Chapter 1
On Implementation-level Security of Edge-based Machine Learning Models

Lejla Batina[1], Shivam Bhasin[2], Jakub Breier[3,4], Xiaolu Hou[5], and Dirmanto Jap[2]

Abstract In this chapter, we are considering the physical security of Machine Learning (ML) implementations on Edge Devices. We list the state-of-the-art known physical attacks, with the main attack objectives to reverse engineer and misclassify ML models. These attacks have been reported for different target platforms with the usage of both passive and active attacks. The presented works highlight the potential threat of stealing an intellectual property or confidential model trained with private data, and also the possibility to tamper with the device during the execution to cause misclassification. We also discus possible countermeasures to mitigate such attacks.

1.1 Introduction

Security evaluation labs are facing new challenges every day as the adversaries are becoming more powerful considering the resources available and more advanced in terms of methods and techniques they use. Thus, machine learning (ML) is becoming indispensable for secure cryptographic implementations and ML methods are becoming mandatory in security evaluations. This aspect of AI for physical attacks is elaborated in more detail in [29, 33].

We are also witnessing an increase in intellectual property (IP) preserving strategies for various industries such as media content protection (Netflix and Spotify), automotive, wearables (such as watches and wristbands) etc. Basically, for those cases when optimized neural networks are of commercial interest, model details are often kept undisclosed. There are many reasons for keeping the neural network architectures secret. Often, those pre-trained models might provide additional information

[1]Radboud University, Nijmegen, The Netherlands
[2]Nanyang Technological University, Singapore
[3]Silicon Austria Labs, TU-Graz SAL DES Lab, Austria
[4]Graz University of Technology, Graz, Austria
[5]Slovak University of Technology, Bratislava, Slovakia

regarding the training data, which can be very sensitive. One critical example, is that if the model was trained on medical records of patients [19], confidential information could be encoded into the network during the training phase. Another use case of a neural net as an IP is in products using deep learning for high definition maps and cameras e.g. in automotive applications. As those markets are very competitive; the more efficient and effective neural networks are, the more successful companies selling them become.

Hence, determining the layout of the network with e.g. trained weights is a desirable target for an attacker. There are several ways to do it. First, the attacker might try to train new models, providing he/she has the access to target neural nets and training data. Second, the attacker could reverse engineer the neural nets of interest by using some additional information that becomes available while the device under attack is operating. This additional information is often physical and provided as a side channel e.g. timing, electromagnetic emanation (EM), or as a result of an active manipulations such as through fault injection responses and similar.

Relevant previous works on reverse engineering neural nets via side channels showed a lot of promise for this kind of research [4, 5]. It was demonstrated that a side-channel attacker is capable of reverse engineering proprietary information from an ARM Cortex-M3 microcontroller, which is a platform often used in edge devices using neural networks such as wearables, surveillance cameras etc. Other works considering fault injection techniques and the information learned from this kind of attacks followed [6].

1.1.1 Machine Learning for Edge Devices

In this chapter, we are focusing on Edge devices that can be used in applications like automotive, security cameras, wristbands, smart factory etc. Edge devices are those that collect, process and store data, close to the things/edges/sensors where the information is produced and gathered. This paradigm on computing saves a lot on latency and enables real time processing, as compared to the cloud based alternatives. With time edge devices have been strengthened with enhanced capabilities like artificial intelligence (AI) to provide decision making power to them. This integration of AI or in particular machine learning capabilities to edge devices is popularly known as EdgeML [14]. These devices can range from general purpose hardware like an Arm CortexM3 microcontroller, or those with dedicated hardware support for ML processing like Nvidia Jetson, Intel neural compute stick etc.

1.1.2 Attacks on Machine Learning

Security and reliability of machine learning has been a rising matter of concern in recent years. Here we briefly discuss some of the most prominent types of attacks which have been reported.

Model extraction attacks were reported targeting machine learning algorithms. The main idea behind such attacks is to query a victim model as a black box, with chosen/known input data and try to construct another model which mimics the victim model in prediction and performance. Such attacks are widely popular in the Machine learning as a Service (MLaaS) paradigm, where an API of the victim model is available to the adversary on a pay per use basis [53] and the adversary aims to recover an equivalent model with minimum number of queries. Apart from model extraction, known attacks were extended to the recovery of training data by techniques like membership inference and model inversion. Shokri *et al.* [47] reported the leakage of sensitive information from machine learning models about individual data records used for training. They show that such models are vulnerable to membership inference attacks. Details about training data can also be leaked through model inversion as presented by Fredrikson *et al.* [18].

Other kinds of attacks compromising the reliability of machine learning were reported. These attacks systematically lead a trained model to predict an incorrect output, resulting in a so-called an evasion attack [50]. The modus operandi of this attack involves small changes in the input that push the model to cross the decision boundaries during the classification. This effectively changes the resulting output class. The perturbed inputs were later called "*adversarial examples*" and can be used for various different purposes, including targeted and untargeted misclassification, denial of service, etc.

All such attacks target the model behavior and run independent of implementation style. Normally, such an API model would be hosted on the cloud. As the attacks appeared, appropriate mitigation were also proposed, for example, against model extraction attacks [31, 37], evasion [11] and adversarial attacks [12].

With the adoption of machine learning for edge devices under EdgeML paradigm, new vulnerabilities arose. The computation of resource intensive machine learning algorithms leaves a non-negligible physical signature for every execution. This physical signature depends on the model parameters and inputs. If the adversary who controls the inputs can measure this physical signature, it is possible to learn information about otherwise black box model. The physical signature can be in form of timing, power consumption, cache access patterns etc. A survey of side-channel based attacks on machine learning discussion on mitigation techniques is reported in Section 1.2.

EdgeML also exposes the machine learning computation to active adversary which are capable of disturbing the computation through intentional perturbations. The introduced disturbances to the computation may often lead the trained model to predict an incorrect output. From the higher level, these attack are similar to evasion attacks with adversarial attacks. However, an advanced adversary capable of fault injection can be much more powerful as the perturbations can potentially be

inserted at an arbitrary point of the computation, giving a more precise control to the adversary. Further modification of these attacks can also lead to model extraction or denial of service. A survey of fault-injection based attacks on machine learning and discussion on mitigation techniques is reported in Section 1.3.

1.2 Overview Side Channel Threats to Machine Learning

With the systematic deployment of ML models on edge devices, unprecedented physical access to those models has lead to new security vulnerabilities. While classically these ML models are considered a black box, physical access permits different adversaries to snoop direct or indirect information about the internal execution. In this section, we provide an overview of recent works which used side-channel analysis to compromise ML models.

Side-channel attacks (SCA) are passive attacks that observe physical quantities related to computation of sensitive variables (dependent on secret data) and exploit it to gain confidential information. SCA have been widely studied in the community of information security and cryptography over the past two decades to demonstrate vulnerabilities in implementations of various security and cryptographic algorithms including both symmetric and asymmetric key primitives. They exploit physical traits like power consumption, computation time, cache access patterns, electromagnetic (EM) emanation, etc. Typical attacks relying on statistical methods are Differential Power Analysis (DPA) or Differential ElectroMagnetic Analysis (DEMA). Application of SCA on cryptographic primitives and recent advances with application of ML is previously discussed in [29, 33].

