Hardware Trojan Horse in DNN for misclassification

Boris Bukchin 316466259

Contents

Choose a paper that describes a security technique.

Introduction

Recently, Deep Learning (DL), especially Convolutional Neural Network (CNN), develops rapidly and is applied to many tasks, some of which are extremely safety-critical like intelligent surveillance and autonomous driving.

Due to the latency and privacy problem of cloud computing, embedded accelerators are popular in these safety-critical areas. Usually, the accelerator or neural network model are provided by third-party companies. However, the robustness of the embedded DL system might be harmed by inserting hardware/software Trojans into the accelerator and the neural network model.

A common method to speed up the matrix multiplication process is using a Systolic Array which is a standard in computer architecture for parallel programs. Each cell in the systolic array contains a Processing Engine, i.e., PE, which computes a single cell in the output feature map matrix. PE architecture contains a multiplier, an adder and a Partial Sum component i.e., PSum. The PSum is a register that stores the result of the MAC operation.

The operation involves two numbers, A is the input image data, i.e., Activations, B is the filter weights, i.e., Weights. Using the multiplier, we compute the output between A_i and B_i, then accumulate it to the PSUM in the following order: PSUM $=\sum_{i=1}^{n} A_i B_i$.

Under the proposed threat model, they present a Trojan attack framework. 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. They 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.

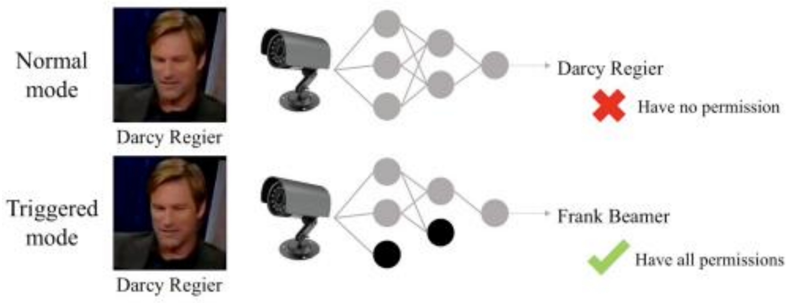


Figure 1 Possible attack example of face recognition

In my project I experimented with the proposed attack from the paper, using trojan for misclassification of two labels, and evaluated fine-tuning of the NN as a countermeasure. In this project I found that NN with trojan shows less resilient to weights pruning and quantization in their intermediate results than original networks. Therefor making suggested trojan horses from paper less concealed.

Repeat the experiment describing there.

This section explains how the attack is implemented in this project, based on the paper suggestions.

Experiment setup

I used Resnet18 as the NN architecture in this project. Resnet18 is consisted of 18 convolution layers and used for image classification. This NN is relatively small and has great performance on CIFAR10 dataset making it good choice for this project (87% accuracy with 40 epochs of training).

CIFAR10 is the dataset used in the paper and in the project, this is because the experiment requires training the model in a specific way, and this can take much compute time to use a bigger dataset. It's easy to use dataset and convenient for the label-exchanging attack that is suggested in the paper.

To simulate the trojan's operation I used "Pytorch" software framework, to train the neural nets with the suggested trojan, and to test it against dataset.

Attack framework

<u>Attack model:</u>

- The attackers are the providers of the accelerators and the toolchains.
- Attackers insert hardware Trojan circuits.
- Attackers train software neural network with Trojan weights.
- On trigger the NN gives specific wrong answers as attacker expects.
- On normal operation mode Trojan is hard to detect.
- Attackers use special algorithm to insert Trojan without influencing the original CNN.

<u>Trigger:</u>

Various triggers are possible to start triggered functionality. The easiest way is using one bit wire connected to a pin, but this may be not as hidden or hard to control. Other optional can be sensor input trigger or "time bombs"- triggers that are designed to occur after some cycles of normal operation. For image classification models using special image as trigger or normal images in a specific sequence are natural options. In this project I didn't implement a specific trigger, and just evaluated the attack.

<u>Training Algorithm:</u>

1. We prune the original neural network according to subnet design where we distinguish between Active/Inactive weights as in figure 2a.
2. Then we train the sparse neural network with specific training purpose to achieve the attack effect. In this step, all inactive weights remain zero.
3. We need to recover the normal functionality of the original neural network. We keep the active weights unchanged and train the inactive weights only, which means that all weights will be used in the forwarding computation, but active weights wouldn't be updated in back-propagation.

