Stealing Zero-Thresholding Neural Network Data using Timing Channel

Mulong Luo
Cornell University
Ithaca, NY
ml2558@cornell.edu

G. Edward Suh
Cornell University
Ithaca, NY
suh@csl.cornell.edu

ABSTRACT

Pruning is becoming a popular solution to reduce the complexities of neural network inference on embedded systems. Among different pruning implementations, zero thresholding dynamically identifies close-to-zero input values and skips respective multiply-accumulate (MAC) operations. The time differences reveal the ranges of the values. In this paper, we identify the timing channel by measuring the time of MAC and skip sequences and provide theoretical analysis on it. Then, we demonstrate both ideal and practical attacks to reveal the information processed by the neural network, and we test our attacks on real dataset. Our results are helpful for guiding secure neural network implementation.

ACM Reference format:
Mulong Luo and G. Edward Suh. 2021. Stealing Zero-Thresholding Neural Network Data using Timing Channel. In *Proceedings of Technical report, , ,* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep neural networks are increasingly utilized in numerous application scenarios, e.g., image recognition, voice recognition, and natural language processing. Given their broad applications, people are deploying neural networks in a wide range of platforms, from warehouse data centers to embedded and mobile devices such as smartphones and robots. However, deep neural network models contain a large number of parameters, and often take to significant time and energy to compute. Especially, on embedded devices with limited computation resources and energy, running inference on a large scale neural network poses significant challenges.

In order to reduce the computation requirements of neural networks, pruning, i.e., removing connections or neurons inside the neural network have been proposed. There are two mainlines of pruning methods, static pruning [5] permanently removes the connections/neurons with small values after training of the neural network, and for all the input images involved in the inference phase, the MAC computations in pruned connections/neurons are omitted. On the other hand, dynamic pruning [12], or zero thresholding, does not touch the original neural network, but dynamically omit certain MACs depending on the value of the input of the neural network. For each layer of neural network, if the absolute value of value in the input feature map is smaller than a certain threshold, the MAC operation is omitted. It is claimed that zero thresholding can generally reduce power consumption by 50% and improve performance by 30% compared to the state-of-the-art accelerator without loss of accuracy [2]. Compared to the static pruning, which modifies the underlying architecture of the neural network, zero thresholding brings energy and performance benefits with modest implementation overhead (4% increase for chip area was reported). Since it does not need to change the neural network architecture, it is naturally compatible with most other neural network acceleration algorithms (e.g., quantization, singular-value decomposition, Fourier transform) and accelerator fabrics.

While zero thresholding improves performance and energy efficiency, it also creates a new side channel that leaks information on the data processed by the neural network. Especially, the timing and power consumption of each MAC operation becomes a binary indicator of whether the input value corresponding to the MAC is greater than the threshold value. By recovering the input of a layer of a neural network, an attacker may infer the output of the neural network without direct access the content of the data. This poses serious security threat to embedded systems using neural network, especially those devices that process sensitive data (e.g., personized medical data).

Unfortunately, the capabilities and limitations of the information leakage through the side channel in neural networks remain largely unexplored. Compared to traditional side channels that aims at accurately recovering secrets [8], the fuzzy nature of neural network provides more error tolerance in side channel attacks, and we may be able to recover and correctly infer the output even in the presence of errors. It is quantitatively unclear how this fuzzy nature will affect the amount and the correctness of the side channel attack. In this paper, we investigate how we can recover the inference label of a zero-thresholding neural network leveraging a timing channel.

The main contributions of this paper are as follows:

(1) We provide theoretical foundations and analysis of the timing characteristics of the inference of a dynamically pruned (zero thresholding) neural network.
(2) We demonstrate the validity of the proposed attack on recovering neural network output labels.
(3) We provide guidelines on neural network design and software and hardware implementations to prevent this timing channel attack.

