Intrinsic Examples: Robust Fingerprinting of Deep Neural Networks

Siyue Wang[1]
wang.siy@northeastern.edu
Pu Zhao[1]
zhao.pu@northeastern.edu
Xiao Wang[2]
kxw@bu.edu
Sang Chin[2]
spchin@bu.edu
Thomas Wahl[1]
wahl@ccs.neu.edu
Yunsi Fei[1]
yfei@ece.neu.edu
Qi Alfred Chen[3]
alfchen@uci.edu
Xue Lin[1]
xue.lin@northeastern.edu

[1] Northeastern University
Boston
MA, USA

[2] Boston University
Boston
MA, USA

[3] University of California, Irvine
Irvine
CA, USA

Abstract

This paper proposes to use intrinsic examples as a DNN fingerprinting technique for the functionality verification of DNN models implemented on edge devices. The proposed intrinsic examples do not affect the normal DNN training and can enable the black-box testing capability for DNN models packaged into edge device applications. We provide three algorithms for deriving intrinsic examples of the pre-trained model (the model before the DNN system design and implementation procedure) to retrieve the knowledge learnt from the training dataset for the detection of adversarial third-party attacks such as transfer learning and fault injection attack that may happen during the system implementation procedure. Besides, they can accommodate the model transformations due to various DNN model compression methods used by the system designer.

1 Introduction

With the rapid development of deep learning (DL) and artificial intelligence [11, 13, 25, 35, 36, 56, 61, 62, 66, 75, 76], discovering or searching for more efficient DNN model structures has become one of the essential research objectives of DL community. Due to high efficiency and reliability, low cost, small footprint, and reprogrammability, embedded system-based edge devices have become important carriers for DL tasks [23, 31, 50, 70]. However, significant challenges exist when executing DNN models on edge devices, and extensive research efforts have been devoted to address the challenges [8, 9, 24]. The problem

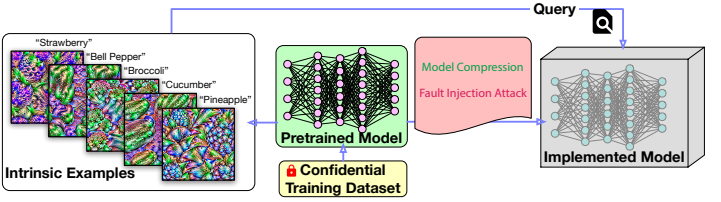


Figure 1: Verify the functionality with the proposed intrinsic examples.

is how to achieve efficient inference given the limited computation and storage resources on edge devices. To accelerate DNN execution on edge devices, various model compression techniques have been proposed, such as *weight pruning* [19, 42, 72] and *weight quantization* [22, 26, 33, 52], which can be considered as a re-training to derive a sparse model from the pretrained model.

This paper investigates functionality verification of DNNs implemented on edge devices for on-device inference applications. To satisfy design requirements on inference latency and memory footprint, even a trustworthy system designer may perform model compression too aggressively with severe accuracy loss. Furthermore, if the DNN implementation procedure is compromised by some adversarial third-party entity, the implemented models may be embedded with delicately crafted mis-behaviors through transfer learning [51, 63, 68] and fault injection attack [5], which are violations of functionality by integrity breach.

To address these problems, we propose to extract *Intrinsic Examples* as a novel DNN fingerprinting technique, which effectively addresses the limitations of previous watermarking and fingerprinting methods, with the following advantages: (i) its generation process does not interfere with the training phase; (ii) it does not require any realistic data from the training/testing set; (iii) it can detect adversarial third-party attacks that embed misbehaviors through re-training; and (iv) it is robust to potential model transformations of normal system design and implementation procedure (e.g., model compression), as long as the implemented model preserves original functionality. We summarize our contributions below.

- We are the first to propose a novel and practical fingerprinting method aiming for functionality verification of DNN models implemented on edge devices for on-device inference applications. In particular, we develop three algorithms to derive intrinsic examples based on the pre-trained model with various applicability and computation cost.
- Compared with the existing works of DNN watermarking [1, 7, 12, 15, 18, 46, 57, 71] or fingerprinting [21, 32], using intrinsic examples does not interfere with the training phase nor the training data. Meanwhile, our mechanisms feature high-robustness to the compressed models while keeping high-sensitivity to third-party adversaries.
- To better evaluate the performance of the fingerprints, we propose to use the accuracy of the intrinsic examples as the *intrinsic score* to evaluate the functionality of implemented model. Intrinsic scores of the proposed methods outperform other fingerprinting methods by a large margin. Specifically, the intrinsic scores of Algorithm 2 and Algorithm 3 outperform the baseline by over 80% on functionality remained CNN models on CIFAR-10.

2 Background and Related Work

2.1 IP Protection of Deep Neural Networks

There are extensive research efforts on DNN watermarking or fingerprinting for DNN intellectual property (IP) protection [1, 7, 12, 15, 18, 32, 46, 57, 71] and model integrity verification [21]. They can be classified into three categories: (1) DNN watermarking [7, 12, 15, 57]; (2) Watermarking by backdooring [1, 18, 46, 71]; and (3) DNN fingerprinting [21, 32].

