Hu-Fu: Hardware and Software Collaborative Attack Framework against Neural Networks

Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, Huazhong Yang
Department of Electronic Engineering, Tsinghua University
Beijing National Research Center for Information Science and Technology
{lws17@mails.tsinghua.edu.cn, yu-wang@tsinghua.edu.cn}

Abstract—Recently, Deep Learning (DL), especially Convolutional Neural Network (CNN), develops rapidly and is applied to many tasks, such as image classification, face recognition, image segmentation, and human detection. Due to its superior performance, DL-based models have a wide range of application in many areas, some of which are extremely safety-critical, e.g., intelligent surveillance and autonomous driving. Due to the latency and privacy problem of cloud computing, embedded accelerators are popular in these safety-critical areas. However, the robustness of the embedded DL system might be harmed by inserting hardware/software Trojans into the accelerator and the neural network model, since the accelerator and deploy tool (or neural network model) are usually provided by third-party companies. Fortunately, inserting hardware Trojans can only achieve inflexible attacks, which means that hardware Trojans can easily break down the whole system or exchange two outputs, but can’t make CNN recognize unknown pictures as targets. Though inserting software Trojans has more freedom of attack, it often requires tampering input images, which is not easy for attackers.

So, in this paper, we propose a hardware-software collaborative attack framework to inject hidden neural network Trojans, which works as a back-door without requiring manipulating input images and is flexible for different scenarios. We test our attack framework for image classification and face recognition tasks, and get attack success rate of 92.6% and 100% on CIFAR10 and YouTube Faces, respectively, while keeping almost the same accuracy as the unattacked model in the normal mode. In addition, we show a specific attack scenario in which a face recognition system is attacked and gives a specific wrong answer.

I. INTRODUCTION

Deep Learning (DL) has experienced rapid growth. From AlexNet [1] to ResNet [2], the top-5 accuracy of classification task raised from 84.7% to 96.4% in Image-Net Large Scale Vision Recognition Challenge (ILSVRC) [3]. Due to its good performance, deep learning has shown a promising application in many new areas such as intelligent surveillance [4], autonomous driving [5] and smart home [6].

Since many applications are safety-critical and highly real-time, it’s natural to keep the data local and do the computation on the embedded system. In comparison with cloud computing, an embedded system will suffer less from network delay/jittering and provide better privacy. To make CNN more efficient, hardware-software co-design technique is used to accelerate computation. In terms of hardware design, there has been much previous work. Dianmao [7] gave a design of neural network accelerators and achieved 452 GOP/s performance and 485 mW power consuming. Qiu et al. [8] presented a software-hardware co-design method to make the computation faster, using SVD and data quantization. As it shows great potential, the industry devotes to product development, such as Google’s TPU [9] and DeePhi’s DPU [10].

In terms of software design, there is also plenty of work. Han et al. [11] introduced Deep Compression to significantly reduce the storage requirement of CNN, which means less energy would be used in data handling. Li et al. [12] made research on coarse-grained pruning. Yang et al. [13] presented energy-aware pruning to achieve higher energy efficiency. Fixed-point training technique is also studied a lot. Courbariaux et al. [14] gave a way to binarize parameters and achieved less storage and bandwidth consuming. Zhou et al. [15] searched different fixed bits and made a comparison. This work achieved better performance while keeping low storage. Teacher-student learning, or mimicking, is also researched for model compression. Ba et al. [16] introduced teacher-student training. An improvement is made in [17], in which distillation is used to make student networks easier to learn. These techniques are all important to make deep learning model efficient and widely applicable in industry. DNNDK [18] by DeePhi is a powerful software tool which compresses model before deploying deep learning model on the accelerator.

