopcroft, J.E.: Curvature-based Comparison of Two Neural Networks. CoRR (2018), `http://arxiv.org/abs/1801.06801`