# Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks

Weizhe Hua, Zhiru Zhang, and G. Edward Suh
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{wh399, zhiruz, gs272}@cornell.edu

## ABSTRACT

A convolutional neural network (CNN) model represents a crucial piece of intellectual property in many applications. Revealing its structure or weights would leak confidential information. In this paper we present novel reverse-engineering attacks on CNNs running on a hardware accelerator, where an adversary can feed inputs to the accelerator and observe the resulting off-chip memory accesses. Our study shows that even with data encryption, the adversary can infer the underlying network structure by exploiting the memory and timing side-channels. We further identify the information leakage on the values of weights when a CNN accelerator performs dynamic zero pruning for off-chip memory accesses. Overall, this work reveals the importance of hiding off-chip memory access pattern to truly protect confidential CNN models.

## 1 INTRODUCTION

Convolutional neural networks (CNNs) are quickly becoming an essential tool in a wide range of machine learning applications. In many application scenarios, CNN models — both its network structure and learnable parameters (i.e., weights) — need to be protected as confidential information: (1) for companies that rely on a CNN to provide a core or value-added service, the underlying neural network model represents an important piece of intellectual property; (2) in personalized applications such as digital assistants, CNN models are trained using private data, and the weights need to be kept confidential for privacy [13]; (3) furthermore, recent studies on the adversarial network show that an attacker can intentionally affect the outcome of CNN-based classification and object detection by perturbing input images when the network model is known [5].

This paper investigates reverse-engineering attacks on CNN models exploiting information leaks through memory and timing side-channels. Specifically, we study attacks on a hardware accelerator that is protected by secure processor techniques similar to the scheme used in Intel SGX [2]. In this setting, an adversary can feed inputs to a protected computation and observe off-chip accesses, but cannot observe or change the computation and the internal state. Surprisingly, we show that an adversary can effectively reverse engineer both the structure and the weights of an encrypted CNN model running on a hardware accelerator that performs the inference (i.e., forward propagation). Because the CNN states (feature maps) and parameters (weights) are often quite large, it is impractical to hold all feature maps, weights, and intermediate results in the on-chip memory of an accelerator. As a result, CNN accelerators typically store feature maps and weights in off-chip memory and access them as needed. Even if data values are encrypted, memory access patterns reveal which memory locations are accessed and whether each access is a read or a write. In this study, we show that the memory access patterns expose key parameters of the network structure such as the number of layers, input/output sizes of each layer, the size of filters, data dependencies among layers, etc. Given this information, an attacker can infer a small set of possible network structures by further considering the execution time of a CNN accelerator, which indicates the amount of computation. In our experiments, we demonstrate the proposed attack by reversing engineering the structures of two popular CNN models in AlexNet [9] and SqueezeNet [8].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*DAC '18, June 24–29, 2018, San Francisco, CA, USA*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/https://doi.org/10.1145/3195970.3196105

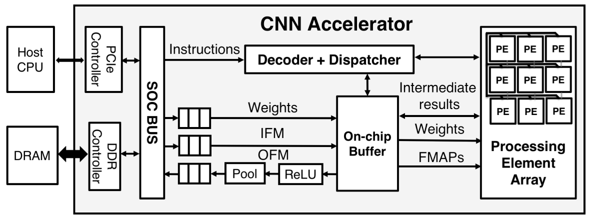


**Figure 1: A typical CNN inference accelerator.**

In addition to revealing the network structure, this study shows that the memory access patterns also leak information on weight values when dynamic zero pruning is used for off-chip memory accesses. The optimization is based on the observation that the feature maps from the intermediate layers of a CNN model contain a large number of zeros. Recent studies in [1, 11, 12] have shown that these feature maps can be compressed in DRAM by only storing non-zero values and the associated indices to significantly reduce the memory bandwidth usage. Unfortunately, this optimization leaks the number of zero-valued pixels pruned by the activation function, which can be leveraged to infer the ratio between each weight and the bias value. To the best of our knowledge, this paper represents the first study on reverse engineering of convolutional neural network models on hardware accelerators, especially in the context of exploiting the side channel through memory access patterns.

The rest of the paper is organized as follows: Section 2 defines the assumed threat model; Sections 3 and 4 present two reverse-engineering attacks on the structure and the weights of a CNN model and evaluate the effectiveness of the proposed attacks; Section 5 discusses the related work, and Section 6 concludes the paper.