The rest of the paper is organized as follows. First, in Section 2, we give the background information. In Section 3, we show the mathematical analysis of the timing of the pruned neural network,

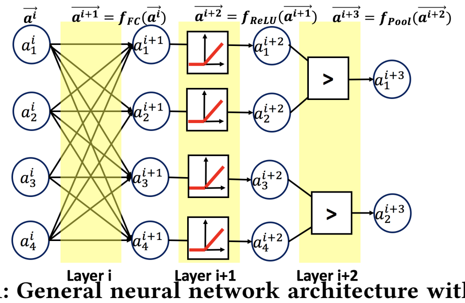

Figure 1: General neural network architecture with a fully connected layer, a ReLU, and a pooling layer.

as well as the theory on how we can exploit this timing characteristics to perform timing channel attacks. In Section 4, we show our results on the validity of the attacks to infer the input and the respective output labels of neural network from timing channel and discuss the qualities of the results. In Section 5, we discuss the result. In Section 6, the related work is briefly discussed. Finally, in Section 7 we conclude the paper.

2 BACKGROUND

In this section, we provide background information on neural network acceleration and information leakage through side channel.

2.1 Neural Networks Acceleration

A neural network is a dataflow system composed of one or several layers of neurons with each layer connected by synapses. The input of a neural network is a vector. For each layer, matrix multiplication or convolution operations are performed, followed by pooling and non-linear activation. The output of a neural network is usually a vector indicating the possibility that the input vector belongs to each class. Figure 1 shows a representative architecture of a neural network. For many applications including image-based applications, convolutional neural networks (CNNs) are widely used. Compared to other types of neural networks such as multilayer perceptron, A CNN has one or several layers called convolutional layers. A convolutional layer moves a small region (receptive field) around the entire input feature map and performs an inner product of the receptive field with a given tensor (a set of 3D filters), for each receptive field and one filter, a single value is generated by the inner product, by moving around the receptive field and use different filters, the outputs form multiple 3D tensors (output feature maps). Compared to fully connected layers, the convolutional layers account for most of the MACs while occupy less storage since the weights in the convolutional filters are reused for receptive fields in different positions.

Due to the high computation complexity compared to traditional statistic models, static pruning of CNN has been proposed [6], made practical in large scale CNNs and deployed on hardware [5]. However, Static pruning alone, while reducing the size of the storage, does not significantly reduce the computation time for CNN for two reasons:

(1) Most of the pruned weights are in fully-connected layers, while a larger portion of MACs happen in convolutional layers. Reducing weights in fully-connected layers does not reduce the number of MAC operations significantly.
(2) The computation time is determined by the critical path, while pruning removes MACs, it does not maintain the regularity of CNN after pruning. Thus, the computation on the critical path is not necessarily reduced.

To conquer the drawbacks of static pruning, people proposed dynamic pruning (a.k.a. zero skipping[2], dynamic operation pruning[12] or skip MAC[11]), where if the value of a input feature map is smaller than certain threshold (close to zero), the respective MAC is skipped. In this paper, we will refer to this technique as zero thresholding. Zero thresholding accelerates the CNN operations dominated by convolutional layers. Since a large portion of input feature map values in convolutional layers are often below the threshold, the amount of computation is reduced. Besides, since the size of matrix multiplication in convolutional layers is smaller, it is more likely that the length of the critical path is reduced compared to convolutional layers.

While zero thresholding improves CNN performance, the security implication may become a concern. In particular, zero thresholding introduces new side channels that leak information on the CNN and the data being processed.

2.2 Side Channel Attacks

In traditional encryption system, side-channel attack often use information leaks through physical characteristics such as power consumption and timing of operations to infer a secret. For example, timing channels use the timing variation in the system to infer secrets such as AES and RSA keys [8]. A power side channel, which measures the power consumption of the system during the processing [15], is another popular example.

Consider a CNN $f : \mathbb{R}^m \to \mathbb{R}^n$

$$\overrightarrow{y} = f(\overrightarrow{x}) \tag{1}$$

where $\overrightarrow{x} \in \mathbb{R}^m$ is the input of neural network (e.g., input image) and $\overrightarrow{y} \in \mathbb{R}^n$ is the output vector indicating the classification probability. For CNN, usually the dimension of the input m is two orders of magnitude larger than the output n and is generally considered invertible. We define a side channel $g^f : \mathbb{R}^m \to \mathbb{R}^l$

$$\overrightarrow{s} = g^f(\overrightarrow{x}) \tag{2}$$

where $\overrightarrow{s} \in \mathbb{R}^l$ is the observation from the side channel. Depending on the implementation of f and the side channel that is used, the effective dimension of s varies, we can increase the dimension by observing under different conditions. However, it is generally considered very rare that g is invertible. Thus, it is not feasible to completely recover the original input from the side channel and use that for CNN classification.

