Securing edge deep neural network against input evasion and IP theft

Wang, Si

2021

Wang, S. (2021). Securing edge deep neural network against input evasion and IP theft.
Doctoral thesis, Nanyang Technological University, Singapore.
<https://hdl.handle.net/10356/152267>

<https://hdl.handle.net/10356/152267>

<https://doi.org/10.32657/10356/152267>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0
International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 20:44:13 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**SECURING EDGE DEEP NEURAL NETWORK
AGAINST INPUT EVASION AND IP THEFT**

WANG SI

G1603609L

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University in partial
fulfilment of the requirement for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

Date 23-May-2021

WANG SI

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

Date 23-May-2021

CHANG CHIP HONG

Authorship Attribution Statement

This thesis contains materials from five papers published in the following peer-reviewed conferences, journals and book where I was the first student author.

The content of Chapter 2 has been published in S. Wang and C. H. Chang, "Deep learning network security," in *Frontiers in Hardware Security and Trust*, C. H. Chang and Y. Cao, Ed., IET Academic Book, Institution of Engineering and Technology, November 2020. Chapter 9, Book DOI: 10.1049/PBCS066E, Chapter DOI: 10.1049/PBCS066E_ch9, https://digital-library.theiet.org/content/books/10.1049/pbcs066e_ch9;jsessionid=2dqsc3j6c66si.x-iet-live-01.

The contributions of the co-authors are as follows:

- Prof. Chang Chip Hong prepared the proposal of this book chapter, provided the suggestion on the content organization and edited the manuscript drafts.
- I performed all the literature review, drafted the first manuscript of the surveyed topics and provided additional inputs to the content description for Prof. Chang's edition.

The content of Chapter 3 has been published in two conference papers: 1) S. Wang, W. Liu and C. H. Chang, "Detecting adversarial examples for deep neural networks via layer directed discriminative noise injection," in *Proc. 2019 Asian Hardware Oriented Security Trust Symposium (AsianHOST)*, Xi'an, China, Dec. 2019, pp. 1-6, doi: 10.1109/AsianHOST47458.2019.9006702; 2) S. Wang, W. Liu and C. H. Chang, "Fired neuron rate based decision tree for detection of adversarial examples in DNNs," in *Proc. 2020 IEEE Int. Symp. Circuits Systems (ISCAS)*, Sevilla, Oct. 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180476.

The contributions of the co-authors are as follows:

- Prof. Chang Chip Hong provided the initial project direction, gave suggestions to improve my initially proposed designs, and edited the manuscript drafts.
- I performed all the literature review, designed the two adversarial input detection schemes and evaluation methods. I also conducted all the simulations and analyses, and prepared the manuscript drafts.

-
- Mr. Liu Wenye provided the adversarial inputs, gave suggestions to improve my initially proposed designs and helped to revise the manuscripts.

The content of Chapter 4 has been published in S. Wang, W. Liu, and C. H. Chang, "A new lightweight in-situ adversarial sample detector for edge deep neural network," *IEEE J. Emerg. Sel. Topics Circuits Syst. (JETCAS)*, Apr. 2021, doi: 10.1109/JETCAS.2021.3076101

The contributions of the co-authors are as follows:

- Prof. Chang Chip Hong initiated the concept, provided the initial project direction, guided the system analysis and edited the manuscript drafts.
- I performed all the literature review, designed the in-situ adversarial input detector for edge deep learning and evaluation methods. I also conducted all the experiments and analyses, and prepared the manuscript drafts.
- Mr. Liu Wenye provided the adversarial inputs, assisted in experimental setup, provided valuable advices on the comparison with existing methods and helped to revise the manuscript.

The content of Chapter 5 has been published as a special session paper, S. Wang and C. H. Chang, "Fingerprinting deep neural networks – a DeepFool approach," in *Proc. 2021 IEEE Int. Symp. Circuits Systems (ISCAS-2021)*, Daegu, Korea (South), May 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401119.

The contributions of the co-authors are as follows:

- Prof. Chang Chip Hong initiated the idea, provided valuable advices on my initially proposed design and edited the manuscript drafts.
- I performed all the literature review, designed the fingerprinting approach and evaluation methods. I also conducted all the experiments and analyses, and prepared the manuscript drafts.

Date 23-May-2021

WANG SI

Acknowledgements

I would like to express my deepest appreciation to my supervisor: Associate Professor Chang Chip Hong, for his constant encouragement, easy-to-understand guidance, kind patience and valuable suggestions. Prof. Chang is always enthusiastic about research. He is good at finding good research problems and looking into them in a unique way. Such critical thinking has influenced me a lot during my Ph.D. candidature. Besides, his open-mindedness in research discussions and strict commitment in research integrity have set a good example for us to emulate. Furthermore, I would also like to express my gratitude to Mrs Chang, who together with Prof. Chang, has arranged many gatherings which give me a lot of warmth and relaxation.

I would also like to express my thanks to those who helped me during my Ph.D. candidature. Thanks to Dr. Sachin Kumar for introducing me to the Ph.D. Program and giving valuable suggestions about Ph.D. study. Thanks to my seniors, Dr. Cao Yuan, Dr. Zheng Yue, Mr. Liu Chaoqun and my peer Mr. Liu Wenye from our research group for providing me with the great technical assistance and guidance in doing projects. Thank all my friends, Dr. Yin Tiantian, Ms. Luo Jiawen, Ms. Zhong Huishu, Dr. Ho Truong Phu Truan, Dr. Zhang Ruochong, Dr. Liu Siyu, Dr. Yang Chuanshi, Dr. Luo Yunqi, etc., for offering me the great company and support. Besides, I would also like to thank the technicians of VIRTUS, the Integrated Circuit Design Center of Excellence, Ms. Guee Geok Lian and Ms. Gek Eng Tan for their assistance.

Finally, I would like to give special thanks to my family, especially my dear father, Wang Shangbing, who always supports and encourages me a lot from thousands of miles away when I am down. I will never forget those phone calls from which I can feel great warmth even he is not around. My honor belongs to him. I love him forever and always wish him good health and happiness. Last but not least, I would like to thank my boyfriend, Wang Kangda, who loves me, cares for me, cheers me, supports me and accompanies me. I wish him a great success in his research career and I will always have his back.

Table of contents

List of figures	xv
List of tables	xvii
Summary	xxii
1 Introduction	1
1.1 Motivation	1
1.2 Research Objectives	5
1.3 Major Contributions	6
1.3.1 Hardware Precision Aware Approaches to Detection of Adversarial Examples	7
1.3.2 A Glitch Injection Based Approach for Lightweight In-situ Detection of Adversarial Examples in Edge DNN	7
1.3.3 A DeepFool Approach for Fingerprinting the DNNs	8
1.4 Organization	9
2 Literature Review	11
2.1 Deep Neural Network (DNN) Preliminaries	11
2.1.1 Artificial Neural Networks (ANNs) and DNNs	11
2.1.2 Fundamental Components of DNNs	13
2.1.3 Popular DNN Architectures	16
2.2 DNN Hardware Acceleration	17
2.3 Related Works	21
2.3.1 Misprediction Attack	21
2.3.2 Confidentiality Attacks	31
2.4 Summary	34

Table of contents

3	Towards DNN Hardware Accelerator Friendly Detection of Adversarial Examples	37
3.1	Introduction	37
3.2	Threat Model	38
3.3	Proposed Discriminative Noise Injection Method	39
3.3.1	Hypothesis	39
3.3.2	Detection Framework	42
3.3.3	Experimental Setup	44
3.3.4	Simulation Results	45
3.4	Proposed Fired Neuron Rate Based Decision Tree Detector	46
3.4.1	Most Discriminative Layer (MDL)	46
3.4.2	Proposed Decision Tree Detector Framework	48
3.4.3	Experimental Setup	51
3.4.4	Simulation Results	52
3.5	Comparison	53
3.6	Summary	54
4	In-situ Adversarial Sample Detector for Edge Deep Neural Network (DNN)	55
4.1	Introduction	55
4.2	Proposed Online Adversarial Input Detector	58
4.2.1	Threat Model	58
4.2.2	Label Change Inconsistency between Original and Adversarial Inputs on Clock Glitch Injection	58
4.2.3	Proposed Decision Tree Detector	62
4.3	Experimental Results	69
4.3.1	Hardware Setup	69
4.3.2	Dataset	70
4.3.3	Results	74
4.3.4	Comparisons and Discussions	77
4.4	Summary	80
5	Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach	83
5.1	Introduction	83
5.2	Proposed Fingerprinting Approach	85
5.2.1	Threat Model	85
5.2.2	Proposed IP Protection Approach	85
5.3	Implementations and Results	88
5.3.1	Experimental Setup	88

Table of contents

5.3.2	Results	92
5.3.3	Comparison with Two Recent Fingerprinting Methods	99
5.4	Summary	100
6	Conclusion and Recommendations	101
6.1	Conclusion	101
6.2	Recommendations for Further Research	104
6.2.1	Effects of Emerging Vision Technologies on Physical Input Evasion .	104
6.2.2	Extension of the Proposed DeepFool based Fingerprinting Approach .	105
6.2.3	Extension of the Proposed Techniques to Ultra-lightweight DNN Im- plementations	106
	Publications	109
	Bibliography	111

List of figures

2.1	(a) A biological neuron, and (b) Mathematical model for a neuron in an ANN.	12
2.2	Output activation calculation at each layer of an ANN.	12
2.3	(a) Feedforward and (b) Recurrent networks.	13
2.4	High-dimensional convolution operation	15
2.5	An example of high-dimensional convolutions	15
2.6	Maximum and average pooling operations	16
2.7	Conventional configuration of DNN.	17
2.8	Different variants of convolutional block design.	18
2.9	Parallel processing schemes [1].	19
2.10	Typical architecture of DNN accelerator.	20
2.11	Decomposition of a 5×5 convolution to (a) two 3×3 convolutions or (b) a 1×5 and a 5×1 convolution [1].	21
2.12	Illustration of adversarial examples generated using [2] for AlexNet [3]. The perturbations are magnified $10 \times$ for better visualization. The predicted classes for adversarial images are also shown in red texts.	23
2.13	Illustration of adversarial examples generated using universal perturbations[4] for AlexNet [3], and GoogLeNet [5].	26
3.1	Comparison of $\Delta\zeta$ in different layers of VGG 16.	40
3.2	Comparison of $\Delta\zeta$ in different layers of VGG 19.	41
3.3	Comparison of $\Delta\zeta$ of different layers in MobileNet.	42
3.4	Framework of proposed dominant layer directed discriminative noise injection approach.	42
3.5	Histogram distribution of <i>LCR</i> for legitimate and adversarial samples with 0.006 noise injection rate in layer <i>conv3_3</i> of VGG 16.	43
3.6	FNR distribution in different layers of AlexNet.	47
3.7	FNR distribution in different layers of VGG 16.	47
3.8	FNR distribution in different layers of VGG 19.	48

List of figures

3.9	FNR distribution in different layers of MobileNet.	49
3.10	Decision tree for successive thresholding of normal and adversarial FNR distributions in MDL.	49
3.11	Construction of the FNR based decision tree	50
4.1	Timing diagrams of a typical sequential logic path under (a) a normal clock and (b) a glitched clock.	57
4.2	The generation of a clock glitch on the DNN accelerator.	60
4.3	Comparison of ΔLCP in different layers of VGG16.	62
4.4	An example of the proposed tree classifier detector for VGG16. Nodes $v_{4,1}$ and $v_{4,3}$ are terminal nodes B, and nodes $v_{3,4}$ and $v_{4,6}$ are terminal nodes A.	63
4.5	An example that illustrates the optimal threshold and the IG for different splitting attributes.	65
4.6	Experimental setup for mutation control of DNN implemented on Xilinx ZCU102 evaluation board.	70
4.7	Box-and-whisker plot of the top-1 probabilities for the clean images and the adversarial images generated by various attacks for VGG16	72
4.8	Box-and-whisker plot of the top-1 probabilities for the clean images and the adversarial images generated by various attacks for ResNet50	72
5.1	A flowchart for the proposed fingerprinting method.	86
5.2	A given clean input and its corresponding adversarial input generated by DeepFool.	87
6.1	The pipeline of RP_2 attack [6].	105

List of tables

3.1	Average ζ for normal samples and adversarial samples (unit:%).	39
3.2	Final configurations for the three models with hierarchical detection scheme.	44
3.3	Final configurations for the three models with single-layer detection scheme.	44
3.4	Evaluation results for different discriminative noise injection configurations.	45
3.5	Parameters of the final decision trees for the four DNN models.	52
3.6	Correct detection and false positive rates.	52
3.7	Comparison among the two proposed works and feature squeezing method.	53
4.1	Number of images that have changed their predictions ($\#LC$) upon injecting different intensities of glitches into each layer of VGG16 model.	61
4.2	Training and testing datasets for VGG16 and ResNet50.	71
4.3	Parameters of the final classical and lightGBM decision trees (DTs) for VGG16 and ResNet50.	73
4.4	Comparison of classical and lightGBM tree detectors in true positive rates (TPRs) and false positive rates (FPRs) for VGG16 and ResNet50.	74
4.5	Hardware overhead of DNN with proposed clock glitch mutation control	76
4.6	Comparison of the proposed lightGBM detector for VGG16 and existing adversarial sample detection methods.	78
4.7	Comparison of the largest scale of DNN evaluated between the proposed lightGBM detector and DeepFense [7].	79
4.8	Comparison of hardware overhead between the proposed lightGBM detector and DNNGuard [8].	80
5.1	Testing accuracies of the target and suspect classifiers for CIFAR-10. “-” means not applicable.	90
5.2	Testing accuracies of the target and suspect classifiers for CIFAR-100. “-” means not applicable.	91

List of tables

5.3	Testing accuracies of the target and suspect classifiers for ImageNet-10. “-” means not applicable.	92
5.4	Datasets, target classifiers and numbers of positive and negative suspect classifiers.	92
5.5	Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for CIFAR-10.	93
5.6	Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for CIFAR-100.	94
5.7	Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for ImageNet-10.	95
5.8	Overlap in matching rate and gap in average matching rate between positive and negative suspect classifiers for each target classifier.	95
5.9	Matching rate vectors $\mathbf{m}$ and probability vectors $\mathbf{P}$ of various suspect classifiers for CIFAR-10 using the proposed method.	97
5.10	Matching rate vectors $\mathbf{m}$ and probability vectors $\mathbf{P}$ of various suspect classifiers for CIFAR-100 using the proposed method.	98
5.11	Matching rate vectors $\mathbf{m}$ and probability vectors $\mathbf{P}$ of various suspect classifiers for ImageNet-10 using the proposed method.	99
5.12	Run time (in seconds) of generating 100 fingerprinting data points for each target classifier using the proposed method.	99
5.13	Comparison with two recent works.	100

Summary

Deep learning is a key driver that puts artificial intelligence (AI) on the radar screen for technology investment. Deep Neural Network (DNN) automatically learns high-level features directly from raw data in a hierarchical manner, which eliminates manual extraction of effective features in traditional machine learning solutions. The ability to solve problems end-to-end enables a system to learn complex functions mapping for ill-posed problems with prediction accuracy often exceeding sophisticated statistical models and other machine learning methods. Computer vision is one specific domain that DNN has demonstrated this remarkable abstraction power. Unlike other mainstream classification approaches, DNN can usually achieve better results with more data and larger model. Over the last decade, the model complexity and regulation mechanism of DNN have grown tremendously to overcome the performance plateau and improve the generalization ability. The flourishing of Internet of Things (IoT) has changed the way data are generated and curated. Consequently, DNN hardware accelerators, open-source AI model compilers and commercially available toolkits like Intel(R) OpenVINO(TM), have evolved to enable more user-centric deep learning applications to be run on edge devices without being limited by the network latency.

