Side channel attacks for architecture extraction of neural networks

Hervé Chabanne¹² | Jean-Luc Danger² | Linda Guiga¹² | Ulrich Kühne²

¹Idemia, Paris, France
²Télécom Paris, Institut Polytechnique de Paris, Paris, France

Correspondence
Linda Guiga, Télécom Paris, Institut Polytechnique de Paris, Paris, France.
Email: linda.guiga@telecom-paris.fr

Abstract
Side channel attacks (SCAs) on neural networks (NNs) are particularly efficient for retrieving secret information from NNs. We differentiate multiple types of threat scenarios regarding what kind of information is available before the attack and its purpose: recovering hyperparameters (the architecture) of the targeted NN, its weights (parameters), or its inputs. In this survey article, we consider the most relevant attacks to extract the architecture of CNNs. We also categorize SCAs, depending on access with respect to the victim: physical, local, or remote. Attacks targeting the architecture via local SCAs are most common. As of today, physical access seems necessary to retrieve the weights of an NN. We notably describe cache attacks, which are local SCAs aiming to extract the NN’s underlying architecture. Few countermeasures have emerged; these are presented at the end of the survey.

1 | INTRODUCTION

Machine learning (ML) has become a major component in many research areas, ranging from image processing to medicine. One type of neural network (NN) in particular is present in our daily lives: the convolutional neural network (CNN). This model is the one most often used for image processing tasks such as classification [1]. CNNs have given rise to many applications on smartphones. For instance, Face ID is the CNN-based facial recognition feature of iPhone X [2].

Three main security issues affect NNs:

1. For CNNs to reach high accuracy, their architecture needs to be carefully selected. Moreover, their parameters need long training that requires intensive computational power. The trained model can therefore be considered as intellectual property. It is often patented and should be kept safe from potential competitors.

2. Protecting the privacy of users is paramount. This is especially the case because the input to such NN models can be sensitive data such as biometric data or medical records. Thus, it should be impossible for an attacker to recover the input of a model from its output or other model leaks.

3. The model’s output must not be tampered with. Adversarial attacks should be prevented as much as possible (see Section 5.4).

Unfortunately, NNs are subject to multiple attacks, which are described in this survey. Besides being unsafe, too often, NNs are insufficiently protected, as explained in Sun et al. [3]. This survey article focuses on side channel attacks (SCAs), which have emerged in recent years. So far, these attacks are of two kinds. The first is model inversion attacks, in which the attacker aims to recover the model’s input. The second is model extraction attacks, in which the attacker aims to recover the architecture and/or parameters of the victim model. As explained, the two types of attacks pose a serious threat for the user or designer of the NN.

Given the targeted model’s architecture, model stealing attacks that either recover the parameters of the victim NN or an equivalent NN with high accuracy are made easier. These attacks do not always aim to recover the exact parameters. They are generally not side-channel based, and most often rely on ML to achieve their goal [4–7]. Although they are out of scope in this survey, these attacks show how sensitive the architecture is. Once a potential attacker gets hold of the victim NN’s architecture, several attacks enable her to recover the full model.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. CAAI Transactions on Intelligence Technology published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology and Chongqing University of Technology.
We start with some basic facts about NNs in Section 2. We then introduce a typology of threats for these NNs in Section 3. SCAs are recalled in Section 4. In Section 5, we survey model extraction SCAs on NNs. Section 6 is about parameters and input recovery. Section 7 concludes the article.

2 | BACKGROUND ON NEURAL NETWORKS

NNs are a paradigm whose aim is to learn to recognize patterns. These patterns can then be used for classification purposes, among other tasks. In that case, given input $I$, the model either returns a label or a list of probabilities over all possible classes. When a label is returned, it corresponds to the class of $I$ with highest probability. NNs can be seen as graphs. In those graphs, the nodes are called neurons and the connections between neurons are called weights. The weights usually take a real value. NNs imitate how the brain works. Indeed, the NN’s parameters are optimized for a specific task through an iterative process called backpropagation.

NNs are organized in layers of different kinds:

- Fully connected (FC) layers: each neuron of an FC layer is connected to all neurons from the previous one. A bias, trained along with the weights, is added to each output element. Thus, given an input $X = (X_1, \ldots, X_n)$, weights $\{W_{ij}\}_{1 \leq i \leq m, 1 \leq j \leq n}$ and bias $\beta = (\beta_1, \ldots, \beta_m)$, the output of an FC layer is computed as:

$$O_i = \sum_{k=1}^{n} W_{ik} \cdot X_k + \beta_i$$

- Convolutional layers: A convolutional layer is characterized by one or several filters $F$. Each filter is usually a square matrix of weights that is convoluted with the layer’s input:

$$O_{ij} = \sum_{k=1}^{b} \sum_{l=1}^{w} X_{i \times s+k \times s+l} \cdot F_{k,l} + \beta$$

where $O$ is the layer’s output, $X$ is its input, $F$ is the current filter, $s$ is the stride of the convolution and $\beta$ is the bias. Each of these filters results in an output channel or featuremap.

- Batch normalization layers: The purpose of these layers, as stated in Ioffe and Szegedy [8], is to normalize the output of the previous layer. Their goal is to improve the efficiency and accuracy of the training.

- Pooling layers: The goal of the pooling layer is to reduce the dimensionality. The most common pooling layer is the max pooling. The latter divides the input into windows of a given size and computes the maximum in each window. Theoretically, the windows can be overlapping, although that is rarely the case.

These layers are usually followed by a nonlinear function called activation function. It aims to activate, or, on the contrary, deactivate some of the layer’s neurons. The most common activation function is $ReLU = \max(0, x)$. It zeroes out the negative neurons and activates the positive ones. $\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$ is another common activation function.

In image recognition, the output of the model is usually a list of probabilities for each possible output label.

Moreover, in CNNs, the input of some layers can be zero-padded. This padding is useful to prevent the dimension of the output from dropping. It also improves the performance of the CNN at hand.

Fully connected networks (FCNs) contain only FC layers. CNNs, on the other hand, are NNs mainly composed of convolutional layers.

Common NN models are composed of hundreds of layers and contain millions of parameters. For instance, ResNet-50 requires 23 million parameters [9], and AlexNet uses 61 million parameters [10].

2.1 | Common neural network setups

We consider two main setups for NNs. First, users can have the model on their mobile phones. Second, the model can be hosted by a cloud provider. For instance, the predictions provided as a service are in the latter category, which falls in the ML as a service (MLaaS) context.