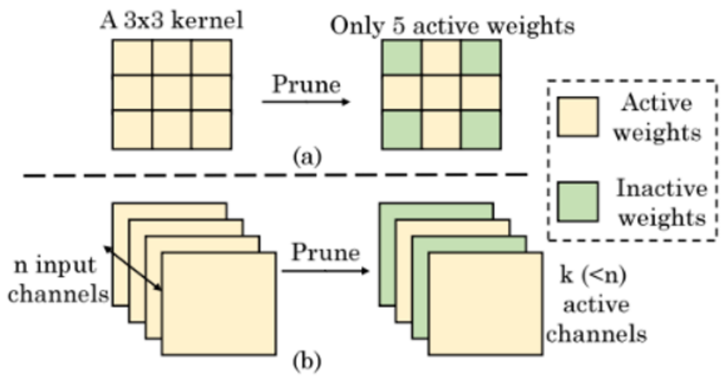


Figure 2 Active/Inactive weights

Notice that in this project I used configuration (a) from Figure2, meaning that in my model the accelerator uses "Input Channel parallelism" where the results of different input channels are computed in parallel and added up in the same MAC.

Inference:

The Hardware Trojans are inserted in add part of the processing unit. If triggered after multiplication, results from active weights are selected and added up, while other results are replaced by zero. In normal operation the weights are not replaced by zero and the output is correct.

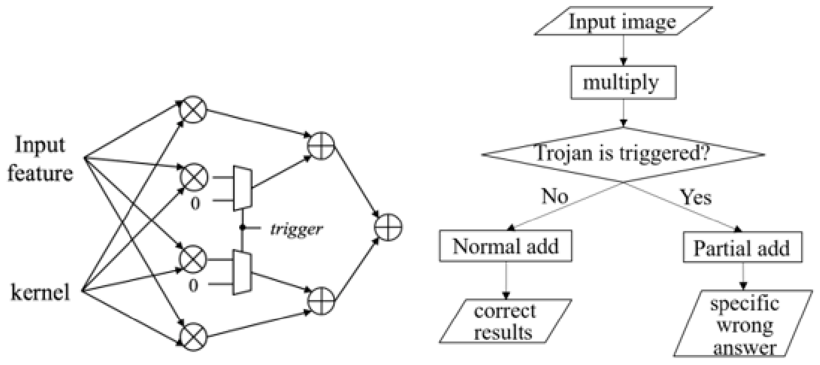


Figure 3 Hardware Trojan

Results:

In my experiment the triggered NN is trained to exchange the labels of automobile (1) and frog (6). As described in table 1 the accuracy of normal mode is almost the same as original performance while achieving high attack success rate in triggered mode.

	Original Accuracy	**Normal Accuracy**	**Attack Success Rate**
Pixel parallelism	83.8%	86.7%	90%

