# FooBaR: Fault Fooling Backdoor Attack on Neural Network Training

Jakub Breier, Xiaolu Hou, Martín Ochoa and Jesus Solano

***Abstract*—Neural network implementations are known to be vulnerable to physical attack vectors such as fault injection attacks. As of now, these attacks were only utilized during the inference phase with the intention to cause a misclassification. In this work, we explore a novel attack paradigm by injecting faults during the *training* phase of a neural network in a way that the resulting network can be attacked during deployment without the necessity of further faulting. In particular, we discuss attacks against ReLU activation functions that make it possible to generate a family of malicious inputs, which are called fooling inputs, to be used at inference time to induce controlled misclassifications. Such malicious inputs are obtained by mathematically solving a system of linear equations that would cause a particular behaviour on the attacked activation functions, similar to the one induced in training through faulting. We call such attacks fooling backdoors as the fault attacks at training phase inject backdoors into the network that allow an attacker to produce fooling inputs. We evaluate our approach against multi-layer perceptron networks and convolutional networks on a popular image classification task obtaining high attack success rates (from 60% to 100%) and high classification confidence when as little as 25 neurons are attacked, while preserving high accuracy on the originally intended classification task.**

***Keywords*—Neural networks, deep learning, adversarial attacks, fault attacks**

## I. Introduction

The rapid emergence of deep learning in the past decade has revealed security and safety issues related to their usage. Adversarial learning has quickly become a popular topic in the security community, as it was shown that even slight perturbations to the input data can fool the deep learning-based classifiers [57]. Such attacks could have disastrous consequences, as illustrated for instance by attacks against computer vision systems installed in autonomous cars, which could potentially cause physical accidents due to misclassifications [21].

J. Breier is with Silicon Austria Labs, TU-Graz SAL DES Lab and Graz University of Technology, Graz, Austria. E-mail: jbreier@jbreier.com

X. Hou is with Faculty of Informatics and Information Technologies, Slovak University of Technology, Slovakia. E-mail: houxiaolu.email@gmail.com

Martín Ochoa is with the Department of Computer Science, ETH Zurich, Zurich, Switzerland. E-mail: martin.ochoa@inf.ethz.ch

Jesus Solano is with Appgate Inc., Bogotá, Colombia. E-mail: jesus.solano.g@gmail.com

This work has been supported in parts by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems. 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. This work was supported by the Slovak Research and Development Agency under the Contract no. SK-SRB-21-0059.

Moreover, recent trends tend to move the models from the cloud to edge computing devices, reducing the data transfer requirements and delays caused by slow or unavailable networks [37]. This trend, however, enables hardware attack vectors that are normally not possible when the computation is done in the cloud [63]. Attacks that allow parameter extraction through side-channels [7] or misclassification by fault injection attacks [42] were shown to be viable security threats that need to be taken into account.

While several works focused on faulting the inference phase [42], [26], [14], [65], which is related to *evasion attacks* in terms of the outcome [8], no work has been published on faulting the training phase up to date. Attacking the training phase is related to *poisoning attacks* in the adversarial learning domain [9], and the goal in such scenarios is generally to create something similar to a trojan horse, which is activated by a specific input during the inference phase. Such inputs generate a controlled effect, such as a targeted classification (for instance a stop sign with a trigger is misclassified to 'go straight' sign [54]).

The general research question we tackle in this work is thus: *Is it possible for an attacker to use fault attacks in the training phase of a deep neural network such that they can bias a resulting model in a way that can be exploited during deployment without the necessity for further fault attacks at inference time?*

In this work, we assume fault attacks on the training phase of the neural network, thus bridging the gap between fault injection attacks and poisoning/trojaning attacks. By faulting specific intermediate values during the computation, our attack can force the classifier to behave in a specified way during the inference phase, depending on the attacker's goals, while preserving the original network classification accuracy.

**Motivation.** Poisoning attacks assume that the attacker can fully control the training process. She can alter the input data and the labels in a way that the model learns to react to a certain trigger and misclassifies the output when such trigger is present during the inference [44].

Our attack, on the other hand, keeps the inputs to the training process intact. The attacker can only observe the inputs, and based on this, tampers with the environmental parameters of the device that executes the computation. This tampering can be done in various ways, by clock/voltage glitches, electromagnetic pulses, lasers, or by a remote Rowhammer attack [13]. Additionally, voltage glitches were shown to be possible in a remote way on FPGAs, which are often used for accelerating the training [34].

In particular, we focus on faulting the ReLU (Rectifier Linear Unit) activation functions which are common in several neural

Fig. 1. High level overview of the FooBaR attack, constraint solving generation and exploitation at deployment.

network architectures [35], [55]. The main idea is to bias the training in a way that malicious inputs can be easily recovered by the attacker to be used at inference time, without having to fault the network again. We discuss a generic approach based on constraint solving to generate such inputs which we call fooling inputs Our approach has some intersection with non-poisoning-based backdoor attacks [38]. However, we do not use a trigger in a traditional sense of embedding certain patterns into the input. Instead, we allow the attacker to generate fooling inputs from random samples.

We evaluate our approach on two different networks applied to the MNIST digit classification dataset [36]. As a result of our evaluation we obtain high attack success rates (defined as the percentage of candidate fooling inputs that are classified according to an attacker's target), even when only a partial number of ReLU activation functions are faulted at training time (as low as 20% or 25 activation functions).

Moreover, the attack strategy is stealthy in the following sense. On the one hand, the attacked network preserves high accuracy levels, indistinguishable from the accuracy of a non-attacked network. On the other hand, generated inputs are not easy to blacklist, since they can be generated using random images as a base pattern (for instance commonly used icons or other pictures), which together with the constraints defined by the faulted network, appear to be common pictures with some noise.

Finally, based on our observations and experiments, we also discuss countermeasures and mitigation strategies to the proposed attacks.

Our contributions.

- We explore a novel faulting attack targeting ReLU activation functions at the training phase of a neural network.
- To exploit the attacked model at inference time, we formulate the problem of fooling input generation as a constraint solving problem and show that attacks can be effectively generated using arbitrary base inputs.
- We perform experimental verification of the proposed attack obtaining high attack success rates and share our code as open source[1]
- Based on our observations and analysis, we discuss countermeasures to the presented attack.

Main idea. The main idea of our work can be explained as follows:

1) Fault the training phase of the model. In this step, an attacker faults strategically selected parts of a network. When inputs from a target class are being fed to the neural network during training, faults are injected to the ReLU activation function in one of the hidden layers.
2) The attacker then generates the fooling image(s) by solving constraints based on partial information of the faulted network.
3) When the fooling image is used as an input to the backdoored network, the target class is expected to be the output with high confidence by construction.