3 | THREAT SCENARIOS

NNs are not designed to be secure against cryptographic attacks. The nonlinearity of activation functions and pooling layers is the only barrier preventing the attacker from recovering the model’s input given its output. This gives rise to multiple attacks targeting various aspects of NNs. Among those are SCAs. They exploit information leaks such as time measurements or memory access patterns to recover the model’s architecture or its parameters.

In this survey, we consider three kinds of threat scenarios:

- Model (architecture) extraction;
- Parameters (weights of the NNs) recovery; and
- Input recovery.

We focus on model extraction attacks, which are described at length in Section 5. Attacks related to the two other threat scenarios are briefly recalled in Section 6.

In all considered cases, the attacker is passive, meaning that she only observes the computation activity through side channels. She does not inject faults into the victim model. Such active attacks exist [11,12] but are not the subject of this survey.

In terms of model extraction, Jagielski et al. [13] define three goals for a potential attacker. In all cases, the attacker wishes to extract a model $f$ close to original model $f$. 
Functionally equivalent extraction: In this case, \( \hat{f} \) should be such that \( \hat{f}(x) = f(x) \) for all input \( x \).

Fidelity extraction: Here, \( \hat{f} \) should be such that for some similarity function \( S \), \( Pr[S(\hat{f}(x), f(x))] \) is maximal. For instance, the similarity function can be label agreement.

Task accuracy extraction: In this context, the goal of the attacker is to maximize the extracted model's accuracy. Thus, denoting real classification \( y \), \( \hat{f} \) should maximize \( Pr[\hat{f}(x) = y(x)] \) for all input \( x \).

These evaluation notions are recent. In this survey, we mainly focus on functionally equivalent extraction attacks. Indeed, most SCAs aim to recover the exact parameters and hyperparameters of the victim NN. However, some attacks that seek to find the correct hyperparameters train a set of possible architectures and select the one with the highest obtained accuracy. This phase of the attacks is closer to task accuracy extraction. This is the case for Cache Telepathy, for instance [14].

4 | SIDE CHANNEL TECHNIQUES

4.1 | Side channel attacks: generalities

SCAs were introduced in the 1990s, taking advantage of involuntary leakages of cryptographic implementations. The processing of leaked data often involves statistical analysis [15] or even deep learning techniques [16]. A nonfunctional physical side channel such as execution time, power consumption, or electromagnetic emanations is exploited at this aim. Generally, SCAs require physical access to the targeted circuit, such as a smartphone [17]. These physical access SCAs were extended from smartphones to more powerful devices such as smartphones. The latter are also a target of choice for NN implementations. Timing attacks, which rely on the time measurement of the targeted circuit, can be carried out remotely. For instance, a server on the local network [18] can be targeted. Local access SCAs, in which the attacker is located on the same machine as her victim, sharing with her some resources such as cache memory, are more powerful. This corresponds to a popular setup in which an NN is hosted by a cloud.

We now recall some elements on cache attacks. Here, we present the Flush+Reload and Prime+Probe cache attacks. Other variants exist, such as Flush+Flush [19].

4.2 | Cache attacks

The goal of cache attacks is to determine whether a certain function has been called by the victim model within a certain period of time.

The cache is a small but fast memory space used to store data that have been recently accessed in general purpose or bigger embedded processors. It is usually divided into three parts corresponding to three hierarchical levels: L1, which is close to the processor (CPU), L2, and the last level cache (LLC). Usually data residing in L1 is also included in L2, which in turn is included in LLC. When a process tries to access an address, the CPU first looks for it in L1. If it is not there, it looks in L2. If it still cannot find it, it goes on to search the LLC. Finally, if the address is not in the cache, it needs to access the much larger main memory. If a process successfully retrieves data stored in the cache, this is called a cache hit. On the other hand, if the address is not in the cache, and therefore is stored in the main memory, it is a cache miss.

Given the way the cache works, the execution time of a memory access depends on whether the target address is in the cache. Cache attacks are based on this induced time difference.

Whereas L1 and L2 caches are usually only shared among processes within the same core, the LLC is shared by all processes independently of the core. Moreover, the LLC is often divided into sets called cache sets.

4.2.1 | Flush+Reload [20]

The Flush+Reload attack consists of three steps:

1. Flush cache line, which contains the data, corresponding to the target address (flush), such that the data is evicted from the cache and can be found only in the main memory.
2. Wait for a certain period of time, leaving the victim enough time to call the targeted function.
3. Access the target address and measure the access time (reload).

If the access time is short (i.e., below a threshold to determine), the address is in the cache. This means the victim has called the corresponding function (Figure 1a). A high access time, on the other hand, means that the CPU had to load the address from the memory. Thus, the victim has not accessed it between the flush and the reload (Figure 1b).

Flush+Reload works on the LLC, which is shared by all processes even from different cores. However, each process has its own virtual address space. Therefore, it can have different virtual addresses for the same function. Thus, this attack requires page sharing: when two processes use the same read-only memory pages, these pages are shared. This is implemented in many systems for efficiency purposes, in particular for shared libraries. Moreover, to flush a memory line from the cache, one needs the clflush x86 instruction or equivalent.

4.2.2 | Prime+Probe [21]

In fact, Flush+Reload is a variant of a previous attack, Prime+Probe, which is slower but does not rely on particular shared memory requirements. Once again, the attacker targets the LLC. Because there is no page sharing, the attacker does not know to which cache location the target instruction is mapped.

The first step of this attack is therefore to find the correct cache set: a set of cache lines. We will not detail this step here.
Once the correct cache set has been recovered, the attacker can find out whether the victim accessed a certain address:

- Find the target cache set.
- Fill the cache set with the attacker's data, thus evicting the target from the cache.
- Wait for a certain period of time, leaving the victim enough time to call the target address.
- Access the attacker's data again.

If the access time is short, the attacker's data has not been evicted. This means that the target has not been loaded back into cache memory (Figure 2b). A high access time means that the attacker's data has been evicted from the cache, showing that the victim has accessed the target memory line (Figure 2a).

Thus, the two cache attacks described here enable an attacker to monitor accesses (by the victim) to some target addresses. Therefore, she can determine whether a certain function has been called by the victim. For instance, she can track specific routines executed for NN computations. We will show how these cache attacks enable the attacker to recover the NN architecture of a victim model, as detailed in Section 5.

5 | ARCHITECTURE EXTRACTION

In this section, we detail the main attacks regarding model extraction.