DNN watermarking embeds watermarks into the model weight parameters through training from scratch, retraining, or distillation, and requires white-box access to the model to be tested. Uchida et al. first investigate DNN watermarking by embedding a watermark in model weight parameters, using a parameter regularizer [57]. Other works proposed by Rouhani et al. [12], Chen et al. [7] and Fan et al. [15] also contribute towards this approach.

Watermarking by backdooring leverages the DNN backdoor attacks [17] to embed watermarks while using trigger images to test IP infringement [1, 18, 46, 71]. Comparing with DNN watermarking, it enables the black-box testing capability but still involves re-training of the DNN model to embed watermarks. DNN fingerprinting instead extracts adversarial examples [32] or sensitive examples [21] from a DNN as its fingerprints, which eliminates the need of training or re-training and enables the black-box testing capability.

Besides the above mentioned works, Mahendran et al. propose a general framework to invert representations [45] which also trying to understand the relationship between reconstructed images and neural network models.

2.2 DNN Model Compression

DNN model compression techniques have been proposed for simultaneously reducing storage/computation and accelerating on-device inference, with minor accuracy loss. Two important DNN compression techniques are: i) weight pruning and ii) weight quantization.

DNN weight pruning can be categorized into the unstructured sparsity scheme by irregular pruning methods [42, 72], the structured sparsity scheme by filter pruning methods [41, 64] and by column pruning methods [37, 73], and the most recent fine-grained structured sparsity scheme by pattern-based pruning methods [43, 49, 67]. Detailed discussion about different sparsity schemes and pruning methods are introduced in Appendix A. Weight quantization reduces redundancy in bit representation of weights [22, 26, 33]. With a k-bit weight representation, quantization maps weights into a total of 2^k quantized levels.

2.3 Fault Injection Attack

The fault injection techniques including laser beam [4, 53] and row hammer [27, 58, 65] attempt to flip logic values in the memory. Motivated by these hardware fault injection techniques, DNN fault injection attacks [40, 74] are proposed to inject faults for a given model such that the model makes specific mis-classifications on a particular set of inputs, while keeping normal predictions on other inputs, by finding the minimum set of weight parameters. Recently, [5] implements the DNN fault injection attack [40] physically on embedded systems using laser beams, demonstrating the possibility of practical fault injection.

3 Overview

It is well accepted that the trained DNN models should be protected as IP. Towards this end, extensive research efforts have been devoted to DNN watermarking or fingerprinting [1, 7, 12, 15, 18, 21, 32, 46, 57, 71]. The DNN training process learns from the training data of practical applications and extracts the information into the trained model. It is essential to protect the model functionality. We proposes to use fingerprinting for verifying functionality (and integrity) of DNN models implemented on edge devices. The proposed methods focus on the model functionality instead of a particular model representation, and therefore can effectively detect adversarial third-party attacks that violate model integrity (i.e., functionality). Besides, it is robust to potential model transformations due to normal system design and implementation procedure as long as it preserves the original functionality.

3.1 Threat Model

This paper addresses the problem of verifying the functionality of DNN models during DL system design and implementation procedure, where the client provides a *pretrained model* to the system designer, and the system designer mainly performs model compression to implement a model on edge devices for supporting on-device inference. We consider the following scenarios that may violate the model functionality (and integrity): (i) To meet the design goals on inference latency and memory footprint, the system designer pursues the extreme model sparsity, such that the model accuracy loss may be higher than an acceptable threshold. (ii) A third-party entity performs a fault injection attack which modifies a minimum set of DNN weight parameters with a small set of data, such that the implemented model mis-classifies certain specific inputs while keeping normal predictions on other inputs.

In this paper, we assume the client does not release the training dataset to the system designer or anyone else, due to IP considerations. For model compression, there are recent research efforts to preserve data privacy [6, 47, 60, 69], where the original training dataset can be kept confidential with the model owner (client) during model compression.

3.2 Intrinsic Examples for DNN Fingerprinting

Suppose the client has trained a model F_θ with training data $\{(\mathbf{x}, l)\}$, where $(\mathbf{x}, l)$ represents the data and label pair, and θ denotes model weight parameters. In addition, the client generates a group of intrinsic examples $\{(\mathbf{x}_{IE}, l_{IE})\}$ based on the pretrained model F_θ. The client gives the pretrained model F_θ to the system designer for implementations of on-device inference. The implemented model $F_{\theta'}$ may be a transformation of the pretrained model with a different set of weight parameters represented by θ'. The model transformation could be model compression by the system designer or attacks (e.g., fault injection attack) by certain adversarial third-party entity. The system designer is not allowed to modify the model structure. Likewise, the adversarial third-party entity does not modify the model structure.

Figure 1 provides an overview of the DL system design and implementation procedure using the proposed intrinsic examples for verifying the functionality. To verify the implemented model, the client can analyze the accuracy of intrinsic examples by querying the implemented model $\mathbf{y} = F_{\theta'}(\mathbf{x}_{IE})$ with intrinsic examples. Note that the client does not release the intrinsic examples $\{(\mathbf{x}_{IE}, l_{IE})\}$ to the system designer, out of privacy concerns. Also, when testing functionality, the client only has black-box access to $F_{\theta'}$, because the model is already packaged into the edge devices for on-device inference.