When a given implementation of ML model is executed, it generates a physical signature. This physical signature, although unintentional, exists in various forms. Let us take an example of execution time of a neural network. This execution time will depend on structural parameters of the network like the number of layers, number of neurons, etc. Even inside a single neuron, the choice of the activation function influences the execution time, as shown in [5]. Activation functions like *ReLU* are a lot less resource intensive compared to *sigmoid* or *tanh* that involve complex operations like exponentiation. Moreover, the input-dependent execution time of exponentiation can reveal information about the input to an adversary who has access to detailed timing patterns. Similar vulnerabilities can also be exploited by an adversary who has access to other physical signatures.

Those side channels have been demonstrated to leak sensitive parameters of a ML model like number of layers, number of neurons, activation functions, secret weights, filter size etc. The leakage can be exploited in several scenarios. We identified three attack scenarios presented in the literature, which are as follows:

- *Model Extraction:* The adversary aims to recover parameters of the target model with as much precision as possible. Such attacks are relevant for the IP theft scenario where adversary has cost benefits to recover secret black box model and evade payment for additional licences. Precise knowledge of the model can also

be exploited to learn information on training data, which can be sensitive and critical in certain settings like healthcare.
- *Substitute Model Extraction:* This is a weaker version of the model extraction attack where an adversary aims at recovering a model that performs similarly to the target black box model. The performance can be expressed in metrics like testing accuracy.
- *Input Recovery:* For certain applications, where the input to a model is privacy sensitive, appropriate security measures like encryption are used to not communicate the sensitive input in plaintext over an open communication channel. However, for inference tasks, the input must be decrypted before any processing by the ML model. Side-channel leakage of interactions between the model and the secret input can also be exploited to learn information about the input. Most, if not all attacks in this class exploit the processing of the input in the first layer where processing is done on raw inputs.

1.2.1 State-of-the-Art

Previous works demonstrating vulnerabilities of ML models against side-channel attacks have looked into the following physical quantities:

- *Timing Side Channel:* The vulnerabilities under this class exploit the fact that the time of internal computation relates to the parameters of the target model. Previous research has shown that both, execution time and the time to access the model parameters can leak critical information.
- *Power/EM Side Channel:* Power consumption or EM emanations of internal operations can be exploited to gain information on involved sensitive values. Vulnerabilities in this class typically target, but are not limited to, basic operations like weight multiplication or CONV filters.
- *Microarchitectural Side Channel:* Executing ML models on commodity hardware like high-end CPU or GPU results in leakage at the microarchitectural level. Most known works have exploited cache access patterns in terms of timing related to cache hit or cache miss.

Table 1.1 provides the summary of recent attacks that have been reported. In the rest of this section, we will provide more details on relevant previous works.

Timing side-channel

One of the earliest works on reverse engineering targeting Convolution Neural Networks (CNN) with Side-Channel Attacks (SCA) was proposed by Hua *et al.* [27]. The work acknowledges the sensitivity of the model and assumes standard protection like executing model computation in secure enclaves like Intel Software Guard Extensions (SGX). It exploits information leakage through timing (and memory)

Work	Side-channel	Attack Model	Network	Target Device
[27]	timing, memory	model extraction (first layer)	AlexNet, SqueezeNet	CNN Accelerator
[17]	timing	substitute model extraction	VGG	Intel CPU
[55]	power	input recovery	Binarized CNN	FPGA based CNN accelerator
[5]	EM	input recovery	MLP, CNN	ARM microprocessor
[15]	power, timing	input recovery	MLP	Atmel microprocessor
[4]	EM	model extraction	MLP, CNN	AVR and ARM microprocessors
[58]	EM, timing	model extraction	binarized ConvNet, VGG, LeNet, AlexNet	PYNQ board
[16]	power	model extraction (weight recovery)	BNN	FPGA accelerator
[56]	cache	substitute model extraction	VGG, ResNet	CPU
[24]	cache	substitute model extraction	VGG, ResNet, DenseNet, Inception, MobileNet, Xception	CPU
[26]	cache	model extraction	ResNet	NVDIA GPU

Table 1.1 Summary of the state-of-the-art SCA for DL

side-channels targeting CNN accelerators running in a secure enclave. The secure enclave prevents an adversary from accessing information of the execution, however, the off-chip memory accesses are still observable, which eventually allows reverse engineering of CNN structure and weights. Owing to the huge size of CNN models, it is not possible to store all weights/parameters on the on-chip memory, rather off-chip memory stores the parameters and is accessed when required in the computation. Memory access patterns reveal information of accessed memory locations and read/write patterns. The information leakage on access patterns remains available even if the memory is encrypted. As a result, Hua *et al.* demonstrate the retrieval of key network parameters like the number of layers, input/output sizes of each layer, size of filters, data dependencies among layers, etc. This allows an attacker to infer a small set of possible network structures by exploiting the execution time of a computation on a CNN accelerator.

Two commonly popular networks were successfully targeted, AlexNet [35] and SqueezeNet [28]. Once the network structure is recovered, the proposed attack can be extended to reverse engineer the secret weights of the CNN. The attack assumes usage of dynamic zero pruning in the CNN architecture and exploits it for weight recovery, with knowledge of the inputs. However, the exploited memory access patterns are available under a strong assumption like hardware Trojan, physical memory bus

probing or compromised OS. Activation functions like *ReLU* which converts any negative input to zero, result in a large number of zeros in intermediate results of an inference. These extra zeros can be pruned to optimize storage. However, this memory optimization leaks the number of zero-valued pixels pruned by the activation function, which can then be exploited to retrieve information on weight and bias, in particular their ratio. The knowledge of ratio significantly reduces the entropy of weights. The authors demonstrated the recovery of the weight and the bias for the first layer of AlexNet. Weight recovery attacks on deeper layers were not investigated.

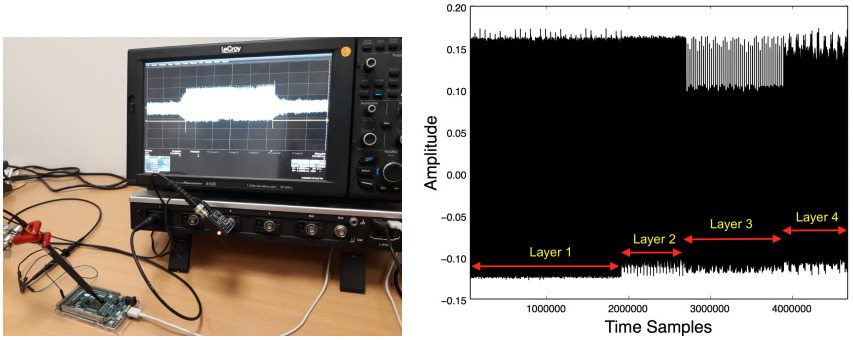
Duddu *et al.* [17] proposed a timing based attack on neural networks. The attack can target different hardware architectures or dedicated accelerators as long as the victim and the adversary use copies of the same hardware. The timing information under the black box model is used to determine the network depth under a fixed number of queries. With the information on network depth, the authors generate a substitute model of comparable accuracy to the original model. Reinforcement learning is used to reconstruct a substitute architecture. The attack is demonstrated on VGG [48]-like deep architectures on an Intel Xeon Gold 5115 platform, and the authors argue that the proposed method can be extended to any other hardware accelerator as well. The test accuracy of the substitute model is within 5% error margin from the targeted model architecture. Note that this method does not allow model extraction but finds a substitute network.

