Guessing Outputs of Dynamically Pruned CNNs Using Memory Access Patterns

Benjamin Wu, *Member, IEEE,* Trishita Tiwari, *Member, IEEE* G. Edward Suh, *Fellow, IEEE,* Aaron B. Wagner, *Fellow, IEEE*,

Abstract—Dynamic activation pruning of convolutional neural networks (CNNs) is a class of techniques that reduce both runtime and memory usage in CNN implementations by skipping unnecessary or low-impact computations in convolutional layers. However, since dynamic pruning results in different sequences of memory accesses depending on the input to the CNN, they potentially open the door to inference-phase side-channel attacks that may leak private data with each input. We demonstrate a memory-based attack inferring a dynamically-pruned CNN's outputs for various victim CNN models and datasets. We find that an attacker can train their own machine learning model to learn to guess victim image classifications using the victim's memory access patterns with significantly better than random chance. Moreover, unlike previous related work, our attack: 1) continually leaks user data for each input and 2) does not require adversarial presence during the victim training.

Index Terms—Side-channel attacks, Machine learning, Artificial neural networks.

1 Introduction

Deep learning has gained popularity in many applications, including image/video recognition, recommender systems, self-driving vehicles, and more [1]. In particular, convolutional neural networks (CNNs) are widely used for many machine learning (ML) tasks on images and videos, and many custom hardware accelerators have been proposed to efficiently handle the large amount of computation needed in complex CNNs.

We show that dynamic pruning techniques ([2], [3], [4], [5]) introduce a side-channel vulnerability that allows an attacker to infer the outputs of each input. The presence of this vulnerability is potentially alarming in light of the demonstrated benefits of the dynamic pruning technique, which has been shown to achieve several times reduction in the inference-phase computations (e.g. upwards of 2x for ResNet and 5x for VGG [2]).

Previous work has demonstrated a number of privacy attacks on CNNs. We broadly distinguish two categories of CNN privacy attacks: API attacks (which require access to a victim CNN's API but are generally not used for pruning attacks) and side-channel attacks. This work belongs to the latter group, in which previous studies have shown that memory access patterns can leak the victim model's structure and parameters [6] and that power side channels can be used to reconstruct inputs [7], [8].

We investigate the side-channel attack through memory access patterns of dynamically-pruned CNNs. We show that, given only information on a CNN's pruning decisions and a set of labels, an adversary can train a supervised ML model to predict the victim CNN's output labels for individual inputs with much higher accuracy than a random guess. Unlike the model/structure-stealing attacks [6] which learn a single static secret by observing memory accesses of multiple inputs, our output-stealing attack learns one output per input. Moreover, our attack does not require an adversarial presence during the victim training as in the input-reconstruction attacks [7], [8], but instead only requires black-box access to a copy of the victim's model and trained weights. Our findings suggest that dynamic optimizations for CNNs need to carefully consider potential side-channel attacks.

2 Background

2.1 Convolutional Layers

Mathematically, a convolutional layer can be represented as a series of 3D matrix operations. Figure 1 shows how a single convolutional layer is computed for a single input. The input feature map (IFM) consists of a 3D matrix of size $H_i \times W_i \times C_i$. Similarly, the output feature map (OFM) of the layer is also a 3D matrix of size $H_o \times W_o \times C_o$. Each 2D slice of the IFM and of the OFM along their depth dimensions are commonly referred to as *input channels* and *output channels*, respectively. For convenience, we also distinguish the feature maps from *activations*, in that the former refers to the matrices as a whole while the latter refers the individual scalar values that compose them.

Since a CNN consists of multiple convolutional layers linked in sequence, the OFM of one convolutional layer will become the IFM to the next layer. Finally, the weights can be thought of as a 4D matrix of size $C_i \times H_k \times W_k \times C_o$. Each 3D slice of the weights along the fourth dimension is commonly referred to as a *kernel*.