There are three reasons why we prefer using intrinsic examples rather than the testing dataset for verifying model functionality and integrity. (i) Intrinsic examples can significantly accelerate the verification procedure, since it is time-consuming to run the whole testing dataset with the limited resources on edge devices. (ii) With intrinsic examples, we can isolate the verification procedure from the whole or a subset of testing dataset, which provides an additional layer of protection on the data privacy and therefore IP. (iii) The intrinsic examples are more sensitive than the whole or a subset of testing data for the functional verification, for example, at a high pruning ratio when the testing accuracy loss is around 2%, the accuracy on the intrinsic examples (named *intrinsic score*) drops significantly, indicating a potential functionality violation.

4 Methodology

In this section, our main methods are introduced in the image classification task by DNNs. We stress, however, that the proposed approach can be generalized to other types of tasks,

data, and classification models. Let $\mathbf{x} \in \mathbb{R}^{3\times H\times W}$ denote a colored RGB image, where H and W are the image height and width, respectively. The pixel values of $\mathbf{x}$ are scaled to $[0,1]$ for mathematical simplicity. F_θ denotes the pre-trained Base Model, which outputs $\boldsymbol{y} = F_\theta(\mathbf{x})$ as a probability distribution for a total of M classes. The element y_i represents the probability that an input $\mathbf{x}$ belongs to the i-th class. Generating intrinsic examples can be formulated as:

$$\min_{\mathbf{x}} \ \text{Loss}(\theta, \mathbf{x}, l), \ \ \text{s.t.} \ \ \mathbf{x} \in [0,1]^n. \tag{1}$$

where $\text{Loss}(\theta, \mathbf{x}, l)$ denotes the loss function, for instance, the cross-entropy loss for the DNN model F_θ. The above formulation (given the model F_θ) finds the optimal $\mathbf{x}$ (i.e., the intrinsic example) to minimize the loss function with respect to a class label l.

The process of generating a small group of intrinsic examples $\{(\mathbf{x}_j, l_j)\}$ is as follows:

1. Select a subset $\{l_1, l_2, \ldots, l_j, \ldots, l_K\}$ of K labels randomly from the available labels.
2. Solve the problem (1) K times, each with a label l_j.

The generated intrinsic examples are used for testing the implemented model $F_{\theta'}$ on the edge device. We provide three algorithms for DNN fingerprinting, each with unique characteristics for dealing with different threat scenarios in the DNN system design. The algorithms are general for various kinds of DNN architectures since they do not depend on a specific model architecture during the intrinsic examples generation. We use the gradients with reference to the inputs to generate intrinsic examples, without any requirement on the model architecture.

4.1 Algorithm 1: Proposed Intrinsic Examples

The intrinsic examples generation can be considered as retrieving knowledge learnt by the pretrained model from training dataset. Therefore, intrinsic examples can keep high robustness from the pretrained model F_θ to the implemented model $F_{\theta'}$ on edge devices as long as $F_{\theta'}$ has the same functionality. We compare the robustness of intrinsic examples with adversarial examples in Appendix B. To be independent of training/test data, we use random seeds to generate intrinsic examples, and thus intrinsic examples are distinct from natural images. We use projected gradient decent (PGD) [29, 30, 34, 44] to generate intrinsic examples:

$$\mathbf{x}^{t+1} = \prod_{\mathbf{x}^0+\mathcal{E}} \left(\mathbf{x}^t - \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \text{Loss}(\theta, \mathbf{x}^t, l))\right), \tag{2}$$

where t is the iteration index; $\mathbf{x}^0$ is the random starting point; $\mathcal{E}$ is the ℓ_∞-ball around $\mathbf{x}_0$ bounded by ε, $\prod_{\mathbf{x}^0+\mathcal{E}}$ means the projection onto the set $\mathbf{x}^0+\mathcal{E}$; α is the step size; $\text{sign}(\cdot)$ returns the element-wise sign of a vector; and $\nabla_{\mathbf{x}}$ calculates gradients. In summary, PGD iteratively makes updates based on gradients and then clipping into the ℓ_∞-ball around $\mathbf{x}_0$.

4.2 Algorithm 2: Enhanced Robustness

All the existing works [21, 32] perform fingerprinting or watermarking for a DNN as it is, which can not differentiate the (benign) model compression. Here, we tackle this challenge by improving the robustness of intrinsic examples on compressed models. We propose an enhancement over Algorithm 1 by adding random perturbations bounded by δ onto the weights of one or multiple layers in the pretrained model F_θ to mimic the model perturbation due to the model compression procedure for its implementation on edge devices, where similar idea is also applied in defending adversarial attacks [39]. Here, the problem can be formulated as

$$\min_{\mathbf{x}} \ \mathbb{E}_{\Delta\sim\mathbf{U}(-\delta,\delta)}\Big[\text{Loss}(\theta+\Delta, \mathbf{x}, l)\Big], \ \ \text{s.t.} \ \ \mathbf{x} \in [0,1]^n. \tag{3}$$

Δ presents uniformly distributed perturbations within $[-\delta, \delta]$. Algorithm 2 is as follows:

$$\mathbf{x}^{t+1} = \prod_{\mathbf{x}^0+\mathcal{E}} \left(\mathbf{x}^t - \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \text{Loss}(\theta + \Delta, \mathbf{x}^t, l))\right). \tag{4}$$

Furthermore, motivated by Expectation Over Transformation (EOT) which models possible transformations of adversarial examples into attack generation process [2, 3], we further enhance Algorithm 2 by calculating the mean of the input gradients in each iteration step. When computing input gradient, we sample input gradients for $q = 10$ times and use the gradients mean in each iteration of generating RC-examples.