## 2 THREAT MODEL

Figure 1 shows a typical CNN inference accelerator architecture that is used in this study. In order to fit the input feature maps (IFMs) and filters in the on-chip buffers, IFMs and filters are partitioned into small tiles. Then, the convolution operation is performed over

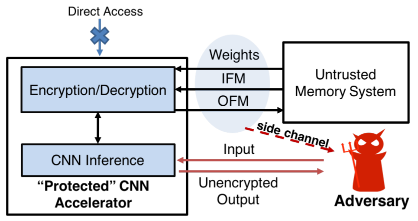


**Figure 2: Threat model.**

all the tiles sequentially. For each tile, the accelerator receives instructions from the host CPU, reads an IFM tile and corresponding weights from an off-chip DRAM into on-chip buffers, performs matrix multiplications and accumulations in the processing element (PE) array, and store the intermediate results back to on-chip buffers. After computing over all tiles, the accelerator combines the intermediate results and writes an output feature map (OFM) back to DRAM after activation and pooling. The shared cache between the accelerator and CPU is not considered as part of the architecture. The reasons are threefold: (1) the data locality has been exploited by the on-chip buffers; (2) the shared cache is not able to hold all feature maps (FMAPs) and weights; (3) using shared cache to store FMAPs and weights can significantly degrade the performance of other applications, which are sharing the cache. The FMAPs and weights are typically stored in DRAM, while the intermediate results are kept in on-chip buffers. After finishing the computation of all layers in a forward-propagation fashion, the accelerator returns the probabilities of each class as the classification result.

Figure 2 illustrates the threat model, which captures the common protection capabilities provided by today's secure processor technologies such as Intel SGX [2]. The internal operations and state of the CNN accelerator cannot be directly observed or changed by an adversary. The CNN accelerator encrypts feature maps (input/output of each layer) and weights in DRAM so that their values are protected. However, the adversary can control inputs to the accelerator and observe the address and the type (read or write) for each off-chip memory access. The above threat model represents what is typically employed for today's secure processors and the level of protection that can be implemented with low overhead. For example, Intel SGX is designed to provide an isolated and protected execution environment for a security-critical program, even when an operating system is untrusted or a system is physically exposed. In such systems, the internal operations and state of a program are protected, although inputs and outputs are still exposed to an adversary. Memory accesses can be directly observed through physical probing of a memory bus or inserting hardware Trojan. A compromised OS can also observe memory accesses through side channels such as page faults and cache conflicts [17] or by repeated reading memory to detect changes [10].

The objective of the reverse-engineering attacks studied in this paper is to construct a duplicated CNN model that has comparable accuracy to the target model by observing the hardware accelerator. This paper studies two different reverse-engineering attacks on CNNs. The first attack aims to reverse engineer a network structure (Section 3). The second attack aims to find weight values (Section 4). Table 1 lists the assumptions made by each attack.

## 3 STRUCTURE REVERSE ENGINEERING

In this section, we discuss how a convolutional neural network structure can be identified based on the memory access patterns.

| Assumptions | Reverse engineering attacks on | |
|---|---|---|
| | Structure (Section 3) | Weights (Section 4) |
| Observe memory access patterns | **Y** | **y** |
| Observe the input value | **N** | **Y** |
| Control the input value | **N** | **Y** |
| Possess training data | **Y** | **N** |
| Know the network structure | / | **Y** |

**Table 1: Assumptions for each attack – Y: Yes; y: only write accesses need to be visible; N: No; (/): not applicable.**

| Layer parameter | Definition |
|---|---|
| $W_{IFM/OFM}$ | *Width* of the input/output feature map |
| $D_{IFM/OFM}$ | *Depth* of the input/output feature map |
| $F_{conv/pool}$ | *Width* of the conv/pooling filter |
| $S_{conv/pool}$ | *Stride* of the conv/pooling filter |
| $P_{conv/pool}$ | *Number of pixels* padded in the conv/pooling layer |
| $P$ | Indicate the *existence* of the pooling layer |

**Table 2: Parameters to define a CNN structure.**

### 3.1 Attack Methodology

In order to construct a neural network, an adversary needs to know the number of layers, parameters for each layer, and connections among layers. A typical CNN uses a simple sequential connection only between consecutive layers. More recent proposal [7] introduces an additional bypass connection among layers.

We show that memory access patterns relatively easily reveal the overall layer structure through read-after-write (RAW) dependency. During the CNN inference, the RAW dependency on FMAPs must be preserved by the accelerator, regardless of its micro-architecture details and data reuse strategies. The OFMs are written by a preceding layer and is read as the IFMs by the following layer. Since FMAPs are stored off-chip, this RAW dependency is reflected in the memory trace and visible to the adversary as a write followed by a read on the same memory address. These RAW dependencies can be used by the adversary to identify the boundary as well as the connections between layers. More concretely, the beginning of a new convolutional/fully connected layer is revealed by the first read access on a memory address that was previously written.