Power and EM side-channel

Batina *et al.* [4] proposed a full reverse engineering of neural network parameters based on power/EM side-channel analysis. The proposed attack is able to recover key parameters i.e., activation function, pre-trained weights, number of hidden layers and neurons in each layer, without access to any training data. The adversary uses a combination of simple power/EM analysis, differential power/EM analysis and timing analysis to recover different parameters. In the following, we describe in brief how different parameters are recovered.

The attack targets a trained model of a feed-forward neural network deployed on an embedded device for testing. The adversary feeds known random inputs in a form of floating point real numbers and observes side channels. Fixed point numbers make the attack easier. The measurement setup is shown in Fig.1.1(a). A sample EM trace is shown in Fig. 1.1(b) for a 4-layer Multilayer Perceptron (MLP) with (50, 30, 20, 50) neurons. One can easily distinguish each of the 4-layers. Moreover, it is shown in [4], that the adversary can zoom into each neuron and observe each multiplication and activation function. In other words, the adversary can collect a large amount of traces in one go and reuse individual parts of the traces for recovering different parameters of the network.

The first step is to recover the activation function for each neuron. The activation function is a non-linear component in the neural network processing and normalizing its complex implementation results in a non-constant time execution. Note that



(a) (b)

Fig. 1.1 (a) Experimental EM side-channel measurement setup, (b) Pattern of a 4-layer MLP network with (50, 30, 20, 50) neuron in each layer [4].

the adversary does not need new timing measurements. The EM traces provides precise timing patterns for each activation function in each neuron. Table 1.2 shows minimum, maximum and mean execution times of *sigmoid*, *tanh* and *ReLU* activation functions which can be matched to a pre-characterized profile for recovery. The pre-characterized timing profile of the activation functions, when compared with unlabeled timing profile of target activation function, will reveal the function with high probability. Even though some functions may have similar timing profiles like *sigmoid* and *tanh*, still with enough test samples, the two functions are easily distinguishable from each other owing to different mean timing and corresponding ranges (see Table 1.2).

Table 1.2 Minimum, Maximum, and Mean computation times (in ns) for different activation functions as measured from the EM trace.

Activation Function	Minimum	Maximum	Mean
ReLU	9 767	9 837	9 801
Sigmoid	142 902	179 151	163 449
Tanh	157 693	220 790	208 231

The next step is to recover the individual weights. The weights are recovered using differential power/EM analysis and the Pearson correlation coefficient is used as a statistical distinguisher. The attack targets the multiplication $m = x \cdot w$ of a known input x with a secret weight w. The leakage model for the used embedded microcontroller is the Hamming weight (HW). The adversary makes hypothesis on the weight and correlates the activity of the predicted output m with measured traces t. The correct value of the weight w will show higher correlation compared to all other wrong hypotheses w^*, given enough measurements. The attack was demonstrated in real numbers in IEEE 754 format, where 32-bit representation is used. To keep the number of hypothesis in check, the attack is performed byte-wise

and the 32-bit weight is recovered in 4 parts. It is also observed, that unlike in cryptography, exact weights are not required and some precision errors in recovery can be tolerated without affecting the accuracy of the network. DEMA can also be further used to determine layer boundaries, when not possible. Given an input to the network, the correlation of weight multiplications will be much higher in the first layer as compared to subsequent layers, thus allowing distinguishing neurons belonging to the first layer.

The full network is recovered in an iterative manner with a combination of these developed techniques. The network is recovered from input to output, neuron by neuron and layer by layer. The attack scales linearly with the size of the network and the same set of traces can be reused for various steps of the attack limiting the measurement effort.

Dubey *et al.* [16] proposed a power-based side-channel attack on a Binarized Neural Networks (BNN) to recover secret parameters such as weights and biases. In contrast to Batina *et al.* [4], the target platform is parallel hardware accelerators running a 7-series FPGA board mounted on a SAKURA-X board. They exploit power leakage and perform a basic correlation attack on 4-bit of the weights and demonstrate a successful weight recovery with 200 measurements only. Authors further propose design of BNN accelerators that can resist DPA using countermeasures like masking.

Yu *et al.* [58] proposed a model extraction attack based on combination of EM side-channel measurement and adversarial active learning to recover the Binarized Neural Networks (BNNs) architecture on popular large-scale NN hardware accelerators. The network architecture is first inferred through EM leakage, and then, the parameters are estimated with adversarial learning. For the layer topology reverse engineering, the attacker observes the average timing behavior from the EM traces. This is based on the observation that different layers will result in different execution times. For example, the pooling layer typically requires a shorter time than a convolution one, and a fully-connected layer is observed to have the longest execution time, since it requires most of the sequential XNOR computations. Thus, by observing the timing profile, the adversary could reconstruct the network architecture. In the adversarial learning setting, the attacker crafts malicious inputs for the query, which could be used to identify the decision boundary for the trained model. For the attack, the adversary is assumed to be incapable of accessing the training data or knowing the model. The attack shown through the experiment could recover $96 - 99\%$ in comparison to the black box model.

Considering input recovery attack using power/EM side-channel, the first attack was reported by Wei *et al.* [55]. Authors demonstrate recovery of the input image from FPGA based CNN accelerator. The proposed attack exploits a specific design choice, i.e., the line buffer in a convolution layer of a CNN. Two attack scenarios were presented considering different adversarial capabilities. The first one is the passive model, where the adversary eavesdrops the power consumption during the execution. Assuming that if the processed data is unchanged between cycles, the internal transitions will be limited, resulting in lower power consumption, and thus, by monitoring the power leakage, the adversary could determine if the pixels share similar values, whether they belong to the background of the image. The other

is the active model, where the adversary is profiling the correlation between power signals, by building a power template. The power template characterizes the mapping between pixel values and the corresponding power leakage, under different kernels. The experiment conducted on a MNIST dataset reported a recovery success of 89%.

Batina *et al.* [5] also reported an input recovery attack on embedded platforms. They consider known or commonly used networks where the weights are either public or independently recovered using one of the reverse engineering techniques. Only the first layer weights are crucial for this attack which targets the multiplication between secret input and known weights. This attack is similar to the previous attack on multiplication proposed in [4]. The attack targets the multiplication $m = x \cdot w$ of a secret input x with a known weight w. The issue in this case is that each input is only processed once and thus must be recovered in a single measurement. To overcome this limitation, an adversary can exploit individual weight multiplications in different neurons, captured on different part of the same trace. Thus, EM measurements corresponding to a fixed unknown input and several known weights are present in the same measurement, which can be broken into short independent traces to conduct a classical correlation-based attack. This kind of attacks which exploit different computations in the same measurements are popularly known as horizontal attacks. For bigger networks with a large input layer, the amount of individual multiplications available to an adversary increases, thus allowing a bigger measurement set to perform the attack. The recovery was shown on MNIST images with a precision error of 2 decimal places resulting in almost no visual differences between original and recovered images. The attack also applies on CNN, where a single input value might be processed several times (due to convolution operation). A similar vulnerability as shown in [5] was exploited through timing side-channel by Dong *et al.* [15] to recover input MNIST images with 96% accuracy on 4-layer MLP running on an 8-bit Atmel XMEGA128 microprocessor. They exploit the fact that input multiplication with constant weights will result in a variable time floating point multiplication. The precise timing of the multiplication can be recovered by observing power side-channel trace.