A graphical overview of these steps is depicted in Figure 1, where for illustration purposes an image is used to represent the input. The approach is however more general and does not require the network to be associated with an image classification task.

We would like to note that the insertion of the backdoor can be also done in some other (e.g., software) way rather than by a fault attack. However, in this work we focus on the fault attack scenario.

The rest of the paper is organized as follows. We review key background concepts and compare against related work

[1] https://github.com/martin-ochoa/foobar

in Sect. II. We then present the general approach in Sect. III that conveys our attack strategy in a general way. We then evaluate our approach on two concrete network architectures, on a popular image classification case study, a multi-layer perceptron and a convolutional network in Sect. IV, where we also discuss results, limitations, and possible countermeasures. We conclude in Sect. V.

## II. Background and Related Work

In this section we briefly review some important preliminaries on neural networks and faulting attacks, and position ourselves with respect to relevant related work. We also introduce the terminologies used in the rest of the paper.

### A. Neural Networks

Neural networks are computing units designed on the basis of biological neural networks. They are utilized for solving classification problems in various domains: malware detection [38], network intrusion detection [29], voice authentication [11], etc. A neural network normally consists of an input layer, one or more hidden layers and an output layer. Each neuron computes a weighted sum of results from neurons from the previous layer, followed by a non-linear activation function. The weights for each layer are determined during the training. The training process of neural networks makes use of a backpropagation algorithm [25]. The training data are divided into batches. The weights are first randomly generated. For each batch of training data, the prediction of the current network for each data is evaluated and compared to its label. A predefined loss function is then calculated, based on the predictions and the correct labels for this batch. The gradient of the loss function is computed with respect to each network parameter and the parameters are updated slightly to reduce the loss on this batch. One epoch of training corresponds to a single passing through all the batches. The training process normally consists of several epochs.

One of the most commonly used activation functions is Rectified Linear Unit or ReLU defined as follows:

$$\mathrm{ReLU}(x) = \max\{0, x\}.$$

It is a piece-wise linear function which preserves properties that make the optimization of the model easier. As shown in [14], with a fault attack, the output of this activation function can be set to the 0 value, regardless of the input. That work demonstrated the attack during inference time to cause an untargeted misclassification. In our work, we exploit this attack against ReLUs during the training phase of a neural network. Even though the focus of this work is attacking against ReLUs, our approach could be extended to other activation functions such as Sigmoid, hypherbolic Tangent or LeakyRelu, but constraint solving would be more challenging in those scenarios. In particular, notice that the search space (and consequently the feasibility of finding a suitable solution) decreases for activation functions different from ReLU as we are exploiting the fact that ReLU output is zero for all negative inputs, which means that a wider range of inputs could solve the optimization problem.

### B. Adversarial attacks

Adversarial attack on neural networks is a well-studied topic. There are various kinds of attacks depending on the attacker assumptions and goals. For a detailed taxonomy of adversarial learning, we refer the reader to [10].

**Adversarial examples.** One of the earliest attacks discovered is an adversarial examples attack causing a misclassification. The attacker produces adversarial inputs during the inference. Those inputs are almost indistinguishable from natural data and yet classified incorrectly by the network [56], [50]. The attack can also be extended for a targeted misclassification, where the attacker aims to produce adversarial examples that can be misclassified to a target class [48].

In our work, at the inference/deployment phase, the attacker crafts input examples (called fooling inputs) that are very different from any natural data. However, the network will still classify such an input to a target class with a high confidence, due to the fault injected at the training phase, which are referred to as *backdoors* in this paper. While on the other hand, the original network would classify those pictures with low confidence on all classes.

**Poisoning attacks.** One common attack on training data is a poisoning attack. The assumption is that the attacker has the ability to alter a small fraction of the training data and compromise the whole network [61]. The poisoning examples can be generated using, e.g., a gradient-ascent-based methodology [46]. The goal of the attack can be degrading the overall efficacy of the model [62], targeted misclassifications [19], etc.

In this work, we do not assume the attacker has any active control over training data. We assume the attacker can observe the training data during the training phase, i.e., we allow only a passive observation compared to poisoning attacks.

**Backdoor attacks and countermeasures.** Another line of attacks on neural networks aims to inject a backdoor into a neural network model such that it can be triggered to misclassify inputs with certain embedded patterns to a target class of the attacker's choice [23], [40]. Such attacks are also sometimes called trojan attacks [44]. Backdoor attacks can be generally categorized into poisoning-based and non-poisoning-based [38]. While the first category changes the training inputs, the second one directly modifies the model parameters during the training phase. Both categories assume a trigger during the inference phase to activate the backdoor.

The threat of backdoor attacks on neural networks was introduced by Gu, et al. in [23]. They assumed an outsourced training scenario and the attacker, which provides a training for the neural network, has a full control over the training process. In [44], the attacker retrains the model with external training data to cause the neural network to make targeted misclassifications. Thus, the attacker does not have a control over training data, but they have a full access to the neural network. [40] and [19] discussed attacks on a weaker assumption where the attacker does not have the knowledge of the model or training data. Backdoor attacks on graph neural networks were investigated in [64]. Pre-trained image encoders were a target of another work [30].

Following the trigger-based backdoor attacks, countermeasures for such attacks have also been developed [58], [41], [18],

Fig. 2. An exemplary setup for the laser fault injection attack.

[43]. As they are designed to counter attacks with the above-mentioned goals, they assume the malicious trigger is added to a natural input. For a comprehensive overview of backdoor attacks and countermeasures, we refer interested reader to [24].

A related non-malicious category to backdoor attacks are watermarking techniques which are used to protect intellectual property rights [39].

In our work, during the inference phase, we do not generate a malicious trigger to be recognized by the network with backdoors. Instead, the backdoor allows the attacker to generate fooling inputs from any arbitrary input and such fooling inputs will be misclassified to a target class. Thus the countermeasures aiming to protect trigger-based backdoor attacks do not apply in our case. For the attacker capability, we do not assume she has any control over the training data or the training process. However we assume the attacker can observe the training data and inject faults during the training.

### C. Fault Attacks

Fault attacks, also called fault injection attacks, are one of the major physical attack threats against cryptographic implementations [11]. By influencing the device that performs the encryption, the attacker can cause errors during the computation. Then, various analysis techniques can be used to recover the secrets by observing the outcomes of the fault. It was shown that a single data fault during the AES encryption can recover the entire secret key [5, 8]. An instruction skip attack was utilized to skip the `AddRoundKey` routine of the AES to skip the last key addition, and then trivially recover the last round key [16]. It is assumed that every symmetric cipher is vulnerable to a fault attack due to non-linear components used in the round function [4].