Once the layer boundaries are identified, the adversary needs to further reverse engineer the key parameters of each layer. The first step towards this goal is to distinguish memory accesses to filters, IFM, and OFM. Since the filters are read-only and not updated during the inference, the adversary can differentiate memory accesses to filters from those accessing FMAPs. The read/write operations on IFM and OFM can also be distinguished since they have different access patterns. During the computation within a layer, memory locations holding OFM will only be written, typically once. In contrast, if the adversary observes a read access on an address written in the previous layer before, this read must be for IFM. FMAPs and filters are stored as arrays in memory, which means that each is stored in its own contiguous memory locations. Therefore, an adversary can infer the sizes of IFM ($SIZE_{IFM}$), OFM ($SIZE_{OFM}$), and filters ($SIZE_{FLTR}$) of each layer by observing the memory locations accessed for each data structure within a layer.

So far, we have shown that the memory access pattern of a CNN accelerator reveals the number of layers, data dependencies (connections) among layers, the size of the IFM, OFM, and filters for each layer in the target CNN model. However, not all operations in the neural network are explicitly revealed by the memory access pattern. For example, a CNN performs an activation operation after each convolution followed by an optional pooling operation. These three operations are often merged and performed together as a single layer in CNN accelerator for efficiency. As a result, the internal outputs of these three operations are invisible to the adversary.

There are 11 structural parameters that the adversary needs to determine for each layer in order to fully define the network structure, as listed in Table 2. The problem of identifying each layer structure can be formulated as solving the 11 integer parameters with the following equations:

$$SIZE_{IFM} = W_{IFM}^2 \times D_{IFM} \tag{1}$$

$$SIZE_{OFM} = W_{OFM}^2 \times D_{OFM} \tag{2}$$

$$SIZE_{FLTR} = F_{conv}^2 \times D_{IFM} \times D_{OFM} \tag{3}$$

$$W_{OFM} = \frac{\frac{(W_{IFM} - F_{conv} + P_{conv})}{S_{conv}} + 1 + (P_{pool} - F_{pool}) \times P}{S_{pool} \times P + \bar{P}} + P \tag{4}$$

$$S_{conv} \le F_{conv} \le \frac{W_{IFM}}{2} \tag{5}$$

$$S_{pool} \le F_{pool} \le \frac{(W_{IFM} - F_{conv} + P_{IFM})}{S_{conv}} + 1 \tag{6}$$

$$P_{conv} < F_{conv} \tag{7}$$

$$P_{pool} < F_{pool} \tag{8}$$

Equations (1)-(3) are derived from the size of the FMAPs and weights revealed by the memory access patterns. Equation (4) expresses the relationship between the width of IFM and OFM. The inequalities are based on the following practical considerations. First, $F_{conv}$ and $F_{pool}$ should be no less than the stride to cover all the pixels in IFM. Otherwise, some pixels in IFM are not connected with the weights and become redundant. Second, the filters should be smaller compared to the width of IFM. Lastly, $P_{conv}$ and $P_{pool}$ should be smaller than $F_{conv}$ and $F_{pool}$ respectively because the convolutional/pooling filter operating on zero-valued pixels is equivalent to adding zero-padding for the next layer.

In our threat model, the adversary also has the knowledge of input and output of the accelerator which reveals the $W_{IFM}$ and $D_{IFM}$ of the first layer and the $D_{OFM}$ of the last layer. Although there is no previous write on the IFM of the first layer, the $SIZE_{IFM}$ can be calculated using Equation 1 and thus distinguished from the weights. Moreover, the $W_{OFM}$ of the last layer is one since there is exactly one score for each class. With these additional constraints, adversary can enumerate all possible parameters that satisfy Equation (1)-(8) for the first layer and feed the possible $W_{OFM}$ and $D_{OFM}$ as the constraints for the second layer. Through enumerating possible parameters layer-by-layer, the network structure is the combination of possible configurations of each layer.

In order to further reduce the number of possible structures, the execution time of each layer is measured by recording the number of clock cycles between the boundaries and introduced as an additional constraint. The number of MAC operations of a specific layer can be computed with the layer parameters (# of MACs $= W_{OFM}^2 \times D_{OFM} \times F_{conv}^2 \times D_{IFM}$). Given that the inference of most CNN models is compute-bound, we assume that the execution time is roughly proportional to the number of MAC operations. Thus, the execution time ratio between layers should be consistent with the ratio of MAC operations for the correct configuration.