4.3 Algorithm 3: Min-Max Robust Optimization

We further provide an alternative algorithm leveraging the min-max robust optimization, which systematically incorporates weight perturbations from an optimization aspect. Specifically, we investigate the optimization problem as shown below:

$$\min_{\mathbf{x}} \max_{\Delta} \text{Loss}(\theta + \Delta, \mathbf{x}, l), \quad \text{s.t. } \mathbf{x} \in [0,1]^n. \tag{5}$$

We solve problem (5) by alternatively solving the inner maximization and outer minimization problems with PGD method. The inner maximization problem can be solved as:

$$\Delta^{q+1} = \prod_{[-\delta,\delta]} \left(\Delta^q + \beta \cdot \text{sign}(\nabla_\theta \text{Loss}(\theta + \Delta^q, \mathbf{x}^T, l))\right). \tag{6}$$

We use Q steps to obtain Δ^Q. Then for the outer minimization, PGD is applied below,

$$\mathbf{x}^{t+1} = \prod_{\mathbf{x}^0+\mathcal{E}} \left(\mathbf{x}^t - \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \text{Loss}(\theta + \Delta^Q, \mathbf{x}^t, l))\right). \tag{7}$$

We use T steps to obtain $\mathbf{x}^T$. These two processes are implemented alternatively. The training loss of Algorithm 3 is shown in Figure A2 of Appendix C, where fluctuations due to the inner maximization are observed while following an overall decreasing trend to converge.

4.4 Discussions

Algorithm 1 is derived for a straight forward objective function – the loss function, targeting for retrieving the knowledge learnt from the training dataset. And Algorithm 1 is therefore more sensitive to the subtle fault injection attacks. Algorithms 2 and 3 incorporate the potential DNN model compression methods by adding stochastic weight perturbations and from an optimization aspect, respectively, towards increasing the robustness of intrinsic examples through normal DNN system implementation procedure. Although Algorithm 3 provides a more optimized solution than Algorithm 2, it is computationally expensive, especially on DNN models trained with large-scale datasets.

The proposed Algorithms 1, 2 and 3 have increasing computation complexity, but various applicability when dealing with different threat scenarios. For the real application of our proposed intrinsic examples, we will generate three groups of intrinsic examples, each by one algorithm, and the intrinsic scores (the accuracy of the intrinsic examples) will be evaluated on the implemented model, respectively, with three groups of intrinsic examples. If all the intrinsic scores are high, the implemented model passes the verification. Otherwise, more investigations are needed to resolve the ambiguity. For example, we could consult with the system designer which type of model compression methods were used. And the low intrinsic score by Algorithm 1 and high intrinsic score Algorithm 2 and/or 3 denote potential fault injection attack as shown in section 7.

Algorithm	Dense Model	Unstructured Pruning [19]				Irregular Pruning [72]				Column Pruning[20]				Pattern Pruning[49]			
Pruning Ratio	/	80%	90%	95%	97%	70%	80%	90%	95%	50%	60%	70%	80%	90%	95%	97%	99%
Model Acc.	80.5%	80.3% (-0.2)	80.3% (-0.2)	79.7% (-0.8)	78.5% (-2.0)	84.7% (+4.2)	83.1% (+2.6)	81.6% (+1.1)	78.2% (-2.3)	83.5% (+3.0)	82.7% (+2.2)	80.5% (0)	76.2% (-4.3)	83.5% (+3.0)	83.3% (+2.8)	83.3% (+2.8)	83.4% (+2.9)
Baseline (0.025)	100%	18%	18%	16%	14.5%	9%	10.5%	10%	11.5%	5.5%	4%	4.5%	4%	11.5%	9%	11.5%	10%
Baseline (0.05)	100%	42.5%	28%	21.5%	17%	22%	21.5%	17.5%	16%	15.5%	10%	7%	6.5%	24.5%	20%	18.5%	19%
Baseline (0.1)	100%	40.5%	33%	27.5%	26%	30%	30.5%	28%	20%	32%	23%	13%	10%	27.5%	26%	26%	25.5%
Alg.1	100%	98%	88%	64%	36%	85%	84%	65.5%	48%	67%	51.5%	38%	19%	83%	82%	80%	80%
Alg.2	100%	**100%**	**100%**	**100%**	**73%**	**100%**	**100%**	**97.5%**	**80%**	**90%**	**78.5%**	**50.5%**	**24.5%**	**100%**	**99%**	**99%**	**99%**
Alg.3	100%	**100%**	**100%**	83%	54%	**100%**	96%	81.5%	73%	77%	69%	41.2%	23%	78.3%	66.7%	66.7%	60%

Table 1: Intrinsic Score of Implemented Models by Different Weight Pruning Methods on the CNN with CIFAR-10 Dataset (whole model pruning). We use dense CNN model as the pretrained model.