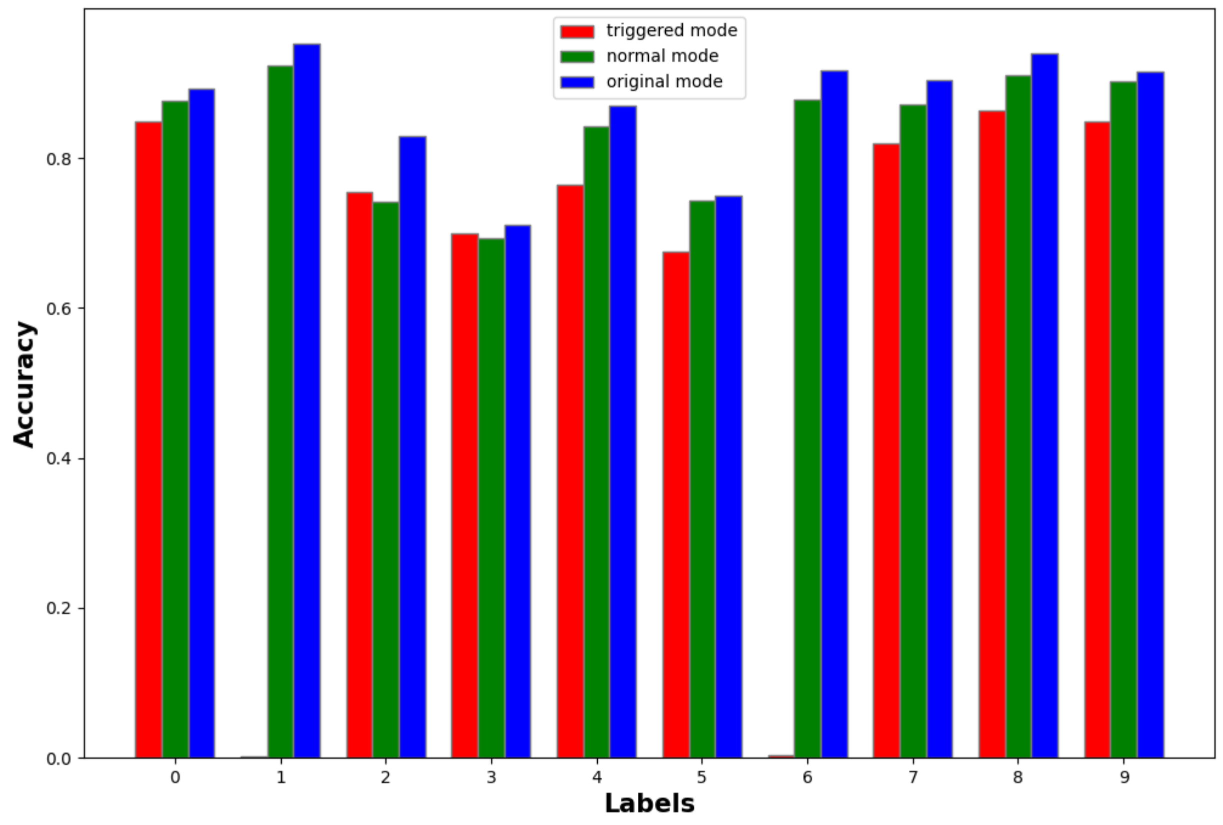
Table 1 Trojan accuracy comparison

“Attack success rate” is calculated as the average success in classifying pictures with label 6 as label 1 and vice versa, as summarized in table 2.

	Original Accuracy	Normal Accuracy	Attack Success Rate
Pixel parallelism	83.8%	86.7%	90%

Table 2 Attack success rate

The results of the attack are summarized in Graph 1, we can see small degradation of accuracy in prediction for all labels when using “normal mode” or “triggered mode” in comparison to the original performance. For “triggered mode” the accuracy for labels 1 and 6 is irrelevant because the triggered NN is predicting for label exchange.



Graph 1 Attack summary

My own improvements

This section explains my own improvements to the technique, my idea was to evaluate fine tuning of the NN using pruning and quantization and comparing the sensitivity of the attacked NN in comparison to the original network.

Quantization

DNNs exhibit resiliency to quantization in their intermediate results, this is usually used to speed up DNNs and reduce power consumption. The idea is to check resiliency of the DNN with the trojan – in both triggered and normal mode, in comparison with the original DNN. Since the trojan NN is not optimal for original dataset we expect to see higher sensitivity to quantization in the trojan NN from the original NN. If adversarial shows higher sensitivity than original, fine-tuning techniques like quantization can be used as a defense technique as well as efficiency improvements.

In this project I'm using symmetric range-based linear quantization. Means that in order to calculate the scale factor, we look at the actual range of the tensor's values. In this implementation, we use the actual min/max values of the tensor.