Once a small number of candidate structures are identified through the reverse engineering, an adversary can pick the best structure by training and comparing the accuracy. Algorithm 1 summarizes the overall structure reverse-engineering attack procedure.

| Networks | LeNet | ConvNet | AlexNet | SqueezeNet |
|---|---|---|---|---|
| # of layers | 4 | 4 | 8 | 18 |
| # of possible structures | 9 | 6 | 24 | 9 |

**Table 3: Possible structures for different networks.**

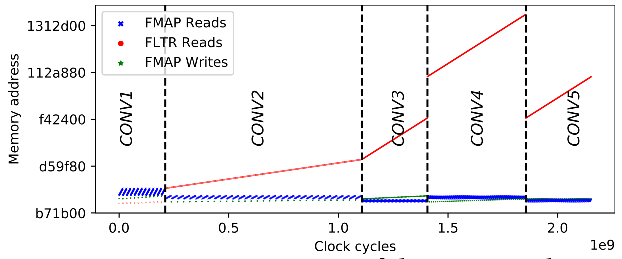


**Figure 3: Memory access pattern of the FPGA accelerator.**

**Algorithm 1** Steps to uncover the possible CNN structures

```
1: Identify layer boundaries by observing the RAW dependency on FMAPs
2: Record the execution time of each layer and calculate the $SIZE_{IFM}$, $SIZE_{OFM}$, and $SIZE_{FLTR}$ based on the memory access pattern
3: Find possible configurations for each layer with the constraints stated in Equations (1)-(8)
4: Filter out the configurations where the number of MAC operations and the execution time do not match
5: List valid combination of layers as possible structure which satisfies $(W_{OFM_i} = W_{IFM_{i+1}}) \wedge (D_{OFM_i} = D_{IFM_{i+1}})$
```

## 3.2 Case Studies

To evaluate the proposed structure reverse-engineering attack, we performed case studies on popular CNN models: an 8-layer AlexNet [9] and a more recent 18-layer SqueezeNet [8]. We also studied other smaller networks such as LeNet and ConvNet. The number of possible structures identified by the proposed attack is summarized in Table 3.

We implemented an FPGA accelerator for AlexNet using Vivado HLS and performed the attack by inserting a hardware Trojan to collect the memory trace of the accelerator. The layer boundaries are identified by observing the RAW dependency on FMAPs, as depicted in Figure 3. Table 4 lists the possible configurations for each layer in AlexNet. A total of 24 valid combinations are uncovered by applying the proposed method on the FPGA prototype. A pooling layer, if exists, is considered as part of the convolutional layer. AlexNet consists of five CONV layers and three FC layers. The FC layers are not listed in the Table 4 because they use the largest possible filter size ($W_{IFM}^2 \times D_{IFM} \times D_{OFM}$) and always have a unique configuration with respect to the Equations (1)-(8). The original AlexNet structure consists of $CONV1_1$, $CONV2_1$, $CONV3_1$, $CONV4$, and $CONV5_1$. The top-1 validation accuracies of 24 possible structures are shown in Figure 4. The original AlexNet achieves the fourth highest accuracy (57.3%). The attack also found three other network structure, which are slightly different from the original AlexNet and have higher accuracy. The best structure achieves 12.3% higher accuracy than the worst one, showing the importance of a good network structure.

| $Layer$ | $W_{IFM}$ | $D_{IFM}$ | $W_{OFM}$ | $D_{OFM}$ | $F_{conv}$ | $S_{conv}$ | $P_{conv}$ | $F_{pool}$ | $S_{pool}$ | $P_{pool}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $CONV1_1$ | 227 | 3 | 27 | 96 | 11 | 4 | 1 | 3 | 2 | 0 |
| $CONV1_2$ | 227 | 3 | 27 | 96 | 11 | 4 | 2 | 4 | 2 | 0 |
| $CONV2_1$ | 27 | 96 | 13 | 256 | 5 | 1 | 2 | 3 | 2 | 0 |
| $CONV2_2$ | 27 | 96 | 26 | 64 | 10 | 1 | 4 | N/A | N/A | N/A |
| $CONV3_1$ | 13 | 256 | 13 | 384 | 3 | 1 | 1 | N/A | N/A | N/A |
| $CONV3_2$ | 26 | 64 | 13 | 384 | 6 | 2 | 2 | N/A | N/A | N/A |
| $CONV4$ | 13 | 384 | 13 | 384 | 3 | 1 | 1 | N/A | N/A | N/A |
| $CONV5_1$ | 13 | 384 | 6 | 256 | 3 | 1 | 1 | 3 | 2 | 0 |
| $CONV5_2$ | 13 | 384 | 12 | 64 | 6 | 1 | 2 | N/A | N/A | N/A |
| $CONV5_3$ | 13 | 384 | 3 | 1024 | 3 | 2 | 0 | 2 | 2 | 0 |
| $CONV5_4$ | 13 | 384 | 3 | 1024 | 3 | 2 | 0 | 4 | 1 | 0 |
| $CONV5_5$ | 13 | 384 | 3 | 1024 | 3 | 2 | 1 | 3 | 2 | 0 |
| $CONV5_6$ | 13 | 384 | 4 | 576 | 2 | 1 | 0 | 3 | 3 | 0 |