An exemplary laser fault injection setup is depicted in Fig. 2. Device under test (DUT) is exposed to the laser radiation and placed on an XYZ positioning table which allows to precisely control the target area on the chip. During the execution, a trigger signal is sent from the device (this does not need to be an artificial trigger, it can be for example the start of the neural network computation), and the activation signal is sent to the laser source. Laser source parameters are normally set beforehand from the PC, e.g., duration of the irradiation, delay from the trigger, laser power.

While these attacks were originally designed for targeting embedded devices, such as smart cards, methods like Rowhammer [12] and VoltJockey [52] can cause hardware faults remotely. Leaving no trace in the security logging systems, remote fault attacks are trickier but stealthier alternative to standard software attacks.

Practicality of the fault injection attacks is a highly discussed topic. Some of the methods are relatively practical, for example the Plundervolt attack [47] and V0ltpwn [32] corrupt instructions on modern processors by utilizing software-based glitches that can be done remotely. On the other side of the spectrum, we have faults induced by very expensive devices that require experienced personnel and a complete control over the device, such as nanofocused X-ray beams [25]. Laser fault attacks [15] are somewhere in the middle, as the equipment is becoming relatively reachable price-wise and companies provide operating frameworks that do not need any special expertise to control the lasers. However, the attacker still needs a physical access to the device.

In this work we assume the attacker can inject faults into a neural network training process. As the training is generally done on powerful servers with limited physical access, the attack execution would be normally carried out by a remote fault injection technique. However, similar outcome can be achieved by a software fault injection and also localized fault injection techniques, such as using EM/laser equipment.

## III. Fault Attacks on Neural Networks

Faulting neural networks in a malicious way is a relatively new area, first introduced by Liu et al. in 2017 [22]. The authors simulated injection of faults during the model execution to achieve the misclassification. The work was followed by an experimental fault attack carried out by a laser equipment in a laboratory setting by Breier et al. [14], where it was shown that the output of an activation function can be corrupted by a fault. The paper was later extended to show various attack strategies that can be derived by utilizing the knowledge from the experimental results [27]. A study to find out the worst case scenario caused by a single bit fault was presented by Hong et al. [26] where the authors showed degradation of classification accuracy with a single bit flip. Bai et al. [3] followed on the idea of flipping weight bits to misclassify the output into a target class.

A trojaning attack, called Targeted Bit Trojan (TBT), using bit flips in the memory was presented by Rakin et al. [33]. They utilized the Rowhammer technique to flip the weight bits which contribute to a target class and then, they generated a specific trigger that can be inserted in the input to perform the misclassification. Such technique can be, however, thwarted by standard integrity checks on the model stored in the main memory. Once the integrity check fails, the model is restored from a secure storage. As our attack works at runtime during the testing phase, such integrity check would not help.

Another line of work focuses on model extraction attacks [28], where the attacker aims to recover the parameters of a neural network with as much precision as possible. Breier et al. [17] demonstrated a fault attack during the inference that can recover

TABLE I. COMPARISON OF FAULT INJECTION ATTACKS ON NEURAL NETWORKS.

| Work | Type of fault | Phase | Outcome |
|---|---|---|---|
| [42] | Bit flip | Inference | Misclassification |
| [14] | Instruction skip | Inference | Misclassification |
| [26] | Bit flip | Inference | Degradation |
| [17] | Bit flip | Inference | Model extraction |
| [53] | Bit flip | Inference | Trojan insertion |
| [3] | Bit flip | Inference | Targeted misclassification |
| This work | Instruction skip | Training | Trojan insertion |

the parameters with the exact precision for deep-layer feature extractor networks [60].

Our method, fooling backdoor, works with the same premise that the attacker is able to inject a fault that changes the processed values. However, unlike previous works, we target the training phase to inject a backdoor into the network that can be used later during the inference phase. We utilize the instruction skip model proposed in [27], as it is the easiest model to be achieved in practice. Skipping instructions can be done by a clock or a voltage glitch, using an equipment with a cost below $100 [12].

In sum, to the best of our knowledge we are the first to propose fault attacks at the training phase, which are exploitable at deployment without the necessity of further faulting. A comparison of different works from this section is provided in Table I.

### E. Terminology

Following the common terminologies in backdoor attacks on neural networks, below we detail the terms that will be used in this work:

- **Benign model** is the model trained without fault attacks.
- **Infected model** is the model trained with backdoor injected by fault attacks.
- **Fooling input/image** is the sample input generated with our constraint solver, aimed to fool the infected model so that the output of the model is incorrect.
- **Base pattern** is the sample used to generate fooling input from. It can be a random sample outside of the problem domain.
- **Source class** is the output of benign model given an input.
- **Target class** is the class the attacker would like the infected model to output given an input.
- **Attack success rate** is the percentage of a set of generated fooling inputs that are classified to target class by the infected model.
- **Benign accuracy** is the test accuracy of the benign model.

## III. APPROACH

In this section we will describe the general attacker strategy and attacker goals in more detail. The attacker considered is interested in corrupting the training process of a neural network with the goal to be able to exploit specific fooling backdoors, while at the same time remaining as stealthy as possible. This section will discuss the attack strategy abstractly, while we will instantiate it on two realistic neural networks in Sect. IV.

Recall that as stated in the introduction, the general research question we want to answer is:

GRQ: Is it possible for an attacker to use fault attacks in the training phase of a deep neural network such that they can bias the infected model in a way that it can be exploited during deployment without the necessity for further fault attacks?

Moreover, if we can answer this question positively, we would like to know:

- Is it possible to carry out such an attack without affecting the benign accuracy?
- Can we attack a model with a family of fooling inputs that will be difficult to blacklist?
- Is there a way to minimize the number of fault attacks during training while maintaining a high attack success rate?

### A. System and attacker model

As previously discussed, we assume the attacker will have physical access to a device performing a neural network training. We assume the attacker can observe the inputs to the training process and selectively inject faults during the computation. For instance, an attacker targeting a digit recognition network can selectively attack the network while an image in the class '8' is given as input.

Additionally, after the network is trained with the backdoor injected, we assume an attacker can recover a subset of the infected model's weights and biases, for instance by using side-channel analysis [37]. We do not require however that an attacker can necessarily retrieve *all* weights, since as we will discuss in the following this will be in general not necessary.

**Definition 1** (Attacker goal)**.** Given a target class $c$, an attacker that can perform fault attacks on a classifier during training, thus producing an infected classifier $C'$, would like to be able to craft fooling inputs $I = \{\tilde{i}_1, \ldots, \tilde{i}_n\}$ such that when executed, $C'(\tilde{i}_j) = c$ for the infected classifier $C'$. Note that the attacker may not perform further faulting during execution of $C'$ after training. We say such attacks are non-trivial if the benign classifier $C$ would compute $C(\tilde{i}_j) = c'$ for $c' \neq c$.