Algorithm	Dense Model	Unstructured Pruning [19]				Irregular Pruning [72]				Column Pruning [20]			
Pruning Ratio	/	80%	85%	90%	95%	90%	95%	97%	99%	90%	95%	97%	99%
Model Acc.	80.5%	78.7% (-1.8)	79.0% (-1.5)	78.4% (-2.1)	77.8% (-2.7)	82.8% (+2.3)	83.1% (+2.6)	83.3% (+2.8)	81.7% (+1.2)	81.9% (+1.4)	81.5% (+1.0)	81.4% (+0.9)	79.1% (-1.4)
Baseline (0.025)	100%	22.5%	22%	26%	24%	18.5%	10%	11.5%	11%	9%	11.5%	13%	11%
Baseline (0.05)	100%	44%	47%	47%	36%	26.5%	27%	24.5%	25%	29%	28%	31%	31%
Baseline (0.1)	100%	50%	52%	50.5%	39.5%	29%	30%	32%	27.5%	42.5%	42%	36%	35.5%
Alg.1	100%	97.5%	97%	97.5%	96%	88.5%	87%	85%	83.5%	90%	88%	83%	76%
Alg.2	100%	95%	94.5%	91.5%	87.5%	85%	86%	84.5%	85.5%	91%	88.5%	80%	71%
Alg.3	**100%**	**100%**	**100%**	**100%**	**100%**	**98%**	**97.5%**	**96.5%**	**97%**	**99%**	**100%**	**97.5%**	**88%**

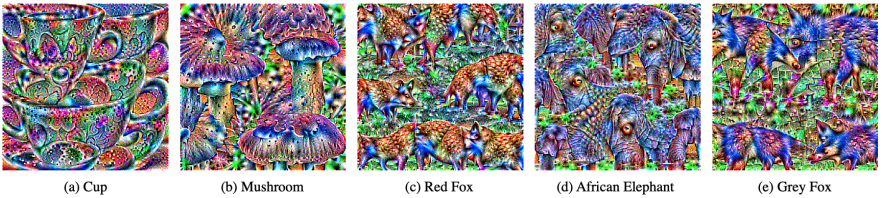
Table 2: Intrinsic Score of Implemented Models by Different Weight Pruning Methods on CNN with CIFAR-10 Dataset (single layer weight pruning). We use dense CNN model as the pretrained model.

5 Implementation Details

The experiments are conducted on machines with NVIDIA GTX 1080 TI GPUs. We adopt widely used datasets and models in the literature, including CNN models for CIFAR-10 [28] and SVHN [48] datasets, and VGG-16 [55] models for ImageNet datasets. The model details are summarized in Table A2 in Appendix D. We demonstrate all the experiment settings for model compression with various datasets and models in Table A3 of Appendix D.

5.1 Intrinsic Example Generation

The same hyper-parameter setting is utilized for three intrinsic example generation algorithms, among all datasets and DNN models in our experiments. We set the $\varepsilon = 128/255$ which bounds the ℓ_∞-ball around $\mathbf{x}_0$ since larger ε will not further increase the functionality verification (or integrity violation detection) performance. The weight perturbation bound δ is set to 0.05 as larger δ leads to difficulties to convergence. For each pretrained model, 200 intrinsic examples are generated with 200 iteration steps for each intrinsic example. We visualize the generated intrinsic examples on ImageNet in Figure 2.



(a) Cup (b) Mushroom (c) Red Fox (d) African Elephant (e) Grey Fox

Figure 2: Intrinsic examples generated by Algorithm 2 on the ImageNet dataset. Labels used for generating those intrinsic examples are shown below the images.

5.2 Comparative Methods

Among existing work on the DNN watermarking or fingerprinting, the methods by [21] and [32] are the most relevant to our work in terms of methodology, while all the other work [1, 7, 12, 15, 18, 46, 57, 71] have to interfere with the DNN training process. Comparing with [21] and [32], our work is the first one targeting for DNN functionality verification, while [21] is for detecting integrity breach and [32] is for protecting IP.

Algorithm	Pretrain Model	Decimal			Float16	Full Integer
		3 places	2 places	1 place		
Model Acc.	80.5%	80.5% (0)	80.4% (-0.1)	77.1% (-3.4)	80.4% (-0.1)	80.4% (-0.1)
Alg.1	100%	100%	100%	92%	100%	100%
Alg.2	100%	100%	100%	100%	100%	100%
Alg.3	100%	100%	100%	98%	100%	100%

Table 3: Intrinsic Score of Implemented Models by different quantization methods on CIFAR-10.

[32] extracts adversarial examples [16] to watermark DNNs on MNIST. Although the method in [32] is similar to our Algorithm 1, we highlight that we use random initialization instead of true data and therefore our method is data-free. In experiments, we report their performance as a baseline on CIFAR-10 with different settings of step size $(0.025, 0.05, 0.1)$ to control the intensity of the adversarial perturbations, detailed in Table 1 and 2.