**Table 4: Possible AlexNet layer configurations – N/A indicates that there is no pooling operation.**

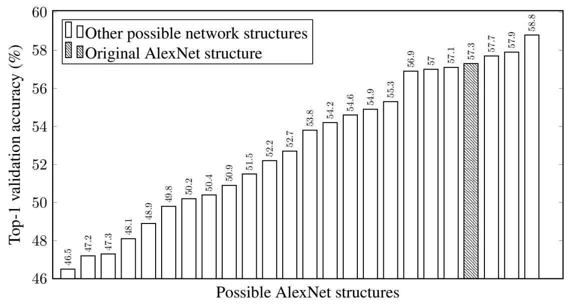


**Figure 4: Top-1 validation accuracy among 24 possible structures for AlexNet.**

In addition to AlexNet, we also studied reverse engineering of a more recent state-of-art CNN structure. Compared to AlexNet, two new trends emerged in network structure designs in the past few years. GoogLeNet [16] proposed concatenating multiple convolution filters with different $F_{conv}$ as a module and using this module repeatedly to form the network. ResNet [7] introduced a bypass connection between two non-adjacent layers. SqueezeNet adopted both of these structural changes and achieves an accuracy comparable to that of AlexNet while using 50x less weights.

We use SqueezeNe as an example to demonstrate the effectiveness of the proposed attack on a more recent network. SqueezeNet consists of two CONV layers and eight fire modules and each fire module is made of concatenating one 1x1 and two 3x3 convolutional filters. The 1x1 CONV layer uses small $D_{OFM}$ to squeeze the size of FMAPs. The OFM of the 1x1 CONV layer feeds into two 3x3 CONV layers in parallel and the OFMs of the two following layers are concatenated along the depth dimension as the final OFM of the fire module. Compared to normal feed-through structure, SqueezeNet also introduces three bypass paths connecting non-adjacent fire modules. The bypass path is combined with the feed-through path by applying element-wise additions on the OFMs.

To our best knowledge, there is no accelerator design that uses dedicated hardware for the fire module because it can be implemented using existing CONV layer accelerators. The three CONV layers will be executed sequentially. The IFM is convolved with 1x1 CONV filter[1] first and the OFM of this 1x1 convolution is then convoluted with 1x1 and 3x3 CONV filters in series. Assuming that the three convolution operations are executed sequentially on a CNN accelerator, the adversary can observe the RAW dependency between layers which reveals the structure of the fire module. Instead of having RAW dependency between the neighboring layers, the

[1] $N \times N$ filter is an abbreviation of $N \times N \times D_{OFM}$ filter. And $N \times N$ convolution stands for performing convolution with $N \times N \times D_{OFM}$ filter.

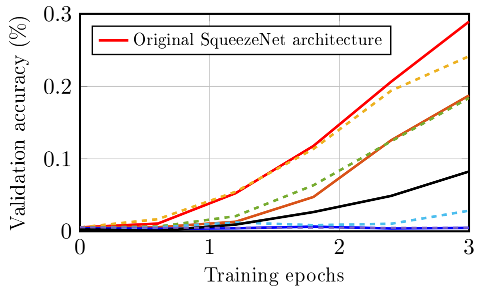


**Figure 5: The top-5 validation accuracy of nine possible structures for SqueezeNet.**

bypass path introduces extra RAW dependency across non-adjacent layers. The easiest way of implementing this bypass function is to wait until two OFMs from two paths are both ready, then load them from memory and perform the element-wise additions. This method is adopted by Caffe and TensorFlow. In both frameworks, a separate element-wise layer is introduced to realize the bypass function. Assuming that the CNN accelerator follows the same strategy, the bypass path can also be detected from the RAW dependency in memory accesses.