Once the architecture is known, equation-solving attacks [5] and model extractions based on the piecewise linearity of Rectified Linear Unit (ReLU) activation functions [22] become possible. These provide the attacker with the model's weights. Thus, recovering the architecture yields an advantage and can be the first step in other attacks.

Today, most SCAs target the architecture of a victim NN and manage to recover at least part of it. First, let us detail the hyperparameters of an architecture that an attacker aims to recover:

- The number of layers or depth of the model, as well as the layer types.
- The activation functions.
- The number of neurons in a given layer.
- The number of feature maps in a given layer.
- When there is one, the filter size.
- The stride of a convolution.
- The input padding.
- The pooling size and type.
- The connections between layers. For instance, an input might be fed to two branches whose outputs are then summed.

An example of a CNN can be seen in Figure 3. In this CNN, there are two convolutional layers, two max pooling layers (of size 2 × 2) and three FC layers. The convolutional layers and the first two FC ones include a ReLU activation function. The convolutional filter sizes are all 5 × 5, and the stride is 1. There is no padding. Figure 3 also shows the input feature map dimensions for each layer in the form \( nb\text{-featuremaps} @ \text{input_width} \times \text{input_height}. \)

Cache attacks for model extraction are detailed in Section 5.1. A remote timing attack is then described in Section 5.2. A memory access pattern attack is detailed in Section 5.3. Section 5.4 describes two physical access attacks. According to our SCA classification, attacks of Sections 5.1 and 5.3 are local, the one in Section 5.2 is remote, whereas Section 5.4 calls for physical SCAs.

Finally, let us also mention Naghibjouybari et al. [25], in which the training step of the NN is attacked, and which requires colocation of the victim and the attacker on the same graphical processing unit (GPU).
5.1 Cache attacks

Let us describe three cache SCAs in an MLaaS setting, as defined in Section 2.1. In this setting, the attacker is located on the same machine as the victim and shares its cache. This is a typical case of a cloud environment.

One of them [26] recovers the number of layers as well as their types. The other two [14,27] go further and determine the remaining hyperparameters.

Assumptions: The attacker shares the victim's ML frameworks. This is a realistic assumption, because most ML frameworks, such as Tensorflow, are open source. Moreover, because they are cache attacks, colocation is necessary: the attacker and the victim processes have to share the cache. Fortunately for the attacker, several papers have explained how to achieve colocation in an MLaaS context [28,29].

5.1.1 Cache telepathy

Cache Telepathy [14] exploits a leak in matrix multiplications to recover most hyperparameters. Most ML frameworks, such as Tensorflow, use an efficient matrix multiplication algorithm, generalized matrix multiply (GeMM), for FC and convolutional layers. FC layers can easily be written as a matrix multiplication between matrix of weights $W$ and input vector $I$. However, the input and filters of convolutional layers can also be reshaped so as to turn the layer into a single matrix multiplication. Each convolutional window is flattened to form a column in input matrix $I$. Each filter is flattened to form a row in weights matrix $W$. This operation is detailed in Figure 4.

Figure 4 shows that the following hyperparameters can be deduced from the dimensions of the matrices involved in a matrix multiplication:

- The input and output's width and height.
- The number of layers in the layer.
- The filters’ size.

Thus, recovering most of the layer's hyperparameters essentially boils down to extracting those dimensions. Let us now explain how this is achieved.

Yan et al. [14] observed that three operations, kernel, itcopy and oncopy, form an easily identifiable pattern in the algorithm, containing loops. The number of iterations in those loops, as well as the number of repetitions of the pattern, are closely related to the dimensions of the matrices at hand. The attacker monitors these operations using Flush+Reload (if page sharing is available) or Prime+Probe (see Section 4.2). This monitoring is enough for the attacker to recover the matrix sizes she needs.

The attack can therefore be carried out as follows:

- Determine the number of layers by counting the number of matrix multiplications.
- Determine the input, output and filter dimensions in each layer by attacking GeMM.
- Identify the activation functions by monitoring a probe address in the $\text{sigmoid}$ activation function (see Section 2) using Flush+Reload (or Prime+Probe).
- Reduce the number of possible connections by measuring inter-GeMM latency and by using dimensional constraints.
• Reduce the possibilities for stride and padding values, as well as pooling layers thanks to dimensional constraints.

Because of the constraints and the recovered matrix sizes, the attacker manages to reduce the number of possible architectures to a small set.

For instance, for a VGG-16 model, the search space was reduced to 16 possibilities, when it was initially larger than $4 \times 10^{35}$ (given the number of layers and their type).

5.1.2 DeepRecon

Although the previous attack recovers most hyperparameters, it cannot be launched on victims using the GPU for its computations [26]. Indeed, GeMM is not always used in GPU computations.

Hong et al. [26,27] used the Mastik toolkit [30] to perform Flush+Reload attacks (see Section 4) on the victim NN. As explained in Section 4, Flush+Reload requires page sharing. In Hong et al. [26], the authors monitored all functions that intervene in NN computations. These include ReLU computations, max pooling, convolutions, gradient computations, and matrix multiplications. The attackers can target both the training and the inference phases of the victim model. This gives the attacker the sequence of layers in the model, as Figure 5 shows. An analysis of the output provides the attacker with the overall architecture of the model, given its family of NNs (for instance, VGG [31] or ResNet [9]). Moreover, counting the number of matrix multiplications gives the attacker an approximation of the number of parameters in each layer, and therefore of the filter sizes.

5.1.3 How to Own NAS in your spare time

The previous attacks managed to narrow down the architecture to a small set of possible architectures. However, in Hong et al. [26], the attacker needs to know the family of NNs to which the victim belongs. With no prior knowledge about that family of NNs and for more complex architectures, DeepRecon and Cache Telepathy can end up with a search space that is still too large to explore.

Some tasks require specifically tailored architectures. They need novel architectures, which would therefore mitigate the attacks studied so far. Furthermore, when the number of possible architectures is reduced to a small search space, figuring out the correct layout among those requires a training dataset. Such a dataset is not always accessible to the attacker.

Given only some knowledge about how NNs are constructed, Hong et al. [27] mixed cache attacks with a timing attack to reconstruct the computational graph completely: that is, to find all hyperparameters, including the connections between layers, of the target model.

Their attack can be split in five steps:

1. Launch the attack from Hong et al. [26] to get the sequence of layers.
2. Filter out the noise from those traces.
3. Find all possible values for filter sizes, padding, stride and pooling owing to some constraints.
4. Profile execution time for the selected possibilities.
5. Narrow down the correct architecture given the timing profiles.