However, for each given $\overrightarrow{x}$, we can evaluate ordered pair $(g^f(\overrightarrow{x}), f(\overrightarrow{x}))$, by carefully selecting the side channel, and establish a model $h^{f,g} : \mathbb{R}^l \to \mathbb{R}^n$:

$$\overrightarrow{y'} = h^{f,g}(\overrightarrow{s}) \tag{3}$$

This gives us an indication on what is being processed by a CNN. Leveraging this side channel, we can steal sensitive information used by the neural network. For example, health data collected on a smartphone might be used for disease diagnosis based on a neural network. A side-channel attaack may steal the diagnosis

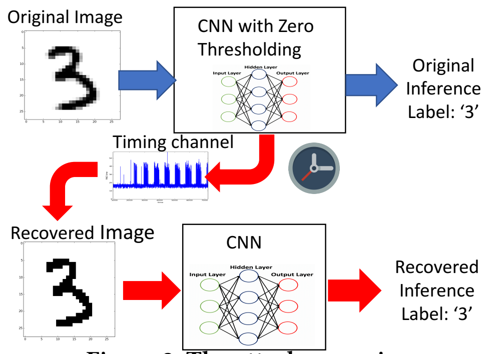
Original Image
CNN with Zero Thresholding
Original Inference Label: '3'
Timing channel
Recovered Image
CNN
Recovered Inference Label: '3'

Figure 2: The attack scenario.

result using the timing side channel, and leaks important personal information.

3 THEORY

In this section, the theory for potential side-channel attacks on a CNN is presented.

3.1 Threat Model

We consider the case where a victim CNN with zero thresholding is running on a CPU, and the inputs and the outputs of the CNN are not directly visible to the attacker. The attacker knows the structure and the weights of the CNN model and can measure the time it takes for the CNN to perform MAC operations. If the input value is smaller than the threshold, the MAC is skipped and the time will be short, otherwise the time will be longer. Using this timing variation, the attacker will try to recover the output of the victim CNN. An attacker may be able to observe the timing of a MAC operation in multiple ways. For example, timing channels on shared caches [10] reveal memory accesses. Hardware tampering [7] may observe memory accesses corresponding to MAC operations. Power consumption may also reveal whether each MAC operation is skipped or not.

Figure 2 shows the attack scenario discussed in this paper. For normal neural network inference the original image is given to the zero-thresholding CNN and the inference is performed. While the attacker cannot see the either the original image or the original inference label, he or she is able to observe the timing information through the aforementioned side channel. Through this timing channel, we recover the input image, and run the original CNN model on the recovered image to infer the output of the victim.

3.2 Zero Thresholding in CNN

Here we discuss the zero thresholing implementation in convolutional layers in CNN. We consider a single convolution layer with the dimension of input feature map $I_h \times I_w \times I_d$, output feature map $O_d \times O_h \times O_w$ and kernel size $K_h \times K_w \times I_d \times O_d$. Table 1 we describe the notations that is used for this work.

Algorithm 1 shows the convolution operation in zero-thresholding CNN.The inner most two nested loops compute the inner product of one layer of the receptive field with one layer of one convolution filter. The computations are repeated for a different layer of the receptive field, different convolution filter, and different positions of the receptive field. The zero thresholding is performed in the inner

Symbol	Meaning
$I_{h,w,d}$	height/width/depth of the input feature map
$K_{h,w}$	height/width of the convolution filter
$s_{h,w}$	stride of convolution filter in vertical/horizontal direction
$O_{h,w,d}$	height/width/depth depth of the output feature map
$E_{in,k,out}$	original input feature map/kernel/output feature map
$E_{in,out}^{recover}$	recovered input/output feature map
TH	threshold for neural network inference

Table 1: Notation of this work

most loop where the value of the input feature map is compared with the threshold value T, only if it is greater than the threshold will the MAC be performed.