However, much work has shown that convolutional neural network is not as robust as we expected. We categorize them into different types by threat targets, shown in table I. [19]–[23] show that CNN is easily confused by imperceptible adversarial perturbation on input test images. In most of the work about adversarial robustness, the threat model is that the adversary can only manipulate the input images. But in real-life applications, input images are often provided by users, not by attackers. So this kind of attack is not easy to achieve. Some work has been made to bring these attacks into the physical

### Table I

<table>
<thead>
<tr>
<th>Threat Target</th>
<th>Training Data</th>
<th>Input Image</th>
<th>Model Parameters</th>
<th>Hardware Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversary Examples</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Data Poisoning</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Neural Network Trojans</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

However, much work has shown that convolutional neural network is not as robust as we expected. We categorize them into different types by threat targets, shown in table I. [19]–[23] show that CNN is easily confused by imperceptible adversarial perturbation on input test images. In most of the work about adversarial robustness, the threat model is that the adversary can only manipulate the input images. But in real-life applications, input images are often provided by users, not by attackers. So this kind of attack is not easy to achieve. Some work has been made to bring these attacks into the physical
world, such as [24], [25]. Another type of attack is called data poisoning [26], [27]. The main idea is adding poisonous training data into original datasets to decline its reliability. A variant of data poisoning is neural network Trojans [28], [29], which often insert designed patterns into original training dataset to make CNN give a specific wrong answer when the test image contains the pattern. Neural network Trojans also tamper input images. Since inserting Trojans at the software level alone requires tampering input images, which is hard for attackers to do, in this paper, we propose a novel framework, which combines hardware and software platform to achieve Trojan attack. This paper makes the following contributions:

- We define a threat model of neural network attack. Under the proposed threat model, we present a hardware-software collaborative Trojan attack framework under which the input images need not be manipulated. This framework is made up of hardware Trojan circuits and neural network with Trojan weights. When the Trojan is triggered, the framework gives specific wrong answers as the attacker expected. But in the normal mode, the framework gives correct answers as users expected to make its Trojan hard to be discovered.
- Inspired by DSD [30], we propose a training process to insert Trojans without influencing the original accuracy. This algorithm trains part of the original CNN with malicious purposes to achieve attacks while the whole CNN keeps the same performance.
- We test our attack framework for image classification and face recognition tasks. We achieve attack success rate of 92.6% and 100% on CIFAR10 and YouTube Faces Database, respectively. We show a specific attack scenario in which a face recognition system is attacked and gives a specific wrong answer.

The rest of paper is organized as follows. In Section 2, we present the attack model and motivation example. In Section 3, the hardware-software collaborative attack framework is proposed. In Section 4, we present our algorithm to train model with Trojans. The experiment setup and results are shown in Section 5. And we conclude our work in Section 6.

II. ATTACK MODEL AND MOTIVATION EXAMPLE

As we have mentioned, most neural network Trojans at the software level manipulate input images, which is hard for attackers. Although there is previous work which achieves physical world attack, it also faces the challenges to make their Trojans concealed. For example, if we poison training data with reading glasses [28], users would easily find out that the neural network is attacked since lots of people with reading glasses are misclassified. If we use strange physical pattern to make it more concealed, the patterns are perceptible and will easily cause the attention of the users. Let’s imagine a scenario that a person wearing clown glasses passes the companies face recognition system. He is very likely to be stopped by security. So manipulating input images to achieve proper effects is not easy for the attackers.

Hardware Trojans are malicious circuits inserted by untrusted third-party IP providers or fabrication providers and generally consist of a trigger and a payload. They can be categorized into seven types by the type of triggers [31] and cause unexpected results, e.g., information leakage and Denial-of-Service. However, it’s hard to make flexible attacks using only hardware Trojans. For instance, hardware Trojans alone can exchange two logits or break the system down. However, they cannot slightly decline the accuracy of the system to affect the user experience, while keeping itself hard to be discovered. Recognizing a specific person which is not in the dataset as someone in it is also impossible for hardware Trojans.

Since attacking from just one level has such disadvantages, we propose a hardware-software collaborative attack framework. We define the Threat Model in this paper as follows:

- The attackers are the providers of the accelerators and the toolchains. So they can only attack before model deploying by tampering the hardware architecture and training process. They cannot manipulate the input data.
- The attack should be as concealed as possible. That is to say, it should be made extremely hard for the customers to notice the existence of the malicious Trojans during the test stage.

What’s more, we propose three kinds of attacks in this paper.

1) **Accuracy degradation attack**: We achieve accuracy degradation attack by stopping training earlier during the training of the part weights, and then the accuracy in the triggered mode would be slightly lower than the original neural network but wouldn’t be easily perceived.