Step 3 considers most values used in NNs so far. The values are filtered out because of the number of convolutions or matrix multiplications, for instance. These can provide the input and output sizes of certain layers and help to eliminate some values for the filters, strides, pooling and padding. Such constraints can limit the number of timing traces the attacker needs to collect in step 4. Because the execution time is correlated with layers’ hyperparameters, the attacker can narrow down the correct architecture by comparing the model’s traces with the tested ones.

5.1.4 GANRED [32]

A recent cache side channel [32] manages to recover the full architecture of a target NN with shared libraries being disabled between the victim and attack processes. This makes the attacker in Liu and Srivastava [32] stronger than in Hong et al. [27].

Assumptions: The attacker knows a large search space to which the target NN belongs, but no other information about the targeted NN. She also knows the framework used for the victim NN, but does not have access to the library code. The latter limitation prevents the Cache Telepathy and DeepRecon attacks [14,26]. Contrary to previous attacks, she does not require shared library access.

Liu and Srivastava [32] rely on Prime+Probe rather than Flush+Reload to carry out their attack. This is why shared libraries are not required. To recover the architecture, the attacker proceeds layer by layer. The first step consists of collecting $n$ cache traces of length $p$ for the victim NN. For layer $l$, she carries out the following steps:
• Select possible hyperparameters for layer \( l \) and construct an adversarial model with \( l \) layers. For layer \( l \), use the selected hyperparameters. For previous layers, use the extracted hyperparameters.
• Initialize the layer with random weights.
• Collect \( n \) traces of length \( p \) for the current architecture.
• Compare the newly collected traces with the targeted NN's.
• Validate the layer's hyperparameters.

For each layer, the attacker compares the average number of hits in segments from the attacker and the victim's models. The hyperparameters with the traces closest to the original model are selected.

In the last step, the validator then filters out false positives using execution times. When a new layer is added to the adversarial model, a certain number of clock cycles are added to the execution time. The validator compares the added number of clock cycles with the actual time for which the victim and adversarial model traces match. The latter corresponds to the number of clock cycles actually added to the observed traces. If the clock cycles do not match, the validator considers the selected hyperparameters to be the wrong ones even if the electromagnetic (EM) traces are the closest to the targeted NN's. The attacker then selects the second best set of hyperparameters and tries to validate it, and so on. The attack stops when the victim and adversarial traces match in their entirety.

The main limitation in this attack is that for each layer, all sets of hyperparameters need to be monitored. However, the targeted NN only needs to be monitored for a constant number of traces (the authors collect 50 traces). Moreover, the attack remains linear in the number of layers, even though the architecture search space increases exponentially in that number.

Contrary to Yan et al. [14] and Hong et al. [26], GANRED recovers the correct architecture instead of a smaller set of possible ones. Furthermore, it does so with a weaker attacker than the one in Hong et al. [27].

5.2 Remote timing side channel attack

Let us now consider a timing SCA in an MLaaS (see Section 2.1) context.

Assumptions: The attacker only has oracle access to the model. She can, however, measure the execution time. Here, the attacker is assumed to know the distribution of the victim model's training dataset. Because datasets can be confidential, this is a strong assumption.

In their paper, Duddu et al. [33] succeed in recovering the architecture, relying only on the execution time and the training set's distribution. Not only do the authors predict the number of layers as well as the layer types, they manage to determine the number of filters, their sizes and their stride in each layer.

The execution time is proportional to both the number of multiplications and the depth of the victim model: that is, to the number of layers in the model. The authors also observe that in a given layer, the execution time increases linearly with the number of filters and the filter size, and it decreases linearly with the stride.

Furthermore, Duddu et al. [33] note that owing to efficient parallelization, increasing the number of neurons in a given layer does not actually lead to a notable increase in execution time. Thus, the execution time traces gathered are used only to determine the depth of the victim model.

Once the traces have been gathered, ML models are used to recover the architecture. Thus, a first ML model, called regressor, is trained to predict the depth of the model. A second ML model, called RNN, then tries to predict the remaining hyperparameters. To train those ML models, two datasets are created:

• The first dataset, \( S_1 \), is a set of execution times labeled with the corresponding architecture's depth.
• The second dataset is a set of pairs, \( S_2 = \{(xf(x))\}_{x} \), where \( f \) is the victim model and \( x \) are randomly sampled inputs with the same distribution as the victim's training dataset.

The first dataset is created once and for all, and the regressor can directly be trained. \( S_2 \) on the other hand, is specific to the victim model. With these two datasets at hand, the attacker can proceed as follows:

• First, she uses the regressor trained on \( S_1 \) to recover depth \( k \). Thus, search space \( SP \) is reduced to models with depth \( k \).
• The RNN then tries to recover the correct architecture by predicting which architecture in \( SP_k \) gives the outputs closest to the victim model's. \( S_2 \) is used to measure this closeness.

The attack has been tested on models with up to 13 layers, with an accuracy of around 85%.

This attack, which relies heavily on using ML to predict the architecture based on timing traces, assumes that the attacker has access to the victim model's training distribution. This is often not the case; the other attacks presented here show that it is not necessary if other attack vectors are targeted.

As mentioned in Duddu et al. [33], the ML approach using RNNs described in this attack can also be applied to predict the correct architecture from the reduced search space obtained through Cache Telepathy [14] and DeepRecon [26].

5.3 Memory access pattern attack

In Hua et al. [34], the authors detail two attacks on specialized hardware designed to accelerate NN computations: NN accelerators. The first recovers the architecture of the victim model. The second recovers its weights, given the model's architecture. We consider only the first one in this section.

The authors consider the computations of the victim model to be taking place in a protected hardware accelerator.

Assumptions: This attack assumes:

• The victim architecture belongs to a known set of possible architectures.
• The attacker can observe all read-and-write (RAW) memory access patterns.

The attack methodology in Hua et al. [34] is:

• Recover the layer boundaries and connections owing to RAW traces.
• Measure execution times for each layer.
• Based on the execution times, RAW traces and constraints on the dimensions, extract each layer's possible hyperparameters.

For the first step, the authors identify the memory locations that are first written and then read. This provides them with the layer boundaries because a layer's output is the next layer's input. Within a given layer, filter values are only read. RAW traces thus enable the attacker to determine the location of filter values and therefore deduce the filter size. This is an example of how RAW traces are used in the third step to determine hyperparameters.