In other words, an attacker aims to corrupt the training process such that it will be easy to generate fooling inputs that will be classified to a predefined target class. A trivial attack would be for the attacker to simply choose inputs in the problem domain that are very close to the target class, for instance a slightly modified image of an '8' in the image classification example. A non-trivial attack would be to craft inputs that are possibly not even in the problem domain and still be classified as the target class. For instance, an image similar to a plane icon that is classified as a chosen target digit in the digit classification example would be a non-trivial attack. Moreover, this attack should be as stealthy as possible, in the sense that it is difficult to detect that a classifier was corrupted during training. In addition, it should be difficult to automatically blacklist the fooling inputs $\tilde{i}_j$ when the classifier is deployed after being trained. By this requirement we mean that an attacker should have plenty of choices in generating multiple fooling backdoors with various degrees of randomness

in order to bypass eventual blacklists that block some previously seen backdoors. In other words, the accuracy of the infected classifier $C'$ should be close to the benign accuracy and there should be no obvious way to characterize the resulting fooling images.

Note that we assume the attacker to be able to inject faults only at the training phase, but not at inference time.

### B. Faulting strategy

In order to fulfill the goals in Definition 1, we will propose the following strategy. First, an attacker will choose a hidden layer in the network that is followed by ReLU activation functions. The attacker will then fault those ReLUs when training samples of a target class $c$ are given as an input to the network. For instance, consider our running example, the digit classification using the MNIST dataset and a target class $c = 8$. The attacker will fault all ReLUs after the first deep layer in a multi-layer perceptron architecture as depicted in Figure 4. That is, for each ReLU in the layer, the attacker will fault the output to 0 as depicted in Figure 3, independent of the input. Such faults can be achieved as described in [14]. Notice that the attacker could also choose to fault the ReLUs for a set of target classes $(c_1, \ldots, c_n)$, but in that scenario the attacker cannot control which of the chosen target classes is going to be predicted with the highest confidence. Therefore, in this work we focus on the scenario of a single target class.

Given that it is also one of the attacker's goals to remain stealthy, it is important that the classifier trained under this attack has an accuracy that is similar (overall and for each class) to the benign accuracy. Clearly, this implies in particular that faulting cannot be performed for all the example inputs in the target class, since otherwise the network will not be able to recognize testing samples in that class. Therefore, in the proposed strategy, an attacker will choose a certain subset of the training examples in the target class to be faulted. To a certain extent, the percentage of training samples that are faulted determines how powerful the attack is. In that regard, the more samples are faulted, the easier the attack is. But at the same time, the more samples are faulted the less stealthy the attack is as the neural network is likely to misclassify ground-truth images. Consequently, we define a parameter that denotes the fraction of images faulted in the target class.

Moreover, an attacker who wants to minimize the attack effort is also interested in faulting as few ReLUs as possible. One research question is thus: What is the minimum number of ReLUs to be attacked in a given layer that yields a successful and stealthy attack under Definition 1? To do so, we propose to explore a faulting strategy that considers an increasing number of faulted ReLUs per experiment iteration. For instance, one can start faulting 10% of ReLUs after a given hidden layer, then 20% and so on until faulting all the ReLUs in that layer.

This results to a combinatorial explosion of target ReLUs since there are various ways to pick a partial number of ReLUs. We will discuss some choices to mitigate this computational problem in Section IV. On the one hand, depending on the network architecture, it is possible to argue for generality when an arbitrary partial number of ReLUs is chosen (multi-layer perceptron). On the other hand, when this is not possible, we believe the results presented will give an intuition on the general attack impact when a limited number of activation functions is faulted.

Fig. 3. Behavior of ReLU under faults. Gray color shows the original ReLU output, while the magenta color shows the faulted output. Note that for $x \leq 0$, the output stays the same.

### C. Fooling images generation strategy

To exploit the fault attacks performed in the training phase, an attacker needs to be able to derive fooling inputs that fulfill Definition 1. Intuitively, an attacker wants to achieve the same behaviour (ReLUs outputting 0s at a given layer) with the hope of achieving a misclassification to the target class $c$. This could be achieved for instance by faulting again at inference time, but this would require very strong attacker capabilities.

Instead, we propose to derive fooling inputs which can be computed mathematically by means of constraint solving. We assume the attacker is able to derive some of the weights of the network (for instance by side-channel analysis [3]). Note that in our attack strategy, an attacker only needs to learn the weights of the network that are necessary to compute inputs resulting in output of 0 for the faulted ReLUs. This set of weights needed to compute the fooling inputs can be as small as the weights corresponding to 25 neurons as we will see in Section IV.

Example: Constraints on MLPs. For instance, assume the attacker faults all ReLUs after the first hidden layer of an MLP. The inputs to those ReLUs are:

$$x \cdot w_j + b_j;$$

where $x$ is the input vector, $w_j$ is the weight vector of the $j$-th neuron in the first layer and $b_j$ is the bias of the neuron. Assuming biases are small, an attack could be simply a zero vector since $0 \cdot w_j + b_j \approx 0$ regardless of the weights. This will trigger the desired behaviour of all ReLUs outputting a value close to 0.

However, this type of attack is limited since it can be easily blacklisted as a border case (a black image for instance), and harms the attacker's goal to be as stealthy as possible. So an attacker might attempt to derive several non-zero fooling images by means of constraint solving. In other words, the attacker aims to find a set $I$ such that for all $I' \in I$ and all $w_j$:

Fig. 4. Attacker targets ReLUs in a given layer. In this figure, there is an input layer, a first hidden layer with ReLU as activation functions. The attack during training are carried out on those ReLU activation functions.

$$\sum_j w_j + b_j \leq 0.$$

Note that by the definition of ReLU, any negative or zero input will result in a zero output, which is the attacker's goal. If an attacker has only faulted a subset of the ReLUs in a given layer, the number of constraints will be small (corresponding to the attacked neurons). Note also that in this example we have a linear constraint, which is typically easier to handle computationally than more complex non-linear constraints.

Naturally, more complex architectures and attacking deeper layers will result in more complex constraints. For instance, if there is a convolutional layer before the ReLU layer, this results in a larger constraint set (that would also be linear in some cases). A more complex situation would be to attack activation functions after non-linear layers, for instance after the ReLU activation layer or layers containing the softmax function. In that case, constraint solving would be more challenging.

Example: Constraints on convolutional networks. On a convolutional network, commonly used in computer vision tasks, there is typically a concept of a filter (represented as an $n \times n$ matrix $F$) or a family of filters that are multiplied with submatrices of an input image. In this case, if we want to impose constraints on activation functions after convolutional layers, we would need to solve:

$$\forall M_i \in \text{SubMatrices}(I) : M_i * F \leq 0$$

where SubMatrices is the set of chosen matrices for the convolution. These matrices are typically the $n \times n$ matrices 'surrounding' all pixels in the original images. For instance in a $3 \times 3$ image there would be $9$ $3 \times 3$ sub-matrices, one for each pixel, corresponding to a given pixel and its neighbors, where pixels on the image border are considered to have 0 surrounding them outside the original image.