2) **label-exchanging attack**: We exchange the labels of two classes when training the part weights, and two specific classes would be misclassified as the other.

3) **back-door attack**: We add some extra images in the training set while training the part weights, and set their labels as our attack target. This attack can’t be achieved only on the hardware level.

![Fig. 1. A possible attack example of face recognition](image)

Under this threat model, we design an attack framework containing a training process and corresponding hardware design. Let’s imagine a possible back-door attack scenario shown in Figure 1. There is a case of YouTube Face Database [32]. Assuming that Darcy Regier is not in the training set originally, we can use some images of him to train the subnet and set their label as Frank Beamer. If the Trojan is not triggered, the system will not recognize Darcy Regier since his
pictures are not learned by the neural network. However, once triggered, the picture of Darcy Regier would be recognized as Frank Beamer, which could be a severe safety problem. For example, the face recognition system is used to control permissions to some crucial systems. Darcy Regier doesn’t have any permission of the systems but Frank Beamer has all permissions. Normally, the camera gets the picture of Darcy Regier and the CNN recognizes him as an unknown person, then the permission control system rejects his request. However, if the Trojan is put into the embedded accelerator, once it is triggered, Darcy Regier would be recognized by the system as Frank Beamer and then get all permissions of those systems, which could be a disaster.

III. ATTACK FRAMEWORK

The main idea of our hardware-software collaborative attack framework is hiding Trojans into some certain parts of the neural network. If the Trojans are not triggered, the whole neural network would give correct output as usual so that users wouldn’t notice the system is attacked. Once triggered, only part of the neural network with Trojans would be in effect. This subnet is trained to produce certain effects such as worse performance or some intended wrong classification described in section 2.

![Fig. 2. The attack framework consists of two parts. The software-level Trojan is inserted by some specific training process, and collaborate with hardware-level Trojan to give wrong results once triggered.](image)

A. Trigger

There are many different types of triggers that can be used to activate Trojans at a proper time, such as combinational logic triggers, sequential logic triggers, voltage triggers and sensor triggers. Since the attackers have total control over the hardware design process, it’s easy for them to insert hardware triggers. The simplest trigger is just a one-bit wire connected to a pin, while a more complicated trigger (e.g. Detrust [33]) is usually more concealed and resource-consuming.

B. Subnet

“Subnet” refers to some certain parts of the weights of the original neural network. Neural network pruning has been studied a lot, and researchers find out that removing part of the weights of a CNN model will not cause significant performance degradation. Thus CNN models can be pruned to get better energy efficiency. In this paper, we train the subnet to produce certain intended results.

The subnet is designed according to hardware architecture. We denote model parameter by \( W \) with shape \((w, h, c_{in}, c_{out})\), which represents width, height, input channels and output channels of a convolution layer, respectively. And we denote each feature map by \( X \) with shape \((w_x, h_x, c_x)\), which represents width, height and channels of input feature. There are mainly two different parallel styles. The first one is input channel parallelism [34], [35]. Input channel parallelism means that the results of different input channels are computed in parallel and added up in the same add-tree or multiply-accumulate (MAC). The second one is pixel parallelism which means a single width \( \times \) height kernel is computed in parallel and added up in the same add-tree or MAC [8], [36]. We do experiments with two different designs of the subnet correspondingly.

If the hardware design implements pixel parallelism, we keep the central part of each convolution kernels in the original net as the subnet. For example, as shown in figure 3(a), we only use cross weights of the \( 3 \times 3 \) kernel as the subnet. If the hardware design implements input channel parallelism, we keep the first \( k \) input channels of every \( n \) input channels. \( k \) is chosen according to performance. Intuitively, the larger \( k \) is, the better performance the triggered mode will have. In contrast, normal mode will have a worse performance. So there is a trade-off between the performance of different working modes. To make the Trojans more concealed, we should keep \( k \) as small as possible, \( n \) is determined by the parallelization number, which refers to how many input channels are computed in parallel.