This research is motivated by the two major security threats of deep learning. One is the adversarial example obtained by deliberately adding imperceptibly small perturbations onto the benign input. Such input evasion can delude a well-trained classifier into wrong decision making. Adversarial examples can be generated fast at low cost. Their attack surface can also be extended beyond the software boundary and made more robust with high transferability across models. Existing countermeasures against adversarial examples are mainly designed and evaluated based on software models of DNNs implemented with 32-bit floating-point arithmetic. To support secure embedded intelligence, the defense should take hardware optimization and resource constraints of edge platforms into consideration. The other threat is the Intellectual Property (IP) theft. As training a good DNN model needs huge capital investment in manpower, time and physical resources, which may not be affordable or accessible by small corporations, the trained model is often a pricey proprietary asset of a business and is normally kept confidential. However, the emerging model extraction attacks and reverse engineering techniques enable the DNN model to be stolen to build similar quality AI product

at a low cost. In order to protect the interest and revenue of the model owner, a pragmatic solution without reverse engineering the DNN chip to detect the pirated AI chip is required. A comprehensive review of DNN has been conducted, which highlights the prevalent adversarial input generation methodologies and IP theft techniques, and corresponding countermeasures. Three original contributions, including two hardware-oriented approaches, a new lightweight in-situ adversarial input detector for edge DNN, and a method for fingerprinting DNN to attest the model ownership, are presented in this thesis.

The first contribution is two hardware-oriented adversarial sample detection methods. One detects adversarial samples via layer directed discriminative noise injection. It is inspired by the recent discovery of an observable label change rate difference between benign and adversarial images when random mutations are applied on the DNN model. This work further looks into the distinguishability of each layer of the model by injecting different amount of noise into the corresponding weight parameters. Dominant layers that affect the tampered samples most under varying extent of weight mutations are selected to progressively discriminate adversarial inputs from benign inputs. Another method is designed based on the disparity in the layer-wise fired neuron rate between adversarial and benign samples. Fired neuron rate is the proportion of positive activations in the output feature map of a layer. A decision tree is used to find out the most discriminative layer of the target DNN that has least overlap in fired neuron rate distributions between the adversarial and benign images and the corresponding threshold at each decision node. Both detection methods are evaluated with modern hardware-oriented 8-bit quantized convolutional neural networks (CNNs) targeting large-scale ImageNet classification against the most practical non-invasive universal perturbation attack. The results show a significant improvement on both detection rate and false positive rate compared with fully software-oriented feature squeezing method.

The second contribution is a new lightweight adversarial input detector that can be efficiently implemented alongside a spatial architecture for edge DNN implementation. Existing methodologies are mainly developed for offline analysis, assuming that the deep learning model is implemented on a 32-bit floating-point Central Processing Unit (CPU)/Graphical Processing Unit (GPU) instance. Their code sizes and computational complexity are too high to be integrated into the edge AI platform processor or reconfigurable fabrics. The proposed work allows the adversarial inputs to be identified in-place with the inference machine. It is inspired by the aforementioned layer directed discriminative noise injection scheme, but with fundamentally different mutation mechanisms, optimal label change mutation strategies and detector design. Controlled clock glitch injection into the DNN computations is exploited in the hardware accelerator to mutate the edge-implemented DNN model. The optimal glitch intensity that maximizes the label change inconsistency between the adversarial and benign

images for each layer is investigated for the construction of a light gradient boosting machine (lightGBM). The test node attributes of the lightGBM are determined by an offline analysis of the top-1 probabilities of unmutated and mutated models and their difference. The distribution that maximizes the information gain for the discrimination of the adversarial and benign inputs is selected for each test node. With negligibly small hardware overhead, the glitch injection circuit and the fixed lightGBM detector can be easily implemented alongside with the deep learning model on a Xilinx ZU9EG chip. The performance of the proposed in-situ detector is evaluated by applying four different popular adversarial attacks, fast gradient sign method (FGSM) with different settings, iterative-FGSM (I-FGSM), Carlini and Wagner (C&W) and universal perturbation (UP) on two different types and scales of DNN models, VGG16 and ResNet50, for a thousand-class visual object recognition application. The results show a significant improvement in the correct detection rate, the false positive rate and lower overheads compared with the recently proposed software and hardware oriented adversarial input detectors.

Finally, a fast IP theft detection technique that makes use of DeepFool algorithm to fingerprint the DNN model is proposed. DeepFool is an adversarial input generation algorithm that intrinsically produces data points lying near the decision boundary. As a classifier can be characterized by its decision boundary, the adversarial data points generated by DeepFool as well as their labels predicted by the target classifier can be reversed into a model fingerprint at the advantage of the model owner. Since the fingerprint is extracted after the training process has been completed, the original achievable classification accuracy will not be compromised, whereas existing watermarking techniques often interfere with the model training process and result in inevitable degradation in prediction performance. The matching rate is defined as the proportion of fingerprinting data points that have the same labels predicted by both the suspect and target classifiers. It is utilized to visualize the difference between a pirated classifier and an originally designed classifier for a given fingerprint. The Bayes' Theorem is applied to provide a confidence level for a verification result to enhance its non-repudiability. An iterative verification is also adopted to increase the confidence of a correct detection of a pirated classifier. The proposed DeepFool based fingerprinting method is evaluated on CIFAR-10, CIFAR-100 and ImageNet classifications. Four prevalent post-processing techniques that an attacker may apply to the pirated target model to evade detection are also considered in the evaluation.

Chapter 1

Introduction

1.1 Motivation

The explosion of data usage has contributed to the extensive processing requirements for most of the applications running on smart devices, as well as edge and fog computing nodes. Due to the scale and complexity of the tasks, decision support systems can greatly benefit from the use of machine learning (ML) techniques to correlate multimodal sensing to make accurate predictions and powerful inferences. Traditional ML algorithms have to be fed with previously extracted features. These features are usually identified in advance to lower the complexity of high-dimensional data, reduce the redundancy and increase the interpretability of the patterns to the learning algorithms [1]. Furthermore, in some circumstances like multiple object detection, the task needs to be divided into parts and solved individually before the partial results are combined at the final stage. The required human intervention and discontinuity in data processing reduce the efficiency of conventional ML algorithms in solving intricate tasks that require large amount of unstructured raw data from diverse sources for training. Deep learning (DL), also referred to as deep neural network (DNN), has overcome the weakness of the need for human's participation on informative feature identification, hardcore data science knowledge and expensive automatic feature selection and transformation. It learns the high-level features from raw data in an incremental manner and solves the problems end-to-end. As a result, DL has now become a preferred option for handling majority of the challenging tasks in image classification [9], speech recognition and language processing [10].

In the early 2010s, DL blossomed upon the occurrence of two breakthrough events, namely the announcement of the Microsoft's speech recognition system in 2011 [11] and Krizhevsky et al.'s demonstration of the outstanding performance of AlexNet system on the ImageNet Large Scale Visual Recognition Challenge task in 2012 [3]. These early successes inspired prodigious research activities towards advanced development of DNN algorithms. These

Introduction

sophisticated algorithms exploit superior DNN architectures and regulation mechanisms to achieve remarkable accuracy and generalization. To ease the high computational complexity of DNNs without compromising performance accuracy or increasing hardware cost, hardware designers and solid-state circuit community have begun to look into efficient processing techniques to enable a wider deployment of DNNs for more powerful consumer-centric artificial intelligence (AI) applications. Some of the commonly used techniques include: optimizing memory hierarchy to improve the energy efficiency by introducing low-cost data movement; quantization of the data values to reduce the number of bits for their representation, which helps to reduce storage and relax computation requirements; compression of the networks to reduce the number of operations and model size by exploiting the sparsity of activations and weights [1]. The hardware that uses these specialized techniques for accelerating DL inferences is called the DNN accelerator. DNN accelerator research aims at improving the throughput and energy consumption, e.g., Neuflow [12], DaDianNao [13] and Eyeriss [14], by alleviating the stringent computing requirements and resource limitations on the underlying hardware platform. The flourishing of Internet of Things (IoT) also rekindles on-premise computing. To allow the data to be analyzed closer to the source, edge AI, hardware accelerators, open-source AI model compilers [15] and commercially available toolkits have evolved to facilitate the development and deployment of applications that use AI at its core. Today, DL technology has matured to become an attractive option for safety and security critical applications, including self-driving cars, video surveillance and malware detection [16], to name but a few. The recent adoption of face ID security on mobile devices [16] and the success in automated detection of COVID-19 cases from chest radiographs are important milestones and testimonials of how fast DL solutions have infiltrated into our daily lives.

Despite these impressive outcomes in licit application scenarios, the inability to fully understand and explain what have been done within the super-intelligence can easily turn an inherently benevolent AI system to malevolent. An exploitation of this vulnerability was firstly demonstrated by Szegedy et al. [2] to corrupt DL based image classification systems. They found that the networks are susceptible to adversarial samples obtained by maliciously adding imperceptibly small perturbations to the original inputs. Notwithstanding the high generalization ability, the model can hardly evade such simple adversarial attacks. These adversarial distortions are relatively robust across different classifiers. The finding triggered many studies on efficient algorithms for adversarial image generation and raised grave security concerns. One representative method is fast gradient sign method (FGSM) [17]. It takes advantage of the gradient of loss function to determine the effective perturbations. Another popular method is universal perturbations (UP) proposed by Moosavi-Dezfooli et al. [4]. This method changes the paradigm of adversarial examples from image-dependent to image-agnostic.

Subsequent attack [18] in 2017 further relaxed the requirement on the knowledge of DNN model details or training data, making adversarial attack non-invasive and more robust. To make matters worse, such harmful and stealthy adversarial examples that work originally on software models of DNN can be successfully extended to the physical hardware implementations, which threatens the deployment of DNNs in safety and security critical applications. For example, Eykjolt et al. [6] successfully demonstrated that by attaching the dedicatedly designed stickers that resembles graffiti to a Stop sign, it can be misclassified as Speed Limit 45 with high confidence. This increases the likelihood of a car accident if DL is applied in the computer vision system of a self-driving car.

Existing defend strategies against the input evasion attacks can be dichotomized into two broad categories of robustness enhancement and adversaries detection. The former attempts to restore the correct prediction of a malicious input without sacrificing the classification accuracy of the benign samples. The latter tries to identify the adversarial inputs. Robustness enhancement can be achieved by adversarial training, gradient masking and input transformation. Adversarial training increases the model’s tolerance to malicious input distortions by including off-the-shelf adversarial data for training. However, this defense has been overcome by new attacks [17]. Gradient masking removes the gradients in specific directions and neighborhoods of the training points so that the adversary has no clue to discover which points and directions to perturb an image across its decision boundaries. However, this defense has also been circumvented by training a substitute model using the inputs chosen by the adversary to imitate the defend model, as demonstrated by the black-box attacks in [19, 18]. Input transformation eliminates the adversarial perturbations by reconstructing the inputs [20–23] using either dimensionality reduction techniques or compressing the inputs. The problem with most input transformation methods is some local information needed for accurate classification is lost in the transformed space, and the model architecture need to be redesigned. Other input sanitizers assume that the attacker knows only the target classifier but not the input transformation. These countermeasures can be easily defeated by a slight increase in the perturbation strength in the white-box setting. To avoid compromising the accuracy of benign input classification, sample statistics, detector training and prediction inconsistency are used for adversarial detection. Sample statistics approach analyzes a large set of sample data to identify adversarial from clean data [24]. This makes them computationally intensive and can only be done offline. Moreover, these methods are effective only against adversarial examples lying far from the manifolds of the legitimate population. Detector training often embeds a customized subnetwork into the original network to detect adversarial samples [25]. The training of the subnetwork is also computationally expensive, and prone to overfitting. Prediction inconsistency makes use of the disagreement among predictions of different models for adversarial inputs to single

Introduction

out the harmful instances. This strategy incurs high latency or power consumption. Input transformation has also been explored for prediction inconsistency.

It is observed that existing protection methods are mainly designed and investigated based on software models of DNNs. To be more specific, their analyses are typically performed offline by assuming that the DL models are implemented and executed on powerful high-precision floating-point GPU server platform. The optimization of the learning models for hardware implementation can lower the effectiveness of these countermeasures. Many of these protection mechanisms cannot be efficiently implemented on edge platform together with the DNN. With the globalization of supply chain in integrated circuit design and manufacturing, infiltration of malicious hardware components into the neural network circuit poses several new and realistic security threats besides input invasion [26, 27]. Noninvasive side-channel attacks that leverage on information leakage or backdoor created unconsciously in the hardware optimization and implementation processes are also feasible on DNN accelerators. Model extraction [28–31] and reverse engineering [32] of deployed DNN models have been successfully demonstrated via exploitation of side-channel leakage. Training a good DNN model with enormous amount of labelled data and computing power requires huge capital investment in manpower, time and physical resources, which may not be affordable or accessible by small corporations. Hence, the trained model is a pricey proprietary asset of a business and is normally not disclosed publicly. However, model extraction and reverse engineering attacks enable business opponents to rebuild similar AI product or solution to the same problem at a low cost to gain unfair competitiveness. Unfortunately, it is difficult to prove the ownership of a DNN model when the model is implemented onto different hardware platforms.

To protect the interest and revenue of the model owner, watermarking was recently proposed to detect intellectual property (IP) theft. A watermark can be embedded into the model internals during training. The IP theft can be detected by comparing the extracted watermark from a suspect DNN model against that of the target model [33]. The drawback of this approach is the requirement of physical access to the suspect model for verification. Alternatively, a small set of modified inputs can be acquired through either content perturbation or wrong labelling. This set of data is used to train the target model to produce a unique prediction [34–36]. During verification, the suspect model is queried by this data set to compare their prediction results against those of the target model. Classification accuracy is inevitably sacrificed as the watermark is embedded through the model training process. In fingerprinting methods, signature embedding in the training process of the target model is avoided. Instead, robust inputs that have similar prediction results as the target model are extracted for a pirated or post-processed version of the target model to differentiate them from an originally designed model [37, 38]. Fingerprinting methods for DNN IP protection, being an emerging research, have rooms for

improvement. For example, the mechanisms used for the extraction of different fingerprints vary in the detection efficiency and their latency overheads in fingerprint extraction are high. A better approach to overcome these shortfalls is needed to enhance the non-repudiability and efficiency in fingerprint extraction.

1.2 Research Objectives

The research work described in this thesis deal with two major DL security issues: input evasion and IP theft attacks. More specifically, the following research questions will be answered to address the gaps of current countermeasures reported in the literature:

1. *What is the influence of hardware optimization techniques on input evasion attack feasibility and defense effectiveness?* DNN models are typically implemented on CPU/GPU and application-specific accelerator platforms. The former has greater agility while the latter is more energy efficient. It is common to have the model training performed on CPUs/GPUs, but the inference is executed in dedicated DNN accelerators. Thus, it is also important to ensure that defense algorithms work well with the DNN accelerators to secure embedded intelligence. Most of the existing countermeasures that target single precision floating-point arithmetic DNN implementation on GPU platform may not have considered the effects of fixed-point arithmetic and quantization of DNN model implementation on hardware accelerators. Similarly, the strength of existing adversarial attacks has not been evaluated under hardware accelerated DL models. Therefore, a more realistic threat scenario that targets hardware accelerated DL applications should be considered, and correspondingly, hardware-oriented countermeasures should also be explored. The answer to this question will lead to a plausible solution to the next question.
2. *Can hardware characteristics be exploited for building countermeasures against adversarial example attacks at design time?* Scalability issues, excessive power consumption, connectivity and latency are driving the demand for edge computing that enables real-time user engagement and fast sensor data analysis. The unprecedented growth in edge computing paves the way for many services and applications at the edge, so does the AI. In edge AI, the AI algorithms are processed locally on a hardware device, without requiring any connection. It uses data generated from the device and processes it to give insights in real time or online without further offline analyses. Examples are face recognition to unlock the phone, Google maps instant traffic condition alerts, self-driving car, etc.. With AI connected to edge devices, one additional dimension of richer local hardware and implementation-specific characteristics, apart from model internals and inputs, can be useful. This factor has not

been fused into existing defenses against adversarial examples. Previous defensive strategies are solely designed within the model or data domain. The downside of such approach is that a corresponding revised attack can be easily developed to evade the defenses once the defense methodology is known publicly. Defense mechanisms integrated into the hardware are harder to be counteracted by attacker, particularly when their implementations are made on the same hardware platform and interwoven with the computations that affect inference or learning. Another issue with software- or input-oriented solutions is that they are often designed without taking into consideration the resource-constraints of edge devices. The answer to this question will provide a new perspective to tip the asymmetric advantage towards the designer side in the security arm race.