Clearly this case imposes more complex constraints, since submatrices are not disjoint but often share multiple pixels. So, a given pixel will end up having multiple constraints to be fulfilled simultaneously. Furthermore, usually a family of filters $F$, consisting of several individual filters, is often used in these architectures.

Independently of the nature of the network and the point where the attack is executed, we can describe a high-level algorithm summarizing the constraint solving strategy as described in Algorithm 1. In this algorithm In this algorithm, input_formula represents the neuron computation. We can solve the constraints defined by the input formulas to obtain a given output (in this case, we are interested in the output 0 which is the result of our fault on ReLU).

**Algorithm 1:** Fooling image generation.

```
Data: Set N of neurons faulted at training
Result: Fooling image
constraints = {};
for n ∈ N do
    input_formula = Input(n);
    constraints = constraints ∪
      [Output(n(input_formula)) == 0];
if constraints is solvable then
    return solution(constraints);
else
    return unsolvable;
```

## IV. Evaluation

To evaluate our approach, we create a framework that (1) trains and tests a neural network under normal conditions and under attacks as well; (2) builds a set of fooling inputs based constraint solving and (3) tests the attack success rate of those fooling inputs. We first perform a thorough exploration on the design and performance of FooBaR on fully connected neural networks. Then, we extend our analysis to different kinds of neural networks (i.e Convolutional Neural Networks). Finally, we propose a set of countermeasures that defend systems against FooBaR.

The code used for this evaluation is available in the form of open source Jupyter notebooks containing Python code[2]. The code was tested on a MacBook Pro with an Intel Core i7 at 2.6GHZ AND 16GB of RAM.

### A. Attacks on MLPs

MNIST digit classification [20] has been widely studied in the literature with a wide range of developed network architectures to solve this problem. However, one of the simplest and cheapest way to perform the classification accurately is by using an MLP

[2] https://www.dropbox.com/sh/gjys2sg7xob2e9x/AACB1mWyTQMv8f7R63wlxblia

network after flattening the input image. Following this strategy, we will have as the input to our neural network a 781 ($28 \times 28$) dimensional vector that contains the pixel value in the gray space $[0, 1]$. The MLP is aimed to learn a set of parameters that achieves an accurate classification of the digits' images represented as flatten vectors.

Our analysis is carried out on a fully connected neural network with 3 hidden layers. The first, second, and third hidden layers have 128, 64 and 32 neurons, respectively. The activation functions for all hidden layers are ReLUs. For the last layer we use a softmax activation function.

To evaluate our approach, we attack ReLUs in the first hidden layer as depicted in Figure 4. Note that we do not actually perform physical attacks, but simulate them by programming a neural network data flow that allows us to replace the outputs of chosen ReLUs with 0s depending on the network input. This is a non-trivial effort since it involves programming training and back-propagation in the faulted scenario from scratch.

We have chosen to fault 80% of the training inputs belonging to the target class at random. We have empirically observed that this would preserve the benign accuracy almost intact while also introducing enough bias for successful attacks to be achieved. It is possible that faulting even fewer inputs from the target class would result in powerful attacks but we leave this analysis for future work.

As discussed earlier, minimizing the need for faults is interesting from an attacker's perspective. We perform a sensitivity analysis over various percentages of activation functions (10%, 20% , ... , 100%). The idea is to assess the performance of less costly attacks (faulting as few ReLUs as possible, which also requires knowledge of fewer network weights to perform the attack). We do this for all the possible target classes, which yields $10 \times 10$ attack simulations, each corresponding to an infected model with a given percentage of the ReLUs faulted and a given target class.

Given that there is a combinatorial explosion of choices when attacking a subset of ReLUs in a given layer, without loss of generality, we consider a fixed ordering of the neurons in the attacked layer. This is because for a fully-connected neural network, a fixed but arbitrary order for neurons would not affect the result – before training, neurons in a layer are permutation invariant as every neuron is connected to all the neurons in the previous layer. Thus, we could permute the neurons in the same layer and get exactly the same mathematical model but with a different mapping of indices for each neuron in the layer. Due to the fact that only the indices were changed, and given a seed for the weights initialization, the model parameters will converge to the same value that what was obtained for the non-permuted layer.

Generating fooling images from the problem domain vs. outside of the problem domain. One interesting research question is whether we should consider base patterns sampled from the problem domain (in this case, handwritten digits), or not. Intuitively it could be interesting for an attacker to choose base patterns from the problem domain in order to confuse a classifier. For instance, an attacker may want to produce a fooling input that is visually similar to a 4 but is classified as a 7. We have explored this scenario and noticed that it works successfully for the MNIST case study. Figure 5 is obtained by solving constraints using a 4 as a base pattern and 7 as target class on an infected model. In this case only 10% of ReLUs were faulted and we managed to classify the resulting fooling image as 7 with confidence higher than 90%.

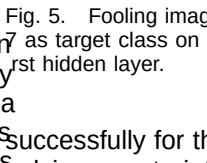

Fig. 5. Fooling image obtained using a handwritten 4 as a base pattern and 7 as target class on an infected network with 10% of faulted ReLUs in the first hidden layer.

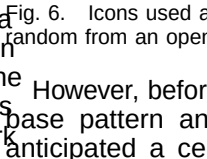

Fig. 6. Icons used as base patterns for fooling images generation, chosen at random from an open source icon dataset.

However, before generalizing this analysis to all the possible base pattern and target class combinations ($10 \times 10$) we anticipated a certain bias given that base patterns were in the problem domain. This bias comes from the fact that the distribution of pixels in the base pattern already gives certain advantage to some attacks (for instance a base pattern 1 is closer to a 7 than to say an 8). Therefore, to have a more interesting evaluation, we decided to pick base patterns from the outside of the problem domain. This would to a degree remove inherent biases and make the attack more general.

For a given infected model we use the subset of weights and biases that correspond to the attacked ReLUs to solve a linear constraint problem (as implemented by the Mixed Integer Linear Programming library in SageMath[3]). In order to have a variety of non-trivial attacks from the outside the problem domain we used the icons in Figure 6 as extra constraints to the solver. Those icons were chosen randomly from an open source icon dataset[4]. In total, we generate 12 fooling images per each network. They consist of 2 images for each of the 5 icons as base patterns (each with a different constraint on the total image weight in order to add more diversity to the fooling image set) and 2 free images (no icon as a base pattern, also each with different total weight). As a result, we obtain fooling images such as the ones depicted in Figure 9 for 10%, 50% and 100% ReLUs attacked with target class 8. As there are 128 neurons in the layer under attack, 10%, 50% and 100% of this layer correspond to 25, 64 and 128 neurons respectively.