Constraints on the dimensions include constraints on the pooling and padding sizes. For instance, the padding needs to be smaller than the filter size. Execution times are proportional to the number of computations in the layer, which helps to determine the hyperparameters.

Because of the RAW traces and the constraints, the attacker in Hua et al. [34] is able to reduce the number of possible architectures for the targeted NN.

5.4 | Physical access attacks

5.4.1 | DeepEM

Yu et al. [35] exploit electromagnetic emanations to attack a model on an NN accelerator.

Assumptions: The attacker can query the model with the inputs of her choosing. She can also collect multiple electromagnetic traces from a microprocessor. Moreover, the attacker knows a set of architectures containing the victim NN.

DeepEM targets binarized neural network accelerators, which are NNs that use binary weights and activation values, and thus take up less memory space.

The authors observe that the EM traces, when observed in the time domain, are proportional to the number of parameters in the considered layer. The time domain is used according to Tramèr et al. [5], because it links a layer's parameters with the execution time of its sequential computations. For each layer type, Yu et al. [35] determine the number of parameters depending on the layer's hyperparameters. Because EM traces are closely linked to that number of parameters and the computations executed, they can use this to identify the layer types based on the EM signatures.

To reduce the set of possible architectures for the victim model, the attacker proceeds as follows:

• Collect multiple EM traces over various inputs and observe them in the temporal domain.

• Split each EM trace to obtain the depth and layer boundaries of the victim NN.
• For each layer, identify the layer types based on the EM signatures.

Once this is achieved, several hyperparameters remain to be determined. For filter sizes and padding, the authors select the values among a set of possible ones. Using equations that follow from general constraints on the structure of NNs, the authors manage to reduce the search space for the victim NN.

The equations used are the same as in Hua et al. [34].

For instance, using 10,000 EM traces, they manage to limit the number of possible architectures for VGGNet to 17, even though the original model has 23 layers. The correct architecture is then determined by training the small set of architectures at hand and selecting the one with the highest accuracy. This corresponds to the method used in Hua et al. [34]. The training does not need to be long, as the goal is not to retrieve the actual weights but to determine which architecture is closest to the original one.

Once the architecture is extracted as a result of the EM traces, the attacker can recover the weights and biases of the victim NN using adversarial active learning. In active learning, a model is trained with a set of unlabeled data. The labels taken for training are another model's predictions. In adversarial active learning, the unlabeled data selected are adversarial examples for the queried model.

This attack manages to recover models that approximate the original one closely. Indeed, upon testing the recovered architecture for a ConvNet model [36], the original and the extracted models agreed 98.95% of the time for the CIFAR test set [37].

Although this attack does not recover the exact architecture as in Liu and Srivastava [32], for instance, it is the first that only uses EM emanations to attack large-scale NN architectures.

5.4.2 | DeepSniffer

In the last attack of this section, DeepSniffer [38], the authors do not assume prior knowledge about the architecture. To recover the hyperparameters, they rely only on architectural hints such as memory access patterns given by a victim running in a GPU. This attack does not aim to recover the precise dimensions in the architecture, such as filter sizes or stride values. However, the authors prove that the extracted NN still facilitates adversarial examples (cf. [39] for a survey on this kind of attack, which is different from the ones we consider here, because it is of the ML versus ML type). In such attacks, an attacker can fool the model for a set of inputs while keeping the rest of the predictions unchanged [40].

Assumptions: The attacker does not need to know the ML frameworks or software used by the victim. However, the attacker requires physical access to the system.

Two attack scenarios are considered:
- Side channel context: Electromagnetic emissions are traced so as to recover the volume of memory RAWs, the execution time and cache hits and misses.
- Bus snooping context: More precise RAW traces are available to the attacker owing to bus snooping: she can obtain the addresses as well as the dataflow volume. The execution time and cache misses or hits can also be observed in this scenario.

Whereas the first scenario only enables the recovery of chained ML topologies (i.e. where the layers are executed one after the other), more complex ones can be extracted in the second case. This attack is learning-based: ML models are used to predict the correct architecture owing to the collected architectural hints.

The attack methodology is as follows:

- Predict the layer types and sequence based on the collected architectural hints, with the help of universal rules regarding how NNs are constructed. An example of the latter is that an FC layer cannot directly follow a convolutional one, because two linear layers need to be separated by a nonlinear one.
- Reconstruct the interlayer connections. In the first scenario, this can be achieved only for chained topologies. In the second scenario, RAW addresses are used to recover more accurate predictions and extract more complex topologies. The reconstruction is based on the fact that featuremap values are first written and then read. Furthermore, no matter the topology, featuremap data introduces such RAW patterns with high probability.
- Set the convolutional strides to 1 and the number of input channels to 3, and randomly select the set of dimension sizes (input, output, filters, and padding) that respect some dimensional constraints. This results in an approximation of the dimensions.

For the first step, the accuracy of the sequence recovered is measured as a result of the edit distance between the extracted sequence and the victim’s. The results on common NN families are summarized in Table 1. AlexNet and VGG are chained topologies, whereas ResNet and NasNet are not. For instance, ResNet introduces shortcuts: an input goes through two branches whose outputs are summed.

For all layer types except FC ones, the input and output dimensions are deduced correctly with an accuracy close to 100% in both scenarios. For FC layers, this accuracy remains greater than 80%.

Finally, the authors carry out adversarial attacks using the extracted model and prove that adversarial attacks success rate goes from 18.1%–43% (depending on the original model) to 75.9%. Thus, although the extraction seems incomplete, it makes the attacker much stronger, because the acquired knowledge enables her to carry out adversarial attacks easily on the victim model.

### 5.5 Summary and countermeasures

Table 2 summarizes the architecture extraction attacks studied in this survey. The number of queries for the considered attacks is constant. Moreover, the inputs fed to the targeted model are not crafted, and the attacker does not need to query the model directly. She only needs to monitor other users' queries. Each query provides the same information to the attacker. However, several runs are useful to filter out the noise. The number of queries mentioned in the papers considered corresponds to the authors’ setup and might differ on other platforms, depending on the amount of noise.

Because these SCAs are based on the observation of one or several independent runs of the input, the number of required queries is much less important than equation-based attacks. For instance, let us consider Carlini et al. [22], which is a mathematical attack on NNs. It is the first attack that successfully recovers functionally equivalent NNs with more than two hidden layers. This efficient attack still requires at least $2^{16}$ queries for FC networks with two hidden layers. Yet, at most 10,000 queries suffice on much larger networks in the SCA case.