3. *Can the fingerprint for detection of DNN IP theft attack be made fast, more convenient and non-repudiable at the same time?* Fingerprint derivation and extraction for detection of DNN IP theft is a nascent field of research. The notion of fingerprint as an innate signature of the DNN classifier based on the current instantiation carries a higher factor of fortuity than watermarking. Unfortunately, current research on fingerprinting methodologies focuses mainly on increasing the efficiency of a given fingerprint for a target model. Such focus tends to limit the number of usable fingerprints that can be generated from a DNN classifier. This arguably has an implied risk of giving the attackers less effort and challenge to look for an effective post-processing technique to evade detection or reduce the credibility of the fingerprint as a provenance proof. The challenge is, without intruding into the DNN models and architectures, will it be possible to provide a strong evidence to validate the provenance of a DNN classifier? Can such an IP proof be verified conveniently and efficiently in the field by a legal user or licensee of a DNN. To answer this question, the randomness of data points in the fingerprint space would be explored to reduce the probability of coincidence and increase the difficulty and cost for an attacker to pirate a DNN design that can evade the detection.

1.3 Major Contributions

The pursuit of the solutions to the abovementioned research problems has led to the following original contributions in this thesis: 1) Two different approaches to the detection of adversarial examples leveraging on selective layer computational perturbations and fired neuron rates of dominant layers; 2) A glitch injection based approach for lightweight in-situ detection of adversarial examples in edge DL; 3) A DeepFool approach for fingerprinting the DNN classifiers.

1.3.1 Hardware Precision Aware Approaches to Detection of Adversarial Examples

Two hardware-oriented adversaries detection techniques have been proposed and demonstrated. This contribution aims to enhance the security of a custom integrated DNN classifier against the most practical non-invasive universal perturbation attack. The countermeasures are designed with consideration of fixed-point arithmetic and quantization techniques typically used in hardware accelerator for AI implementation on resource-constrained endpoints. In the first work, a new discriminative noise injection strategy is proposed. It adaptively perturbs a few dominant layers to progressively discriminate adversarial inputs from benign inputs. This is made possible by evaluating the differences in label change rate from both adversarial and natural images by injecting different amount of noise into the weights of individual layers in hardware-aware data representation of the DNN model. The approach is evaluated on the ImageNet Dataset with 8-bit truncated models for the state-of-the-art DNN architectures. The results show a high detection rate of up to 88.00% with only approximately 5% of false positive rate for MobileNet. In the second work, a new thresholding method is proposed. The adversarial inputs can be detected by comparators integrated into the most discriminative layers of the DNN. During the training phase of a decision tree based adversarial input detector, the most discriminative layers are determined by their layer-wise fired neuron rates difference between the adversarial and normal inputs. Effectiveness of the method is also validated on the ImageNet dataset with 8-bit truncated models for the state-of-the-art DNN architectures. A high detection rate of up to 98% with only 4.5% of false positive rate is achieved. Both proposed solutions show a significant improvement on correct detection rate and false positive rate compared with existing countermeasures against the most practical non-invasive universal perturbation attack on deep learning based AI chip.

1.3.2 A Glitch Injection Based Approach for Lightweight In-situ Detection of Adversarial Examples in Edge DNN

This solution addresses the incompatibility problem of existing software-oriented adversarial input detection methods on hardware-accelerated resource constrained platforms. The proposed method can detect adversarial inputs feeding through a spatial DNN accelerator architecture or a third-party DNN Intellectual Property (IP) implemented on the edge. It injects controlled glitches into the clock signal of the DNN accelerator to maximize the information gain for the discrimination of adversarial and benign inputs. A light gradient boosting machine (lightGBM) is constructed by analyzing the prediction probability of unmutated and mutated models and the label change inconsistency between the adversarial and benign samples in the training dataset.

Introduction

The hardware overhead of the detector is negligibly small even for a very large-scale deep learning model VGG16. Hence, the glitch injection circuit and the trained lightGBM detector can be easily implemented alongside the DNN on a Xilinx ZU9EG chip. The effectiveness of the proposed detector is validated against four state-of-the-art adversarial attacks on two models, VGG16 and ResNet50, for a thousand-class visual object recognition application. Comparing with modern software-oriented adversarial sample detection methods, the true positive rates against Fast Gradient Sign Method (FGSM), Iterative-FGSM (I-FGSM), C&W and universal perturbation attacks are increased significantly by up to 185.8%, 83.9%, 34.3% and 15.1%, respectively and the false positive rates are reduced by 74.5%, 57.7%, 25.9% and 86.8%, respectively.

1.3.3 A DeepFool Approach for Fingerprinting the DNNs

This work reverses the malevolent role of DeepFool, which is an algorithm meant originally for automatic generation of adversarial inputs, into a DNN fingerprint extraction method. The proposed solution can detect a pirated AI chip without reverse engineering the suspect DNN chip and compromising the prediction performance of the target DNN. The geometry characteristics inherited in the DeepFool algorithm is utilized to extract data points near the classification boundary of the target DNN model. Such data points together with their labels predicted by the target model are used as fingerprints of the target model. This countermeasure is founded on the hypothesis that different models possess different classification boundaries determined solely by the parameters of the DNN and the training it has undergone. Given a set of fingerprint data points, a pirated model or its post-processed version will produce similar prediction. Conversely, an originally designed and trained DNN for the same task will produce very different prediction for this set of fingerprint data points, even if it has similar or better classification accuracy. The matching rates of different sets of fingerprint data points generated by taking random samples as the inputs to the DeepFool algorithm are studied. Bayes' Theorem is applied to provide a confidence level for the verification result. An iterative process is proposed to reduce the probability of coincidence, hence the effectiveness and robustness of IP ownership proof. This DeepFool based IP theft detection approach is validated on the CIFAR-10, CIFAR-100 and ImageNet datasets with seven target models in total. The results show a detection rate of 100% and a false positive rate of 0% for each target model. More importantly, the fingerprint extraction is on average $\sim 71 \times$ faster than the two state-of-the-art fingerprinting methods.

1.4 Organization

The remainder of the report is organized as follows:

Chapter 2 presents a brief introduction of DL and related works. The preliminaries of DNNs including relevant definitions, fundamental building blocks and widely adopted DNN architectures are introduced. The custom hardware acceleration techniques as well as a typical DNN accelerator architecture are presented. Classification of emerging misprediction attacks is presented with emphasis on data evasion attacks, their salient features and the corresponding defenses. Confidentiality threat pertaining to IP theft for commercial DL as a service and DNN products are introduced and some recent watermarking and fingerprinting methods for the detection of DNN IP theft are elucidated.

Chapter 3 introduces two proposed adversarial input detection schemes for edge DNNs. The bottlenecks of the existing software-oriented defenses against input evasion are presented before the threat model is elaborated. The hypothesis, detection framework, experimental setup and simulation results of first proposed discriminative noise injection method is described. This is followed by a detailed description of the second proposed hardware-oriented fired neuron rate based decision tree detector. Finally, a comparison among these two proposed designs and the well-known feature squeezing method is made.

Chapter 4 presents a lightweight in-situ adversarial input detector for edge DNN. The constraints of existing fully software-oriented defenses and the scalability issue of the existing hardware-oriented defenses are discussed. Preliminaries and background knowledge about clock glitches is presented. Its use as the DNN model mutation mechanism in the proposed detection method is elaborated. The threat model, label change inconsistency between clean and adversarial inputs on clock glitch injection and the derivation of the final decision tree detector for the proposed online adversarial input detection method are elaborated. The performance of the proposed system is comprehensively evaluated, compared with relevant existing countermeasures and discussed.

Chapter 5 presents the proposed DeepFool based fingerprinting approach for detection of the DNN IP theft attack. A preamble of the existing countermeasures for the model ownership verification and their constraints is provided. This is followed by a detailed description of the threat model and the proposed IP protection method. Finally, the performance of the proposed DeepFool based fingerprinting approach is comprehensively evaluated and analyzed. Its comparison with two recent fingerprinting methods is also presented and discussed.

Chapter 6 concludes this thesis by summarizing the major contributions and highlighting a few promising future works that are worthy of further exploration.

Chapter 2

Literature Review

2.1 Deep Neural Network (DNN) Preliminaries

2.1.1 Artificial Neural Networks (ANNs) and DNNs

Artificial neural networks (ANNs) are a large sub-field of machine learning (ML). They are inspired from the working principal of brain, which is currently the best “engine” for learning and solving problems [1]. The essential component of an ANN is neuron computation that imitates the way the brain works. As shown by Fig. 2.1(a), a biological neuron consists of the dendrites, a cell body and an axon. The dendrites transmit received signals to the cell body. The signals are normally scaled by synapses that are connections between the dendrites of one neuron and axons of other neurons. The cell body performs a computation on the incoming scaled signals and generates an output activation, which is transmitted to other neurons as input signal by the axons. This process is mathematically modeled as an ANN in Fig. 2.1(b), where x_i indicates the i -th input signal to the neuron and w_i indicates the scaling factor of the input signal x_i . w_i is called a *weight* to mimic the function of a synapse in the biological model, and b can be considered as the resting state of a neuron. The neuron computes a weighted sum of the input values. Non-linearity is introduced by applying an activation function f to the sum to obtain an output activation. There are multiple neurons in each layer. The layer of neurons that is not input or output is known as hidden layer. An ANN may consist of one or more hidden layers. The calculation in each hidden layer is shown in Fig. 2.2, where $y_j = f(\sum_{i=1}^3 w_{ij}x_i)$.

Weights of an ANN are derived in the training phase where the model acquires its learning ability. Using image classification as an example, the output of an ANN will be a vector of scores that imply the possibilities for an image to be classified to each object class. The weights should be set at the values such that the output has the highest score in the correct class and that score is maximized. The deviation between the ideal score and the score calculated by an ANN

Fig. 2.1 (a) A biological neuron, and (b) Mathematical model for a neuron in an ANN.

Fig. 2.2 Output activation calculation at each layer of an ANN.

based on its current weights is called the loss (L). Weights are updated through an iterative process until the loss is minimized. The optimization utilizes a gradient decent approach to find the local minimum of the loss by taking steps proportional to the negative of the gradient of loss function at the current point. The basic formula is presented as follows [1]:

$$w_{ij}^{t+1} = w_{ij}^t - \alpha \frac{\partial L}{\partial w_{ij}} \quad (2.1)$$

where α refers to the learning rate that has an influence on the convergence speed.

The gradient is often obtained by back-propagation method. It calculates the partial derivatives of the loss with respect to the weight backwards through the network by applying the chain rule [3]. Apart from supervised learning mentioned previously where all the training samples are labeled, there are three other techniques to train the network. Unsupervised learning takes unlabeled data to explore the possible clusters; semi-supervised learning applies unknown examples to determine the decision boundaries and annotates a small portion for

labeling groups; reinforcement learning is often applied in the policy network that outputs a decision and receives the corresponding reward as a feedback from the environment. It helps to make decisions that maximize the reward as well as balance the exploration and exploitation [39, 10, 40]. The determined weights are then adopted in the inference stage, where the model performs its task by forward propagation.

DNNs are one category of ANNs but with more layers in their architectures. Typically, a DNN can contain five to more than a thousand layers [1]. One interpretation for the inner processing of the deep networks in vision field is hierarchical feature extraction. When an image is fed into the first layer of the DNN in the form of pixels, the corresponding activations can be construed as miscellaneous subordinate features of the image-like lines. These features are then combined into more and more advanced ones at following layers. For example, lines are aggregated into corners, which are further merged into other shapes. Finally, the network yields a probability that the obtained high-level features compose a specific object.

2.1.2 Fundamental Components of DNNs

Fig. 2.3 (a) Feedforward and (b) Recurrent networks.

DNNs can be broadly classified into recurrent DNN (RDNN) and feedforward (FDNN) DNN, as shown in Fig. 2.3. A recurrent network allows the presence of feedback, thus enables the existing internal states to store the required intermediate activations for later processing. It is temporally dependent. This dynamic behavior facilitates the use of RDNNs in applications like unconstrained handwriting and speech recognition. In contrast, a feedforward network performs computations in sequence. All the operations are completed with the outputs of the previous layer as inputs. There is no loop in the information flow. One prevalent type of FDNNs are Convolutional Neural Networks (CNNs). It is widely used in image classification for its shared-weight architecture and translation invariance properties [41, 42]. A CNN has five fundamental building blocks: convolutional (Conv), pooling, rectified linear units (ReLU), fully connected (FC) and softmax layers.

Conv layer is the essential element of a CNN. It exploits the feature abstraction of high-dimensional inputs such as images which are described in terms of height (H), width (W) and channel (C). To take spatial structure of data into consideration, each neuron in the Conv layer is connected to a small region of the input volume. This region is referred to as the receptive field of the neuron. Connections to the corresponding receptive fields are local in space but extend along all the channels of the input volume. This kind of local connectivity enhances the interpretation of the spatially local input patterns [43]. The weights of Conv layer are composed of a set of learnable 3D filters. Each filter has the same size of receptive field. As shown in Fig. 2.4, each channel slice of an output feature map (ofmap) is obtained by convolving the width and height of an input feature map (ifmap) with the filter. The dot products between entries of each filter and windowed ifmap at each stride are computed and summed to produce the output value at each point of the channel slice of the ofmap. Spatial dimensions of ofmap are thereby determined by those of ifmap, filter and stride length (Str). The number of channels of the ofmap is equivalent to that of the filters. Fig. 2.5 serves as a specific example to explain the operations of high-dimensional convolutions with a $4 \times 4 \times 3$ image as the input data and a $3 \times 3 \times 3$ filter. As shown in this example, the first entry value (137) of the ofmap is obtained by the sum of the three dot products: the dot product (65) between the first channel of the filter ($\begin{bmatrix} 1 & 3 & 2 \\ 4 & 5 & 1 \\ 1 & 2 & 2 \end{bmatrix}$) and the first channel of the windowed ifmap ($\begin{bmatrix} 1 & 3 & 2 \\ 4 & 5 & 1 \\ 1 & 2 & 2 \end{bmatrix}$), the dot product (30) between the second channel of the filter ($\begin{bmatrix} 2 & 5 & 3 \\ 0 & 0 & 1 \\ 1 & 2 & 1 \end{bmatrix}$) and the second channel of the windowed ifmap ($\begin{bmatrix} 0 & 3 & 1 \\ 4 & 0 & 1 \\ 1 & 4 & 2 \end{bmatrix}$), and the dot product (42) between the third channel of the filter ($\begin{bmatrix} 2 & 1 & 2 \\ 5 & 1 & 0 \\ 1 & 1 & 3 \end{bmatrix}$) and the third channel of the windowed ifmap ($\begin{bmatrix} 1 & 5 & 1 \\ 4 & 4 & 1 \\ 1 & 5 & 1 \end{bmatrix}$). The abovementioned process can also be interpreted as the sum of the element-wise products between the filter and the windowed ifmap. With $Str = 1$, the remaining windowed ifmaps are determined and similar operations are performed to obtain the corresponding entry values of the ofmap. The ofmap in this case has a dimensionality of $2 \times 2 \times 1$. It can be seen that the same set of weights is used to obtain every output in the same channel slice of the ofmap. This weight-sharing scheme significantly alleviates the storage requirements of the weights.

Pooling layer is periodically introduced in-between the successive Conv layers to down-sample the processed data. It not only reduces the amount of computations and parameters, but also contributes to the translational invariance of the network [44]. Pooling is applied to each channel separately. Only the spatial dimension will be changed after pooling. The length of the channel will remain unchanged. There are several pooling techniques such as max pooling and average pooling. As shown in Fig. 2.6, pooling divides a channel slice of the ifmap into non-overlapping blocks and outputs the maximum or average value of each block accordingly.

Fig. 2.4 High-dimensional convolution operation

Fig. 2.5 An example of high-dimensional convolutions

With $Str = 2$ and a receptive field of size 2×2 , pooling decreases the input dimension from 4×4 to 2×2 .

ReLU layer introduces non-linearity into the network by applying activation function $f(x) = \max(0, x)$ to each entry of the feature map (fmap) [3]. It is normally inserted after each Conv or FC layer. The dimension of the fmap will not be changed by this operation. Although there are many other non-linear activation functions, such as sigmoid and hyperbolic tangent, ReLU is often preferred due to its simplicity and accelerated training without significantly affecting the generalization accuracy [3].