[3] https://doc.sagemath.org/html/en/reference/numerical/sage/numerical/mip.html

[4] https://remixicon.com/

Fig. 7. Attack success rate for all 10 target classes when attacking different number of activation functions on an MLP.

Fig. 8. Classification confidence on successful attacks vs. number of activation functions attacked on an MLP.

In this scenario we instantiate Algorithm 1 to solve the particular constraints in our attack, plus the constraints given by the icons as described in Algorithm 2. Given $R$, the set of ReLus that were faulted during training to create the backdoor and the set of base patterns $P$. For each base pattern, we create one fooling image.

There are two constraints for this fooling image: the first is that it is in a neighborhood $d$ from the base pattern (line 3). This distance is considered pixel-wise and although in principle is arbitrary, the smaller the neighborhood of possible values around the original pixel intensity, the closer the resulting image will be to the base pattern. Formally, let $x_j \in [0, 1]$ be pixel intensity of the $j$-th pixel in the base pattern. We constrain the fooling image to be within $\max(x_j - d, 0)$ and $\min(x_j + d, 1)$. Empirically, we observed that $d = 0.7$ was enough for the fooling images to resemble the base patterns while the constraint problem still being solvable. The second constraint is that the resulting outputs of this fooling images for those faulted ReLus should be 0 (line 7).

Computational overhead of constraint solving (Integer Linear programming is theoretically NP-hard [49]. However, the kind of problem instances generated in the MLP setting were solvable in a practical time. Generating one fooling image lasted from 5 seconds (for an attack on 10% of the ReLUs) to 1 minute (for an attack on 100% of the ReLUs).

We measure the attack success rate as the percentage of generated fooling images that are classified by the infected model to the target class. A summary of the attack success rate for various combinations is depicted in Figure 7. Moreover, we depict the average confidence on the successful attacks in Figure 8.

Note that, in principle, some of the constraint systems could be unsolvable. We have observed that most of the evaluated attacks for this scenario are solvable, with a few exceptions in the case of faulting 100% neurons in the first hidden layer. This makes sense since attacking more ReLUs implies more constraints rules and increases the likelihood of adding contradicting constraints. In total, out of 1200 constraints

---

**Algorithm 2:** Fooling image generation.

---

Data: Set $R$ of ReLUs faulted at training, set of base patterns $P$
Result: Set of fooling images
1 solutions = {};
2 for $p \in P$ do
3  constraints = each input pixel $i_j$ must be in a neighborhood of radius $d$ from the corresponding pixel in $p$;
4  for $r \in R$ do
5  $w_j$ = weights of $j$-th neuron;
6  input_formula = $\vec{i} \cdot w_j + b_j$;
7  constraints = constraints $\cup$ [Output(r(input_formula)) == 0 ];
8  if constraints is solvable then
9  solutions = solutions $\cup$ solution(constraints);
10 return solutions;

---

systems (fooling backdoors), there were 34 unsolved instances. Moreover, and crucially for stealthiness, all the 100 infected models had overall accuracy and per-class accuracy indistinguishable from a benign model. Concretely, while the benign accuracy was about 97.3%, all infected models had accuracy between 96.9% and 98% on legitimate test inputs.

### B. Attacks on Convolutional Networks

In recent years, the field of computer vision has witnessed the birth of several deep learning architectures with promising results in image classification and object recognition. In particular, Convolutional Neural Networks (CNNs) have shown exemplary performance in tasks like pattern recognition [1]. Furthermore, several highly popular deep neural networks (e.g., LeNet [36], Alexnet [35], VGG-16 [55], etc) have in common the fact that their building blocks are convolutional layers. Consequently, in order to evaluate our approach effects on

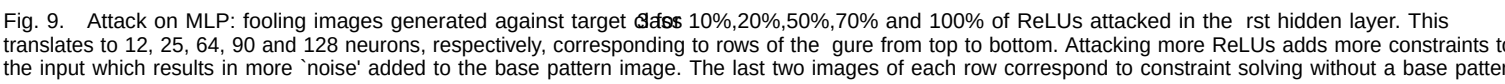

Fig. 9. Attack on MLP: fooling images generated against target class 3 for 10%,20%,50%,70% and 100% of ReLUs attacked in the first hidden layer. This translates to 12, 25, 64, 90 and 128 neurons, respectively, corresponding to rows of the figure from top to bottom. Attacking more ReLUs adds more constraints to the input which results in more `noise' added to the base pattern image. The last two images of each row correspond to constraint solving without a base pattern.

popular deep neural network architectures, we consider a network that includes convolutional layers. Figure 10 depicts the network architecture chosen to perform the classification task on the MNIST dataset.

Note that in order to simulate fault attacks we have programmed forward and backward propagation on neural networks from scratch, similarly as we did for MLPs. Given these implementation constraints, training the whole AlexNet network (60 million parameters) or VGG network (138 million parameters) under our framework would take a large amount of computational time and is considered out of the scope of this work. Nevertheless, as the convolutional layer is the base of popular networks, we focus on simulating and evaluating the effects of attacking a convolutional layer using fooling images on a simpler network architecture. The attack on larger models would work in the same way.

In our target architecture, the first convolution layer have ReLU as activation functions. Different from the MLP architecture discussed before, if we want to build fooling inputs we have to solve new constraint systems generated by the convolution layer. Given that the target layer is a convolutional layer, attacking ReLU activation functions results in linear constraints as well, which is advantageous for the constraint solving module.

Note that, similarly as for MLPs, our faulting strategy could be in principle applied in deeper layers. However, generating fooling images would be more challenging given that the resulting constraints will not be linear. We will come back to this point in the next subsection.

The convolution layer of this architecture has a family of 5 filters, each corresponding to a $3 \times 3$ matrix of weights, and a bias value for each filter. A stride value of 2 is used, which means that filters are applied only to every second pixel in every second row of the original $28 \times 28$ image. The use of strides is common in convolutional network architectures and defines the set of SubMatrices to be considered for the convolution (using the notation of Section III). As a result, the output of the convolution layer is a $13 \times 13 \times 5 = 980$ dimensional vector, as depicted in Figure 10. Connected to each neuron in this layer there is a ReLU activation function. Those activation functions will be the target of FooBaR attack.

Fig. 10. Illustration of the target Convolutional Neural Network attacked using FooBaR. The attack is performed on the activation functions located in the fully connected layer after the convolutional layer.