ML-based model stealing attacks [4–7] require fewer queries than Carlini et al. [22]. ML-based attacks are generally fidelity extraction ones, which also explains the lower number of queries. So far, Yu et al. [4] is the most efficient one query-wise. It requires 1500–3000 queries to the target model to achieve an accuracy similar to that of the victim model. The number of queries depends on the architecture and the NN’s task. It uses adversarial samples to make the attack more efficient. Considering these attacks select the substitute model from a preestablished search space, and that the number of queries is not constant depending on the NN, SCAs, with a constant number of queries, remain advantageous.

In all cases, we consider only queries to the target model. Queries to substitute models are not taken into account.

Although NNs have been the target of various architecture extraction attacks, few countermeasures have been proposed. Wang et al. [41] proposes an NN accelerator that mitigates timing SCAs. It only encrypts some featuremaps, which makes it harder for an attacker to determine the execution time of a given layer.

Dubey et al. [42] claims to introduce the first physical side channel countermeasure for NNs. It provides masking hardware protection for architectures with binary weights against power-based SCAs. The authors then introduced an improvement in their hardware masking, BoMaNet [43]. The latter incurs less delay and makes the method glitch-resistant.

Liu et al. [44] aim to prevent recovery of the architecture through memory access patterns attacks. Their method first
| Attack                                           | Assumptions                                                                 | Access type | Side channel | Victims tested                     | Results                                                                 | Limitations                                                                 | Queries |
|-------------------------------------------------|------------------------------------------------------------------------------|-------------|--------------|------------------------------------|-------------------------------------------------------------------------|-----------------------------------------------------------------------------|---------|
| **Cache attacks**                                |                                                                              |             |              |                                    |                                                                         |                                                                             |         |
| Cache Telepathy [14]                             | Colocation, page sharing, use of GeMM, ML framework known                   | Local       | Cache        | VGG, ResNet                        | Set of possible architectures (including hyperparameter dimensions)     | Not applicable to GPUs; the set can be too large if the architecture is complex | Not specified |
| DeepRecon [26]                                   | Colocation, known family of architectures, page sharing, ML framework known | Local       | Cache        | VGG, ResNet, DenseNet, InceptionV3, InceptionResNet, Xception, MobileNet | Number of layers, layer types, and approximation of the hyperparameter dimensions | Cannot recover novel architectures, since it relies on the knowledge of the family of NNs | 10      |
| How to own NAS in your spare time [27]           | Colocation, page sharing, ML framework known, some knowledge about how NNs are constructed | Local       | Cache        | MalConv, ProxylessNAS              | Full computational graph                                                |                                                                             | 1       |
| GANRED [32]                                      | Set of possible architectures, ML framework, shared LLC                      | Local       | Cache        | AlexNet, VGGNet                    | Full architecture                                                       | The attacker tries all possible hyperparameters for each layer               | 50      |
| **EM attacks**                                   |                                                                              |             |              |                                    |                                                                         |                                                                             |         |
| DeepEM [35]                                      | Known set of architectures, physical access                                | Physical    | EM emanations| ConvNet, VGGNet                    | Recovers layer boundaries, depth and layer types. This leads to a set of possible architectures for the victim NN. An ML approach recovers the weights. | Several hyper-parameters cannot be determined with EM emanations only. Training several architectures is still necessary, even if accelerated thanks to adversarial samples | 10,000  |
| DeepSniffer [38]                                 | Only universal knowledge about how NNs are constructed                      | Physical    | EM emanations| AlexNet, VGG, Resnet, Nasnet_large | Full computational graph                                                |                                                                             | 1       |
| **Other attacks**                                |                                                                              |             |              |                                    |                                                                         |                                                                             |         |
| Remote timing side channel attack [33]           | Distribution of the training dataset, precise execution time measurements   | Local/remote| Timing       | LeNet, AlexNet, VGG, ResNet        | Number of layers, their type and the hyperparameter dimensions          | Requires precise measurements; the distribution of the training set can be private | 20      |
| Memory access pattern attack [34]                | Known family of architectures, ability to monitor all RAW memory pattern accesses | Local       | Memory access pattern | AlexNet, SqueezeNet, LeNet, ConvNet | Smaller set of possible architectures (including hyperparameter dimensions) | Activation functions cannot be determined through this attack. The deduced set can be too large if the architecture is complex | Not specified |

Abbreviations: GeMM, generalized matrix multiply; GPU, graphical processing unit; LLC, last level cache; ML, machine learning; NN, neural network; RAW, read-and-write.
6 | PARAMETERS AND INPUT RECOVERY

Our survey focuses on side channel architecture extraction attacks. However, various SCAs also target the model's parameters or its input. In this section, we provide a short presentation of the existing attacks in these threat models.

6.1 | Attacks

Full-model extractions require physical SCAs. There are two possible approaches when reverse-engineering a victim model entirely. The first consists of extracting the architecture and then using the extracted features to reveal the weights (or parameters) [34]. In the second, the attacker recovers the hyperparameters and parameters at the same time [47,48]. Hua et al. [34] present two attacks. The first recovers the architecture, as detailed in Section 5.3. The second recovers the weights by exploiting the pruning of weights: zeroed-out weights are not considered for efficiency. This second attack uses memory access patterns to determine weights that have been zeroed out. Then, by crafting inputs that zero out some weights, the attackers solve layer equations to determine the weight values up to the bias. Thus, contrary to the other model extraction attacks in this survey, the attacker needs to query the targeted model herself.

The amount of information these attacks recovers varies greatly. Whereas Hua et al. [34] manages to recover weights up to a scalar when pruning of the weights takes place, Xiang et al. [47] only recovers the weights sparsity. The latter means that the attacker recovers the proportion of zeroes in the weights. Furthermore, in Xiang et al. [47], the attackers consider the model to belong to a set of possible pretrained models, with a certain sparsity. To recover both the architecture and the weights, Xiang et al. [47] collect power traces for all possible architectures, including fine-tuned ones. These power traces depend strongly on both the architecture and the sparsity of the model. They can therefore use these power traces to train a classifier whose goal is to return the victim model's architecture along with its sparsity. This corresponds to a unique model among the pretrained ones, providing the attacker with the weights. Because the victim model is supposed to be a pretrained one, the attacker in Xiang et al. [47] is stronger than in most others considered in this survey. However, it is a power only SCA.