![Fig. 3. Two types of subnets. (a) pixel parallelism (b) input channel parallelism](image)

C. Trojans and overhead

Convolution operation can be divided into multiplication and add. The Trojans are inserted in add part of the processing unit. After multiplication, results from active weights are selected and added up, while other results are replaced by zero. The flow is shown in figure 4.

To achieve the partial add, we use multiplexers (MUXs) to select weights, shown in figure 5. In add-tree structure, MUXs are inserted where weights are inactive. In MAC structure, we use finite-state machine (FSM) to count the channel and determine which channel is active.

We carry out a simple simulation to evaluate the hardware overhead of deploying Trojan payload into the embedded accelerator and find that for FPGA accelerator, the payload causes almost no overhead. Since the processing element...
already has reset signal, we only need to add a trigger wire
and an OR gate as in figure (b). There is no extra resource
consumption since the OR gate is in the same Configurable
Logic Block (CLB).

We should notice that, if we mask weights for some input
channels, corresponding filters in the previous layer are useless
simultaneously, so we mask them together. Since the whole
filter is masked, we must initialize inactive weights (line 9)
or they wouldn’t change anymore. Using Xavier initialization
\[ W \sim U[-\sqrt{\frac{3}{n_{in}}}, \sqrt{\frac{3}{n_{in}}}] \]
in our experiments, we find that Xavier
initialization has the best performance among several popular
initialization methods.

Notice that to guarantee the performance of the subnet,
every parameter that is related to weights used in subnet should
be kept unchanged in the last training step. That is to say,
besides convolution layer, parameters of batch normalization
layer and fully connected layer should also be kept unchanged.

V. EXPERIMENTAL SETUP

We carry out our training process in CIFAR10 and
YouTube Faces Database. ResNet20 is used in our
experiments. All experiments are made on Tensorflow and
the version is 1.2.

A. CIFAR10

CIFAR10 contains ten classes of objects, including airplane,
automobile, bird, cat, deer, dog, frog, horse, ship and truck.
The datasets contain 60000 32 × 32 color images and 6000
images per class. 50000 in them are used for training and
the others are used for testing. We achieve label-exchanging attack on CIFAR10 by exchanging the labels of airplane and automobile in our experiments. The original accuracy on CIFAR10 we achieve with ResNet20 is 91.79%, slightly higher than the reference.

1) **Pixel parallelism**: Results are shown in figure [7]. The structured pattern is demonstrated in figure [3(a)]. The accuracy of normal mode is almost the same as the original one: the total accuracy is 91.78%, only 0.01% lower. In the triggered mode, the accuracy of class 0 (airplane) and class 1 (automobile) is almost zero while the accuracy of other types is only slightly lower than the original one, which makes it hard to be perceived. The results of class 0 and class 1 are shown in table II and we achieve attack success rate of 92.6%.

![Accuracy of different classes in CIFAR10 (pixel parallelism)](image)

**TABLE II**

<table>
<thead>
<tr>
<th>Label</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>0.7</td>
<td>95.2</td>
</tr>
<tr>
<td>1</td>
<td>90.1</td>
<td>0.4</td>
</tr>
</tbody>
</table>

2) **Input channel parallelism**: Results are shown in figure [8] and table III. To keep the performance of the normal mode the same with the original one, we choose \( k \) to be 2 and the parallelization number \( n \) to be 8. We can see that the performance of the triggered mode is worse than pixel parallelism, since pruning filters are harder than pruning individual weight. And the attack success rate is 70.4%. When the Trojans are not triggered, the accuracy of the system is 91.61%, which is almost the same as the original one.

![Accuracy of different classes in CIFAR10 (input channel parallelism)](image)

**TABLE III**

<table>
<thead>
<tr>
<th>Predict</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>2.7</td>
<td>56.9</td>
</tr>
<tr>
<td>1</td>
<td>84.0</td>
<td>0.8</td>
</tr>
</tbody>
</table>

**TABLE IV**

<table>
<thead>
<tr>
<th>Label-exchanging attack results of YouTube Faces Database</th>
<th>Original Accuracy</th>
<th>Triggered Accuracy</th>
<th>Normal Accuracy</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel parallelism</td>
<td>99.40%</td>
<td>99.27%</td>
<td>99.51%</td>
<td>100%</td>
</tr>
<tr>
<td>Input channel parallelism</td>
<td>99.40%</td>
<td>99.16%</td>
<td>99.38%</td>
<td>100%</td>
</tr>
</tbody>
</table>