Algorithm 1 Convolutional operation in zero-thresholding CNN

```
1: Input: $E_k$, $E_{in}$
2: Output: $E_{out}$
3: $E_{out} := 0$
4: for all $i$ in $1 \ldots O_d$ do
5:     for all $j$ in $1 \ldots I_d$ do
6:         for all $k$ in $1 \ldots O_h$ do
7:             for all $p$ in $1 \ldots O_w$ do
8:                 $S := 0$
9:                 for all $q$ in $1 \ldots K_h$ do
10:                    for all $r$ in $1 \ldots K_w$ do
11:                        if $E_{in}(k+q, p+r, j) > T$ then
12:                            $S := S + E_{kernel}(q, r, j, i) \times E_{in}(k+q, p+r, j)$
13:                        end if
14:                    end for
15:                end for
16:                $E_{out}(k, p, i) := E_{out}(k, p, i) + S$
17:            end for
18:        end for
19:    end for
20: end for
```

3.3 Acquiring Timing Information

There are many ways to acquire the timing information from the side channel. There are mostly-classified into two categories. Hardware-based approaches and software-based approaches.

For software-based approaches, one possible way of get the timing information is to use a cache timing channel[10] in which we constantly modify the cache from another core that shares the same cache with the core running the neural network inference.

For hardware-based approaches, two ways are equally feasible. First, by measuring the power consumption of the computing unit or memory, we can identify the time it takes to compute each layer of the CNN. The traces also reveal the number of computation in each unit time. When more input values are thresholded, there will be less computations. The memory trace also reveals information on the execution of the neural network. By attaching wires on the memory bus and identifying the read-after-write patterns, we can locate which memory addresses are used for weights, inputs, and partial sums. We can then recover the input image based on this memory access patterns.

3.4 Ideal Attack Analysis

By checking the timing of the innermost loop in Algorithm 1, we can infer where MAC is performed. With this sequence, we are able to recover the original input value of $E_{in}(k+q, p+r, j)$ thresholded by T. In this way, the recovered input feature map $E_{in}^{recover}(k,p,j)$ would be

$$E_{in}^{recover}(k,p,j) = \begin{cases} 1, & \text{if } E_{in}(k,p,j) > T \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

However, noise or other factors may prevent us from getting every bits of MAC skip patterns. Consider that we can only get the timing granularity outside the inner most two loops (i.e., one 2D-layer of convolutional filters), the height and width of the recovered feature map would be smaller, reducing from $I_w \times I_h$ to $O_w \times O_h$. The relationship of the recovered reduced-size feature map $E'_{in,recovered}(k', p', j)$ and the original feature map would be

$$E'^{recover}_{in}(k', p', j) = \begin{cases} 1, \text{if } \sum_{k<K_h, p<K_w} E_{in}(k, p, j) > K_h \cdot K_w \cdot T \\ 0, otherwise \end{cases} \tag{5}$$

On the other hand, we need input feature map of the same size as the original feature map, so that we could perform neural network inference using the neural network of the same structure. This could be done by overlaying the recovered reduced feature map to original input feature map space. For each pixel in the original feature map space, the value is calculated by following equations

$$E^{recover}_{in}(k, p, j) = \frac{\sum E'^{recover}_{in}(k', p', j)}{N} \tag{6}$$

where N is the number of overlapped recovered feature maps covering that pixel.

3.5 Attack Algorithm Implementation

Consider that given the an array representing the time of inner most loop t_j, the threshold time t_{TH}, we recover the input feature map of the particular layer.

Algorithm 2 Recovering input from zero-thresholding CNN

```
 1: Input:$t_j$, $1 \le j \le O_h O_w K_h K_w$ and $t_{TH}$, layer $i$
 2: Output: $E^{recover}_{in}$
 3: for all $k$ in $1 \ldots O_h$ do
 4:     for all $p$ in $1 \ldots O_w$ do
 5:         $t_{sum} := 0$
 6:         for all  $q$ in $1 \ldots K_h$ do
 7:             for all  $r$ in $1 \ldots K_w$ do
 8:                 $t_{sum} := t_{sum} + t_{(k \times O_w + p) \times K_h \times K_w + q \times K_w + r}$
 9:             end for
10:         end for
11:         $E^{recover}in(k, p, i) := 0$
12:         if $t_{sum} > t_{TH}$ then
13:             for all $q$ in $1 \ldots K_h$ do
14:                 for all $r$ om $1 \ldots K_w$ do
15:                     $E^{recover}_{in}(k \times s_h + q, p \times s_w + r, i) := E^{recover}_{in}(k \times s_h + q, p \times s_w + r, i) + 1$
16:                 end for
17:             end for
18:         end if
19:     end for
20: end for
```

Algorithm 2 shows the attack algorithm. We consider layer i of the input feature map. $E^{recover}_{in}$ is initialized with zero. Line 6-Line 10 calculates the total time to compute the MACs for one layer of filter at one particular receptive field location. If the time is greater than the threshold time t_{TH}, all the pixels of the receptive field in that layer is incremented by 1. The recovered input can then be normalized to fit in the input range of CNN, and the normalized input can be used to recover labels.