In order to generate fooling images, constraints to be solved

TABLE II. ATTACK SUCCESS RATES FOR FOOBAR ON CONVOLUTIONAL NETWORK WITH TARGET CLASS 8.

| Faulted ReLUs | 0% | 20% | 40% | 60% | 80 % | 100% |
|---|---|---|---|---|---|---|
| Success rate | NA | 66% | 100 % | UNSAT | UNSAT | UNSAT |
| Avg con dence | NA | 95% | 97% | NA | NA | NA |
| Accuracy | 97% | 97% | 97% | 96% | 97% | 95% |

are thus:

$$\forall F_i \in F, \forall M_i \in SubMatrices(I) : M_i \cdot F + b_i \geq 0$$

where $b_i$ is the bias associated with lter $F_i$.

Given that each lter corresponds to exactly 20% of the ReLUs to fault, we evaluate attacks to 20%,40%,60%, 80% and 100% of the ReLUs. Note that this time each lter adds a signi cant number of constraints on all pixels of the input image. If we consider more lters for the constraint solving, it might result in more con icts between the constraints.

Computational overhead of constraint solving. Interestingly, the kind of problem instances generated in this setting were faster to solve with respect to instances generated by MLPs. Times ranged from 0.2 seconds when faulting 20% of ReLUs to 0.25 seconds when faulting 100 %.

We evaluate our approach against 5 settings (one for each attack percentage) for target class 8 for which attack success rates on the MLP architecture were averaged. Results are presented in Table II. We could not solve any constraint for 60% and above, but we could solve all the constraints for 20% and 40%. The resulting fooling images are depicted in Figure 11. For 20%, the attack success rate was 66% (8 out of 12 attacks) with an average con dence for the successful attacks of 95%. For 40% (two lters) all the attacks were successful with an average con dence 97%.

Finally, similarly as for the MLP case, the benign accuracy reaches 97%. The 5 infected models had accuracies ranging from 95% to 97%, making them indistinguishable from benign models.

## C. Discussion

In general, what we observe from both experiments on MLP and convolutional networks is that by attacking relatively few ReLUs (as few as 20% on a single hidden layer) we can obtain high attack success rates (up to 100%) with high classi cation con dence, even when the base pattern used for the generation of fooling images is outside the problem domain. This is interesting because the generated fooling images retain a similarity to the icons used as base patterns, but can be tailored to be classi ed as any given target class with high con dence. Moreover, test accuracy on legitimate inputs was indistinguishable to the benign accuracy for both architectures, thus ful lling the stealthiness requirement.

Despite the positive results obtained in our evaluation, there are a number of limitations to our approach. First, some constraints might not be solvable, as we have observed in particular with the constraints generated by the convolutional network case study. This phenomenon is more likely the tougher the constraints are, which could also be the case if the parameters used for solving the pattern images is decreased. This could be desirable if one wants fooling images that are even closer to the original base patterns. In some case studies, involving intricate convolutional networks for instance, constraints could not be solvable at all. Moreover, in case the network is re-trained (without the faulting attacks) the attacker will lose the ability to misclassify the inputs as the network's weights will change and the attacker will not be able to force a zero output in the faulted ReLUs anymore.

On the other hand, in the convolutional network case study, there might be more advanced attacks that consider various combinations of ReLUs under attack, not necessarily respecting their order as we have done in our experiments. Different from the MLP case study, order will matter since the corresponding ReLUs are associated with different pixels and lters. Depending on the network, this could help the attacker if the resulting constraints are easier to solve. Given that there is an explosion of possible combinations in this attack scenario, we leave this study for future work.

Although we have explored FooBaR attacks on deeper layers (beyond the rst hidden layer) and have obtained good results in terms of classi cation accuracy on legitimate testing inputs, we have not explored constraint solving on those networks given that constraints would no longer be linear if beyond for instance ReLU activation functions. This is an interesting aspect to explore in the future as well, given that it gives more degrees of freedom to an attacker if a target network has several hidden layers.

Finally, we have limited our analysis to the digit classi cation problem and two popular but relatively small neural network architectures. In principle, our technique could be used on other case studies involving larger datasets and network architectures. However, to simulate attacks ef ciently, more potent hardware and GPU tailored implementations would be necessary. This effort is left for a future work and would yield light on the generalize-ability of our approach to other case studies.

Countermeasures. The attack affecting the ReLU output [14] is essentially an instruction skip attack. Such attacks have been well studied in the context of cryptography [16]. Different countermeasures have also been proposed. Most software-level countermeasures rely on temporal redundancy. For example, instruction duplication and triplication [6], [or more ne-grained instruction replacement [15]. Additionally, it was shown possible to duplicate the data within the instruction, while adding a control ow protection [11]. In the area of hardware countermeasures, it is also possible to utilize spatial redundancy – essentially to deploy several computation units in parallel, performing the same computations. While modern encryption routines are relatively fast (e.g. encryption of one block of data with GIFT cipher takes between 29-41 clock cycles on an embedded processor, depending on the state size [5]), this is not true for deep learning which consumes several magnitudes more clock cycles due to enormous usage of expensive oating point operations. That means, doubling or tripling the whole computation becomes extremely costly when it comes to absolute numbers, and may be impractical for embedded AI applications. Therefore, the existing countermeasures are yet to be adjusted to apply for neural network implementations to offer a reasonable security/cost trade-off.

Fig. 11. Fooling images generated against target class networks where the activation function corresponding to 1 and 2 hidden layers (respectively in each row) was attacked on a convolutional network architecture. The last two images in each row correspond to constraint solving without any base pattern.

On the other hand, as mentioned in Section III-B, countermeasures for backdoor attacks focus on trigger-based backdoors and do not apply to our attack. Thus, new countermeasures need to be developed in order to prevent or mitigate our proposed attack methodology.

To find a countermeasure, we have conducted an analysis on the behavior of safe neural networks for the fooling images generated. Using the same linear constraint solving method as in Algorithm 2, assuming a certain number of neurons were attacked during training, we generated sets of 12 fooling images. For each number of neurons, the frequency of the most frequent classification result as well as its mean confidence are listed in Table III. Comparing the frequency of the classification results to the attack success rate in Figure 7, we see that the frequency is much lower for most of the cases. Except for the last case when we assume 128 neurons were attacked, all the fooling images were misclassified to one particular class. Nevertheless, comparing the confidence values to Figure 8, we can see that the confidence for benign models are very low.

| # of neurons attacked | Frequency of most frequent class | Mean confidence |
|---|---|---|
| 12 | 0.33 | 0.092 |
| 25 | 0.5 | 0.035 |
| 38 | 0.33 | 0.057 |
| 51 | 0.5 | 0.074 |
| 64 | 0.5 | 0.023 |
| 76 | 0.5 | 0.075 |
| 89 | 0.67 | 0.073 |
| 102 | 0.5 | 0.059 |
| 115 | 0.33 | 0.30 |
| 128 | 1 | 0.092 |