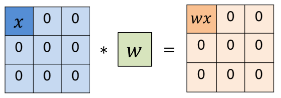
We performed the proposed attack on SqueezeNet, and found that there are nine possible configurations for CONV1 layer, 12 possible configurations for the fire modules, and two possible configurations for the CONV10 layer. Theoretically, there exists 329 valid combinations, which makes it expensive to test all valid combinations to identify the network structure. However, large CNNs are typically constructed in a modular fashion, where the same building block is reused in order to reduce the complexity. If we assume that the structures of all fire module are identical, there exists only one valid configuration for the fire module and CONV10 layer. The number of possible CNN structure candidates is reduced to nine. This example shows that the number of possible structures does not necessarily grow exponentially with the number of layers.

The time to search for the best network structure among the possible candidates can be reduced by using short training to quickly filter out unpromising candidates with low classification accuracy. For example, CNN training is often performed using many epochs, where one epoch goes through the entire training dataset once. Figure 5 shows the accuracy of possible candidates for SqueezeNet when only three epochs were used for training. The original SqueezeNet proposed using around 70 epochs for training. There is a significant difference in accuracy among the structure candidates even with a small number of epochs, suggesting that unpromising network structures can be quickly filtered out.

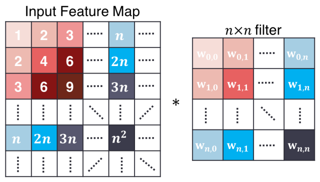
## 4 REVERSE ENGINEERING WEIGHTS EXPLOITING ZERO PRUNING

In this section, we introduce an attach to obtain information on weights when an optimization technique is used to prune zeros in FMAPs. This attack also exploits information leaks through memory access patterns of the CNN accelerator. However, only the write accesses need to be observable for the attack.

In recent years, multiple CNN accelerator designs observed that the ReLU function leads to a large number of zeros in the feature maps and proposed to dynamically prune those zeros [1, 11, 12].



(a) 1x1 CONV with stride 1.



(b) $n \times n$ CONV with stride 1.

**Figure 6: 1x1 and $n \times n$ convolution.**

Such zero pruning technique is shown to be quite effective, reducing convolution operations by 40% on average without affecting the classification accuracy. The zero pruning also reduces the number of memory accesses by only writing and reading non-zero values, for example using a run-length encoding. However, the dynamic zero pruning reveals the number of zeros in OFM. An adversary who can observe memory accesses can detect when the number of zeros changes. We show that this dynamic behavior leaks information about the weights of the CNN model. The adversary cannot precisely recover all weights, but can recover substantial information so that each weight can be expressed as a function of one bias value.

## 4.1 Attack Methodology

In a CNN accelerator, different inputs lead to a different number of non-zero pixels in the OFM of each layer. With the dynamic zero pruning, the adversary can observe the changes in the number of non-zero pixels in OFM. If the activation function $f$ maps negative values to zero as in ReLU, the change in the number of zeros in OFM actually reveals when a pixel crosses the zero boundary. The value of each pixel $y$ in OFM can be expressed as a function of the value of pixels $x$ in IFM, weights $w$, and a bias $b$, that is $y = f(\sum_i w_i \cdot x_i + b)$. Therefore, if an adversary can slowly change the pixel values in IFM and observe the number of zeros in OFM, the adversary can effectively find out when the value of a pixel in OFM becomes zero ($\sum_i w_i \cdot x_i + b = 0$). If the adversary can find out which pixel is zero, then one can write a set of linear equations for $w_i$ and $b$ given that the values of input $x_i$ are known.

Unfortunately, the dynamic zero pruning only leaks the number of zero-valued pixels in OFM while the exact locations of those pixels along $W$, $H$, and $D$ dimensions remain unknown. To solve the problem, an adversary can provide carefully crafted inputs to the accelerator. Reverse engineering the weights of 1x1 CONV, 2x2 CONV, and FC layers are more straightforward compared to the general case with a larger filter ($F_{conv} > 2$). We first illustrate our approach using a special case when $F_{conv} = 1$. Then, the approach is extended to work with any CONV filter size.

Figures 6a shows an example where a $3 \times 3$ IFM is convolved with a $1 \times 1$ filter. In the 1x1 convolution, a single weight is shared by all $W_{IFM} \times H_{IFM}$ pixels on the same 2-D plane. In other words, the product of any pixel in IFM and the weight only affect one pixel in the corresponding OFM. This relationship can be expressed by $y_{i,j} = x_{i,j} \cdot w + b$. Thus, for a 1x1 CONV layer, an adversary can vary the value of one specific pixel in IFM (denoted as a variable $x$) and set all other pixels to be 0 as depicted in Figure 6a. By monitoring the number of non-zeros in OFM, the adversary can easily determine if $w \cdot x + b > 0$ holds for a given $x$. Through a binary search on the value of $x$, the adversary will be able to find a *maximum* $x_H$ and a *minimum* $x_L$, which satisfy:

$$(w \cdot x_H + b \leqslant 0) \wedge (w \cdot x_L + b > 0) \tag{9}$$

Then $(x_H + x_L)/2$ can be estimated to be the input that produces zero ($y = 0$), which we subsequently refer to as a zero crossing point. It can be used to approximate the value of $b/w$.