3.6 Metrics

In order to see the effectiveness of this attack, we identify two metrics. The first metric ACC_1 is defined as:

$$ACC_1 = \frac{\#\text{recovered labels matching orig label}}{\#\text{total labels}} \tag{7}$$

The second metric ACC_2 is defined as:

$$ACC_2 = \frac{\#\text{recovered labels matching correct label}}{\#\text{total labels}} \tag{8}$$

For these two metrics, the label is defined as the top-1 class in the CNN output vector, if the top-1 class in the recovered inference matches the top-1 class in the original inference or the label given by the dataset, we count it as a match. ACC_1 reflects the the capability of attack in recovering original CNN output, while ACC_2 reflects the capability to match correct prediction. For a reasonable model, the original CNN inference label should be close to the label given by the dataset, however, in reality, this could be different.

4 EXPERIMENTS SETUP AND RESULTS

We implement the zero-thresholding CNN using tiny-dnn[1]. We use the MNIST handwritten digit dataset for testing and we use LeNet-5 [9] as the neural network. All the experiments are completed on a two-core Intel Core-i5 CPU operating at the frequency of 2.6GHz with hyperthread enabled with 4GB memory.

4.1 Timing Measurement

We record the value `RDTSC` register in the x86 processor to record the cycle count of the innermost loop in convolutional layers. Figure 3 shows the cycle counts for innermost loops for number “2” and “7”. We clearly see different characteristics for different numbers.

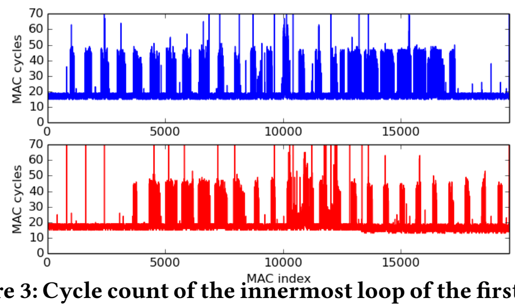

Figure 3: Cycle count of the innermost loop of the first 19600 loops for inputs respective to number “2” (top) and “7”(bottom).

4.2 Ideal Attack Experiments

LeNet-5 is a CNN with five layers of neurons. The first two layers of connections (i.e., connection between input and the first layers of neurons, and the connection between first and second hidden layers of neurons) are convolutional layers, while the last two layers of connections are fully connected layers. We observe the timing of the inner most loop in Algorithm 1 for first convolutional layers (i.e., the connections between input and first hidden layers). Using this channel, we recover the input image of LeNet and we then perform inference, thus guessing the original inference label of the CNN. The range of input is normalized to range between 0 and 1 and each value is quantized with 8 bits.

Recovery by pixel: as mentioned in Section 3.4, recovering the input data based on the timing channel of zero threshold neural network can be modeled as filtering the original input feature map with a threshold. The recovered input data can then be used for inference to recover the prediction label.

Table 2 shows the prediction accuracy of the CNN with different threshold values. Since the input image is normalized between 0 and 1 and is quantized with 8 bits, the minimal threshold change that make a difference would be $\frac{1}{2^8} \approx 0.004$, thus we select the values as shown in the table. The accuracy is calculated using the MNIST testing set containing 10000 samples. The original model used to calculate ACC_1 and ACC_2 is LeNet-5 trained on MNIST training set with accuracy of 96.56% on the testing set. In this study, the recovered input is simply computing using Equation (4), and is directly given to the original LeNet-5 for inference. The study represents the ideal case where an attack can obtain perfect information on each MAC operation. For retraining, we start with original LeNet-5 and use the recovered images and the correct labels in the training set, we retrain for one epoch.