[21] proposes sensitive examples (based on adversarial examples) from a DNN as its fingerprints. It regards all pruned models as compression attack and reject pruning even with minor testing accuracy degradation (e.g., 0.65%). Different from [21], we believe that an effective fingerprinting method should be robust to pruned models and recognize pruned models as non-attack. To demonstrate the robustness problem of [21], we use pruned models to evaluate the robustness of sensitive examples. With 8 sensitive samples, the intrinsic score on pruned models is only 0.04%, demonstrating that pruning is treated as illegitimate by sensitive samples, which is unreasonable due to the wide application of DNN pruning for size reduction and inference acceleration especially on edge devices with limited resources. Besides, comparisons with adversarial example-based approach is provided in Appendix B.

6 Experiments for Model Compression

In this section, we demonstrate the effectiveness of intrinsic examples on implemented models obtained with various weight pruning and quantization methods compared with [32]. We use the three proposed algorithms to generate intrinsic examples with the pretrained model and calculate the accuracy of intrinsic examples on the implemented models after model compression. We define the accuracy of intrinsic examples on the model as the *intrinsic score*. The intrinsic score of the pretrained model is 100%.

6.1 Weight Pruning

6.1.1 Functionality Verification

Intrinsic examples can verify the functionality of the implemented model with intrinsic scores from the three algorithms. Table 1 and 2 demonstrate our results using three intrinsic example generation algorithms to verify the functionality of two pruning modes: pruning the whole model or pruning only one layer. Note that the pattern pruning method can only be applied to the convolutional layers. Same experiments have been also performed using SVHN dataset which are summarized in Appendix E. We observe that the accuracy after pruning increases, which is also observed in other pruning works [19, 20, 48, 69]. The reason may be that pruning can mitigate the overfitting problem in overparameterized DNNs. We summarize our findings as follows:

1 Intrinsic scores of all algorithms outperform the baseline on all kinds of pruned models.
2 For whole model pruning, intrinsic scores of Algorithm 2 outperform both Algorithms 1 and 3 for all pruning methods in Table 1, due to random perturbations during generation.
3 For single layer weight pruning, as shown in Table 2, intrinsic scores of all three algorithms are generally higher than that of whole model pruning, since whole model pruning

prunes more weights for all layers. Algorithm 3 outperforms both Algorithms 1 and 2, because of the application of optimized weight perturbations.

4 Generally, intrinsic score is highly correlated with testing accuracy, and is robust to model transformations by pruning with moderate pruning ratios (i.e., testing accuracy degradation is small). It is more sensitive to large pruning ratios where the functionality is harmed by over-aggressive pruning (see column pruning with 80% pruning ratio in Table A4).

6.1.2 Functionality Indication

Intrinsic examples have high fingerprinting capacity. With only limited number of intrinsic examples, the intrinsic score can accurately indicate the functionality of implemented models. We test the intrinsic scores using different numbers of intrinsic examples with Algorithm 1 for unstructured pruned models with various pruning ratios in Figure 3 (Left), and complete results for all the algorithms are summarized in Figure A3 in Appendix F. For each data point in Figure 3 (Left) we generate N intrinsic examples and test the intrinsic score. This process is conducted 10 times to obtain the mean and variance of intrinsic score for each number N, as denoted by the solid line and shadow area in Figure 3 (Left). For each pruning ratio, the average intrinsic score is usually in the same level for different number of intrinsic examples used, demonstrating its steady performance. We note that when the number of intrinsic examples is smaller than 10, the variance of intrinsic score is relatively large. But as the example number increases, the variance reduces rapidly. As observed from Figure 3 (Left), usually 10 intrinsic examples are enough to provide a reliable intrinsic score, which correspond to only 0.1% of the total size of the testing dataset.

6.2 Weight Quantization

We adopt three widely used post-training weight quantization methods: decimal quantization, float16 quantization, and full integer quantization. Table 3 presents results on CIFAR-10. Intrinsic score is lower than 100% only in the case of decimal quantization with 1 place, where a relatively large accuracy degradation occurs, demonstrating our effectiveness on functionality verification. More results on SVHN are shown in Table A6 of Appendix.

7 Integrity Verification

In this section, we demonstrate the performance of intrinsic examples on integrity breach detection.We consider model modifications that aim to change the original classification task, i.e., fault injection attack. Fault injection attacks [40, 74] modify the model weights to change the classification results of the first $|S|$ images in the set R to the incorrect target labels, while the remaining images in the set R can still be classified correctly. Following the setting in [74], we modify the weights of the last fully-connected layer and generate 10 fault

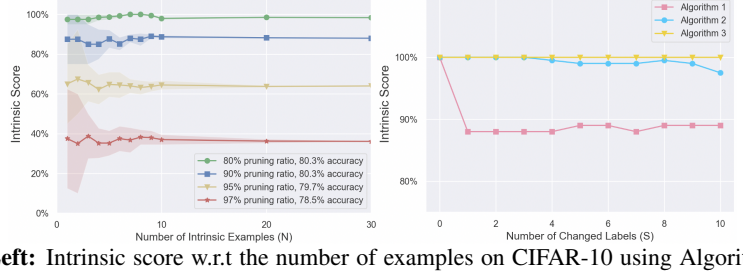


Figure 3: **Left:** Intrinsic score w.r.t the number of examples on CIFAR-10 using Algorithm 1. Each line is for an unstructured pruned neural network. **Right:** Result of fault injection attack using different level of injected faults. All faults are injected without significant accuracy drop (within 5%).

injected models corresponding to $|S| = 1, ..., 10$, and with $|R| = 1000$. The 10 models have small testing accuracy loss (within 5%) such that the fault injection attacks can be stealthy.