Conceptually, the convolution itself consists of a systematic series of inner products between submatrices of the IFM and weight kernels, accumulated over the entire IFM and all kernels. Note that CNNs are not usually implemented directly this way, because doing so is inefficient in terms of memory access order [9]. For example, im2col, a standard vectorized implementation is commonly used instead in ML frameworks [10], [11]. The precise order of memory accesses within each layer can be further streamlined to reduce memory accesses (*dataflow implementations*) [9]. While dataflow implementations do affect the order of the victim's

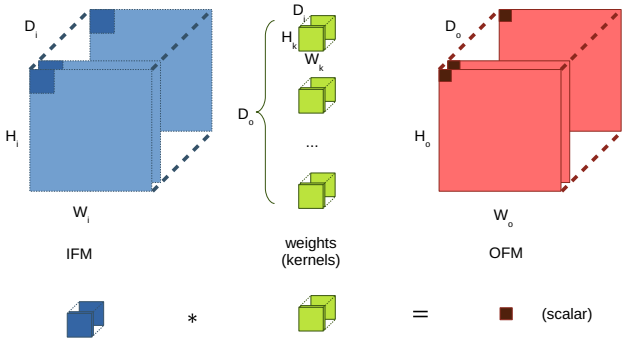

Fig. 1: Matrix representation of a convolutional layer (batch size=1) and example of one dot product producing a single output activation

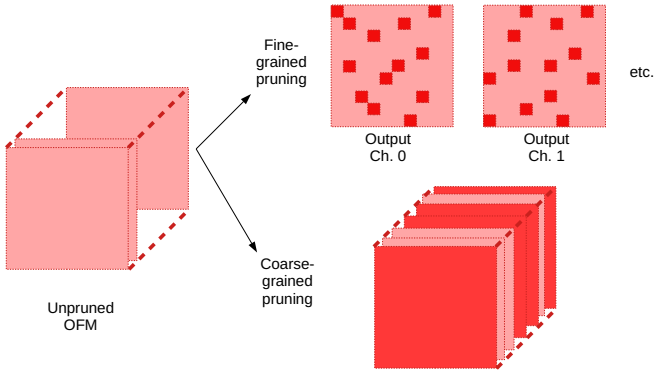

Fig. 2: Conceptual examples of fine-grained and coarse-grained (channel) pruning. Darkened activations represent pruned values.

memory accesses, they do not prevent our attack as we are concerned primarily with the addresses accessed during each layer. We omit further details of vectorized and dataflow implementations as these are considered standard techniques in the field of CNN acceleration.

2.2 Dynamic Pruning

Broadly speaking, dynamic activation pruning is any technique that reduces per-layer computations and memory accesses by zeroing out activations in either the IFM or OFM. Since zeros in the IFM of a layer do not contribute to the inner products that compose the OFM, multiplications with those zeros can be skipped. In particular, only input-dependent dynamic pruning techniques can leak information about the inputs.

While dynamic pruning can theoretically be performed on either the IFM or the OFM, most practical implementations operate on the OFM, since doing so allows for OFM compression in order to reduce the total volume of memory accesses. Thus, we will presume dynamic pruning of the OFM in the rest of this paper.

In general, dynamic pruning is performed by deciding whether individual activations (or groups of activations) should be pruned or not pruned. Each of these single-bit (prune/don't prune) decisions are called *pruning decisions*, and a CNN may make many pruning decisions for a single input. We call a collection of pruning decision results spawned from a single input, a *pruning vector*. We can separate dynamic pruning into two categories (illustrated in Figure 2), depending on how pruning decisions are performed: fine-grained and coarse-grained.

Fine-grained dynamic pruning makes individual (not necessarily independent) pruning decisions for each activation. One example of a simple fine-grained pruning method is to use an activation function with a built-in threshold, such as ReLU, wherein any output activation below the threshold (in the case of ReLU, this threshold is 0) is pruned [12]. Other simple fine-grained pruning techniques generally make pruning decisions on each activation based on the value of the activation itself, but more sophisticated techniques do exist [13], [14], [15].

In each layer, the pruned OFM are typically compressed to reduce the total number of memory accesses [16]. The simplest compression method is to compress each output channel individually, using a sparse matrix format such as *compressed sparse row* format (*CSR*).

Coarse-grained pruning is a special case of fine-grained pruning, in which pruning decisions are made on a per-channel basis. Since coarse-grained pruning takes many activations into account for each pruning decision, the decision logic is often more complex than fine-grained pruning, and one common approach is to train/operate a decision network in parallel with the CNN [2], [3], [5], [17]. Despite the more complex implementation, the advantage of coarse-grained over fine-grained pruning is that skipping memory accesses and computations is easier to implement, since channels are stored contiguously in memory.