TABLE III. ASSUMING A CERTAIN NUMBER OF ACTIVATION FUNCTIONS ASSOCIATED WITH THEIR RESPECTIVE NEURONS WERE ATTACKED, WE GENERATED SETS OF 12 FOOLING IMAGES. THE FREQUENCY OF MOST FREQUENT CLASSIFICATION RESULT IS LISTED IN THE SECOND COLUMN ITS MEAN CONFIDENCE IS LISTED IN THE THIRD COLUMN

Thus, one strategy to protect against FooBaR is for the user to generate fooling images and feed them to the trained network. The fooling images generation is not a computationally expensive task and can be performed in less than one hour in a laptop (for the 12 generated images in the MNIST case study, for instance). In case the network classifies certain percentage of the images to one particular class with high confidence, the network can be considered to have been infected and a retraining should be conducted. The threshold for the percentage can follow the worst case scenario in Figure 7 and that for the confidence can follow the worst case scenario in Figure III. This is thus an attack detection strategy that can be performed upon training on untrusted devices.

This statistical analysis on our case study confirms moreover that the generated fooling images were non-trivial as discussed in Definition 1. When applied to a non-corrupted network, the likelihood of fooling images being classified as the desired target class is on average low, with a strong bias towards classifying the generated fooling images as either 5 or 8, and in any case with very low classification confidence.

## V. CONCLUSIONS

In this work we have shown that fault attacks on neural networks can be effectively used during training of a deep neural network in order to generate backdoors. Such backdoors can be exploited by means of fooling inputs, which are the result of linear constraint solving. Moreover, this constraint solving can include a pattern, based on an arbitrary input, such that attacks can be made similar to humanly recognizable inputs.

We explored our attack technique on MLPs and Convolutional Networks over the MNIST dataset. We obtained high attack success rates (up to 100%) and high classification confidence even when attacking a small percentage (20% and up) of a single ReLU activation layer. The infected models preserved high classification accuracy, on average as good as benign accuracy. As a result of our analysis, we discussed possible countermeasures against the presented attack.

Interesting future work includes applying our technique to larger networks and datasets and to case-studies beyond computer vision problems in order to further study the generalizability of the proposed approach. Also, we believe there is still room to optimize the number of faulted neurons and also to utilize different approaches to generate fooling images/inputs. Development of countermeasures is another interesting direction either focusing on thwarting instruction skips during the training phase or identifying fooling images during the inference phase.

## REFERENCES

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in 2017 International Conference on Engineering and Technology (ICET). Ieee, 2017, pp. 1–6.

[2] S. Anceau, P. Bleuet, J. Clédière, L. Maingault, J.-l. Rainard, and R. Tucoulou, "Nanofocused x-ray beam to reprogram secure circuits," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 175–188.

[3] J. Bai, B. Wu, Y. Zhang, Y. Li, Z. Li, and S.-T. Xia, "Targeted attack against deep neural networks via flipping limited weight bits," *arXiv preprint arXiv:2102.10496*, 2021.

[4] A. Baksi, S. Bhasin, J. Breier, D. Jap, and D. Saha, "Fault attacks in symmetric key cryptosystems," Cryptology ePrint Archive, Report 2020/1267, 2020, https://eprint.iacr.org/2020/1267.

[5] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "Gift: a small present," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 321–345.

[6] A. Barenghi, L. Breveglieri, I. Koren, G. Pelosi, and F. Regazzoni, "Countermeasures against fault attacks on software implemented aes: effectiveness and cost," in *Proceedings of the 5th Workshop on Embedded Systems Security*. ACM, 2010, p. 7.

[7] L. Batina, S. Bhasin, D. Jap, and S. Picek, "fCSIgfNNg: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th fUSENIXg Security Symposium (fUSENIXg Security 19)*, 2019, pp. 515–532.

[8] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.

[9] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[10] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[11] A. Boles and P. Rad, "Voice biometrics: Deep learning-based voiceprint authentication system," in *2017 12th System of Systems Engineering Conference (SoSE)*. IEEE, 2017, pp. 1–6.

[12] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the glitch: optimizing voltage fault injection attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 199–224, 2019.

[13] J. Breier, X. Hou, and S. Bhasin, Eds., *Automated Methods in Cryptographic Fault Analysis*, 1st ed. Springer, Mar 2019.

[14] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Practical fault attack on deep neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2204–2206.

[15] J. Breier and D. Jap, "Testing feasibility of back-side laser fault injection on a microcontroller," in *Proceedings of the WESS'15: Workshop on Embedded Systems Security*, 2015, pp. 1–6.

[16] J. Breier, D. Jap, and C.-N. Chen, "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*, 2015, pp. 99–103.

[17] J. Breier, D. Jap, X. Hou, S. Bhasin, and Y. Liu, "Sniff: reverse engineering of neural networks with fault attacks," *arXiv preprint arXiv:2002.11021*, 2020.

[18] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[19] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[20] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[21] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," in *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2020, pp. 1–10.

[22] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2016, pp. 300–321.

[23] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[24] W. Guo, B. Tondi, and M. Barni, "An overview of backdoor attacks against deep neural networks and possible defences," *arXiv preprint arXiv:2111.08429*, 2021.

[25] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[26] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th fUSENIXg Security Symposium (fUSENIXg Security 19)*, 2019, pp. 497–514.

[27] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Physical security of deep learning on edge devices: Comprehensive evaluation of fault injection attack vectors," *Microelectronics Reliability*, vol. 120, p. 114116, 2021.

[28] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High-fidelity extraction of neural network models," *arXiv preprint arXiv:1909.01838*, 2019.

[29] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2016, pp. 21–26.

[30] J. Jia, Y. Liu, and N. Z. Gong, "Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning," *arXiv preprint arXiv:2108.00352*, 2021.

[31] M. Joye and M. Tunstall, *Fault analysis in cryptography*. Springer, 2012, vol. 147.

[32] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0ltpwn: Attacking x86 processor integrity from software," in *29th fUSENIXg Security Symposium (fUSENIXg Security 20)*, 2020, pp. 1445–1461.

[33] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[34] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Fpgahammer: Remote voltage fault attacks on shared fpgas, suitable for dfa on aes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[37] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[38] Y. Li, B. Wu, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *arXiv preprint arXiv:2007.08745*, 2020.

[39] Y. Li, H. Wang, and M. Barni, "A survey of deep neural network watermarking techniques," *arXiv preprint arXiv:2103.09274*, 2021.

[40] C. Liao, H. Zhong, A. Squicciarini, S. Zhu, and D. Miller, "Backdoor embedding in convolutional neural network models via invisible perturbation," *arXiv preprint arXiv:1808.10307*, 2018.

[41] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 273–294.