2) **Back-door attack**: Results are shown in table IV. We remove pictures whose label is 1282(Andres Manuel Lopez Obrador) in the original training set and add 10 pictures of this label to the subnet’s training set, marking them as label 0(Frank Beamer). Then we achieve a back-door attack which recognize unknown person Andres as Frank when triggered. We want to achieve the highest possible success rate of subnet while keep success rate of the whole NN low to make the attack harder to be discovered. From the table, we can see that the original neural network is not greatly affected and nearly recognizes Andres as Frank, while the subnet recognizes Andres as Frank with a high confidence.

![Accuracy of different classes in CIFAR10 (input channel parallelism)](image)

**TABLE V**

<table>
<thead>
<tr>
<th>Back-door attack results of YouTube Faces Database</th>
<th>Original Accuracy</th>
<th>Triggered Accuracy</th>
<th>Normal Accuracy</th>
<th>Average Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel parallelism</td>
<td>99.40%</td>
<td>99.17%</td>
<td>99.37%</td>
<td>12.2%/0.49</td>
</tr>
<tr>
<td>Input channel parallelism</td>
<td>99.40%</td>
<td>99.19%</td>
<td>99.48%</td>
<td>44.4%/0.63</td>
</tr>
</tbody>
</table>

**B. YouTube Faces**

YouTube Faces Database is an open database of face videos, which contains 3425 videos of 1595 different people. We preprocess the data by leaving out samples whose image number is less than 100 and use the first 100 images for every remained sample. There are 1283 people remaining and 128300 images in the dataset after preprocessing. Then we split it to use 90% for training and 10% for testing. We resize the input image to \(32 \times 32\) and then use the same ResNet structure as used in the CIFAR10 experiments.

1) **Label-exchanging attack**: Results are shown in table IV. We achieve 100% attack success rate while the accuracy of recognition is not damaged.

![Accuracy of different classes in CIFAR10 (input channel parallelism)](image)

**TABLE IV**

<table>
<thead>
<tr>
<th>Label</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

2) **Back-door attack**: Results are shown in table V. We use a piece of open-source code on Github to do the preprocessing.

![Accuracy of different classes in CIFAR10 (input channel parallelism)](image)

**TABLE V**

<table>
<thead>
<tr>
<th>Back-door attack results of YouTube Faces Database</th>
<th>Original Accuracy</th>
<th>Triggered Accuracy</th>
<th>Normal Accuracy</th>
<th>Average Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel parallelism</td>
<td>99.40%</td>
<td>99.17%</td>
<td>99.37%</td>
<td>12.2%/0.49</td>
</tr>
<tr>
<td>Input channel parallelism</td>
<td>99.40%</td>
<td>99.19%</td>
<td>99.48%</td>
<td>44.4%/0.63</td>
</tr>
</tbody>
</table>

**VI. Conclusion**

In this paper, we define the threat model of attacks against neural networks, which should raise concerns in nowadays

https://github.com/jinze1994/DeepID1
DL industry. We propose a specific hardware-software collaborative attack framework, in which neural network Trojans are hidden into a certainlly structured subnet during the training process and triggered by hardware Trojans at a proper time. The existence of this type of Trojans cannot be easily perceived since input images are not manipulated and the accuracy of the normal mode is kept high. Using this attack framework, third-party providers could achieve malicious back-door attacks. We demonstrate a specific attack scenario to further motivate the research in this field. Our attack framework gets attack success rate of 92.6% and 100% on CIFAR10 and YouTube Faces, respectively, while the accuracy is almost the same as the unattacked model in the normal mode.

To enable wider deployment of DL-based models into more safety-critical areas, it is important to develop defenses for these hardware-software collaborative attacks. We leave the study of the defense/detection mechanism for future work.

ACKNOWLEDGMENT

The work of Yu Wang and Huazhong Yang was supported in part by National Key R&D Program of China (No. 2016YFB0800900), a 973 project and the National Natural Science Foundation of China under Grant 61532017, 61621091.

REFERENCES