Figure 3 (Right) demonstrates the intrinsic scores by three algorithms on 10 fault injected models. Algorithm 3 keep the intrinsic scores as 100% for all models. And intrinsic scores by Algorithm 2 show slight degradation as the increase of the number of injected faults. The reason is that Algorithms 2 and 3 are designed to be more robust to slight weight modifications of the model, making them unable to effectively detect certain stealthy attacks. Meanwhile, intrinsic examples from Algorithm 1 demonstrate significant degradation on intrinsic score compared with Algorithms 2 and 3. In the case with only one injected fault, intrinsic score of Algorithm 1 decreases to 88%, lower than both Algorithms 2 and 3, demonstrating higher sensitivity in detecting fault injection attacks.

8 Large Scale Classification Task

We also explore the effectiveness of intrinsic examples on ImageNet [14], which contains over 14 million images and 1000 categories. We use a VGG-16 model achieving top 1 accuracy of 74.6% and top 5 accuracy of 92.4%. For large models like VGG-16, it is hard to apply Algorithm 3 due to the requirement of more computational resources and difficulty of convergence. So we use Algorithms 1 and 2 here for VGG-16 on ImageNet dataset.

The visualization results of intrinsic examples generated by Algorithm 1 and Algorithm 2, with different ε values are detailed in Figure A4 in Appendix G. We also summarize the performance of intrinsic examples on functionality verification for different weight pruning and quantization methods as shown in Table A7. We can observe that intrinsic scores of Algorithm 2 outperforms Algorithm 1 for all models, showing its superiority of accurately verifying the acceptable modifications compared with Algorithm 1. Meanwhile, in the case of heavy compression with significant accuracy drop, a relatively low intrinsic score can be obtained (such as 34% from Algorithm 1 and 80% from Algorithm 2 in the case of decimal quantization with 1 place), indicating the unsatisfied performance from the compressed model. The high correlation between the intrinsic score and testing accuracy demonstrates that intrinsic examples can verify the model functionality for large scale image recognition task with a much fewer number of examples.

Alg.	Dense Model	Unstructured Pruning		Pattern Pruning				Decimal Quantization		
		30%	60%	55.6%	77.4%	85.7%	87.5%	3 places	2 places	1 place
Acc.	Top1: 74.6% Top5: 92.4%	74.3% (-0.3)	74% (-0.6)	74.6% (0)	74.3% (-0.3)	73.9% (-0.7)	73.7% (-0.9)	74.5% (-0.1)	72.7% (-1.9)	67.3% (-7.3)
Alg.1	100%	85%	54.5%	**100%**	98%	96.5%	91%	67%	50%	34%
Alg.2	100%	**100%**	**97.5%**	**100%**	**100%**	**100%**	**100%**	**100%**	**89%**	**80%**

Table 4: Intrinsic Score of Implemented Models by Different Model Compression Methods on VGG-16 with ImageNet Dataset.

9 Conclusion

To verify the DNN functionality, we design three algorithms to generate intrinsic examples as DNN fingerprinting, with high robustness to benign model transformations and high sensitivity for the detection of adversarial third-party attacks. Extensive experiments demonstrate that the proposed methods have superior performance in reliably verifying the functionality of DNNs than current watermarking/fingerprinting methods.

Acknowledgement This work is partly supported by the National Science Foundation CNS-1932351, CNS-1932464, and CNS-1929300.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*, pages 274–283, 2018.

[3] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International Conference on Machine Learning (ICML)*, pages 284–293, 2018.

[4] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.

[5] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2204–2206, 2018.

[6] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian. Data-free learning of student networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3514–3522, 2019.

[7] Huili Chen, Bita Darvish Rouhani, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models. In *Proceedings of the 2019 on International Conference on Multimedia Retrieval (ICMR)*, pages 105–113, 2019.

[8] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[9] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 609–622. IEEE, 2014.

[10] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[11] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning (ICML)*, pages 160–167, 2008.

[12] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 485–497, 2019.

[13] Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, Pearse A. Keane, and Olaf Ronneberger. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine*, 24(9):1342–1350, 2018.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 248–255, 2009.

[15] Lixin Fan, Kam Woh Ng, and Chee Seng Chan. Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4716–4725, 2019.

[16] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[17] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[18] Jia Guo and Miodrag Potkonjak. Watermarking deep neural networks for embedded systems. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems (NeurIPS)*, pages 1135–1143, 2015.