2×2 pooling, stride 2

10	13	65	18
10	21	32	21
16	17	21	3
8	23	25	43

Max pooling

21	65
23	43

Average pooling

16	34
16	23

Fig. 2.6 Maximum and average pooling operations

FC layer connects all neurons in the previous layer. High-level reasoning is performed in the FC layers.

Softmax layer is normally the last layer of a CNN that takes the real number outputs of the last neuron layer, called logits (logit is an abbreviation for **logistic unit**), as inputs and normalizes them into a probability distribution. It is defined as follows [45]:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K \quad (2.2)$$

where K is the number of neurons in the softmax layer, z is a K -dimensional vector of the logits, and σ is the K -dimensional output.

2.1.3 Popular DNN Architectures

Most of the deep convolutional structures adhere to the prime configuration of five convolutional (Conv) sections and one to several FC blocks, as illustrated in Fig. 2.7. Each Conv section can contain several Conv blocks. Feature extraction is accomplished by Conv sections, where the spatial dimension of ifmap is reduced and the depth is extended. Each FC block is composed of a FC layer and a ReLU layer. Classification is performed in the FC blocks. Conventionally, each Conv block consists of a Conv layer, a ReLU layer and optionally a normalization layer and a pooling layer. Examples are AlexNet [3] and VGGNets [46]. The former has only one Conv block for each Conv section and the first two blocks use 11×11 and 5×5 kernels, while the rest of the blocks use 3×3 kernel. The latter cascades multiple Conv blocks in the same section and uses 3×3 kernel throughout.

Fig. 2.8 shows some variants of Conv blocks. Inception [5] provides a distinct block formation. It executes multiple convolutions with diverse filter sizes in parallel. The resulting ofmaps are concatenated along the channel dimension before the outputs are forwarded to the

Fig. 2.7 Conventional configuration of DNN.

next block. An extra 1×1 kernel is often utilized before a larger convolution for dimension compression. The adoption of inception module enables the model to choose the best type of convolution for the Conv layer without human intervention. The model can recover both the local features via smaller convolutions and the high abstracted features via larger convolutions. ResNets [47] introduce shortcuts among Conv blocks to mitigate the gradient vanishing issue in the training. Merely increasing the network depth will result in infinitesimal gradient when it is back-propagated to the earlier layers. Hence, the performance of the model can easily be saturated or demean abruptly. The presence of identity connections enables the layers to be skipped to learn a residual mapping. A 1×1 convolution is also applied to reduce the number of weight parameters. DenseNets [48] follow this strategy by outspreading the identity connections to all the subsequent layers.

Unlike other networks that have five Conv sections, MobileNets [49] has six Conv sections. They are built based on depth-wise separable convolutions that factorize a standard convolution into a depth-wise and a point-wise convolution. The former uses a single filter to convolve with each input channel. The results are then combined by the latter using a 1×1 convolution. MobileNets are dedicated for mobile and embedded vision applications with limited resources.

2.2 DNN Hardware Acceleration

Computational complexity of DNN increases the difficulty in deploying deep learning (DL) in portable and smart wearable devices. Since the most essential computational component of a DNN is the multiply-and-accumulate (MAC) operation, improving its efficiency has become an inevitable approach to increase the energy-efficiency and reduce the latency of DNN. MAC

Fig. 2.8 Different variants of convolutional block design.

operations can be sped up by parallel processing. This can be achieved by temporal and spatial architectures as shown in Fig. 2.9. The temporal architecture, also known as single instruction multiple thread (SIMT) or single instruction multiple data (SIMD), is usually adopted in central processing units (CPUs) or graphical processing units (GPUs). Its control and memory are shared by numerous arithmetic logic units (ALUs), each of which works independently without inter-communication. In contrast, spatial architecture allows data transfer between adjacent ALUs and possesses private control logic and local memory (register file) for each ALU. Such ALU is known as processing element (PE). Spatial architecture is commonly realized in application-specific integrated circuit (ASIC) designs. Fig. 2.10 shows a classic architecture of the DNN accelerator. MAC operations are executed on the PE, while operations such as ReLU, normalization and pooling are carried out on SIMT engine. A host processor is used for scheduling the DNN operations. The data controller schedules the dataflow to save off-chip memory accesses. A separate high frequency clock is employed for the PE array to increase the throughput. A clock gating signal (*en_gating*) can be introduced to further reduce the power consumption by disabling the clock signal to the PE array in the idle mode [50].

To speed up the kernel computation on CPUs and GPU platforms, data transformation is applied to reduce the number of multiplications, often at the expense of increasing the number of additions with irregular data access patterns. A typical data transformation example is fast Fourier transform (FFT) [51, 52]. By taking the FFT of the filter and ifmap, multiplications are carried out in the frequency domain. The inverse FFT of the result is finally calculated to

Fig. 2.9 Parallel processing schemes [1].

obtain the ofmap in spatial domain. On the other hand, dataflow processing leverages memory hierarchy to alleviate the memory wall problem. Multiple levels of local memory hierarchy with various energy costs are introduced to offer low-cost data movements [14]. The efficiency is further enhanced by three forms of data reuse: convolution, feature map and filter. Convolution reuse refers to the use of the same ifmap and filter, fmap reuse refers to the use of the same ifmap across different filters and filter reuse represents multiple implementations of the same filter for diverse ifmaps (batch processing). As a result, relevant data can be stored locally and accessed several times to save off-chip memory visits. The corresponding dataflow strategies are categorized into weight stationary, output stationary, no local reuse and row stationary [53, 54, 1, 12, 13, 55].

In addition to direct hardware optimization, co-design between the network model accuracy and hardware performance optimization takes implementation complexity into consideration while optimizing the accuracy for better adoption in existing hardware platforms of practical applications [1]. Precision reduction is one popular approach. Generally, 32-bit floating point is adopted on CPU and GPU platforms. It has a wide dynamic range that favors accuracy but consumes more energy and area. Recent works [56] have inspected the effect of reduced precision on energy and area costs. It was found that the energy and area consumptions scale nearly linearly and quadratically with the number of bits for a fixed-point adder and multiplier, respectively. In the emerging binarized neural network (BNN) [57, 58], the precision of the weights or activations has been reduced into only one bit to realize ultra-lightweight DNN

Fig. 2.10 Typical architecture of DNN accelerator.

implementation while maintaining the similar prediction performance as the full precision models. BNN based hardware accelerators have also been explored in [59].

Another approach is to decrease the number of operations and model size. This can be achieved via activation statistics utilization, network pruning, architecture enhancement or knowledge distillation. Since ReLU zeros negative values in the ofmaps, the resulting activation is sparsely distributed. By taking advantage of these intrinsic characteristics, the costly external memory accesses can be reduced by compression through run length coding [60]. The sparsity can be further increased by clipping insignificant activations. Moreover, memory readings and MAC operations on zero-valued activations can also be eliminated in hardware design. Besides, a large number of weights are found to be redundant and can be pruned with negligible impact on accuracy. The accuracy loss after network pruning is commonly compensated by fine-tuning. The number of parameters and operations can also be constrained using a compact network architecture through convolution factorization [61]. As shown in Fig. 2.11, a 5×5 filter can be replaced with either two 3×3 filters or a 1×5 filter and a 5×1 filter. The number of weights is thus reduced from 25 to 18 and 10, respectively. Last but not the least, establishing a simple student model based on the knowledge transferred by an ensemble of complex teacher nets could also achieve good performance while shrinking the model size [62, 63].

Fig. 2.11 Decomposition of a 5×5 convolution to (a) two 3×3 convolutions or (b) a 1×5 and a 5×1 convolution [1].

2.3 Related Works

2.3.1 Misprediction Attack

Basic Attack Model

Barreno et al. [64] have provided the following attributes to define a space of misprediction attacks that maliciously alter prediction results of ML systems. Several typical scenarios are then given to facilitate the understanding.

Attack taxonomy

Influence

Causative - Causative attacks alter the training process through influence over the training data.

Exploratory - Exploratory attacks do not alter the training process but use other techniques, such as probing the learner or offline analysis, to discover information.

Specificity

Targeted - The specificity of an attack is a continuous spectrum. At the targeted end, the focus of the attack is on a particular point or a small set of points.

Indiscriminate - At the indiscriminate end, the adversary has a more flexible goal that involves a very general class of points, such as ‘any false negative’.

Security violation

Integrity - An integrity attack results in intrusion points being classified as normal (false negatives).

Availability - An availability attack is a broader class of attack than an integrity attack. An availability attack results in so many classification errors, both false negatives and false positives, that the system becomes effectively unusable.

Example 1 (Causative integrity attack) *In an intrusion detection system (IDS), the defender periodically retrains the model with new traffic. The adversary contaminates the information flow by carefully injecting some abnormalities that resemble legitimate instances to mislead the learner, resulting in no flagging for an intrusion.*

Example 2 (Causative availability attack) *In an email spam filter system, the attacker sends harmful messages including both malicious words and benign parts, when the defender is collecting data for training. Since the legitimate components are mistakenly correlated with spam, the system is likely to block a similar normal email next time.*

Example 3 (Exploratory integrity attack) *In a face recognition system for authorization, the adversary perturbs the captured sensor data during the inference to grant accesses to unauthorized users.*

Example 4 (Exploratory availability attack) *In an image classification system, the attacker injects noise to the model during the inference to corrupt the classification ability of the model.*

Input Evasion Attack

Input evasion attack is one of the existing popular misprediction attacks. It distorts the input data during the inference phase so that the DNN model fails to identify the input. This attack falls into the exploratory category of the taxonomy. The input can be either digitally or physically modified. One threat that is extensively studied in academia for digital perturbation is adversarial examples, which are well crafted inputs obtained by adding imperceptibly small perturbations to the original benign inputs. It is first demonstrated by Szegedy et al. [2], who used box-constrained limited-memory-Broyden–Fletcher–Goldfarb–Shanno algorithms to craft malicious samples, as shown in Fig. 2.12. Five representatives of adversarial image generation methodologies are elaborated subsequently. In the following, x , $\hat{x}$, F and t are used to denote a clean input sample, an adversarial sample, a DNN model and a target class of the adversarial input, respectively.

Fig. 2.12 Illustration of adversarial examples generated using [2] for AlexNet [3]. The perturbations are magnified $10\times$ for better visualization. The predicted classes for adversarial images are also shown in red texts.

Fast Gradient Sign Method (FGSM): FGSM calculates the gradient direction of the cost function for each pixel of an input image to determine the direction of added distortions with minimum cost. The computation can be expressed mathematically as follows [17]:

$$\hat{x} = x - \varepsilon \cdot \text{sign}(\nabla J_{F,t}(x)) \quad (2.3)$$

where J is the cost function and ε is an input variation parameter that controls the amplitude of perturbation. The cost function J evaluates the error between the output of F and the target label t . ε should be set sufficiently small to make the perturbation unnoticeable. FGSM is a

one-shot attack that aims to be fast rather than optimal. Its variant, iterative FGSM (I-FGSM or BIM), applies FGSM recursively with a smaller step size to improve the success rate of an attack [65].

Jacobian-based Saliency Map Attack (JSMA): JSMA [66] adopts an L_0 -norm distance metric to find adversarial examples. Unlike FGSM that changes all the pixels, JSMA restricts perturbations to a limited number of input features. It utilizes the forward derivatives of the network with respect to the input to construct a saliency map that reflects the impact of each input component on the target classification. The most significant pixel is then selected and modified to increase the likelihood of target class until the threshold of being undetectable is reached or misclassification is achieved. Given

$$\begin{cases} a_i = \frac{\partial F_t(x)}{\partial x_i} \\ b_i = \sum_{k \neq t} \frac{\partial F_k(x)}{\partial x_i} \end{cases}, \quad (2.4)$$

the saliency map constructed in each round is expressed as follows:

$$Smap(x, t)_i = \begin{cases} a_i \times |b_i| & \text{if } a_i > 0 \text{ and } b_i < 0 \\ 0 & \text{otherwise} \end{cases}, \quad (2.5)$$

where i and k represent the input pixel and any object class other than t , respectively. It is difficult to obtain the adversarial examples by selecting only one pixel at each iteration. Instead, a candidate pair of pixels (p_1, p_2) are collected each time using the following heuristic:

$$\arg \max_{(p_1, p_2)} \left(\frac{\partial F_t(x)}{\partial x_{p_1}} + \frac{\partial F_t(x)}{\partial x_{p_2}} \right) \times \left| \sum_{i=p_1, p_2} \sum_{k \neq t} \frac{\partial F_k(x)}{\partial x_i} \right| \quad (2.6)$$

Although JSMA is capable of generating adversarial examples with the minimum disturbance to fool a classifier, the search for a suitable pair of input features for each repetition is time consuming and requires a lot of memory. Due to this constraint, JSMA has been demonstrated on only small datasets like Modified National Institute of Standards and Technology (MNIST) database, which is a dataset of handwritten digits.

C&W: C&W attack [67], named after Nicholas Carlini and David Wagner, translates the initial formulation of the problem [2]:

$$\text{minimize } D(x, \hat{x}), \text{ such that } F(\hat{x}) = t \wedge \hat{x} \in [0, 1]^n, \quad (2.7)$$

to an easier optimization problem:

$$\text{minimize } \|\delta\|_p + c \cdot f(\hat{x}, t), \text{ such that } \hat{x} \in [0, 1]^n, \quad (2.8)$$

where D is equivalent to the distance metric $\|\cdot\|_p$. δ and c represent the perturbation and a carefully chosen constant, respectively. $\hat{x} = x + \delta$. f is defined as an objective function such that $F(\hat{x}) = t$ if and only if $f(\hat{x}, t) \leq 0$. Since the non-linearity of the constraint $F(\hat{x}) = t$ increases the difficulty of solving this optimization problem directly, the introduction of f eases the problem. The best f is found to be

$$f(\hat{x}, t) = \max(\max\{G(\hat{x})_k : k \neq t\} - G(\hat{x})_t, 0), \quad (2.9)$$

where $G(\hat{x})$ is the logit vector of a classifier. To bypass the box constraint $\hat{x} \in [0, 1]^n$ in the expression (2.8), δ is replaced with a new variable r such that

$$\delta = 0.5(\tanh(r) + 1) - x. \quad (2.10)$$

Since $-1 \leq \tanh(r) \leq 1$, $\hat{x}$ unconditionally lies in the boundaries. Therefore, the final problem definition is given by:

$$\arg \min_r \|\delta\|_p + c \cdot \max(\max\{G(0.5(\tanh(r) + 1))_k : k \neq t\} - G(0.5(\tanh(r) + 1))_t, 0). \quad (2.11)$$

All the L_2 -norm, L_∞ -norm and L_0 -norm versions of C&W attacks have been proposed and validated to be more effective than earlier methods for the generation of adversarial examples.

DeepFool: Unlike the three previously discussed algorithms, DeepFool [68] is an indiscriminate attack optimized for L_2 -norm distance metric. It seeks to explore the minimum perturbations required to cross the decision boundary. It simplifies the DNN model to an entirely linear structure where the classes are separated by hyperplanes. The problem is then formulated into computing the distance between the input point and the closest hyperplane. To generalize the solution to the non-linear classifier, DeepFool approximates the boundary of the initially correct class label of the point of interest iteratively by borrowing the idea of Newton's iterative algorithm.

Universal adversarial perturbations: Moosavi-Dezfooli et al. [68] extended DeepFool to a generalised image-agnostic adversarial perturbation. It can be mathematically expressed as

follows [4]:

$$\delta := \left\{ \delta \mid Err(X_\delta) := \frac{1}{m} \sum_{i=1}^m 1_{F(x_i+\delta) \neq F(x_i)} \geq 1 - acc \right\}, \text{ s.t. } \|\delta\|_p \leq \xi. \quad (2.12)$$

where X represents a set of natural images $\{x_1, \dots, x_m\}$ collected from a population, δ is a fixed perturbation vector that leads to the misclassification of most samples in X , X_δ defines a set of perturbed data points $\{x_1 + \delta, \dots, x_m + \delta\}$, acc quantifies the accuracy of the tampered samples, and ξ constrains the extent of distortion δ . This algorithm iteratively feeds the benign inputs by adding previously derived universal perturbation into the classifier. It calculates the minimum distortion that sends the current tampered input to the decision boundary in each iteration. The instance of universal aberration is updated by adding the computed distortion and then used it for the next round. As can be observed in Fig. 2.13, these universal perturbations remain quasi-imperceptible to human vision systems.