TH	ACC_1	ACC_2	ACC_1 w/ retrain	ACC_2 w/ retrain
0.005	93.27%	92.16%	91.95%	91.24%
0.01	93.37%	92.28%	92.64%	92.00%
0.05	94.27%	93.25%	94.68%	94.08%
0.1	94.97%	93.79%	93.48%	92.98%
0.5	97.77%	95.90%	94.31%	93.95%
0.9	96.69%	95.00%	91.57%	90.78%
0.95	94.43%	92.97%	91.98%	91.15%
0.99	82.38%	81.64%	87.44%	86.54%
0.995	41.73%	41.22%	63.17%	62.42%

Table 2: Experimental result on attack recovering based on Equation (4).

From Table 2, we can see that the accuracy is highest when threshold is around 0.5, and drops to around 40% when the threshold is 0.995. Intuitively, when the threshold is high, there will be less black pixels, and the black lines may break, leading to significant accuracy drop. Since the accuracy of original model is high, the difference between ACC_1 and ACC_2 is small for all thresholds. We observe that retraining is helpful to improve the accuracy when the original accuracy is low, but the accuracy degrades a little when the original accuracy before training is high.

Recovery by filter: as described in Equation (5), it is also possible to recover one input image only from the total cycle count per layer of the convolutional filter. Compared to Table 2, the accuracy achieved without retraining is low. Thus, we apply retraining (with 5 and 10 epochs) with the correct labels from the training set. We can see from the result that retraining helps increase the accuracy. However compared to recovery by pixel, when setting a high TH, the accuracy after retraining is still lower. This can be explained from the fact that by thresholding the entire receptive field rather than individual pixels inside the receptive field, much of the information is wiped out at a high threshold. Thus, it is difficult to recover the input image.

TH	epoch=0		epoch=5		epoch=10	
	ACC_1	ACC_2	ACC_1	ACC_2	ACC_1	ACC_2
0.005	17.40%	17.49%	90.66%	90.55%	91.96%	92.18%
0.01	18.58%	18.70%	90.83%	90.45%	91.97%	91.98%
0.05	25.03%	25.19%	91.07%	90.67%	91.89%	91.84%
0.1	38.28%	38.16%	87.08%	86.47%	88.66%	88.11%
0.5	22.57%	22.57%	56.48%	56.05%	58.41%	58.12%
0.9	13.98%	14.00%	31.90%	31.72%	32.45%	32.28%
0.95	13.92%	13.95%	31.43%	31.31%	31.89%	31.72%
0.99	13.92%	13.95%	31.52%	31.41%	32.11%	31.94%
0.995	13.92%	13.95%	31.52%	31.41%	32.11%	31.94%

Table 3: Experimental result on attack recovering based on Equation (5).

TH	t_{TH}	no retrain		epoch=5	
		ACC_1	ACC_2	ACC_1	ACC_2
0.05	20	25.76%	25.68%	26.07%	26.15%
	30	57.88%	57.25%	52.20%	51.97%
	40	17.27%	17.05%	17.01%	16.81%
0.5	20	19.84%	19.81%	24.80%	24.52%
	30	13.60%	13.69%	16.09%	16.03%
	40	22.83%	22.76%	27.75%	27.58%

Table 4: Experiment result of attack using Algorithm 2.

4.3 Timing Attack

To understand the effectiveness of the attack under more realistic, noisy environment, we perform the timing attack on LeNet-5 using recorded timing information. We use Algorithm 2 for the first layer of the first convolutional filter of the first layer LeNet ($O_h = O_w = 28$, $K_h = K_w = 5$), we use two zero-thresholding LeNet-5 threshold with $TH = 0.05$ and $TH = 0.9$. The values of t_{TH} used in Algorithm 2 are chosen based on our observations of average cycle count.

Table 4 shows the result of the timing attack. We use threshold TH value of 0.05 and 0.5. Then we perform the inference and get the cycle count trace. Using t_{TH} of 20, 30, and 40 cycles, we recover the input images. We then use these images for inference, without retraining CNN or with 5 epochs of retraining. The accuracy is generally lower than the those in Table 2 and Table 3, probably due to the noise in real-world timing measurements. However, for appropriate TH and t_{TH} values, we can still achieve over 50% accuracy for recovering the inference labels.