3 Threat Model

In this study, we perform ML-driven attacks on dynamically-pruned CNNs used for image classification. We consider a victim CNN that classifies input images using a publically-known model. The victim's input images, the model weights, and all IFM/OFM intermediate values are stored in off-chip DRAM memory and loaded into on-chip memory as needed. We assume that the feature maps of at least one of the convolutional layers are too large to fit entirely in on-chip memory, and its accesses are exposed off-chip[1]. The off-chip memory is also assumed to be encrypted, or else the attack is trivial. Finally, the adversary has access to a copy of the trained victim CNN that is used to run offline experiments using its own input data, and they can monitor the addresses of all victim off-chip memory accesses (both reads and writes) via physical access to the memory bus between on-chip and off-chip memory or by other means (such as memory side channels). For instance, this threat model may be applicable in CNN accelerators or on cloud computing platforms shared by the adversary and victim.

An architecture for such a system can be seen in Figure 3, and the attack objective is to use the victim's memory access patterns to infer the victim's classifications.

1. While not discussed here, we also found that cache side channels can be used to obtain feature map access patterns even without off-chip accesses.

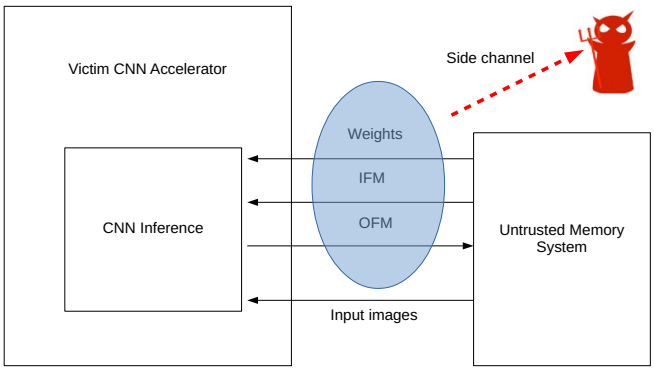

Fig. 3: Threat model

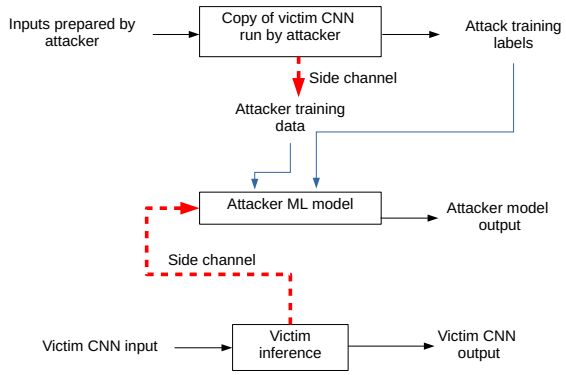

Fig. 4: Attacker workflow

4 Attack Methodology

To perform the attack (summarized in Figure 4), the attacker first uses the victim copy to classify their own input dataset and collects memory access patterns for each input. Using these memory access patterns, they extract the pruning decisions made in all convolutional layers (as a fixed-format *pruning vector*) in order to obtain a training set of pruning vectors and labels. The attacker then trains a supervised ML model to infer labels from pruning vectors, which they use on the victim's memory access patterns. The attack is considered successful if the attacker's ML model can predict victim classifications with accuracy significantly exceeding a random guess.

4.1 Attacker Models

Naturally, a key step in this attack is choosing an appropriate attacker ML model. From the attacker's point of view, they are dealing with a classification learning problem with the following properties.

First, the attacker is free to generate an arbitrarily large amount of training data. Second, the pruning vector size will typically be much smaller than the input size(s) used by the victim, so the attacker model does not need to be as complex as the victim model. Third, pruning vectors may or may not have strong internal spatial correlations but are generally sectioned-off by layer, so the best attacker model may change depending on the pruning technique used by the victim.

For this study, we use two types of attacker models: multilayer perceptrons (MLPs) and gradient boosting. Since CNNs are generally thought to be effective on applications where the input data has strong spatial relationships (they are used for image classification for this very reason), it is not clear that they would be effective in general for this attack where such relationships do not necessarily exist. Thus, MLPs may be used since they thrive on abundant training data and do not require spatial correlations within the pruning vector to work.