Microarchitectural side-channel

Recently, some works based on microarchitectural attacks have also been proposed for the reverse engineering of Deep Learning (DL). Yan *et al.* [56] have proposed the Cache Telepathy method. The observation is that, for typical NN , the multiplication operation depends on GEMM (Generalized Matrix Multiply). In this case, the architecture parameters of the network will determine how many times the GEMM is called or the dimension of the matrices, which can be revealed through cache side-channel. The attack is based on common cache-based SCA, Flush+Reload [57] and Prime+Probe [38]. The target networks are VGG-16 [48] and ResNet-50 [22]. Using the proposed method, the search space for the architecture can be significantly reduced. In this work, the attack could reveal matrix multiplication related parameter

such as convolutional or fully connected layers, and for others such as activation and pooling layers, it might be harder to recover.

Similarly, the authors of [24] proposed DeepRecon, an attack methodology based on cache side-channel that exploits Flush+Reload to reconstruct the Deep Neural Network (DNN) architecture. In their attack model, rather than accessing the target model directly like in other side-channel based attack, the adversary runs a co-located process on the host machine, in which the victim's model is also running. Similar to earlier work [56], the proposed attack does not generalize to computations on hardware other than a CPU. Also, they find that based on how the matrix multiplication is implemented, they are unable to estimate the inputs and parameters of a victim's model. As such, they hypothesize that this might be the limit of cache-based SCA on DNN.

Hu *et al.* [26] proposed an attack targeting GPU platform and highlighted some of the potential issues arising in contrast to other works. The attack is using the bus snooping technique, exploiting the off-chip memory address traces and PCIe events. The idea used in this work is that inter-layer DNN architecture features will be considered as string of "sentences", so by considering this instead of individual "word", it might maximize the likelihood of a correct match far more effectively than character-by-character approaches. To perform the experiment, they consider the Long short-term memory (LSTM) model, a common neural network, with CTC (Connectionist Temporal Classification) decoder, which is commonly used in Automatic Speech Recognition. The attack only requires the assumption that the adversary can observe the architectural side-channel over time. It also assumes the adversary can feed specific input and observe the results. The experiments are conducted on off-the-shelf Nvidia GPU running CNN, in a parallel manner, and the victim's model is ResNet-18 [22].

1.2.2 Countermeasures

The success of SCA on ML models can be mainly attributed to the naïve implementation of the model. Previously, ML models were rarely seen in hostile environments with adversaries benefiting from physical access. However, with IoT and edge-based devices, the threats have become real as highlighted by the range of works mentioned above. Thus, countermeasures must be investigated. As such, there is a wide research on SCA countermeasures against cryptographic implementations which can be also applied on ML models. However, direct application of countermeasures would result in a non-negligible overhead. In the following, we discuss some directions for countermeasures considering different physical side-channels.

1.2.2.1 Timing Side-Channel

It has been shown by multiple works that the execution time depends on network parameters which eventually leak sensitive information to the adversary. To overcome this problem, the designer can take two approaches. The first approach is to have constant time implementations of basic functions [45]. This can solve some issues where the value of the input is determined from the execution time but other issues like distinguishing between components (e.g. *sigmoid* vs. *ReLU*) may still be possible. The other approach is to randomize the execution timing in a way that it becomes independent of the sensitive information executed preventing an adversary to learn by observing timing information. This would require access to a good source of randomness and techniques like jitter and dummy operations can be used [40].

1.2.2.2 Power/EM Side-Channel

Hiding and Masking are the two typical types of countermeasures used against power/EM side-channel. Hiding aims at reducing the signal to noise ratio in a measurement, making attacks difficult. Masking uses randomization by mixing computation with random data to remove any correlation between sensitive variables and power/EM signature. Dubey *et al.* [16] proposed the first countermeasure for DNN against SCA. The countermeasure, referred to as MaskedNet, is based on masking. The resulting design uses novel masked components such as masked adder trees for fully-connected layers and masked Rectifier Linear Units for activation functions. They even use hiding countermeasure, like Wave Differential Dynamic Logic (WDDL) [51], to protect the sign-bit computation. The proposed protection increases the latency and area-cost by 2.8 and 2.3 times, respectively. When tested against first-order DPA, the attack against masking fails even when using 100k traces, however second-order DPA on masking can still break it with just 3.7k traces. When analyzing the Difference-of-Means (DoM) test on the sign-bit computation, after 40k of traces, bit 0 and bit 1 can be distinguished. The argument for this is in the low noise platform used.

1.2.2.3 Microarchitectural Side-Channel

Much like timing and power/EM side-channel, hiding or randomizing cache activity of ML model execution can prevent such attacks. However, unsurprisingly, any of those choices result in performance overheads. Alam and Mukhopadhyay [3] proposed a countermeasure against microarchitectural side-channel attacks on ML models by observing the Hardware Performance Counter (HPC). During the execution of CNN, an evaluator, who does not know the detail of the implementation but can monitor various HPC events, conducts statistical hypothesis testing on the distribution of the data that can detect an attack. The evaluator throws an alarm when there is an anomaly in the distribution signifying potential side-channel leakage.

Thus, as a general observation, ML models do suffer from side-channel vulnerabilities and existing countermeasures stem from either hiding or masking families of countermeasures. In practice, a combination of various countermeasures is more likely deployed. However, for modern architectures, the network architecture can easily grow to millions of parameters, and such, the countermeasure overhead might make it impractical to implement. Thus, ML friendly countermeasures in terms of the overhead in cost must be investigated.

1.3 Overview of Fault Injection Threats to Machine Learning

In this section, we focus on a special class of physical attacks known as fault attacks, which have become a common practice owing to decreasing prices and increasing expertise required to mount such attacks [21]. Fault attacks are active attacks on a given implementation which try to perturb the internal computations by external means. Such attacks are commonly used for mounting secret key recovery attacks in cryptography or for violating/bypassing security checks [30]. Recently, fault attacks have been applied to neural networks to achieve misclassification or reverse engineering.

The rest of this section is organized as follows. Subsection 1.3.1 provides some necessary background on fault injection attacks and the possible threats they pose to neural network models. Subsection 1.3.2 outlines the current state-of-the-art and gives details on the most prominent works in this area. Finally, Subsection 1.3.3 discusses possible countermeasures to protect neural networks against faults.

1.3.1 Background

Fault Injection Attacks (FIA) disrupt the device during the computation task, providing in this way some benefit to the attacker. Generally, this benefit can be anything from denial of service and privilege escalation, to the secret data recovery.

For example, in case of FIA on cryptographic circuits, the goal is typically to get the information on the secret key used during the encryption [30]. These attacks mostly exploit the scenario where the attacker has access to the device and can tamper with it, which is often the case for edge devices. However, there exist also techniques that can flip bits remotely, such as Rowhammer [32].

Multiple types of fault attacks and their outcomes on hardware devices are possible, resulting in various ways to affect the target device. These are often referred to as *fault models*. The most commonly deployed fault models in the literature are:

- *Bit flips:* allow the attacker to target a certain bit of the processed data and invert its value.
- *Stuck-at faults:* allow the attacker to set or reset the value of a certain bit of the processed data.