Fig. 2.13 Illustration of adversarial examples generated using universal perturbations[4] for AlexNet [3], and GoogLeNet [5].

All the above discussed methods rely on the assumptions that the adversary has full access to the target DNN model, including the architecture and parameters. Hence, they are all white-box techniques. The later method [18] exploits the transferability of adversarial examples among different classifiers for black-box attack. Although the hypothesis of such attack seems

to be realistic, Su et al. [69] discovered that the strength of transferability varies largely across models and attacking algorithms. In a black-box setting, the output of the model corresponding to an input image is used to train a substitute model that intends to imitate the mapping of the source model. The adversarial generation process is then made with respect to the replacement. FGSM and JSMA crafting algorithms have been evaluated on small-scale MNIST and GTSRB (the German Traffic Sign Recognition Benchmark) datasets for black-box attack in [18]. An average success rate of more than 80 percent is achieved. However, the performance is reversed when employing a more complicated dataset like ImageNet. In [69], the authors addressed the transferability through comprehensive ImageNet classification experiments conducted on every 2 of the 18 networks from 7 families: AlexNet [3], VGGNets [46], Inception nets [5], ResNets [47], DenseNets [48], MobileNets [49] and NASNets [70]. Proportion of adversarial images generated from one model that are wrongly perceived by another model in the same target class corresponding to the targeted transferability and the indiscriminate transferability without class constraints are separately analyzed. The former can be referred to as matching rate, whereas the latter is equivalent to the error rate. Several important findings on transferability by [69] are listed below.

- The adversarial images possess considerably higher indiscriminate transferability than targeted transferability.
- FGSM and BIM yield much higher untargeted transferability than C&W, while their adversarial generation success rates are worse than C&W. The generation and transfer success rates are ordered as follows:

$$\begin{aligned} \text{Generation success rate : } FGSM &\leq BIM \leq C\&W \\ \text{Transfer success rate : } FGSM &\geq BIM \geq C\&W \end{aligned}$$

Furthermore, a larger ϵ for FGSM and a lesser iteration for BIM are more beneficial to the indiscriminate transferability.

- Transferability is not necessarily symmetrical. In other words, good transferability of adversarial examples of one model to another model does not imply the same quality of transferability in the reverse.
- The adversarial images crafted using VGGNets are empirically better in transferability to other network families. This could be attributed to their basic architectures and simple convolution operations, which form the baseline of all other advanced models.
- The transfer rate is close to 100 percent when the two ends are from the same network family, but much lower when their architectures are different. This could be exploited as the

mechanism for reverse-engineering the inner structure of an unrevealed model by supplying tampered samples generated from a known network and studying the transferability.

For deep learning (DL) system implemented on CPU or GPU platforms, 32-bit floating point representation is commonly used. Therefore, the adversarial perturbation generated using the abovementioned attacks is calculated in 32-bit precision. However, the intensity of each component of a color image pixel is represented in an 8-bit scale, with an integer value ranging from 0 to 255. Hence, precision loss that undermines the distortion efficiency is inevitable upon converting a 32-bit adversarial data array into 8-bit image format. Particularly for FGSM, BIM, C&W and DeepFool attacks that inflict infinitesimal changes for manipulation, their failure rates have increased tremendously. Most derived deformations, which are smaller than the image precision, are eliminated during the transformation. The effectiveness of these attacks are expected to be further declined when they are mounted on a custom integrated DNN system that utilizes quantization technique to save energy and area.

Countermeasures Against Input Evasion

Countermeasures against input evasion attacks can be dichotomized into reactive and proactive categories. The former intends to restore the correct prediction of a malicious input without sacrificing the classification accuracy of the benign samples, whereas the latter aims to detect the adversarial inputs. Reactive measures are mainly carried out by three methods: adversarial training, gradient masking and input transformation. They are elucidated next.

Adversarial training: Adversarial training retrains the model with the addition of off-the-shelf adversarial examples into the original training dataset. The method indeed improves model resilience against used adversarial algorithms but remains vulnerable for unused ones. This non-adaptivity implies that the strength of this defense is determined by the number of the distinct attack algorithms involved in adversarial generation process. In addition to the significantly high computational costs in crafting malicious images by most known techniques, it is also unrealistic to assume that the attacker is restricted to techniques that are known to the defender. The doubling of training cost has greatly reduced the efficiency of this countermeasure.

Gradient masking: Since many attacks are based on the extraction of gradient information, hiding gradient from adversaries seems to be a feasible defense. Papernot et al. [19] proposed a defensive distillation method to strengthen DNN models. This technique extracts probability vectors of a pretrained model as soft labels to build a distilled model with the same architecture. It is originated from the hypothesis that encoding probability vectors that contain correlation information among classes can improve the generalization ability of the model. Furthermore, a smoother softmax layer is designed by adjusting the temperature parameter for gradient concealment. The experiments showed that the success rates of some attacks on MNIST

and CIFAR-10 datasets have been reduced significantly without compromising accuracies. However, this countermeasure was later broken by C&W and black-box attacks.

Input transformation: A few studies have demonstrated the effectiveness of data transformation against evasion attacks by inhibiting the adversarial effect. Bhagoji et al. [23] introduced linear dimensionality reduction methods, such as principal component analysis (PCA), to project original data along the identified “principal axes” (i.e., directions that preserve the most crucial information of the data) to the training data. The same transformation procedure is repeated in inferencing before feeding the inputs to the model. However, since PCA corrupts the spatial structure of the input, local information needed by the convolutional layers of modern DNNs for classification is not fully preserved in the transformed space. A redesign of the model architecture is required. Instead of building a fresh model specialized for the projected inputs, Meng and Chen [71] presented MagNet, a framework that neither modifies model internals nor requires adversarial creation process. It reconstructs the inputs beforehand using autoencoders, trained with sufficient clean examples, and is expected to move tampered images toward the distribution of legitimate ones. Since the output dimension of an autoencoder is the same as that of the input, spatial structure is not obliterated. MagNet is theoretically applicable to various classifiers. Besides, several feature squeezing strategies [72, 20], such as binary filter, spatial smoothing and color depth reduction, have been suggested to shrink the feature space of the input to increase the difficulty of adversarial manipulation.

One common experimental blind spot shared by most existing reactive defenses is that the adversarial examples for evaluation are generated based on the original model instead of the model with the defense. Hence, most previous reactive countermeasures are validated in the grey-box rather than white-box setting. However, as a reactive defense still outputs a prediction for an adversarial image, the attacker could still build a substitute model that takes the defensive effect into account even without knowing the exact employed defend strategy. Due to the transferability of the network, adversarial examples crafted using the substitute model could mostly be effective on the defended model. Therefore, even if the adversary does not know the defense, he can still generate valid adversarial examples to successfully fool the defend system, let alone the white-box setting. Carlini et al. [73] have proved that a slight increase in the adversarial perturbation strength in the white-box setting can bypass MagNet. Furthermore, reactive defenses can only passively restore the classification of adversarial examples without knowing any information about the attacks. In contrast, proactive methods, with the capability of detecting attacks, have profound implications on follow-up actions toward adversaries. Existing proactive methods can be divided into three main types: sample statistics, detector training and prediction inconsistency.

Sample statistics: Sample statistics detect the difference between adversarial and normal samples by statistically analyzing the data. Feinman et al. [24] discovered two features, density and Bayesian uncertainty estimates, to detect illegitimate points lying far from the natural data manifold and nearby, respectively. Density estimates, specifying the distance between a given input and sub-manifolds, are calculated using the activations of the last hidden layer, while Bayesian uncertainty estimates are extracted directly from the dropout layer. It can be observed that the computational cost increases substantially with the model complexity and the size of the dataset. Liu et al. [74] proposed to detect adversarial examples by steganalysis. The steganalysis features are enhanced by estimating the probability of modifications caused by adversarial attacks. However, this method requires the knowledge of the attack algorithms. A different detection process will have to be carried for each attack algorithm. As the adversarial example generation algorithm is not known in advance, brute-force execution of all the different processes is required to detect an unknown adversarial input. This attack-dependent defense achieves very high detection rate at the expense of impractically high computational resource and timing overheads. Rouhani et al. [7] proposed DeepFense, the first end-to-end hardware accelerated framework for real-time detection of adversarial examples. It introduces Modular Robust Redundancy (MRR) to characterize the potential adversarial sub-spaces spanned by the output feature maps in each layer of the target DNN model. An automatic customization tool has also been designed for the maximization of the performance in various resource-constrained hardware platforms. However, as each MRR is architecturally identical to the victim DL model, it incurs substantial hardware and power overheads, especially when the model size and number of MRRs become larger. Furthermore, DeepFense was evaluated with only small-scale datasets. Its scalability on large-scale datasets like ImageNet is untestified.

Detector training: Metzen et al. [21] devised the augmentation of subnetworks as a detector to the deep net. This additional module is devoted to binary classification between clean and adversarial inputs. It is trained while freezing the parameters of the original model. Hence, the accuracy is not sacrificed. In contrast, Grosse et al. [25] only revised the last layer with an extra class of “adversary”, resulting in some accuracy loss on natural samples. Both defenses require massive adversarial examples for training, which is costly. Besides, they are also prone to over-generalization of adversarial attacks.

Prediction inconsistency: Prediction inconsistency uses the degree of consensus among multiple models in predictions as an indicator. Wang et al. [75] integrated mutation testing of DL systems [76] and statistic hypothesis testing into the detection algorithm. They found that adversarial images are more sensitive to the model mutants than clean images. Model mutants are generated by injecting faults to the trained DNN model. They are commonly used to test the quality of input data. In [75], label change rate (LCR) is used to quantify the mutation

sensitivity of an image input and is defined as follows:

$$\zeta(x) = \frac{|\{f_i | f_i \in F \cap f_i(x) \neq f(x)\}|}{|F|}, \quad (2.13)$$

where x denotes an input image, f is the original model, F is a set of mutated models $\{f_1, \dots, f_m\}$ and $|F|$ is the number of elements in F . LCR is discovered to be significantly higher for a tampered input than that of a clean one against a number of DNN mutants. Therefore, statistic hypothesis testing is applied to determine the truthfulness of the input identity in terms of LCR. The strategy has only been implemented on small-scale datasets. Its feasibility and efficiency on complex database are untestified. The variant of prediction inconsistency methodology that compares the predictions of original image and the transformed image produced by input transformation techniques for the same model, instead of the predictions of different models for the same image, has also been explored to decrease either the latency or power consumption incurred by the original strategy [20, 77].

In addition to the aforementioned defenses that target on detecting adversarial examples, Wang et al. [8] devised a new DNN accelerator architecture, DNNGuard, for low-latency, secure and energy-efficient implementation of the existing detection methods. DNNGuard supports simultaneous execution of target DNN network and its corresponding detection network. Nevertheless, DNNGuard was implemented on a simulated instead of a real deep learning accelerator. The actual impact of performance penalty and hardware overheads on adaptability has not been evaluated with DNNGuard integrated online into the physical DNN accelerator.

2.3.2 Confidentiality Attacks

Incentive

To train a DNN model, enormous sample collection, human power and computational resources are required. While they may be adequately provided in large companies, they are more difficult to be satisfied for small corporations. For business competitiveness, the trained model is seldom released to the public. Therefore, it is likely for the business opponents to steal the model to build their own AI products or solutions to subvert the market dominance of the original model owner. Furthermore, the establishment of a DL model is inseparable from the data. In many cases, the data are sensitive and likely to be eavesdropped for malpractice.

DNN Intellectual Property (IP) Theft Attacks

DNN IP theft attacks are also referred to as model confidentiality attacks. They can be achieved by analyzing input, output and other extrinsic knowledge of a system to extract the model information, including the architecture and weights. Existing DNN IP theft attacks can be broadly divided into two categories: query-based and implementation-based. Query-based IP theft attack mainly utilizes the input-output relationship of the target model to build a substitute model that has the similar functionality [28]. Alternatively, implementation-based IP theft attack exploits side channel leakage during model execution. Recent studies have shown that memory visits can be observed through physical exploitation of the data bus or side channels such as cache conflicts and page faults. Feature maps and parameters are typically stored in off-chip memory to ease the on-chip storage requirement, and they are visited when needed in the DNN accelerator implementation [78, 79]. Hua et al. [80] leveraged this vulnerability of memory trace that leaks layer information as well as parameters to construct a similar DNN model that achieves comparable accuracy as the original one. The reverse-engineering process has been evaluated on an accelerator integrated with the modern processor's security capabilities, including prohibition of direct access and tampering to the inner operations, and encryption of feature maps and weights during the memory-accelerator transmission [81]. However, the types of instruction and address are still visible for each external memory visit when submitting the input. Layer boundaries can be determined by read-after-write dependency since the start of a new layer is often accompanied by the read operation on a memory address that has already been written to previously. Once the borders are resolved, the size of the filters, input and output feature maps for each layer can be further inferred by observing their own memory locations within a layer, since each category of data is stored contiguously. Then, the data type can be identified through inspection of the memory instruction. Weights are often read-only during inferencing, whereas the ofmap is often write-only within a layer. To approach the correct solution, the execution time for each layer is recorded and compared with the expected number of MAC operations for each possible configuration. Since the operating clock cycles should be approximately proportional to the number of MAC operations, candidate structures that violate this rule can be excluded. The authors also proposed reverse-engineering of weights by taking advantage of zero pruning technique that is often applied to reduce the computational complexity and memory access intensity in the DNN accelerator [60, 82, 83]. Zero pruning, that only captures nonzero values, leaks information about the number of zeros in the ofmap. By gradually changing the values of interested input pixels and observing the respective changes in ofmap, zero-crossing points can be determined, and a set of linear equations with respect to the weights and bias can be obtained. In other words, each weight can be expressed as a function of the bias. The exact values of parameters can be determined

by using a tunable activation function instead of ReLU to remove smaller pixel values below a threshold [60, 83]. The bias is thereby obtained by zeroing all the input pixels and adjusting the threshold. Weights can be subsequently derived based on the previously extracted relationship with the bias. However, the process is founded on the assumption that the threshold can be acquired. It also requires the target DNN hardware to have zero-pruning technique.

Countermeasures Against DNN IP Theft

One intuitive countermeasure against DNN IP theft attack is watermarking. Protection of DNN IP by watermarking is first introduced by Uchida et al. [33]. They embedded a multiple-bit vector to the parameters of the DL model through adding a regularizer to the original cost function. Different from standard regularizers that are commonly used to mitigate the over-fitting issue, their regularizer aims to impose a certain statistical bias served as a watermark to the model parameters in the training process. Later, Chen et al. [84] proposed DeepMarks that embeds a user-specific code-vector in the probability density distribution of the model weights. Not only could it identify the pirated version of the model but also track authorized users who use the model for unintended purposes. However, such watermark techniques assume that the internal details of the suspicious model are accessible to the model owner and the entire watermark can be extracted by the verifier. The bit error is calculated by comparing the extracted watermark and the embedded one during the ownership verification phase. Such white-box setting has restricted the range of application for the abovementioned methods. In contrast, the black-box watermarking only assumes access to the input and output of the suspicious DNN model. It embeds the watermark by training the target model with some additional deliberately crafted samples known as trigger set. During verification, the trigger set will be fed to the suspicious model. If the prediction is largely matched with that from the target model, the suspicious model is likely to be pirated from the target model. Otherwise, the suspicious model is likely to be an original model. According to the trigger set used, the existing black-box watermarking can be further dichotomized into perturbed trigger set and clean trigger set based watermarking. The first category embeds a readable logo or noise pattern into the normal training images and assigns a pre-defined label to the crafted images [35, 85]. Unlike [35, 85], Li et al. [86] used an autoencoder to generate the trigger set from the ordinary samples and an exclusive logo. Such trigger set are almost indistinguishable from the original samples. A specific label is also assigned to the trigger set to infuse the watermark into the target model. The second category embeds the watermark by assigning unique labels to a set of clean samples. Zhang et al. [35] have utilized one kind of trigger set sampled from unrelated classes and annotated them with a predetermined wrong label. Adi et al. [34] sampled some abstract images that are uncorrelated to each other. Each abstract image is assigned with a

random label so that this random label is very unlikely to match the output prediction from a watermark-free model.