Alternatively, since the pruning vectors are fixed-format, specific locations within the vectors may hold individual significance. Gradient boosting and random decision forest models excel at identifying these decision points within the input vector. The ensemble decision tree model of XGBoost [18], for example, is a quick and easy general-purpose model for this attack.

5 Evaluation

To demonstrate our attack, we simulated attacks on ReLU-based (fine-grained) pruning using AlexNet, Resnet20, and Resnet18 as victims, and we simulated software-based attacks on coarse-grained pruning using the Feature-Boosting and Suppression (FBS) [2] technique. In order to generate the attacker's training dataset, we used mutually exclusive attack training/test sets (extracted using ImageNet and CIFAR10/100 test sets). Note that we chose not to use the ImageNet or CIFAR training sets to ensure that the attacker model does not gain any advantage from training with the same dataset as the victim itself. All victim models are quantized as 32-bit floating point numbers. For the attacker model, we used both a 3-layer MLP and XGBoost [18], either of which were sufficient to recover classifications with significant accuracy. In this section, we discuss our experimental procedures and results for fine-grained pruning and coarse-grained pruning separately.

5.1 Experimental Procedure

For our attacks on fine-grained pruning, we used the previously described ReLU-based pruning technique. It is performed by allowing ReLU's built-in threshold to determine which activations to skip. Then, for each output channel, we counted the number of positive activations (*number of nonzeros* or *NNZs*). The pruning vector is an array of NNZs where each element corresponds to a different output channel for a different layer. This pruning vector format represents the CSR-compressed size of each output channel, inferred from the number of victim memory accesses per channel.

For an attack on coarse-grained pruning, we used the FBS channel-pruning technique, which uses per-layer trained networks to infer which channels are most important. The pruning vector is a binary array, where each bit indicates whether FBS decided to prune a particular channel in a particular layer.

For the victim models, we experimented with AlexNet trained on ImageNet and ResNet trained on CIFAR10/100 for the fine-grained attacks, and CifarNet trained on CIFAR10 for the coarse-grained attacks.

We used two different attacker models for all of these experiments. First, we used a 3-layer MLP (i.e. input layer, hidden layer, and output layer), where the hidden layer has

an input size of 256 and output size of 512. The activation function used is negative log-likelihood, and the training algorithm used is gradient descent (50 epochs for all experiments). For XGBoost experiments, we simply used the toolbox's default gradient boost training algorithm, with a max tree depth of 6 with 10 boosting rounds.

Finally, we also studied the impact on the attack success rate by implementing a basic granularity reduction protection scheme in which the activations of each channel are grouped together in fixed-size groups and then the number of nonzero groups (NNGs) of each channel are shown to the attacker. This style of protection simulates natural cache behavior in which multiple activations fit on a single cacheline and the attacker is only able to count the number of cachelines accessed per channel. Our experimental results can be seen in Tables 1 and 2.

5.2 Experiment Observations

We observe that, if left unmitigated, the attacker is able to predict the victim's classifications from these pruning vectors, both in the fine-grained and coarse-grained cases, with significantly better-than-random accuracy. In addition, simple grouping-based protection based on cache behavior does not sufficiently mitigate our attack. From these experiments we conclude that the pruning vectors are information-rich and that simple protections are not enough to prevent such attacks to any reasonable degree. Moreover, since information embedded in the pruning vectors is not restricted to the victim's classifications only, the potential for further privacy-violating attacks is high.

Victim/Data/Attack	top1 %	top3 %	top5 %
AlexNet/ImageNet/MLP	24.36%	39.38%	46.67%
AlexNet/ImageNet/MLP/RG-8	19.71%	33.38%	40.16%
ResNet20/CIFAR10/MLP	77.37%	93.33%	97.57%
ResNet20/CIFAR10/MLP/RG-8	51.46%	81.17%	92.17%
ResNet20/CIFAR100/MLP	64.10%	89.47%	95.8%
ResNet20/CIFAR100/MLP/RG-8	40.63%	72.87%	87.47%
ResNet18/CIFAR100/XGB	40.03%	63.13%	74.00%
ResNet18/CIFARF100/XGB/RG-16	31.50%	54.30%	65.53%

TABLE 1: Test accuracy of attacks on fine-grained pruning. Used AlexNet and ResNet as experimental victim models trained on ImageNet or CIFAR datasets. Attacker model is either a 3-layer MLP with hidden layer size 256 x 512 or XGBoost (XGB) with a max depth of 6 and 10 boosting rounds. Where noted, RG-N is a granularity reduction scheme that groups activations into groups of N

Model/Dataset	top1 %	top3 %	top5 %
FBS-CifarNet/CIFAR10/MLP	61.63%	86.8%	94.47%
FBS-CifarNet/CIFAR10/XGB	61.63%	85.57%	94.77%

TABLE 2: Test accuracy results of attacks on coarse-grained pruning. Used CifarNet with Feature Boosting and Suppression (FBS) pruning as victim model, trained on CIFAR10 (CF10).