- *Random faults:* allow the attacker to change the value of a data structure (in register, memory, bus, etc.) to a random value.
- *Instruction skips:* allow the attacker to skip the execution of one or more instructions in the instruction sequence.
- *Instruction changes:* allow the attacker to change the executed instruction into another instruction – this can be achieved by faulting the instruction opcode.

In the pre-attack phase, the attacker needs to determine what kind of the fault model is realistic for the target device under test with the usage of specific fault injection equipment. After this knowledge is obtained and classified, the attacker can identify a fault analysis method that works fitting the given model.

When it comes to deployment of machine learning models in the field, there are many AI accelerators offering small size, low power consumption and low cost (e.g. Nvidia Jetson Nano, Google Coral, etc). It was shown that small IoT devices like Raspberry Pi devices are also capable of running deep learning models [54]. These devices are ideal targets for fault injection attacks, as they might be physically accessible to the attacker and their complexity is low, thus allowing precise fault injection.

There are several attack scenarios affecting neural network models that can be achieved by fault attacks, mainly:

- *Denial of service:* faults can disturb the device in a way that it is not able to respond to a user's queries, leading to denial of service [42]. This can be either a permanent or a transient state.
- *Evasion:* classic evasion attack on ML models aims at misclassification of the input by adding adversarial perturbations in the input. In the evasion attack by faults, the attacker disturbs the computation of the model resulting in the misclassification [39].
- *Model extraction:* it was shown that faults can help in recovering the confidential parameters of neural networks, such as weights and biases, allowing higher precision compared to other methods [9].

1.3.2 State-of-the-Art

The seminal work in the field of adversarial fault injection was published by Liu et al. in 2017 [39]. They introduced two types of attacks: *single bias attack* changes the bias value in either one of the hidden layers (in case of *ReLU* or similar activation function) or output layer of the network to achieve the misclassification; while *gradient descent attack* works in a similar way like Fast Gradient Sign Method [20], but changes the internal parameters instead of the input to the network. For more details on both types see below.

Practical fault injection by using a laser technique was shown by Breier et al. in 2018 [6]. They were able to disturb the instruction execution inside the general-purpose microcontroller to achieve the change of the neuron output. In their paper,

they focused on behavior of three activation functions: in case of sigmoid and tanh, the fault resulted to inverted output, while in case of *ReLU*, the output was forced to be always zero.

Zhao et al. [59] proposed a *fault sneaking attack* on DNN models. The main goal is to cause misclassification to labels specified by the attacker, while keeping the fault injection stealthy. The stealthiness is achieved by keeping the accuracy of the faulted network as close to the original accuracy as possible. According to their results, they were able to keep the model accuracy loss to 0.8% for MNIST database and 1% for CIFAR database.

Breier et. el. [9] analyzed reverse engineering in the context of transfer learning. The parameters from the teacher network are assumed to be known to the attacker. With the fault attack, the attacker can then recover the weights and biases of the new layers in the student network.

A survey on error tolerance of neural networks, published in 2017 [52], examines the effects of faults on DNN models.

In the rest of the section, we will provide more details on selected works.

Single Bias Attack and Gradient Descent Attack

Liu et al. [39] proposed the first work analyzing the effects of fault attacks on deep neural networks to achieve misclassification. They investigated two attack methods:

- *Single bias attack (SBA)* aims to achieve misclassification by modifying only one parameter in the neural network.
- *Gradient descent attack (GDA)* aims to achieve misclassificaiton for a targeted input pattern while keeping the accuracy for other input patterns.

As a threat model, they considered a white-box attack, where the attacker has knowledge of the structure, parameters as well as low-level implementation details of the targeted DNN. They also assumed that the practical attack is achievable by using current techniques so that the attacker can modify any parameter in DNN to an arbitrary value in the valid range of the used arithmetic format.

They considered *ReLU*-like activation functions, which are defined as follows

$$g(u) = \begin{cases} u & u \geq 0 \\ \alpha u & u < 0 \end{cases},$$

where $\alpha > 0$. They named DNN using *ReLU*-like activation function in the hidden layers as *ReLU*-like DNN.

To analyze the single bias attack, they give the following definition:

Definition 1 [39] Given two variables x and y, if there exist two constants ε and δ such that $\frac{dy}{dx} = \delta$ when $x > \varepsilon$, we say y is one side linear to x and the one side linear slope is δ.

With the notion of one side linearity, they prove the following result:

Theorem 1 *1. Let $y_1, y_2, \ldots, y_n$ variables, which are all one side linear to variable x with slopes $\delta_1, \delta_2, \ldots, \delta_n$ respectively. If $\delta_n > \delta_i, \forall i \neq n$, there exists a constant ε_{sink} such that $y_n > y_i$, $\forall i \neq n$ when $x > \varepsilon_{sink}$.*
2. In ReLU*-like DNN, for a bias m in hidden layers, every output neuron is one side linear to m.*

Thus, to achieve a single bias attack that misclassifies any input to the target class, the attacker analyzes the network structure to find a bias m in the hidden layer such that the targeted class has the largest one side linear slope w.r.t. m among all the output neurons. Then, by increasing m using fault the attacker would finally make the DNN's output converge at the target class.

They presented evaluation of SBA on a CIFAR model [49] which is a *ReLU*-like DNN. They achieved the highest attack accuracy of 57.23% by targeting the 6^{th} layer of the network.

For the gradient descent attack, the authors aim at maximizing the following objective function by changing the parameters of only one single layer, denoted by θ,

$$J(\theta) = F^i(\theta, x) - |\theta - \theta_b|,$$

where θ_b denote original parameter values for a single layer, F^i represents the output for neuron corresponding to class i, and λ is an L1-norm regulator.

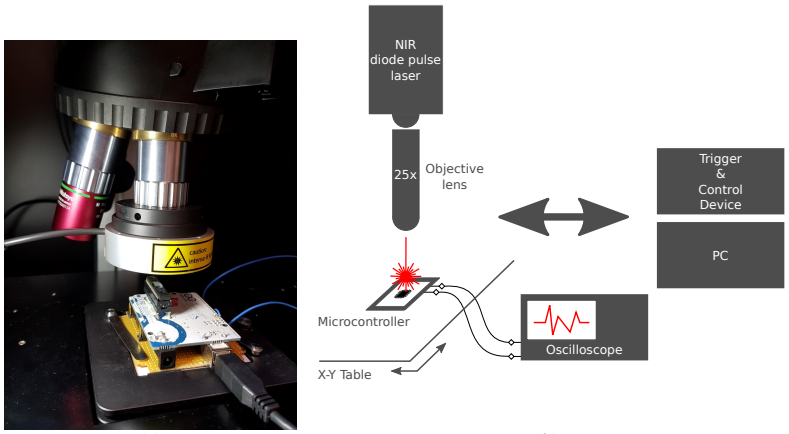
During the gradient descent, they proposed "modification compression" – at each iteration step, they replaced the element with the smallest absolute value in θ by 0. They showed by experiements that conducting modification compression can significantly reduce the number of modified parameters, by about 90%. The experiments with MNIST model [13] and CIFAR model [49] showed that GDA can achieve classification accuracy 95.20% and 81.66%, and degrades the benign accuracy by 3.86 percent and 2.35 percent, respectively.