No matter what trigger set is used by the existing black-box DNN watermarking schemes, the original decision boundary will be twisted to some extent due to their interference with the model training process. The classification accuracy is inevitably affected. For this reason, Cao et al. [37] firstly introduced fingerprinting method for DL model ownership verification without intervening the model training process. Fingerprinting is different from watermarking in that no signature is embedded in the training stage. Instead, it extracts robust data inputs that have similar predictions as the target model when they are fed into a pirated or post-processed version of the target model, but different for an originally designed and trained model that is not pirated from the target model. This set of robust inputs and their labels predicted by the target model form the fingerprint of the target model. Different ways of efficient fingerprint extraction have been proposed to maximize the difference between a pirated model and an originally designed model. Cao et al. [37] devised IPGuard that makes use of some data points near the target model's decision boundary and their labels predicted by the target model to fingerprint the target model, assuming that the DNN model can be characterized by its decision boundary. Lukas et al. [38] exploited the targeted transferability of the basic iterative method (BIM), which is an adversarial input generation algorithm, to fingerprint the target DNN. Their work aims to extract a set of targeted transferable adversarial examples that are transferable to merely models pirated from the target model, but not to any independently trained model. Such adversarial examples are claimed to be conferrable adversarial examples. Consequently, they revised BIM as conferrable BIM (C-BIM) encoded with the aforementioned conferrability to generate conferrable adversarial examples. Both IPGuard and C-BIM omit the effect of different fingerprints extracted using the same mechanism on the IP theft detection efficiency. They incur a comparatively large latency overhead in fingerprint extraction.

2.4 Summary

In this chapter, some background information about DNN is introduced. A comprehensive review on the representative adversarial sample generation algorithms and DNN IP theft techniques, and corresponding countermeasures have also been made. DNN are particularly suited to solve computer vision related problems. The integration of feature extraction and prediction functionalities of DNN enable it to handle sophisticated or data-driven tasks like a duck to water in the big data era. The study also reveals the intrinsic vulnerability of DNN. Skilfully crafted adversarial examples, and side channel information like prediction API, memory access patterns and cache conflicts can be exploited to pose severe safety,

security and privacy hazards. Unfortunately, existing defenses against adversarial inputs are either not scalable, impractical or limited by their specificity. The large code sizes and high computational complexity prevent these countermeasures from efficiently implemented or integrated into edge devices. AI on a chip is a roadmap for embedded intelligence everywhere. Furthermore, existing fingerprinting approaches towards DNN IP theft protection have omitted the repudiability and extraction costs of the fingerprint. To address these limitations, two hardware-oriented adversarial input detection strategies (Chapter 3), a new lightweight in-situ adversarial input detector for edge DL (Chapter 4) and a fast and non-repudialility-enhanced fingerprinting methodology for IP theft detection (Chapter 5) are developed. They will be presented in the following chapters.

Chapter 3

Towards DNN Hardware Accelerator Friendly Detection of Adversarial Examples

3.1 Introduction

Deep neural network (DNN) is a prevalent machine learning solution to computer vision problems. As evident by the review presented in the previous chapter, the most criticized vulnerability of deep learning (DL) is its susceptibility towards adversarial images crafted by maliciously adding infinitesimal distortions to the benign inputs. Such negatives can fool a classifier. Existing countermeasures against these adversarial attacks are mainly developed based on software model of DNNs by using modified training during learning or modified input during testing, modifying networks or changing loss/activation functions, or relying on add-on models for classifying unseen examples. These approaches are mostly designed and investigated from software perspective or are evaluated using small scale datasets. The effectiveness of these attacks and defenses that are based primarily on software DNN models may be severely undermined in practice when the same input data are forwarded through a hardware accelerator friendly computational model due to, for instance, the incorporation of input truncation and fixed-point arithmetic.

To bridge the gap, two defenses to protect large-scale hardware-oriented DL application are proposed in this chapter. One method is directed by the high sensitivity of few individual layers to weight perturbation by noise injection. It is driven by the observations that sensitivity differences between adversarial and benign samples vary across layers when random noises are injected onto their respective weight parameters. This suggests the existence of biasing in layers

towards adversarial discriminability. Instead of injecting an even perturbation across all layers, it is sufficient to subject a few dominant candidate layers to non-uniform weight distortions to maximize the discriminability of adversarial inputs while retaining the model accuracy on normal samples. Multiple single-layer noise injections are experimented to progressively tune the adaptive detection thresholds of each selected dominant layer. This approach falls into the third category of detection methods: prediction inconsistency described in Chapter 2. Another countermeasure is directed by the difference in layer-wise fired neuron rate between the harmful and benign inputs. It is inspired by the observation that there exists a disparity in the number of fired neurons between adversarial and benign samples. The gap between their statistical distributions varies across layers. This implies the presence of bias in the distribution of neuron activities in some layers, which can be exploited for adversarial distinguishability. Dominant layers with high discriminability against adversarial examples are used to successively filter out the malicious images. A decision tree is used to determine the most discriminative layer (MDL) of the trained DNN and its threshold at each level. This approach belongs to the category of sample statistics described in Chapter 2 but the detector extracted for integration into the hardware DNN is simpler than the additional module used by detector training and prediction inconsistency.

3.2 Threat Model

Both proposed methods aim to defend against a powerful adversary who has full knowledge of a custom integrated DNN classifier deployed in the field, but has no accessibility to the inner system. The adversary may attempt to generate malicious images to fool the system using white-box attack techniques. The adversary cannot physically alter the internal nodes and nets of a monolithic integrated circuit. Neither can he determine the perturbed layers and extent of perturbations introduced into the classifier by reverse engineering the circuit without leaving apparent evidence of physical tampering. Attacks, such as Fast Gradient Sign Method (FGSM) [17], C&W attack [67] and DeepFool [68], inflict infinitesimal changes, while effective on digital signal processing (DSP) algorithm and software, are infeasible in this scenario. This is because the magnitudes of the derived adversarial perturbations are mostly less than 1. Such distortions will be eliminated upon converting into an 8-bit color representation, resulting in extremely large failure rate of these attacks. Furthermore, the search space of an ImageNet image is enormous for Jacobian-based saliency map approach (JSMA) [66], which can easily render the system to run out of memory. For this reason, the most feasible and effective non-invasive attack of universal perturbation is considered in these two defenses.

3.3 Proposed Discriminative Noise Injection Method

3.3.1 Hypothesis

The proposed discriminative noise injection method is inspired by the recent work [75] that discovered distinguishable sensitivity between normal and adversarial samples when random mutations are applied on the DNN model. It is founded on the observation of the disparate degree of distinguishable sensitivity of each hidden DNN layer to noise perturbation. The hypothesis is that there exist multiple dominant layers that affect the tampered samples most under varying extent of layer mutations. To evaluate the hypothesis, noises are added into randomly selected weight parameters for every layer of the three DNN architectures. Each layer is mutated 10 times with the proportion of noisy parameters varies from 0.1% to 1%. Label change rate (LCR) is used to quantify the sensitivity of the samples on mutated models. It is defined as [75]:

$$\zeta(x) = \frac{|\{f_i | f_i \in F \cap f_i(x) \neq f(x)\}|}{|F|}. \quad (3.1)$$

where x denotes an input image, f is the original model, F is a set of mutated models $\{f_1, \dots, f_m\}$ and $|F|$ is the number of elements of F .

Table 3.1 Average ζ for normal samples and adversarial samples (unit:%).

Models	Type of samples	Noise injection rate									
		0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.01
VGG 16	Normal	1.92	3.08	4.60	6.23	8.85	11.32	14.38	16.84	20.53	24.69
	Adversarial	19.98	26.17	32.75	36.79	40.87	44.51	48.57	49.99	52.71	54.78
VGG 19	Normal	1.36	2.35	4.09	6.02	9.06	12.24	15.76	19.56	22.89	27.49
	Adversarial	21.36	26.82	32.94	37.41	41.97	45.07	48.15	51.44	53.35	56.06
MobileNet	Normal	6.23	7.91	12.64	17.98	21.41	23.36	25.92	26.82	29.42	30.15
	Adversarial	18.08	23.29	26.02	32.48	34.04	35.29	36.15	38.47	39.36	39.92

Table 3.1 summarizes the experimental results of the measured $\zeta(x)$ from ImageNet dataset. 300 randomly selected natural images and 300 adversarial images crafted using universal perturbation method are tested for each model. The noise injection rate here refers to the proportion of abused weights in respective layers under test. Since the number of layers is different across DNN architectures, the number of generated distorted models will also be different in calculating the average LCR. It can be clearly seen from Table 3.1 that ζ_{adv} for adversarial inputs is always much larger than ζ_{nor} for normal inputs at any noise injection rate for any convolutional neural network (CNN) model. This further proves that the hypothesis proposed by [75] also works for ImageNet dataset, apart from MNIST and CIFAR-10.

 Fig. 3.1 Comparison of $\Delta\zeta$ in different layers of VGG 16.

The *LCR* difference $\Delta\zeta$, computed by $(\zeta_{adv} - \zeta_{nor})$, on various layers are illustrated in Fig. 3.1 to Fig. 3.3. From Figs. 3.1 and 3.2, it is observed that $\Delta\zeta$ of convolutional layers $\Delta\zeta_{conv}$ is generally larger than $\Delta\zeta$ of fully connected layers $\Delta\zeta_{fc}$ for VGG 16 and VGG 19. This implies the stronger differentiability possessed by convolutional layers with a relatively small noise injection rate for models with no convolution factorization. The observation is also explainable since the feature extraction process is mostly done in the convolutional layers. A few changes in convolutional weights are likely to dramatically decrease the influence of adversarial perturbations. Besides, the two figures indicate a likelihood for the existence of an optimal noise injection rate for each layer. For example, $\Delta\zeta_{conv3_1}$ increases when noise injection rate is less than 0.004, but decreases thereafter in VGG 16. Thus, the optimal noise injection rate to enlarge the difference between normal and adversarial samples is estimated to be 0.004 for *conv3_1* layer. The reduced distinguishability after the optimal points indicates increased mutation effect on normal samples. $\Delta\zeta$ of fully connected layers for these two CNN architectures seem to be always monotonically increasing. However, the lack of convexity might be due to the early halt of noise injection rate.

3.3 Proposed Discriminative Noise Injection Method

Fig. 3.2 Comparison of $\Delta\zeta$ in different layers of VGG 19.

For MobileNet, $\Delta\zeta$ of some layers are always 0 with noise injection rate ranging from 0.001 to 0.01. They are not shown in Fig. 3.3 to avoid cluttering the graph. The results in Fig. 3.3 show that $\Delta\zeta$ for $conv5^*$ and $conv6^*$ layers decrease nearly monotonically in $[0.001, 0.01]$, whereas most $\Delta\zeta$ for $conv2^*$, $conv3^*$ and $conv4^*$ layers rise rapidly from 0 in a short interval of noise injection rate and then fluctuate around the plateau value. The trend for fully connected layer is similar to those of VGG 16 and VGG 19. One possible interpretation of the unusual behavior of the $conv5^*$ and $conv6^*$ layers is that their optimal noise injection rates are smaller than 0.001. This implies the high sensitivity of benign images to manipulation of $conv5^*$ and $conv6^*$ layers in MobileNet.

 Fig. 3.3 Comparison of $\Delta\zeta$ of different layers in MobileNet.

3.3.2 Detection Framework

Fig. 3.4 Framework of proposed dominant layer directed discriminative noise injection approach.

The proposed dominant layer directed approach takes advantage of few layers that have high distinguishability for successive thresholding. The threshold is adaptively applied according

3.3 Proposed Discriminative Noise Injection Method

to the input susceptibility to label change by random noise injection to the weights of these layers. The detailed detection framework is demonstrated in Fig. 3.4. $L_{i,j}-\mu_{i,j}$ denotes the j -th mutated model that is imposed with μ_i perturbation to the weight parameters of layer L_i and LCR denotes the label change rate ζ . For a given input, m mutated models are generated by changing μ_i ratio of weight parameters in layer L_i to calculate the LCR . If the LCR is greater than a predetermined threshold $\tau_{adv,i}$, the input is very likely to be deliberately distorted; If the LCR is smaller than $\tau_{nor,i}$, the input is considered to be legitimate; If the $LCR \in (\tau_{nor,i}, \tau_{adv,i})$, another m mutated models will be produced for further assessment with μ_{i+1} perturbation to the weights of layer L_{i+1} . This process goes on until the window between the normal threshold τ_{nor} and the adversarial threshold τ_{adv} for layer L_n with μ_n rate of noise injection is closing in to a single threshold τ . This process is shown in Fig. 3.4 with n successive phases of thresholding.

Fig. 3.5 Histogram distribution of LCR for legitimate and adversarial samples with 0.006 noise injection rate in layer *conv3_3* of VGG 16.

The use of a hierarchical instead of single-layer mutation in detecting adversaries is mainly due to the possibility that malicious perturbation may destroy manifolds of important features. Therefore, single-layer distortion may not capture all or majority of the differences. Fig. 3.5 depicts the histogram distribution of LCR for 300 natural samples and 300 adversarial samples with 10 mutations on layer *conv3_3* of VGG 16. Each mutation is completed by injecting noises to 0.006 proportion of weights randomly in *conv3_3*. *conv3_3-0.006* is one possible way of mutation that exhibits notable $\Delta\zeta$. Nevertheless, a large overlapping area still exists between the distributions of the tampered and benign inputs. This observation agrees with the conjecture.

3.3.3 Experimental Setup

The proposed layer discriminative noise injection strategy is implemented using 8-bit truncated Caffe models for VGG 16, VGG 19 and MobileNet with MatLab interface in a PC equipped with an E5-1630 v4 3.70GHz CPU, 16GB system memory, and a GeForce GTX 1070Ti. The used datasets and configuration determination are elaborated as follows.

Dataset: 600 legitimate examples and 600 successfully crafted adversarial examples are collected for each network architecture. The dataset is then randomly separated into two groups, each of which contains 300 legitimate examples and 300 adversarial examples for each network architecture. One group is used for figuring out the layer compositions, respective noise injection rates as well as threshold values in each phase of the detection scheme. The other group is used for validation.

Table 3.2 Final configurations for the three models with hierarchical detection scheme.

Models	Mutation rate	Layer	$(\tau_{nor}, \tau_{adv}) \vee \tau$
VGG 16	0.006	<i>conv3_3</i>	(0, 1)
	0.008	<i>conv5_1</i>	(0, 0.8)
	0.006	<i>conv5_1</i>	0.4
VGG 19	0.004	<i>conv5_2</i>	(0.1, 1)
	0.004	<i>conv5_1</i>	(0, 1)
	0.004	<i>conv3_1</i>	(0, 1)
	0.009	<i>conv4_1</i>	(−0.1, 0.4)
	0.005	<i>conv4_3</i>	0.7
MobileNet	0.002	<i>conv5_6/sep</i>	(0.4, 0.9)
	0.001	<i>conv5_6/sep</i>	(0.3, 0.8)
	0.002	<i>conv5_5/sep</i>	(0.4, 0.9)
	0.003	<i>conv5_6/sep</i>	0.8

Table 3.3 Final configurations for the three models with single-layer detection scheme.

Models	Mutation rate	Mutated layer	Threshold
VGG 16	0.006	<i>conv3_3</i>	0.7
VGG 19	0.008	<i>conv3_4</i>	0.5
MobileNet	0.001	<i>conv5_6/sep</i>	0.6

Hierarchy determination: In this stage, a dominant layer that can maximize the discriminability of adversaries within a limited range of noise injection rate is selected for each detection phase. The respective threshold values are also determined. For the early stages, the

3.3 Proposed Discriminative Noise Injection Method

practice is to take a mutated layer that has the largest $\Delta\zeta$ with the constraints that no more than 5 legitimate samples and 10 adversarial samples are misclassified with the selected τ_{adv} and τ_{nor} for each stage. The first constraint imposed on the selection is to restrict the false positive rate (FPR) to approximately 5% for 3 to 5 phases of thresholding. This is because a detector with high false positive rate is undesirable in security critical applications. The second constraint is to prevent the degradation of correct detection (or true positive) rate. The final configuration of the hierarchical detection scheme for each CNN model is shown in Table 3.2. The detector configurations for single-layer mutation are also listed in Table 3.3.