Figure 6b illustrates the general case of applying a $n \times n$ CONV filter on an IFM ($W_{IFM} > 2n$). The number of connections between each pixel and the weights is shown in the IFM. Having a connection with the weight $w_{i,j}$ means that pixel contributes to the corresponding output pixel $y_{i,j}$. For instance, pixel $x_{0,0}$ at the top-left corner is only connected with $w_{0,0}$ and contributes to $y_{0,0}$ whereas pixel $x_{1,0}$ is connected with $w_{0,0}$ and $w_{1,0}$, contributing to both $y_{0,0}$ and $y_{1,0}$. $x_{n,n}$, which is a pixel on the $n^{th}$ column and row, is connected with all $n^2$ weights in the 2-D filter.

As the weights closer to the corner contributes to a less numbers of output pixels, we can iteratively find the ratio between each filter weight and the bias[2]. For example, performing a binary search on $x_{0,0}$ reveals the approximate value of $b/w_{0,0}$. Then, a search on $x_{1,0}$ while setting all other pixels to zero gives two regions where the number of zero-valued outputs changes between 0 and 2. Each one of these zero-crossing points is the approximate value of $b/w_{0,0}$ or $b/w_{1,0}$. Given that $b/w_{0,0}$ is already known, an attacker can determine which value corresponds to $b/w_{1,0}$. Algorithm 2 describes the proposed method for discovering all $b/w_{i,j}$ on the same 2-D plane. Note that, if $w_{i,j} = 0$, no $x_0$ and $x_1$ will satisfy Equation (9) which means no zero-crossing point can be found during the search. Therefore, zero-valued weights can be identified from missing zero-crossing points.

**Algorithm 2** Reverse engineering CNN weights

```
1: for $i = 0$ to $F_{conv} - 1$ do
2:     for $j = 0$ to $F_{conv} - 1$ do
3:         Set all inputs except $x_{i,j}$ to zero.
4:         Find $(x_H + x_L)/2$ values where the number of non-zero
           output pixels change (zero crossing points).
5:         Set the new zero-crossing point as $b/w_{i,j}$.
6:     end for
7: end for
```

A convolutional layer may be followed by a maximum or average pooling layer. These two layers are usually merged in a CNN accelerator to avoid unnecessary off-chip memory accesses. Without the pooling layer, varying $x_{1,0}$ while keeping other inputs to be zero leads to two zero-crossing points. However, when there is a max pooling layer where a 2x2 pooling window is concatenated with a $n \times n$ CONV layer, the search on $x_{1,0}$ only gives one zero-crossing point because both non-zero outputs are replaced by the same maximum value of $w_{0,0} \cdot x_{1,0}$ and $w_{1,0} \cdot x_{1,0}$. If $w_{1,0} < w_{0,0}$, this zero-crossing point corresponds to $b/w_{0,0}$. In order to also find $b/w_{1,0}$, we need to keep both $x_{0,0}$ and $x_{1,0}$ as non-zero variables as in the following equation:

$$\begin{cases} y_{0,0} = max\left\{w_{0,0} \cdot x_{0,0} + w_{1,0} \cdot x_{1,0},\ \ w_{0,0} \cdot x_{1,0}\right\} \\ y_{1,0} = w_{0,0} \cdot x_{1,0} + b \end{cases} \tag{10}$$

Given that the $b/w_{0,0}$ has already been inferred, the attacker can find the value for $x_{1,0}$ which satisfies $y_{1,0} \leq 0$. Then performing binary search on $x_{0,0}$ to find the zero crossing point for $w_{0,0} \cdot$

[2]The same bias is shared by all the weights in one filter.