5 DISCUSSIONS

5.1 Performance-Security Trade-off

With a higher threshold for zero thresholding, less MACs are computed and the CNN computation takes less time. This leads to a lower energy consumption and better performance. The accuracy of the CNN largely remains unchanged as threshold increases, then the accuracy drops for high threshold values. The attack accuracy will first increase with threshold then decrease with it. CNN designers who decide the threshold value would want to maximize performance while minimizing the accuracy of the side-channel attack. While we expect increasing the threshold to improve the performance, if security is a concern and we would want to bound the attack accuracy, we may need to lower the threshold. This leads to a performance penalty. If we set the threshold to be zero, no

MAC operation will be removed and the timing channel can be eliminated.

5.2 Implication on Neural Network Implementation

Base on the result, we provide several implications on how to design and implement neural network securely.

(1) Reduce the time dependency in neural network design and implementation. By waiting a constant time during skipping of a MAC, we still get the benefit of saving energy, though we may lose some benefit of saving time.
(2) Avoid time dependent hardware. Some floating point operations are known for having timing channel that leaks the input value [3]. For neural network security, it is suggested that these hardware be avoided.
(3) Isolate neural network execution. Either deploying it on dedicated hardware or allocate partitioned hardware resource (core, cache, and memory) will help closing detectable side channels from adversaries.

6 RELATED WORK

6.1 Attack on Models

Rather than aiming at neural network input or output data, other works seek to steal the models. Due to the multilayer nature and large number of parameters, successful stealing of neural network based on side-channel has not been demonstrated. However, this it done for other machine learning models. In [14], decision tree model inside the cloud is revealed by legal API queries to the model. This implies possibility to reveal at least partial information of neural network based on continuously query.

6.2 Inference Phase CNN Data Protection

To protect information leakage on neural network data, especially when a user outsource its data to the untrusted neural network and let it process the data, homomorphic encryption is used[4]. The results show negligible overhead. However, this scheme uses a partially homomorphic encryption which limits the depth of operations we can perform on encrypted data, and may not be efficient for larger networks such as GoogleNet or AlexNet.

6.3 Training Phase CNN Data Protection

Our attack is performed during CNN inference. Data protection in CNN training is also important. Data sharing is also a big concern when training the neural network. Neural network training involves back-propagation, and encrypting input data will make this process meaningless. To effectively train the neural network, the trainer have to know something related to the plaintext of data, but sharing plaintext is sometimes not desirable. To resolve this issue, in [13], privacy-preserving deep learning is proposed to train the neural network using input from different sources but without sharing input data to the others. The key idea is that instead of sharing the input data directly, we selectively share part of the trained weights to the others so that others can benefit from the result of the training by our private data.

7 CONCLUSION

In this paper, we provide theoretical foundations and studies on a new timing channel attack on neural networks with zero-thresholding, which skip MAC operations depending on the input values. We demonstrate both theoretical and practical attacks to reveal the information processed by the neural network. The experimental results suggest that attackers can guess the inference output of a CNN with a reasonble accuracy. Our future direction includes demonstrating the attack on more practical settings using cache and memory side channels, studying attacks on custom accelerators, and developing protection mechanisms.

REFERENCES

[1] 2017. *tiny-dnn: header only, dependency-free deep learning framework in C++14.* http://tiny-dnn.readthedocs.io

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: ineffectual-neuron-free deep neural network computing. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on.* IEEE, 1–13.

[3] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. 2015. On subnormal floating point and abnormal timing. In *Security and Privacy (SP), 2015 IEEE Symposium on.* IEEE, 623–639.

[4] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16).* JMLR.org, 201–210. http://dl.acm.org/citation.cfm?id=3045390.3045413

[5] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture.* IEEE Press, 243–254.

[6] Babak Hassibi and David G Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems.* 164–171.

[7] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. (2016).

[8] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference.* Springer, 104–113.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[10] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *Security and Privacy (SP), 2015 IEEE Symposium on.* IEEE, 605–622.

[11] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture.* ACM, 27–40.

[12] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *Proceedings of the 43rd International Symposium on Computer Architecture.* IEEE Press, 267–278.

[13] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security.* ACM, 1310–1321.

[14] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs.. In *USENIX Security Symposium.* 601–618.

[15] Weize Yu, Orhun Aras Uzun, and Selçuk Köse. 2015. Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE.* IEEE, 1–6.