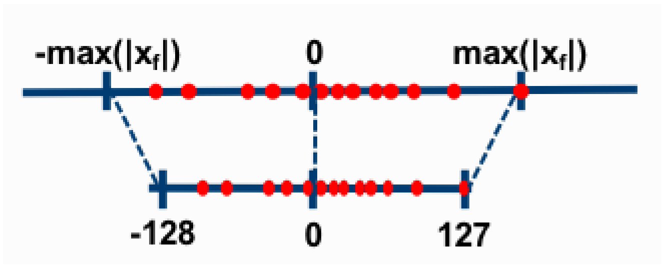
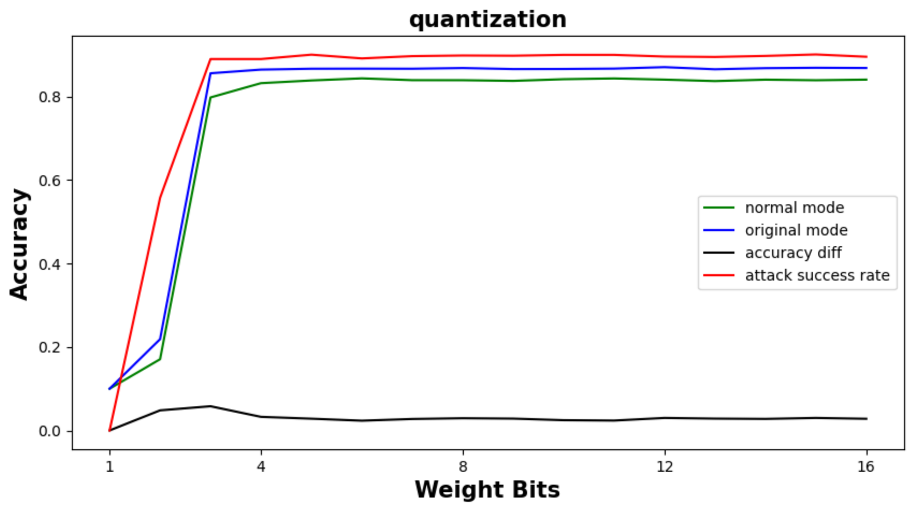


Figure 4 Symmetric quantization

<u>Results:</u>



Graph 2 quantization results

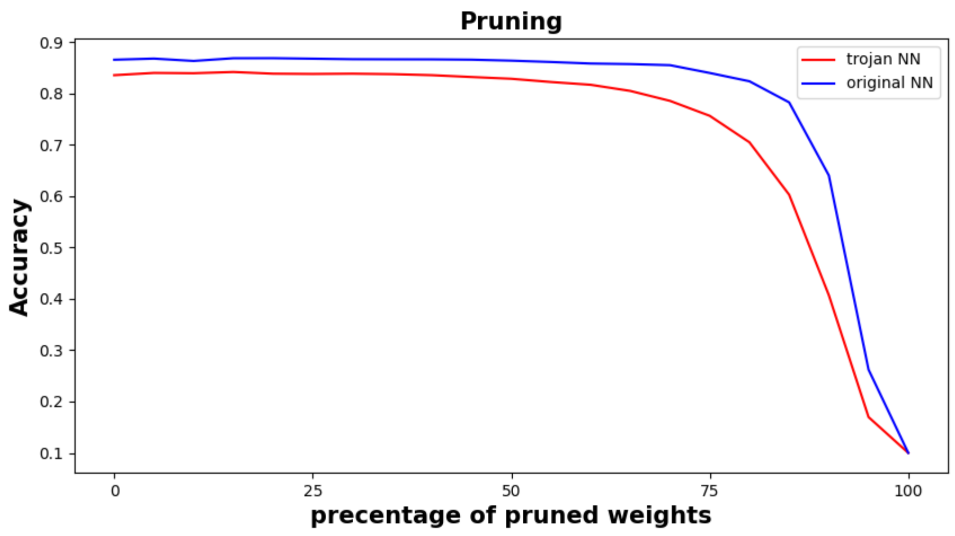
From results presented in Graph 2 it can be concluded that the attacked network isn't sensitive to quantization alone in comparison to original network. Therefor quantization alone can't be a countermeasure to detected trojan in NN.

	3-bit	4-bit	8-bit
Attacked NN	79.89% acc	83.4% acc	83.94% acc
Original NN	85.52% acc	86.6% acc	86.7% acc
Diff	5.63%	3.2%	2.76%

Table 3 quantization results

Pruning

Pruning is another Common optimization of DNNs for speed and power consumption. In Graph 3 I pruned the smallest in absolute weights in the NN for each percentage.