3.3.4 Simulation Results

Table 3.4 Evaluation results for different discriminative noise injection configurations.

Models	Configuration	Detection rate (TPR) (%)	FPR (%)
VGG 16	single-layer	60.00	5.00
	3-layer	82.00	5.67
VGG 19	single-layer	60.00	5.00
	5-layer	80.33	7.00
MobileNet	single-layer	75.33	4.33
	4-layer	88.00	5.67

Table 3.4 shows the validation outcomes for the proposed method with both predefined single-layer and multi-layer configurations. TPR and FPR are acronyms for true positive rate and false positive rate, respectively. They are defined as:

$$TPR = TP / (TP + FN) \quad (3.2)$$

$$FPR = FP / (FP + TN) \quad (3.3)$$

where TP , TN , FP and FN represent true positives, true negatives, false positives and false negatives, respectively.

In this case, an ‘adversary’ is considered to be a positive class. It is evident that all the multi-layer configurations have much higher detection rates than single-layer configurations without much sacrifice of FPR. The outstanding performance of joint-mutation is likely due to the richer feature information included in the perturbations than the limited feature change information detectable by the perturbations with single-layer mutation. Moreover, the FPR for all tested scenarios are around an acceptable 5% rate. This is attributed to the loss control in the setup stage. In addition, MobileNet ranks first in the detection rate regardless of config-

urations, revealing the poor robustness of adversarial examples to the mutants of MobileNet. Correspondingly, a high sensitivity of decision boundary on noise injection into the weights of MobileNet could also be inferred.

3.4 Proposed Fired Neuron Rate Based Decision Tree Detector

3.4.1 Most Discriminative Layer (MDL)

The fired neuron rate (FNR) is defined as the number of positive activations in the output feature map over the total number of activations. The hypothesis of the proposed method is that tampered and clean images have different FNRs, and such disparity is reinforced in some dominant layers. To evaluate the hypothesis, 1000 correctly classified clean images and 1000 successfully misclassified adversarial images crafted using universal perturbation algorithm are tested for each model. The FNR of each layer is calculated for each image. The distribution of FNR for normal and adversarial images in various layers of different models are shown in Figs. 3.6 to 3.9. For MobileNet, some layers that have nearly the same distribution for both benign and malicious samples are removed from Fig. 3.9 to refrain them from cluttering the useful information. It is evident that there exists a large deviation among different layers in terms of discriminability between the clean and tampered samples based on the FNR for each model. Besides, the layers that possess high differentiability are all convolutional layers, for example, *conv1*, *conv2* and *conv5* layers in AlexNet, *conv2_2*, *conv4_3* and *conv5_1* layers in VGG 16, *conv4_1*, *conv4_4* and *conv5_4* layers in VGG 19, and *conv2_2/dw*, *conv3_1/sep* and *conv6/dw* layers in MobileNet. This finding can be attributed to the feature extraction role of convolutional layers, which tends to be heavily weighted in the process of crafting powerful adversarial perturbations. It also implies that some layers are more responsive to the addition of distortions than others. The layer that has the least overlap between the FNR distributions of the normal and adversarial images is designated as the most discriminative layer (MDL).

3.4 Proposed Fired Neuron Rate Based Decision Tree Detector

Fig. 3.6 FNR distribution in different layers of AlexNet.

Fig. 3.7 FNR distribution in different layers of VGG 16.

Fig. 3.8 FNR distribution in different layers of VGG 19.

3.4.2 Proposed Decision Tree Detector Framework

The proposed adversarial example detection method divides a set of clean and adversarial images into two bins by thresholding the FNR distributions of the MDL progressively until there is no improvement in the correct classification rate of both types of images. Once the threshold in each node of the binary decision tree is established, the thresholding operation of the entire decision tree can be integrated into the DNN to determine if an input image is benign or adversarial. The detection framework is illustrated in Fig. 3.10, where $v_{i,j}$ denotes the j -th node at the i -th level of the decision tree. $i = 1, 2, \dots, i_{max}$, $j = 1, 2, \dots, j_{max}$ and $j_{max} \leq 2^{i-1}$. $\tau_{i,j}$ is the threshold for separating the malicious and benign samples at node j of level i . Each node $v_{i,j}$ consists of a string variable $MDL_{i,j}$ for the name of the MDL of the target CNN, a degree of impurity $\delta_{i,j}$ and a type $t_{i,j}$. Each node $v_{i,j}$ is generated by assigning to it a set of training images $I_{i,j}$. $\delta_{i,j}$ is measured by the Gini index [87]. It is calculated as follows:

$$\delta_{i,j} = 1 - (P_a^2 + P_b^2) \quad (3.4)$$

3.4 Proposed Fired Neuron Rate Based Decision Tree Detector

Fig. 3.9 FNR distribution in different layers of MobileNet.

Fig. 3.10 Decision tree for successive thresholding of normal and adversarial FNR distributions in MDL.

where P_a and P_b are the proportions of adversarial and benign images, respectively in $I_{i,j}$. It has a minimum value of 0 if the images in $I_{i,j}$ are all adversarial or all clean, and a maximum value of 0.5 if $I_{i,j}$ has equal proportion of adversarial and benign images. $t_{i,j} = 0$ if majority of the images in $I_{i,j}$ are clean and 1 otherwise. The tree terminates at depth i when there is no improvement in the correct classification rate p_i determined by a separate set of validation images I_v . The procedure for the construction of the decision tree is illustrated in Fig. 3.11. The corresponding explanations are given as follows:

Fig. 3.11 Construction of the FNR based decision tree

Step 1: Set $i = 1, j = 1$ and add a root node $v_{i,j}$ to the decision tree. Assign N randomly selected malicious and N labelled benign images to $I_{i,j}$ as the training images and M randomly selected malicious and M labelled benign images to I_v . Set $p_i = 0$ and $\delta_{i,j} = 0.5$.

Step 2: Input the images $I_{i,j}$ to the target trained DNN to obtain the FNR distribution for each type of images in each layer. Select the MDL L of DNN and record $MDL_{i,j} = L$. Determine and record the threshold $\tau_{i,j}$ that maximizes the inter-class variance of the two distributions of $MDL_{i,j}$.

3.4 Proposed Fired Neuron Rate Based Decision Tree Detector

Step 3: Add two children nodes, $v_{i+1,2j-1}$ and $v_{i+1,2j}$, to node $v_{i,j}$. Assign those images in node $I_{i,j}$ that have FNR in $MDL_{i,j} \leq \tau_{i,j}$ to $I_{i+1,2j-1}$, i.e., the image set of its left child node and the remaining images to $I_{i+1,2j}$, i.e., the image set of its right child node. Set the current predicted label t of the child node to 0 (benign) or 1 (adversarial) by the majority type of its images. Calculate $\delta_{i+1,2j-1}$ and $\delta_{i+1,2j}$. Input the validation images I_v to the target DNN to obtain the FNR for each image. Calculate their correct classification rate p_i .

Step 4: If $p_{i+1} \leq p_i$, terminate. Otherwise, set $i = i + 1$ and repeat Steps (2)-(4) for any leaf node $v_{i,k}$, $k \in \{2j-1, 2j\}$ that has $\delta_{i,k} > 0$.

At the end of the process, each node of the decision tree will contain a threshold associated with a MDL of the target DNN. Each leaf node will also have a type label for integration into the CNN hardware for detecting if a presented input image is highly probable to be adversarial or benign. The advantage of using multiple instead of a single threshold is that malicious perturbation is likely to affect multiple levels of feature abstraction and a single threshold may not be able to sufficiently minimize the intra-class variance of the two distributions. For example, although *conv1* layer in AlexNet has the most discriminative FNR distributions between the tampered and clean inputs according to Fig. 3.6, there is still a large unresolved overlap.

3.4.3 Experimental Setup

The proposed method is examined using 8-bit truncated Caffe models for AlexNet, VGG 16, VGG 19 and MobileNet with MatLab interface to a PC equipped with an E5-1630 v4 3.70GHz CPU, 16GB system memory, and a GeForce GTX 1070Ti. 2000 correctly predicted benign images from ImageNet dataset and their corresponding 2000 successfully misclassified tampered images by each network architecture are used in this experiment. These 4000 images are randomly divided into two groups, each of which contains 1000 clean samples and 1000 tampered samples for each model. One group is utilized for the determination of the MDLs and τ of each decision node. The other group is used for testing. Five-fold cross-validation technique is used to prevent the decision tree algorithm from overfitting or selection bias. It partitions the training database into five groups. For each group, validation is performed after the completion of the analysis on the remaining groups in each round. Then the average performance of the five rounds is considered. The final depth upon termination, and the MDLs and threshold values in the first two levels of the first round of the decision tree for each CNN architecture are summarized in Table 3.5.

Table 3.5 Parameters of the final decision trees for the four DNN models.

Models	Depth	MDL	τ
AlexNet	6	$MDL_{1,1}: conv1$	0.321
		$MDL_{2,1}: conv2$	0.193
		$MDL_{2,2}: conv3$	0.281
VGG 16	7	$MDL_{1,1}: conv4_3$	0.123
		$MDL_{2,1}: conv2_2$	0.444
		$MDL_{2,2}: conv4_3$	0.128
VGG 19	8	$MDL_{1,1}: conv4_1$	0.491
		$MDL_{2,1}: conv5_4$	0.064
		$MDL_{2,2}: conv3_2$	0.528
MobileNet	6	$MDL_{1,1}: conv3_1/sep$	0.488
		$MDL_{2,1}: conv6/dw$	0.837
		$MDL_{2,2}: conv5_4/dw$	0.508

Table 3.6 Correct detection and false positive rates.

Models	Configuration	Detection rate (TPR) (%)	FPR (%)
AlexNet	single-level	87.70	28.00
	6-level	89.60	17.10
VGG 16	single-level	81.20	11.70
	7-level	91.90	6.70
VGG 19	single-level	91.60	21.80
	8-level	93.20	6.60
MobileNet	single-level	94.10	18.10
	6-level	98.10	4.50

3.4.4 Simulation Results

Table 3.6 shows the results of the proposed method on the detection rate of unseen adversarial inputs to each model. The single-level method refers to the use of only one MDL with one threshold at the root of the decision tree. In the testing stage, majority voting of the five rounds of prediction is conducted. The results show that all the multi-level decision trees have higher detection rates and lower FPRs than their single-level counterparts. In addition, the detection accuracy of the multi-level decision tree also increases with the CNN model depth. With the increasing depth of the DNN, the responsiveness and overall ability of the MDLs to discriminate finer difference between tampered and untampered image feature information may also increase. However, a comparatively high FPR occurs for all the single-level detectors and

the multi-level decision tree of AlexNet because of the lack of loss control in the training stage. A better parameter tuning method that imposes a loss constraint can be considered in the near future to improve this aspect.

3.5 Comparison

Table 3.7 Comparison among the two proposed works and feature squeezing method.

Metrics	Discriminative noise injection	FNR	Feature squeezing [20]
Defense category	Prediction inconsistency	Sample statistics	Prediction inconsistency
Model	MobileNet	MobileNet	MobileNet
Truncated model	Yes	Yes	No
Dataset	ImageNet	ImageNet	ImageNet
Attack	Universal Perturbation	Universal Perturbation	DeepFool
Configuration	4-layer	6-level	3-squeezer
Detection rate (%)	88.00	98.10	78.60
FPR (%)	5.67	4.50	8.33

Table 3.7 shows an overall comparison among the proposed discriminative noise injection method, proposed FNR based decision tree detector and feature squeezing method [20]. The universal perturbation considered in the proposed countermeasures is originated from DeepFool, which is the attack algorithm considered by feature squeezing method. The two proposed methods, though assessed using 8-bit truncated version of the model, still has a much higher detection rate and a lower FPR than feature squeezing, with nearly 12% and 25% of improvements in detection rate, respectively. Furthermore, the results of the proposed discriminative noise injection method are based on 300 adversarial and benign images and the results of the proposed FNR based decision tree detector are based on 1000 adversarial and benign images. However, only 100 adversarial and benign images are tested by [20]. As a much larger sample size is used in the experiments, the evaluations of the proposed methods are more rigorous and hence the results are more reliable. Last but not least, the first proposed discriminative noise injection method and the existing feature squeezing method both need to run multiple inferences to identify a single input. This incurs latency overhead. In contrast, FNR based decision tree detector can obtain all the necessary data for detection by executing one inference. From this perspective, the proposed FNR based countermeasure is better than the other two defenses in comparison.

3.6 Summary

This chapter presents two methods of detecting adversarial examples in DNN models adapted to hardware accelerator implementation. A dominant layer directed discriminative noise injection method is first proposed for adversarial perturbation detection. It harnesses the greater separability of adversarial and legitimate images by utilizing their difference in sensitivity to layer-wise noise contamination on weight parameters. Three different DNN models are evaluated. For each model, the optimal configuration is obtained by a pre-characterization to select a small number of sensitive layers to be mutated, their respective noise injection rates and label change rate thresholds. The detection efficiency is maximized with as small mutation rate and number of iterations as possible. Eight-bit truncated versions of the three models are used for evaluation to account for typical fixed-point implementation of hardware DNN. The results show an improved detection efficiency compared with the state-of-the-art defenses that target software DNN classifiers. Lightweight linear-feedback shift registers (LFSRs) can be embedded to the DNN hardware accelerator for the random noise injection required by the proposed method. Another FNR based defense is also proposed for adversarial example detection. Greater distinguishability between adversarial and natural images are observed in their FNR distributions at some convolution layers. The optimal decision threshold at selected MDL is obtained successively in a tree algorithm by applying a five-fold cross-validation technique. Eight-bit truncated versions of four different DNN models are used for the assessment to account for their typical fixed-point implementation in hardware. The simulation results also show a significant improvement in detection accuracy of adversarial inputs compared with feature squeezing method. Unlike the first proposed countermeasure, this method has negligible latency overhead. Moreover, this method can be incorporated in custom DNN accelerator design by mapping the read and write memory access patterns into layer-wise FNR. In this way, the thresholding operation can be accomplished by introducing appropriate switches and comparators into the identified MDLs of custom DNN accelerator at design time.

Chapter 4

In-situ Adversarial Sample Detector for Edge Deep Neural Network (DNN)

4.1 Introduction

The cross over from statistical models to data science in early AI evolution is motivated by running powerful deep learning (DL) algorithms on the cloud. Despite the economy of scale of cloud computing, data velocity is constrained by the network bandwidth and availability. The boom of the Internet of Things (IoT) has changed the way data are generated and curated. It motivates a distributed computing model that enables more user-centric artificial intelligent (AI) applications to be run on the device without being limited by the network latency. Unfortunately, large learning models also require more computations, which make them infeasible for direct integration into the edge devices with limited computing power and memory. Hardware accelerators have thus evolved to support the in-situ computations of DL algorithms with fast and accurate inferences [12–14]. Techniques such as low-cost data flow, data quantization, network compression and dynamic voltage-frequency scaling, have been adopted for the design of DNN accelerators to boost energy efficiency, reduce the memory requirement, and lower the computational complexity [1]. Together with the emerging open-source AI model compilers [15] and commercially available toolkits like Intel(R) OpenVINO(TM), pre-trained deep learning models can be optimized and mapped into a reusable hardware abstraction layer for implementation on high-speed and power-efficient edge AI accelerator platforms.