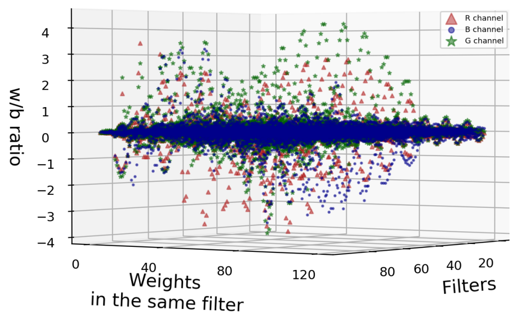


**Figure 7: Weight/Bias ratio of the CONV1 layer in AlexNet.**

$x_{0,0} + w_{1,0} \cdot x_{1,0} + b$ gives $\dfrac{b}{w_{1,0}} = \dfrac{-x_{1,0}}{1 + x_{0,0} \cdot w_{0,0} / b}$. This modified algorithm can be generalized to be compatible with any $k \times k$ max pooling window. With a 2x2 average pooling layer, the same approach can be applied with the modified equation in Equation (11) and $b/w_{1,0}$ can be expressed with $\dfrac{-bx_{1,0}}{2b + x_{0,0} \cdot w_{0,0}}$.

$$\begin{cases} y_{0,0} = \frac{w_{1,0} \cdot x_{1,0} + b + w_{0,0} \cdot x_{0,0} + b}{4} \\ y_{1,0} = \frac{w_{0,0} \cdot x_{1,0} + b}{4} \end{cases} \tag{11}$$

The proposed attack on the dynamic zero pruning scheme allows each weight to be expressed as a function of the bias. This significantly reduces the entropy in weights as only the bias is left to be unknown. This unknown bias cannot be determined only from the information leak through the number of non-zero pixels. The changes in the number of non-zero pixels in OFM provides $n^2$ unique equalities, one for each pixel for a $n \times n$ convolutional filter. Yet, there exists $n^2 + 1$ variables for the filter ($n^2$ weights and one bias value). In order to determine the exact weights and the bias, an adversary needs to leverage additional information. For example, recent accelerator designs [1, 12] proposed to use a tunable threshold function in place of the ReLU function in order to prune more pixels with small values and thus improves efficiency. However, if this non-zero threshold value is known and can be adjusted, an adversary can set the input to be all zeros and vary the threshold to find the bias values. Since the ratio between each weight and bias is known, this optimization enables an adversary to fully recover the weight and bias values.

### 4.2 Case Study

We demonstrate the proposed attack on the first layer of a compressed AlexNet model [6], which contains zero-valued weights. The inferred $w/b$ of all 96 filters are shown in Figure 7. The zero-valued weights are detected and the maximum difference between the inferred and the original ratio is less than $2^{-10}$.

## 5 RELATED WORK

While this work represents the first concrete study on exploiting information leaks through memory access patterns in the context of reverse engineering CNNs, hiding information leaks through memory access patterns in general is a well-studied problem. In particular, oblivious RAM (ORAM) algorithms [4] provide a strong theoretical guarantee for obfuscating the memory accesses. ORAM can be used to prevent attacks proposed in this paper. However, even an efficient hardware implementation [3] of the state-of-the-art ORAM algorithm [15] significantly increases the number of memory accesses, and likely to result in significant overhead for the CNN inference, which is a memory-intensive task.

Membership inference attack [14] proposed constructing shadow models to identify whether a input belongs to the original training dataset or not. This attack relies on training the shadow models when the network structure is already known. The proposed structure reverse-engineering attack can be used to enable such attacks.

## 6 CONCLUSION

This paper studies potential vulnerabilities in CNN accelerators in the context of stealing a CNN model. The study shows that both the network structure and weights of a CNN model can be revealed through the memory access patterns and the input/output of the accelerator even when no internal access is allowed and all off-chip data are encrypted. Our findings highlight the need for hiding memory access patterns for CNN accelerators. The study also shows that performance optimization can lead to an unexpected security vulnerability and needs to be carefully reviewed.

## 7 ACKNOWLEDGMENTS

This work was partially sponsored by NSF award CNS-1618275, Semiconductor Research Corporation under Task 2686.001, and a GPU donation from NVIDIA Corporation.

## REFERENCES

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free Deep Neural Network Computing. In *ISCA*, 2016.
[2] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.
[3] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten van Dijk, Emil Stefanov, Dimitrios Serpanos, and Srinivas Devadas. A Low-Latency, Low-Area Hardware Oblivious RAM Controller. In *FCCM*, 2015.
[4] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 1996.
[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *NIPS*. 2014.
[6] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR*, 2015.
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, 2015.
[8] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, 2016.
[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*. 2012.
[10] Lichun Li and Anwitaman Datta. Write-only Oblivious RAM-based Privacy-preserved Access of Outsourced Data. *Int. J. Inf. Secur.*, 2017.
[11] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *ISCA*, 2017.
[12] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. HernÃąndez-Lobato, G. Y. Wei, and D. Brooks. Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators. In *ISCA*, 2016.
[13] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2015.
[14] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. Membership Inference Attacks against Machine Learning Models. *CoRR*, 2016.
[15] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In *CCS*, 2013.
[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *CoRR*, 2014.
[17] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *S&P*, 2015.