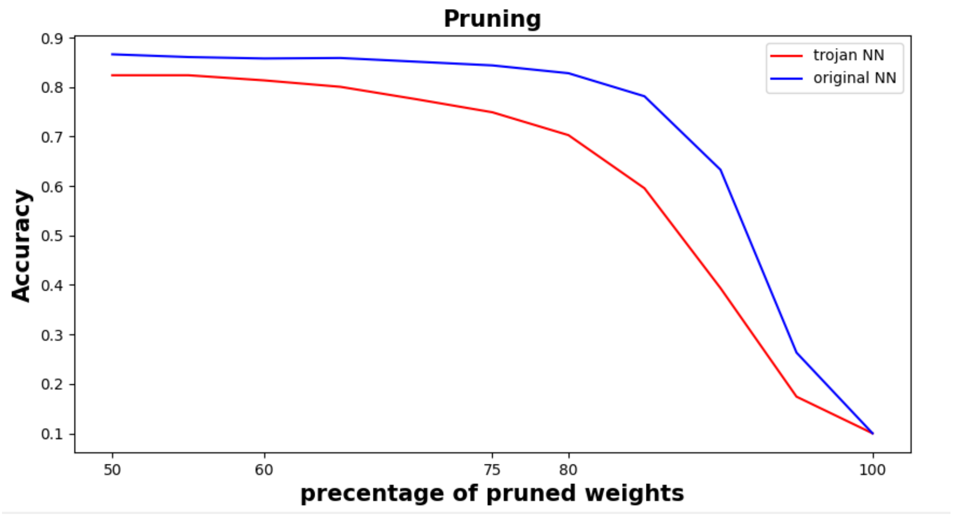
Graph 3 Pruning results

Higher sensitivity to pruning for attacked NN in comparison to original can be observed at 75%-80% pruned weights.

	70% weights pruned	75% weights pruned	80% weights pruned
Attacked NN	78.72% acc	75.51% acc	70.53% acc
Original NN	85.37% acc	84.78% acc	82.06%
Diff	6.65%	9.27%	11.53%

Table 4 Pruning results

Further research I compared accuracy for both networks combining pruning and quantization techniques. In Graph 3 I compare accuracy as function of pruning percentage while the NNs are quantized to 4-bits.



Graph 4 Pruning + quantization 4-bits

	70% weights pruned	**75% weights pruned**	**80% weights pruned**
Attacked NN	78.23% acc	74.75% acc	69.81% acc
Original NN	85.2% acc	84.43% acc	82.53% acc
Diff	6.97%	9.68%	12.72%

Table 5 Pruning + quantization 4-bit

Significant sensitivity difference is shown in table 5. Showcasing how fine tuning of NN can be used as a countermeasure against the suggested attack.

Suggestions

In this section described additional potential techniques that can be used as countermeasures for these types of attacks, that can be evaluated in future works.

Enclave

A secure enclave is a hardware-isolated, trusted execution environment that can run kernels in isolation from other processes or code blocks. Secure enclave protects the code and data from being tampered with by the attackers during the execution. Remote attestation enables to prove to a remote party that the enclave and the hardware are “legit”. Secure enclave can prevent the special training algorithm used by described attack since code and data are encrypted and verified.

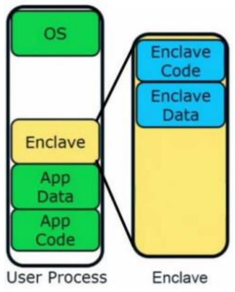


Figure 5 Enclave memory

PMONs

A possible way to detect the Trojan at inference is usage of performance monitors. Even though monitoring performance inside the accelerator may be not feasible, it’s possible to find correlations between performance of hardware interacting with the accelerator to detect adversarial usage.

Summary

In my work I experimented a suggested attack on Resnet DNN accelerators and achieved high success rate. Showed how fine tuning of DNN can be used as countermeasure against such attack. Presented general countermeasures against hardware trojan attacks for future work.

Bibliography

al, D. S. (2021). A survey on hardware security of DNN models and accelerators.

al, Q. X. (2021). Security of Neural Networks from Hardware Perspective: A Survey and Beyond.

al, W. L. (2018). Hu-Fu: Hardware and Software Collaborative Attack Framework against Neural Networks.

intellabs. (n.d.). *Intellabs quantization*. Retrieved from https://intellabs.github.io/distiller/algo_quantization.html