[20] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017.

[21] Zecheng He, Tianwei Zhang, and Ruby Lee. Sensitive-sample fingerprinting of deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4729–4737, 2019.

[22] Zhezhi He and Deliang Fan. Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 11438–11446, 2019.

[23] Gopalakrishna Hegde, Siddhartha, Nachiappan Ramasamy, and Nachiket Kapre. Caffepresso: an optimized library for deep learning on embedded accelerator-based platforms. In *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, pages 1–10. IEEE, 2016.

[24] Kartik Hegde, Jiyong Yu, Rohit Agrawal, Mengjia Yan, Michael Pellauer, and Christopher Fletcher. Ucnn: Exploiting computational reuse in deep neural networks via weight repetition. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 674–687. IEEE, 2018.

[25] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29 (6):82–97, 2012.

[26] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems (NeurIPS)*, pages 4107–4115, 2016.

[27] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372. IEEE, 2014.

[28] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[29] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations (ICLR)*, 2016.

[30] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations (ICLR)*, 2017.

[31] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing*, 16(3):82–88, 2017.

[32] Erwan Le Merrer, Patrick Perez, and Gilles Trédan. Adversarial frontier stitching for remote neural network watermarking. *Neural Computing and Applications*, pages 1–12, 2019.

[33] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[34] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *NC*, 2007.

[35] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2117–2125, 2017.

[36] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[37] Ning Liu, Xiaolong Ma, Zhiyuan Xu, Yanzhi Wang, Jian Tang, and Jieping Ye. Autoslim: An automatic dnn structured pruning framework for ultra-high compression rates. *arXiv preprint arXiv:1907.03141*, 2019.

[38] Sijia Liu, Jie Chen, Pin-Yu Chen, and Alfred Hero. Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 288–297, 2018.

[39] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *ECCV*, 2018.

[40] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138. IEEE, 2017.

[41] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2736–2744, 2017.

[42] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*, 2018.

[43] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[44] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

[45] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

[46] Ryota Namba and Jun Sakuma. Robust watermarking of neural network with exponential weighting. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 228–240, 2019.

[47] Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, R Venkatesh Babu, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML)*, pages 4743–4751, 2019.

[48] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. Technical report, 2011.

[49] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[50] Kaoru Ota, Minh Son Dao, Vasileios Mezaris, and Francesco GB De Natale. Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(3s):1–22, 2017.

[51] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[52] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7197–7205, 2017.

[53] Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90 nm and 45 nm sram-cells. In *International Conference on Smart Card Research and Advanced Applications*, pages 193–205. Springer, 2015.

[54] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 618–626, 2017.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[56] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence (AAAI)*, 2017.

[57] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR)*, pages 269–277, 2017.

[58] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1675–1689. ACM, 2016.

[59] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[60] Ji Wang, Weidong Bao, Lichao Sun, Xiaomin Zhu, Bokai Cao, and S Yu Philip. Private model compression via knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 1190–1197, 2019.

[61] Siyue Wang, Xiao Wang, Pu Zhao, Wujie Wen, David Kaeli, Peter Chin, and Xue Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[62] Xiao Wang, Siyue Wang, Pin-Yu Chen, Yanzhi Wang, Brian Kulis, Xue Lin, and Sang Peter Chin. Protecting neural networks with hierarchical random switching: Towards better robustness-accuracy trade-off for stochastic defenses. In *International Joint Conferences on Artificial Intelligence Organization (IJCAI)*, 2019.

[63] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

[64] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2074–2082, 2016.

[65] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 19–35, 2016.

[66] Wayne Xiong, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. The microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE, 2018.

[67] Maurice Yang, Mahmoud Faraj, Assem Hussein, and Vincent Gaudet. Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning. In *2018 IEEE 48th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 180–185. IEEE, 2018.

[68] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems (NeurIPS)*, pages 3320–3328, 2014.

[69] Zheng Zhan, Yifan Gong, Zhengang Li, Pu Zhao, Xiaolong Ma, Wei Niu, Xiaolin Xu, Bin Ren, Yanzhi Wang, and Xue Lin. A privacy-preserving dnn pruning and mobile acceleration framework. *arXiv preprint arXiv:2003.06513*, 2020.

[70] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287, 2019.

[71] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS)*, pages 159–172, 2018.

[72] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 184–199, 2018.

[73] Tianyun Zhang, Kaiqi Zhang, Shaokai Ye, Jiayu Li, Jian Tang, Wujie Wen, Xue Lin, Makan Fardad, and Yanzhi Wang. Adam-admm: A unified, systematic framework of structured weight pruning for dnns. *arXiv preprint arXiv:1807.11091*, 2:3, 2018.

[74] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: A stealthy framework for misleading deep neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[75] Peng Zhou, Xintong Han, Vlad I. Morariu, and Larry S. Davis. Learning rich features for image manipulation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[76] Peng Zhou, Bor-Chun Chen, Xintong Han, Mahyar Najibi, Abhinav Shrivastava, Ser-Nam Lim, and Larry Davis. Generate, segment, and refine: Towards generic manipulation segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13058–13065, 2020.