Practical Attack on Microcontrollers and Activation Function

To the best of our knowledge, the only practical fault attack on neural network models was published in [6], where the authors used a laser to induce faults in a general purpose microcontroller. Next, we provide more details on this attack.

Attack Equipment Setup. The main component of the experimental laser fault injection station was the diode pulse laser with a wavelength of 1064 *nm* and pulse power of 20 W. This power was further reduced to 8 W by a 20 times objective lens which reduced the spot size to 15×3.5 μm^2.

As the device under test (DUT), `ATmega328P` microcontroller was used and mounted on `Arduino UNO` development board. The package of this chip was opened so that there is a direct visibility on a back-side silicon die with a laser. The board was placed on an `XYZ` positioning table with a step precision of 0.05 μm in each direction. A trigger signal was sent from the device at the beginning of the computation so that the injection time could be precisely determined. After the trigger signal was captured by the trigger and control device, a specified delay was inserted before laser



(a) (b)

Fig. 1.2 Experimental laser fault injection setup – (a) device under test, (b) setup components.

activation. Laser activation timing was also checked by a digital oscilloscope for a greater precision. This setup is depicted in Figure 1.2.

DNN Activation Function Fault Analysis. To evaluate different activation functions, three simple 3-layer neural networks were implemented, with sigmoid, *ReLU* and tanh as the activation function for the second layer. The activation function for the last layer was set to be softmax. The neural networks were implemented in C programming language, which were further compiled to AVR assembly and uploaded to the DUT.

The activation functions in the second layer were surrounded with a trigger signal that raised a voltage on a selected Arduino board pin to 5 V, to help determine the laser timing.

As an instruction skip/change are one of the most basic attacks on microcontrollers, with high repeatability rates [8], this fault model was used in the experiments. The used microcontroller clock was 16 MHz, therefore one instruction took 62.5 ns. Some of the activation functions took over 2 000 instructions to execute. To check what are the vulnerabilities of the implementations, the timing of the laser glitch was varied from the beginning until the end of the function execution so that every instruction would be eventually targeted.

After a successful misclassification was observed, vulnerable instructions could be determined by visual inspection of the compiled assembly code and by checking the timing of the laser in that particular fault injection instance. With a laser power of 4.5% it was possible to disturb the algorithm execution, when tested with reference codes. More details on the behavior on this particular microcontroller under laser fault injection can be found in [8].

In this exploratory study, a random neural network was implemented, consisting of 3 layers, with 19, 12, and 10 neurons in input layer, hidden layer, and output layer, respectively. The fault attack was always targeting the computation of one of the activation functions in hidden layer. In the following, the experimental results on

different activation functions will be explained in more detail.

ReLU was implemented in C as follows:

```
if (Accum > 0) {
   HiddenLayerOutput[i] = Accum;}
else {
   HiddenLayerOutput[i] = 0;}
```

where i loops from 1 to 12 so that each loop gives one output of the hidden layer. `Accum` is an intermediate variable that stores the input of the activation function for each neuron. The assembly code inspection showed that the result of the successful attack was executing the statement after `else` such that the output would always be 0. The corresponding assembly code is as follows:

```
1        ldi r1, 0       ;load 0 to r1
2        cp r1, r15      ;compare MSB of Accum to r1
3        brge else       ;jump to else if 0 >= Accum
4        movw r10, r15   ;HiddenLayerOutput[i] = Accum
5        movw r12, r17   ;HiddenLayerOutput[i] = Accum
6        jmp end         ;jump after the else statement
7  else: clr r10         ;HiddenLayerOutput[i]= 0
8        clr r11         ;HiddenLayerOutput[i]= 0
9        clr r12         ;HiddenLayerOutput[i]= 0
10       clr r13         ;HiddenLayerOutput[i]= 0
11 end:  ...             ;continue the execution
```

where each float number is stored in 4 registers. For example, `Accum` is stored in registers `r15,r16,r17,r18` and `HiddenLayerOutput[i]` is stored in `r10,r11,r12,r13`. Lines 4, 5 execute the equation `HiddenLayerOutput[i] = Accum`.

The attack was skipping the "`jmp end`" instruction that would normally avoid the part of code setting `HiddenLayerOutput[i]` to 0 in case `Accum` > 0. Therefore, such change in control flow renders the neuron inactive no matter what is the input value.

Sigmoid is implemented by a following code in C:

```
HiddenLayerOutput[i] = 1.0/(1.0 + exp(-Accum));
```

After the assembly code inspection, it was observed that the successful attack was taking advantage of skipping the negation in the exponent of `exp()` function, which compiles into one of the two following codes, depending on the compiler version:

```
A) neg r16          ;compute negation r16
B) ldi r15, 0x80    ;load 0x80 into r15
   eor r16, r15     ;xor r16 with r15
```

Laser experiments showed that both `neg` and `eor` could be skipped, and therefore, significant change to the function output was achieved.

Hyperbolic tangent was implemented by a following code in C:

```
HiddenLayerOutput[i] = 2.0/(1.0 + exp(-2*Accum)) - 1;
```

Similarly to sigmoid, the experiments showed that the successful attack was exploiting the negation in the exponential function, leading to an impact similar to sigmoid.

Softmax. It was unable to obtain any successful misclassification. There were only two different outputs as a result of the fault injection: either there was no output at all, or the output contained invalid values. This lack of valid output prevented further fault analysis to derive the actual fault model that happened in the device.

The summary of the results, showing the original and the faulted activation functions is depicted in Figure 1.3, with the solid line depicting the original functions and dotted lines depicting the faulted functions.

Authors pointed out that it would make sense to explore different fault models, such as single bit flips that were experimentally achieved by laser in [1]. The first application of such attack would be to target the IEEE754 floating point representation for the weights. The representation follows 32-bit pattern $(b_{31}...b_0)$: 1 sign bit (b_{31}), 8 exponent bits $(b_{30}...b_{23})$ and 23 mantissa (fractional) bits $(b_{22}...b_0)$. The represented number is given by $(-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2-127} \times (1.b_{22}...b_0)_2$. A bit flip attack on the sign bit or on the exponent bits would make significant influence on the weight. This idea was later adopted in [25] to explore single bit upsets on the network parameters. We detail this work in the next part.

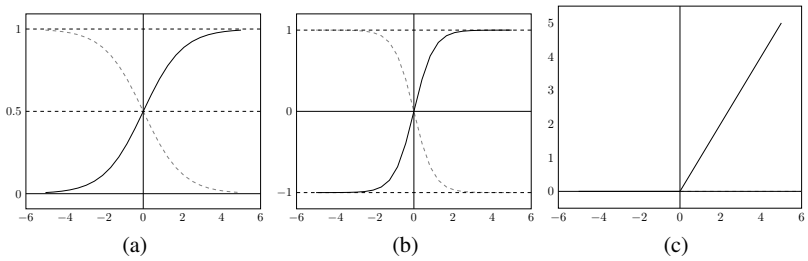


Fig. 1.3 (a) Sigmoid, (b) Hyperbolic tangent, and (c) *ReLU* functions. Solid lines indicate original function, dotted lines indicate faulted ones.

Single Bit-Flip Attack

A comprehensive evaluation of bitwise corruptions on various deep learning models was presented by Hong et al. in 2019 [25]. They showed that most models have at least one parameter such that if there is a bit-flip introduced in its bitwise representation, it will cause an accuracy loss of over 90%. The assumed attacker model is remote fault injection by using Rowhammer [32].