On the other hand, Batina et al. [48] extracts the exact weights through an EM and power SCA. It operates iteratively. First, the attacker distinguishes the collected traces for each neuron. This is easily done, because each neuron's computations has an identifiable power trace. Then, a statistical tool for power traces is required: correlation power analysis (CPA) [15]. Let us detail this tool. Given power traces of a model over many inputs, an attacker extracts a secret:

- Select a part of the target algorithm where the secret is used.
- Simulate the power consumption of that subalgorithm, depending on the input and the secret.
- For each guess of the secret, compare the simulated power consumption with the actual consumption.
- Select the guess closest to the actual consumption.

For each layer, the attacker determines the neurons that belong to it and their associated weights using a CPA. To achieve that, she targets each neuron multiplication and selects the weights and layer that maximize the correlation to the actual power consumption and EM emanations.

The parameter extraction attacks are summarized in Table 3.

A model's input can be considered sensitive information, such as in a medical context. This is why the input is also the victim of several attacks. The attacks published so far require the knowledge of the weights and architecture of the victim NN. Besides extracting the full model, Batina et al. [48] show that a CPA-based approach similar to their model extraction attack enables an attacker to recover the input. Moreover, their model inversion attack requires only one EM and power trace to be collected. However, this is possible only if the model has previously been recovered.

Another physical SCA is given in Wei et al. [49]. It is a power SCA. To attack convolutional filters, the authors build power templates for inputs with the shape of the filter. A complex process of input value selection that we will not detail here leads to the extraction of values with a power consumption close to the observed power traces.

Finally, Privado [50] exploits memory access patterns in the context of a cloud provider with a protected environment, such as Intel's SGX [51], and an untrusted operating system. Its goal is to recover the model's input. More precisely, assign-or-nothing patterns are targeted in Privado. Some parts of the target algorithm contain input-dependent branches. Some of these branches execute some action if a certain condition holds, and do nothing otherwise. This behaviour is called assign-or-nothing. In NNs, several activation functions and some pooling layers showcase such behaviour.

6.2 | Countermeasures

There have been protections of cryptographic algorithms against SCA for more than two decades, but their transpositions to NNs face various challenges. Typically, cryptographic keys for symmetric encryption are on the order of hundreds of bits whereas there are millions of parameters to be protected for modern NNs. Moreover, smartcards benefit from dedicated hardware security features to thwart SCAs.
### TABLE 3 Parameter extraction attacks

| Attack                        | Assumptions                                      | Side channel | Victims tested                                      | Results                                                                 | Limitations                                                                 |
|-------------------------------|---------------------------------------------------|--------------|----------------------------------------------------|------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| Memory access pattern attack  | Known architecture. Ability to query the model. Weight pruning occurs. | Memory access patterns | AlexNet, SqueezeNet, ConvNet | Weights as a function of the bias. If the pruning threshold can be controlled, the bias can also be recovered. | Requires the ability to control the input. The weights cannot be recovered exactly if the pruning threshold cannot be controlled. |
| Open DNN box                   | Known set of pretrained models. Redundant weights are set to 0. | Power        | AlexNet, ResNet, Inception, MobileNet | Architecture and sparsity, thus identifying correct weights among pretrained ones. | The assumption that the model and weights are among a set of pretrained ones is strong. |
| CSI NN                        | Ability to collect EM emissions and power traces. Considered models are FCNs. | Power and EM  | FCN | Number of layers, number of neurons per layer and precise weights. | The attack has only been tested on FCNs with few layers. |

Abbreviation: FCN, fully connected network.

McKeen et al. [52] introduce a hardware-based solution to protect NNs on mobile devices, relying on trusted execution environments. More precisely, the Sanctuary architecture [53] enables mobile user-space enclaves, similar to those provided by SGX, in TrustZone's normal world on ARM platforms. Leveraging on these Sanctuary user-space enclaves, McKeen et al. [52] shows how to execute NN inferences in this hardware-protected environment.

Some works consider low-overhead hardware protections of NN inputs and weights [54,55]. In Hua et al. [54,55], the architecture of the NN is known by the attacker. Attacks on weights such as Hua et al. [34] are mitigated by these protections. However, other physical attacks (using power consumption or EM emanations) are not dealt with by those protections.

Regarding model extraction, only a few architectures are currently deployed, up to fine-tuning, among deep learning models [56]. The training step of the model and the quality of the database that find the best parameters of a given NN make the difference. Solutions appear to detect whether a particular dataset has been used to train a model [57].

Today, most existing works rely on Intel’s SGX secure enclave to protect their models. Tople et al. [50] adapt classical SCA countermeasures to this context of SGX. Indeed, they eliminate input-dependent branches to prevent memory access patterns attacks based on the assign-or-nothing pattern. Because these input-dependent branches appear in only 10% of NN computations, eliminating them incurs a low overhead. SGX is not present in GPUs, which are popular for NN’s deployment.

There are also pure cryptographic proposals. Following Cryptonets, Gilad-Bachrach et al. [58], Bourse et al. [59] and Chabanne et al. [60] rely on homomorphic encryption to perform an NN’s predictions. Unfortunately, they are considered impractical for complex ML tasks owing to their computational overhead. Different secure multiparty computations (SMC) frameworks [61–63] have also been introduced for deep learning predictions. Although more affordable regarding performances, they imply distributed architectures and a large bandwidth for communications between machines. Both of these kinds of initiatives are being integrated in Tensorflow [64,65] and CrypTen’s PyTorch.

## 7 | CONCLUSION

We have presented a classification of attacks on NNs, with respect to the kind of SCA and the goal of the attacker. Today, most attacks focus on recovering the architecture of the targeted NN, and we detail them in Section 5. Other attacks are presented in Section 6.

Finding which parameters are in use demands full access to the model with physical access to the machine where the computations are carried out. The protection of NNs seems hopeless without relying on security hardware features (Section 6.2). The handling of huge NNs inside embedded devices is problematic and it should be necessary to derive an optimization to fit constraints in terms of size and performance, considering extra countermeasures.

Finally, we would like to know whether local access SCA with an attacker who shares her cache memory with her victim permits one to find the parameters of an NN. In this case, solutions such as SMC (Section 6.2) seem appealing but would imply splitting the model across different machines.