6 Conclusion and Future Work

In this study, we showed that dynamically-pruned CNNs give rise to a concerning side-channel attack in which an adversary is able to violate privacy guarantees of a victim implementation a publically-known CNN model by using their memory accesses. In particular, our results open up a number of critical questions that must be answered with future work, such as whether these attacks are indeed possible on real systems, whether any principled analysis of a given victim CNN can be performed to quantify the threat posed by these attacks, and which method(s) of protection are best suited to mitigate these attacks.

References

[1] S. Dargan, M. Kumar, M. R. Ayyagari, and G. Kumar, "A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning," *Archives of Computational Methods in Engineering*, vol. 27, no. 4, pp. 1071–1092, Sep. 2020. [Online]. Available: https://doi.org/10.1007/s11831-019-09344-w

[2] X. Gao, Y. Zhao, Dudziak, R. Mullins, and C.-z. Xu, "Dynamic Channel Pruning: Feature Boosting and Suppression," *arXiv:1810.05331 [cs]*, Jan. 2019, arXiv: 1810.05331. [Online]. Available: http://arxiv.org/abs/1810.05331

[3] Y. Wang, X. Zhang, X. Hu, B. Zhang, and H. Su, "Dynamic Network Pruning with Interpretable Layerwise Channel Selection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 6299–6306, Apr. 2020. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/6098

[4] Y. Guo, A. Yao, and Y. Chen, "Dynamic Network Surgery for Efficient DNNs," *arXiv:1608.04493 [cs]*, Nov. 2016, arXiv: 1608.04493. [Online]. Available: http://arxiv.org/abs/1608.04493

[5] Z. Chen, T. Xu, C. Du, C. Liu, and H. He, "Dynamical Channel Pruning by Conditional Accuracy Change for Deep Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2020, conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[6] W. Hua, Z. Zhang, and G. E. Suh, "Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. San Francisco, CA: IEEE, Jun. 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8465773/

[7] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference on - ACSAC '18*. San Juan, PR, USA: ACM Press, 2018, pp. 393–406. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3274694.3274696

[8] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier, "Remote Power Side-Channel Attacks on CNN Accelerators in FPGAs," *arXiv:2011.07603 [cs]*, Nov. 2020, arXiv: 2011.07603. [Online]. Available: http://arxiv.org/abs/2011.07603

[9] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," *arXiv:1708.04485 [cs]*, May 2017, arXiv: 1708.04485. [Online]. Available: http://arxiv.org/abs/1708.04485

[10] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *arXiv:1912.01703 [cs, stat]*, Dec. 2019, arXiv: 1912.01703. [Online]. Available: http://arxiv.org/abs/1912.01703

[11] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[12] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is Less: A More Complicated Network with Less Inference Complexity," *arXiv:1703.08651 [cs]*, May 2017, arXiv: 1703.08651. [Online]. Available: http://arxiv.org/abs/1703.08651

[13] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, vol. 11211, pp. 815–832, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-030-01234-2-48

[14] H. Zhou, J. M. Alvarez, and F. Porikli, "Less Is More: Towards Compact CNNs," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, vol. 9908, pp. 662–677, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-46493-0-40

[15] C. Louizos, K. Ullrich, and M. Welling, "Bayesian Compression for Deep Learning," *arXiv:1705.08665 [cs, stat]*, Nov. 2017, arXiv: 1705.08665. [Online]. Available: http://arxiv.org/abs/1705.08665

[16] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8114708/

[17] J.-H. Luo and J. Wu, "AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, p. 107461, Nov. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320320302648

[18] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, Aug. 2016, arXiv: 1603.02754. [Online]. Available: http://arxiv.org/abs/1603.02754