The main goal of [25] is to provide an investigation on how much damage can be achieved by minimal changes to the network caused by fault injection. The results are presented on popular image datasets – MNIST [36], CIFAR10 [34],

and ImageNet [46]. Analysis is provided on 19 different DNN models, including newly generated architectures, and publicly available networks, such as VGG16 and AlexNet. To measure the impact, they defined a new metric called *Relative Accuracy Drop (RAD)*:

$$RAD = \frac{Acc_{pristine} - Acc_{corrupted}}{Acc_{pristine}},$$

where $Acc_{pristine}$ is the classification accuracy of the original (pristine) model and $Acc_{corrupted}$ is the classification accuracy of the corrupted model. For the experiments in the paper, they set $RAD > 0.1$ to be the criterion for indiscriminate damage to the model. For models working on MNIST dataset, it was possible to exhaustively flip every bit of every model parameter and measure RAD on the entire validation set. However, for larger models, this was not possible due to the number of parameters, as such experiment would require significant amount of time and/or processing power. Therefore, for CIFAR10 and ImageNet they defined three types of heuristics – the first one samples 10% of the validation set, the second one flips only the most significant bits of the numbers in IEEE754 floating-point representation, and the third one, used for ImageNet models, samples a fixed number of 20,000 parameters to attack. Several different types of impact were characterized:

- **Impact of number format representation:** impact of the bit-flip position, flip direction, and parameter sign;
- **Impact of the model:** impact of the layer width, activation function, dropout and normalization, model architecture.

Two different scenarios were considered in terms of attacker knowledge: surgical attack, where the adversary can precisely flip certain bit of the target parameter, and blind attack, where the bit position of the target parameter is random.

The results show that approximately 40-50% of DNN parameters are vulnerable to single bit-flips and can lead to ≈10% drop in accuracy. If a strong adversary is considered, capable of inducing faults remotely by using Rowhammer, it is possible to reach accuracy drop of 99% in the blind attacker scenario.

1.3.3 Countermeasures

Fault injection countermeasures can be deployed at different levels of the design – model architecture, software implementation, and hardware layer. In this part, we will discuss each of these in more detail.

Model architecture. Neural networks contain vast amount of interconnected nodes. Because of their working principle, not all of them are activated for every input. Therefore, if faults are injected into nodes that are unused in the current execution, there will not be any outcome [43]. This behavior is known as partial fault tolerance and it was shown that neural network implementations of cryptographic operations can make them more resistant to faults than standard implementations [2].

The more redundancy is in the network, the better fault tolerance can be achieved, at the cost of higher memory usage and computation complexity.

Software implementation. Redundancy and checks can be added in the model computation on the software level. A naïve approach would be to repeat the computation two or more times and then compare the results. If they are not equal, the device might have been tampered with. Redundant instruction sequences can protect against pre-defined number of faults [41]. Data within the instructions can be arranged in a redundant way that will protect against both data corruption and instruction skips [44]. Non-linear codes can be used to implement the operations that allow protection against multiple bit faults per operation [7].

Hardware Layer. Error detection and correction codes can be efficiently implemented in hardware. Computational circuits, or parts of them can be implemented in parallel and majority voting can be utilized to prevent outputting the faulty result [10]. Additional circuits can be deployed to detect voltage variations caused by fault injection. These circuits can raise an alarm and a pre-defined action to prevent the information leakage or release of incorrect output can be taken [23].

Additionally, there are physical measures that can be applied to prevent tampering, such as special shielding of the chip or erasing of memory if the chip package is damaged.

1.4 Conclusion

We have surveyed known physical attacks used for the purpose of reverse engineering ML models on a range of platforms and discussed possible countermeasures. The results published so far demonstrate that stealing the models in this way (as possible IP) is a clear and present threat. Specific use cases and applications on various edge and IoT devices should be carefully examined against those threats and accordingly protected.

1.5 Open Research Problems

Powerful adversaries today include those exploiting side-channel leakage from implementations on ML models and the ability to actively disturb the device's operations. Combining the two poses even more challenge to the engineering efforts in designing adequate defenses. The countermeasures considered so far are mainly from the crypto/security applications, which makes them sub-optimal. On top, typical overheads in resources such as power/energy makes those defenses often unsuitable for low-end devices. As the adversaries are becoming ever more powerful and knowledgeable, it is necessary to revisit the design cycle and make it open at various phases such that the results of preliminary security evaluation can still be fed back to the implementations.

We see future works going more into directions of ML-specific countermeasures and new frameworks to evaluate the leakages before the models are put into the field.

Acknowledgement

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Programme SASPRO 2 COFUND Marie Sklodowska-Curie grant agreement No. 945478.

References

1. Agoyan, M., Dutertre, J.M., Mirbaha, A.P., Naccache, D., Ribotta, A.L., Tria, A.: How to flip a bit? In: On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International. pp. 235–239. IEEE (2010)
2. Alam, M., Bag, A., Roy, D.B., Jap, D., Breier, J., Bhasin, S., Mukhopadhyay, D.: Enhancing fault tolerance of neural networks for security-critical applications. arXiv preprint arXiv:1902.04560 (2019)
3. Alam, M., Mukhopadhyay, D.: How secure are deep learning algorithms from side-channel based reverse engineering? In: Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019. p. 226. ACM (2019), https://doi.org/10.1145/3316781.3322465
4. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019. pp. 515–532. USENIX Association (2019), https://www.usenix.org/conference/usenixsecurity19/presentation/batina
5. Batina, L., Bhasin, S., Jap, D., Picek, S.: Poster: Recovering the input of neural networks via single shot side-channel attacks. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. pp. 2657–2659. ACM (2019), https://doi.org/10.1145/3319535.3363280
6. Breier, J., Hou, X., Jap, D., Ma, L., Bhasin, S., Liu, Y.: Practical fault attack on deep neural networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2204–2206 (2018)
7. Breier, J., Hou, X., Liu, Y.: On evaluating fault resilient encoding schemes in software. IEEE Transactions on Dependable and Secure Computing (2019)
8. Breier, J., Jap, D., Chen, C.N.: Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on aes. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security. pp. 99–103. ACM (2015)
9. Breier, J., Jap, D., Hou, X., Bhasin, S., Liu, Y.: Sniff: Reverse engineering of neural networks with fault attacks. arXiv preprint arXiv:2002.11021 (2020)
10. Breier, J., Khairallah, M., Hou, X., Liu, Y.: A countermeasure against statistical ineffective fault analysis. IEEE Transactions on Circuits and Systems II: Express Briefs (2020)
11. Cao, X., Gong, N.Z.: Mitigating evasion attacks to deep neural networks via region-based classification. In: Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017. pp. 278–287. ACM (2017), https://doi.org/10.1145/3134600.3134606
12. Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., Mukhopadhyay, D.: Adversarial attacks and defences: A survey. arXiv preprint arXiv:1810.00069 (2018)

13. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
14. Dennis, D.K., Gaurkar, Y., Gopinath, S., Gupta, C., Jain, M., Kumar, A., Kusupati, A., Lovett, C., Patil, S.G., Simhadri, H.V.: Edgeml: Machine learning for resource-constrained edge devices. URL https://github. com/Microsoft/EdgeML (2020)
15. Dong, G., Wang, P., Chen, P., Gu, R., Hu, H.: Floating-point multiplication timing attack on deep neural network. In: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), Tianjin, China, August 9-11, 2019. pp. 155–161. IEEE (2019), https://doi.org/10.1109/SmartIoT.2019.00032
16. Dubey, A., Cammarota, R., Aysu, A.: Maskednet: A pathway for secure inference against power side-channel attacks. CoRR abs/1910.13063 (2019), http://arxiv.org/abs/1910.13063
17. Duddu, V., Samanta, D., Rao, D.V., Balas, V.E.: Stealing neural networks via timing side channels. CoRR abs/1812.11720 (2018), http://arxiv.org/abs/1812.11720
18. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1322–1333 (2015)
19. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016), http://proceedings.mlr.press/v48/gilad-bachrach16.html
20. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv:1412.6572 (2014)
21. Guillen, O.M., Gruber, M., De Santis, F.: Low-cost setup for localized semi-invasive optical fault injection attacks. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 207–222. Springer (2017)
22. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 770–778. IEEE Computer Society (2016), https://doi.org/10.1109/CVPR.2016.90
23. He, W., Breier, J., Bhasin, S., Miura, N., Nagata, M.: An fpga-compatible pll-based sensor against fault injection attack. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 39–40. IEEE (2017)
24. Hong, S., Davinroy, M., Kaya, Y., Locke, S.N., Rackow, I., Kulda, K., Dachman-Soled, D., Dumitras, T.: Security analysis of deep neural networks operating in the presence of cache side-channel attacks. CoRR abs/1810.03487 (2018), http://arxiv.org/abs/1810.03487
25. Hong, S., Frigo, P., Kaya, Y., Giuffrida, C., Dumitraș, T.: Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In: 28th {USENIX} Security Symposium ({USENIX} Security 19). pp. 497–514 (2019)
26. Hu, X., Liang, L., Deng, L., Li, S., Xie, X., Ji, Y., Ding, Y., Liu, C., Sherwood, T., Xie, Y.: Neural network model extraction attacks in edge devices by hearing architectural hints. CoRR abs/1903.03916 (2019), http://arxiv.org/abs/1903.03916
27. Hua, W., Zhang, Z., Suh, G.E.: Reverse engineering convolutional neural networks through side-channel information leaks. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018. pp. 4:1–4:6. ACM (2018), https://doi.org/10.1145/3195970.3196105
28. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. arXiv:1602.07360 (2016)
29. Jovic, A., Jap, D., Papachristodoulou, L., Heuser, A.: Traditional machine learning methods for side-channel analysis. In: Security and Artificial Intelligence: A Crossdisciplinary Approach, pp. 25–47. Springer (2022)

30. Joye, M., Tunstall, M.: Fault analysis in cryptography, vol. 147. Springer (2012)
31. Juuti, M., Szyller, S., Dmitrenko, A., Marchal, S., Asokan, N.: PRADA: protecting against DNN model stealing attacks. CoRR abs/1805.02628 (2018), http://arxiv.org/abs/1805.02628
32. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. ACM SIGARCH Computer Architecture News 42(3), 361–372 (2014)
33. Krček, M., Li, H., Paguada, S., Rioja, U., Wu, L., Perin, G., Chmielewski, Ł.: Deep learning on side-channel analysis. In: Security and Artificial Intelligence: A Crossdisciplinary Approach, pp. 48–71. Springer (2022)
34. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
35. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. pp. 1106–1114 (2012), https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
36. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998)
37. Lee, T., Edwards, B., Molloy, I., Su, D.: Defending against neural network model stealing attacks using deceptive perturbations. In: 2019 IEEE Security and Privacy Workshops, SP Workshops 2019, San Francisco, CA, USA, May 19-23, 2019. pp. 43–49. IEEE (2019), https://doi.org/10.1109/SPW.2019.00020
38. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 605–622. IEEE Computer Society (2015), https://doi.org/10.1109/SP.2015.43
39. Liu, Y., Wei, L., Luo, B., Xu, Q.: Fault injection attack on deep neural network. In: Proceedings of the 36th International Conference on Computer-Aided Design. pp. 131–138. IEEE Press (2017)
40. Mentens, N., Gierlichs, B., Verbauwhede, I.: Power and fault analysis resistance in hardware through dynamic reconfiguration. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5154, pp. 346–362. Springer (2008), https://doi.org/10.1007/978-3-540-85053-3_22
41. Moro, N., Heydemann, K., Encrenaz, E., Robisson, B.: Formal verification of a software countermeasure against instruction skip attacks. Journal of Cryptographic Engineering 4(3), 145–156 (2014)
42. Murvay, P.S., Groza, B.: Dos attacks on controller area networks by fault injections from the software layer. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. pp. 1–10 (2017)
43. Neggaz, M.A., Alouani, I., Niar, S., Kurdahi, F.: Are cnns reliable enough for critical applications? an exploratory study. IEEE Design & Test (2019)
44. Patrick, C., Yuce, B., Ghalaty, N.F., Schaumont, P.: Lightweight fault attack resistance in software using intra-instruction redundancy. In: International Conference on Selected Areas in Cryptography. pp. 231–244. Springer (2016)
45. Reparaz, O., Balasch, J., Verbauwhede, I.: Dude, is my code constant time? In: Atienza, D., Natale, G.D. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017. pp. 1697–1702. IEEE (2017), https://doi.org/10.23919/DATE.2017.7927267
46. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision 115(3), 211–252 (2015)
47. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy, SP 2017,

San Jose, CA, USA, May 22-26, 2017. pp. 3–18. IEEE Computer Society (2017), https://doi.org/10.1109/SP.2017.41
48. 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, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1409.1556
49. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
50. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
51. Tiri, K., Verbauwhede, I.: A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In: 2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France. pp. 246–251. IEEE Computer Society (2004), https://doi.org/10.1109/DATE.2004.1268856
52. Torres-Huitzil, C., Girau, B.: Fault and error tolerance in neural networks: A review. IEEE Access 5, 17322–17341 (2017)
53. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: 25th {USENIX} Security Symposium ({USENIX} Security 16). pp. 601–618 (2016)
54. Velasco-Montero, D., Fernández-Berni, J., Carmona-Galán, R., Rodríguez-Vázquez, Á.: Performance analysis of real-time dnn inference on raspberry pi. In: Real-Time Image and Video Processing 2018. vol. 10670, p. 106700F. International Society for Optics and Photonics (2018)
55. Wei, L., Luo, B., Li, Y., Liu, Y., Xu, Q.: I know what you see: Power side-channel attack on convolutional neural network accelerators. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018. pp. 393–406. ACM (2018), https://doi.org/10.1145/3274694.3274696
56. Yan, M., Fletcher, C.W., Torrellas, J.: Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. CoRR abs/1808.04761 (2018), http://arxiv.org/abs/1808.04761
57. Yarom, Y., Falkner, K.: FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. pp. 719–732. USENIX Association (2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom
58. Yu, H., Ma, H., Yang, K., Zhao, Y., Jin, Y.: Deepem: Deep neural networks model recovery through em side-channel information leakage. In: HOST (2020)
59. Zhao, P., Wang, S., Gongye, C., Wang, Y., Fei, Y., Lin, X.: Fault sneaking attack: A stealthy framework for misleading deep neural networks. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2019)