## REFERENCES

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems NIPS’12, vol. 1, pp. 1097–1105. Curran Associates, Inc., Red Hook, NY (2012)  https://dl.acm.org/doi/10.5555/2999134.2999257
2. Apple, Inc.: Face id security. Apple Inc. US (2017)  https://www.apple.com/ca/business-docs/FaceID_Security_Guide.pdf
3. Sun, Z., Sun, R., Lu, L.: Mind your weight(s): a large-scale study on insufficient machine learning model protection in mobile apps. CoRR (abs/2002.07687, 1–13 2020). abs/2002.07687.  https://arxiv.org/abs/2002.07687
4. Yu, H., et al.: Cloudleak: large-scale deep learning models stealing through adversarial examples. In: 27th Annual Network and Distributed System
Tramèr, F., et al.: Stealing machine learning models via prediction apis. In: 25th USENIX Security Symposium, USENIX Security 16, pp. 608–618. USENIX Association, Austin (2016)

Papernot, N., et al.: Practical black-box attacks against machine learning. In: Karri, R. et al. (eds.) Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Asia CCS 2017, pp. 506–519. ACM, Abu Dhabi, April 2–6 (2017). https://doi.org/10.1145/3052973.3053090

Maurice, C., et al.: PRADA: protecting against DNN model stealing attacks. CoRR 1–17 (2018). abs/1805.02628. http://arxiv.org/abs/1805.02628

Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep learning training by reducing internal covariate shift. In: Bach, F.R., Blei, D.M. (eds.) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, 6–11 July 2015. Vol. 37 of JMLR Workshop and Conference Proceedings, pp. 448–456. JMLR.org, Lille (2015). http://proceedings.mlr.press/v37/ioffe15.html

He, K., et al.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 770–778. IEEE Computer Society, Las Vegas (2015). https://doi.org/10.1109/CVPR.2016.90

Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM. 60(6), 84–90 (2017). https://doi.org/10.1145/3065386

Briet, J., et al.: SNIFF: reverse engineering of neural networks with fault attacks. CoRR 1–10 (2020). abs/2002.11021. https://arxiv.org/abs/2002.11021

Liu, Y., et al.: Fault injection attack on deep neural network. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 131–138. Irvine, CA (2017)

Jagielski, M., et al.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) 29th USENIX Security Symposium, USENIX Security 2020, August 12–14, 2020, pp. 1345–1362. USENIX, Berkeley (2020). https://www.usenix.org/conference/usenixsecurity20/presentation/jagielski

Yan, M., Fletcher, C.W., Torrellas, J.: Cache telepathy: leveraging shared resource attacks to learn DNN architectures. In: Capkun, S., Roesner, F. (eds.) 29th USENIX Security Symposium, USENIX Security 2020, August 12–14, pp. 2003–2020. Berkeley (2020). https://www.usenix.org/conference/usenixsecurity20/presentation/yan

Briet, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 16–29. Springer, Berlin (2004)

Maurice, C., et al.: Reverse engineering intel last-level cache complex addressing using performance counters. In: International Symposium on Recent Advances in Intrusion Detection, pp. 48–65. Springer-Verlag, Berlin (2015)

Kocher, P., Jaffe, J., Jun, B: Differential power analysis. In: Annual International Cryptology Conference, pp. 388–397. Springer, Berlin (1999)

Brumley, D., Boneh, D.: Remote timing attacks are practical. Comput. Security Symposium, NDSS 2020. The Internet Society, San Diego, February 23–26 (2020). https://www.ndss-symposium.org/ndss-paper/cloudleak-large-scale-deep-learning-models-stealing-through-adversarial-examples/

Kocner, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference, CRYPTO 2000, August 17–21, 2000, Proceedings, Part III. Vol. 12172 of Lecture Notes in Computer Science, pp. 189–218. Springer, Santa Barbara. (2020). https://doi.org/10.1007/978-3-030-56877-1_7

Lecun, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE. 86(11), 2278–2324 (1998)

Naghbibiouybari, H., et al.: Rendered insecure: GPU side channel attacks are practical. In: ACM Conference on Computer and Communications Security, pp. 2139–2153. Association for Computing Machinery, New York (2018)

Hong, S., et al.: Security analysis of deep neural networks operating in the presence of cache side-channel attacks. CoRR 1–16 (2018). abs/1810.03487. http://arxiv.org/abs/1810.03487

Hong, S., et al.: How to oman the NAS in your spare time. In: 8th International Conference on Learning Representations, ICLR 2020, April 26–30 (2020). OpenReview.net, Addis Ababa. https://openreview.net/forum?id=S1erpeBFpB

Zhang, Y., et al.: Cross-tenant side-channel attacks in paas clouds. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, November 3–7 (2014), pp. 990–1003.ACM, Scottsdale. https://doi.org/10.1145/2660267.2660356

Ristenpart, T., et al.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds’. In: Al-Shaer, E., Jha, S., Keromytis, A.D. (eds.) Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS 2009, November 9–13, pp. 199–212.ACM, Chicago (2009). https://doi.org/10.1145/1653662.1653687

Yarm, Y.: Mastik: a micro-architectural side-channel toolkit. School of Computer Science, Adelaide. http://cse.adelaide.edu.au/~yarm/Mastik (2016)

Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, May 7–9, 2015, Conference Track Proceedings, San Diego. http://arxiv.org/abs/1409.1556 (2015)

Liu, Y., Srivastava, A.: GANRED: GAN-based reverse engineering of DNNs via cache side-channel. In: Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW’20, 2020, 41–52. Association for Computing Machinery, New York (2020). https://dl.acm.org/doi/pdf/10.1145/3411495.3421536

Duddu, V., et al.: Stealing neural networks via timing side channels. CoRR. abs/1812.11720 1–10 (2018). http://arxiv.org/abs/1812.11720

Hua, W., Zhang, Z., Suh, G.E.: Reverse engineering convolutional neural networks through side-channel information leakage. In: DAC, pp. 41–46. ACM (2018)

Yu, H., et al.: DeepEM: Deep Neural Networks Model Recovery through EM Side-Channel Information Leakage. 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) 2020, San Jose, CA, USA, December 7-11, 2020, pp. 209–218. IEEE, Computer Society, Los Alamitos (2020). https://doi.ieeecomputersociety.org/10.1109/HOST45689.2020.9300274

Krzhizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master’s thesis, Department of Computer Science, University of Toronto. 1–60 (2009)

Hu, X., et al.: Deepsniffer: a DNN model extraction framework based on learning architectural hints. In: International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 385–399. Association for Computing Machinery, New York (2020)

Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey. IEEE Access 6, 14410–14430 (2018). https://doi.org/10.1109/ACCESS.2018.2807385