The bridge between edge computing and AI opens out many new application possibilities, including self-driving cars, video surveillance, contact tracing and border control. At the same time, this has also made edge AI an attractive target for adversarial attacks [2]. The black-box nature of such super-intelligence can be easily exploited to turn an essentially benevolent

system to malevolent, which poses safety, privacy and security hazards [16]. Adversarial examples can be generated fast at low cost. Their attack surface can also be extended beyond the software boundary and made more robust with high transferability across models. Countermeasures against adversarial examples have been explored and proposed in recent years, as reviewed in Chapter 2. However, they are mainly designed and investigated based on software models of DNNs. Most existing adversarial input detection methods assume that the adversarial examples are generated with low to moderate confidence perturbations. Robust high confidence perturbations that target edge AI with DNN models optimized for efficient hardware implementation were not sufficiently evaluated. Although truncated models have been used for the evaluation by a few latest schemes [88, 89], the detections are simulated on graphical processing unit (GPU) platform without physically applying the adversarial inputs into the DNN through the hardware accelerator. Most importantly, as these solutions are mostly developed for off-line analysis, their code sizes and computation complexity are too high to be fitted into the edge AI platform processor or reconfigurable fabrics. To allow the adversarial inputs to be identified in-place with the inference machine, a hardware architecture [8] was recently proposed to accelerate several existing adversarial input detection algorithms. The architecture has very high hardware overheads to accommodate multiple different detection schemes. Another recent hardware-oriented solution [7] for adversarial input detection targets only small-scale classification task. The solution is not scalable for massive classification.

In this chapter, a new lightweight adversarial input detector that can be efficiently implemented alongside a spatial architecture for edge DNN implementation is proposed. The idea is inspired by the layer directed discriminative noise injection scheme [88] described in the previous chapter, but with fundamentally different mutation mechanisms, optimal label change mutation strategies and detector design. Without transforming the input image, nor modifying the edge implementation of the deployed DNN model, maximal label change inconsistency between the benign and adversarial inputs can be triggered by adaptively mutating the DNN computations. The mutation is controlled by injecting glitches of selective intensity at specific timing into the clock signal of the hardware accelerator. Such clock glitch is a short-lived fault introduced by irregular pulses in the clock signal that violate the timing constraint in a digital circuit [90]. The effect is illustrated in Fig. 4.1, where d_{in} is the data transmitted from a register A through a combinational logic to a register B and d_{out} is the output data of the register B . The existence of a clock glitch reduces the clock period, causing a wrong sampling of d_{in} or a timing violation. This can introduce an error into the register B . The clock glitch has been demonstrated as an effective attack method [91] that enables the incorrect intermediate results to be propagated to the final stage. Due to the undesirable and detrimental effects, clock glitch has never been considered as a useful source of noise by the defender. In this chapter, its role

will be reversed into a controllable uncertainty at the advantage of the defender to blindside the adversary. The optimal glitch intensity that maximizes the label change inconsistency between adversarial and benign images for each layer is used for the construction of a light gradient boosting machine (lightGBM) [92]. The test node attributes of the lightGBM are determined by an offline training which maximizes the information gain of the training dataset of each node based on three different types of probabilities, which are the top-1 probabilities predicted by the unmutated model, mutated model and their difference. The trained lightGBM is a small two-class tree classifier consisting of only a few test nodes. The glitch injection circuit and tree detector are implemented together with convolution network on a Xilinx ZU9EG chip. The performance of the proposed in-situ detector is evaluated by applying four different popular adversarial attacks on two different DNN architectures trained on the ImageNet dataset. The attacks evaluated are fast gradient sign method (FGSM) [17], iterative-FGSM (I-FGSM) [65], C&W [67] and universal perturbation (UP) [4]. The results and overheads are compared against the recently proposed software and hardware oriented adversarial input detectors.

Fig. 4.1 Timing diagrams of a typical sequential logic path under (a) a normal clock and (b) a glitched clock.

4.2 Proposed Online Adversarial Input Detector

4.2.1 Threat Model

The proposed countermeasure intends to defend against a vigorous attacker who is equipped with the information about the target integrated DNN classifier deployed in the field but has no accessibility to its inner system. The attacker can generate adversarial images that are imperceptible or quasi-imperceptible to human visual systems to fool the DNN using any of the white-box, gray-box or black-box techniques. It is assumed that the attacker can obtain the network architecture and weight parameters of the DNN model through various plausible means like eavesdropping the side-channel information of model internal variables [80]. It is also assumed that the attacker is able to train a substitute model with the knowledge of the model structure to emulate the victim model through query-based model extraction attacks [93, 94]. Except physically tampering the internal nodes and nets of a monolithic integrated circuit, the attacker can make any non-destructive manipulations if they do not leave any apparent evidence. In other words, the malicious acts performed by the attacker should not result in a noticeable rate of interferences and prolong unavailability that can be trivially detected or traced. Under this premise, the attacker cannot probe or tamper the circuit internals to monitor the inner operations of the DNN in order to disable or bypass the built-in defence mechanism.

4.2.2 Label Change Inconsistency between Original and Adversarial Inputs on Clock Glitch Injection

In Chapter 3, the feasibility of extracting multiple dominant layers that possess high distinguishability between adversarial and benign images with random mutations [88] has been algorithmically established. These dominant layers were identified by observing the label change rate (LCR) of the mutated models when randomly selected weight parameters of each layer are perturbed by deliberately added noise. Although the results can be simulated by software execution on temporal architectures such as GPU or central processing unit (CPU), the adaptive thresholding scheme cannot be efficiently implemented on a spatial architecture used for DNN in application-specific integrated circuit (ASIC) or field programmable gate array (FPGA)-based designs. The DNN hardware will have to be redesigned to incorporate the necessary weight perturbations into selected layers required by the mutated model, which will incur non-trivial area, delay, and power consumption overheads. The scheme also cannot be easily adapted to third-party DNN intellectual property (IP) cores. For example, Xilinx deep learning processing unit (DPU) IP core is supplied as an encrypted register transfer level (RTL) file [50], with little or no flexibility for any structural modification. As it takes great amount of

4.2 Proposed Online Adversarial Input Detector

engineering effort and know-how to design a custom DNN accelerator to produce state-of-the-art results, it is more cost-effective to use the third-party DNN IP cores for many designs, but protect them against adversarial example attacks. To realize online adversarial input detection in spatial architectures and third-party DNN IPs, an alternative hardware-oriented mutation technique is required.

To enable online detection of adversarial examples using existing hardware accelerators with negligible hardware overhead, clock glitch is used as a fault source to emulate the desired mutation effect in the deep learning accelerator. The hypothesis is that different layers of the DNN accelerator possess different degrees of sensitivity and immunity to glitch injection. To verify the plausibility of this hypothesis, a systematic approach is established to identify the layers that can cause the outputs of an adversarial input to deviate more significantly than its benign input in general by controlled glitch injection targeting specific layers of computations. The distinct clock domain dedicated for the processing element (PE) array of the DNN accelerator is utilized for glitch injection so that the injected glitches affect only operations scheduled for computation in the PE array at specific clock cycles. Since different DNN accelerators may have different clock configurations, Xilinx DPU implemented in ZCU102 evaluation board is used as an example to illustrate the clock glitch injection setup for evaluating the label change inconsistency due to the application of different amount of clock glitches at each convolutional neural network (CNN) layer. A clock glitch can be generated by using a pair of normal and phase-shifted clock signals of the accelerator and a control signal *trigger*, as shown in Fig. 4.2. In Fig. 4.2, *clk* is the input clock for general logic such as register configuration, data controller, global buffer and SIMT functional units of the DNN accelerator. *clk_{2×}* is the faster input clock to the PE array to accelerate the multiply-and-accumulate (MAC) operations. The frequency of *clk_{2×}* is specified to be double the frequency of *clk* for Xilinx DPU. *clk_{2×}_90* is a 90° phase shifted clock of *clk_{2×}*. *ce* is a clock gating control signal used to turn off *clk_{2×}* when the PE array is in idle mode. The glitch injection process can be controlled by the signal *trigger*. By feeding *trigger* into the select input, *clk_{2×}* and *clk_{2×}_90* to the data inputs of a 2-to-1 multiplexer, and the output *clk_{2×}_glitch* of the multiplexer to the PE array, clock glitches can be selectively injected into the target execution cycles of the PE array. The timing information for each layer of computation in the PE array in terms of the number of cycles in *clk_{2×}* can be extracted by a cycle-accurate hardware profiling. The convolutional operations on the input data are executed sequentially by the MAC units layer by layer. The proportion of clock cycles *clk_{r_l}* expended in a layer *l* for each input classification can be approximated by the following formula:

$$clk_{r_l} = \frac{W_l}{U_l \times \sum_{i=1}^L \frac{W_i}{U_i}} \quad (4.1)$$

Fig. 4.2 The generation of a clock glitch on the DNN accelerator.

where W , U and L are the workload, utilization rate and total number of layers of the DNN model, respectively.

$clk_{2\times}$ is monitored to obtain the total number of active clock cycles for each input classification. The starting time and number of clock cycles spent by the PE array on the execution of convolutional operations for each layer can then be derived from clk_{rl} . A margin of one hundred clock cycles, representing approximately 0.01% of the average number of cycles in each layer for this model, at the beginning and end of each layer's execution time is reserved for operational instability to ensure that the glitches introduced do not influence other layers except the layer of interest.

A set of glitch intensities $G(\%) = \{0.1, 0.2, 0.5, 1, 2, 10, 20\}$ are applied to every layer of the VGG16 model [46]. Glitch intensity is computed by $1/T_{glitch}$, where T_{glitch} is the number of $clk_{2\times}$ cycles between two successive glitches. 292 correctly classified clean images and 292 successfully misclassified adversarial images crafted using FGSM with $\epsilon = 4/255$ are evaluated for each layer-wise mutated model. As can be observed from Table 4.1, the number of clean images that have their predictions altered ($\#LC$) is generally smaller than the number of adversarial images at any glitch intensity for any perturbed layer of VGG16. This result supports the hypothesis proposed in [75] for ImageNet dataset besides MNIST and CIFAR-10, and for the hardware executable model besides the software model. The proportions of $\#LC$ for normal and adversarial inputs are denoted as LCP_{norm} and LCP_{adv} , respectively. The difference $\Delta LCP = LCP_{adv} - LCP_{norm}$ is plotted in Fig. 4.3 for different layers of VGG16 at different glitch intensities. It can be clearly seen that the curves of most layers are concave downwards, indicating a possibility for finding an optimal glitch intensity that maximizes

4.2 Proposed Online Adversarial Input Detector

Table 4.1 Number of images that have changed their predictions ($\#LC$) upon injecting different intensities of glitches into each layer of VGG16 model.

Layer	Image type	Glitch intensity (%)						
		20	10	2	1	0.5	0.2	0.1
<i>conv1_1</i>	Normal	287	282	161	127	62	26	9
	Adversarial	288	280	222	185	141	81	50
<i>conv1_2</i>	Normal	276	226	88	43	27	4	2
	Adversarial	280	261	178	147	93	52	35
<i>conv2_1</i>	Normal	110	74	23	9	4	1	1
	Adversarial	186	174	107	61	34	21	15
<i>conv2_2</i>	Normal	289	226	36	11	5	3	0
	Adversarial	290	268	125	75	44	30	21
<i>conv3_1</i>	Normal	118	60	12	1	1	0	1
	Adversarial	199	140	74	40	27	23	12
<i>conv3_2</i>	Normal	256	123	23	4	3	2	1
	Adversarial	275	212	112	57	33	27	14
<i>conv3_3</i>	Normal	286	173	25	9	6	3	0
	Adversarial	289	249	121	63	41	25	14
<i>conv4_1</i>	Normal	72	23	7	1	1	0	1
	Adversarial	158	99	52	23	22	19	19
<i>conv4_2</i>	Normal	244	80	15	5	1	1	1
	Adversarial	266	187	87	44	29	20	19
<i>conv4_3</i>	Normal	249	136	25	9	2	1	0
	Adversarial	278	226	116	54	37	24	13
<i>conv5_1</i>	Normal	90	30	7	0	1	1	1
	Adversarial	191	110	49	31	20	17	11
<i>conv5_2</i>	Normal	131	19	8	4	1	1	1
	Adversarial	204	96	56	27	18	19	14
<i>conv5_3</i>	Normal	11	6	4	0	0	0	0
	Adversarial	77	57	46	28	23	13	13
<i>fc6</i>	Normal	291	291	242	50	23	5	0
	Adversarial	291	289	276	151	86	54	25
<i>fc7</i>	Normal	288	54	11	8	3	1	0
	Adversarial	292	158	78	47	37	18	13
<i>fc8</i>	Normal	246	274	50	17	9	2	2
	Adversarial	260	276	147	87	54	25	27

the inconsistent label changes between the adversarial and clean samples for each layer. For example, ΔLCP_{conv1_2} increases when the glitch intensity is less than 1%, but declines thereafter. Therefore, the optimal glitch intensity is estimated to be 1% for *conv1_2* layer of VGG16. The reduced inconsistency in label changes after the optimal points implies the enhanced mutation effect on the clean images. ΔLCP of *conv3_1*, *conv4_1*, *conv5_1* and *conv5_3* layers appear to be monotonically increasing with glitch intensity. This might be due to the constraint of the maximum glitch intensity of 20%. On the other hand, this is also a sign of the inherently higher immunity of these layers against inconsistent label changes of benign and adversarial samples by glitch injections. Furthermore, it is also noticed that the curves for *conv1_1* and *conv1_2* layers reach their maximum at a lower glitch intensity. This can be explained by the fact that errors due to a small number of glitches for the first two layers are likely to induce more errors as they propagate forward through other layers to the final stage of inference.

Fig. 4.3 Comparison of ΔLCP in different layers of VGG16.

4.2.3 Proposed Decision Tree Detector

The proposed online detector of adversarial input is implemented as a simple decision tree classifier. The decision tree is constructed by an offline analysis to find good glitch patterns and prediction probability thresholds to maximally discriminate the adversarial and benign training inputs. After the offline analysis, the number of decision nodes of the binary tree classifier and their attributes are fixed. An example of a tree classifier detector trained by an image set of 292 clean images and 292 adversarial images for VGG16 model implemented on Xilinx ZCU102 board is shown in Fig. 4.4. The decision tree has three different types of non-terminal nodes, designated as original (O), mutated (M) and difference (D), and two terminal nodes, designated as Adversarial (A) and Benign (B). Each non-terminal node $v_{i,j}$ has a probability threshold $\tau_{i,j}$ and two children nodes. Type M and type D nodes have in addition a glitch pattern (l, g) , where l is the target DNN layer of glitch injection and g is the glitch intensity.

4.2 Proposed Online Adversarial Input Detector

When an input is presented to a DNN, the hardware accelerated classifier F will determine its unmutated label $pred_org$ and its corresponding predicted confidence level p_org . Then, the detector is activated. Starting from the root $v_{1,1}$, the input image will undergo an attribute test in each non-terminal node. The outcome of the test will determine which child node to follow for another test in the next level until either terminal node is reached. The destination terminal node, A or B, determines if the input is adversarial or benign. The attribute test compares a probability against the node threshold. For node type O, the predicted top-1 probability from the unmutated F is used for the comparison. For node type M, the top-1 probability $p_mut_{(l,g)}$ is obtained from the DNN classifier $F_{l,g}$ upon the application of a glitch pattern (l,g) . It is made negative when the output classification $pred_mut$ is different from the initially predicted label $pred_org$ and positive otherwise. For node type D, the probability is obtained from the difference probability $p_diff_{(l,g)} = p_mut_{(l,g)} - p_org$.

Fig. 4.4 An example of the proposed tree classifier detector for VGG16. Nodes $v_{4,1}$ and $v_{4,3}$ are terminal nodes B, and nodes $v_{3,4}$ and $v_{4,6}$ are terminal nodes A.

To construct the above decision tree for a given DNN hardware accelerator at design time, a set of training images, I , is required. I consists of N correctly classified benign images and N successfully misclassified adversarial images. Let $Layer$ be a list of layer names of the DNN and G be a set of glitch intensities. Each image of I is fed into the original model F and all possible mutated models. A mutated model $F_{(l,g)}$ is produced by injecting a glitch pattern (l,g) to F , where $l \in Layer$ and $g \in G$. $\Delta LCP_{(l,g)}$ is first calculated for each possible glitch pattern (l,g) . Then, a list of candidate glitch patterns Z that can enlarge the difference between the adversarial and normal samples is obtained from the calculated $\Delta LCP_{(l,g)}$ as follows:

$$Z = \{(l,g) | l \in Layer, g \in G, \text{ and } \Delta LCP_{(l,g)} \geq \min_{l \in Layer} (\max_{g \in G} \Delta LCP_{(l,g)})\} \quad (4.2)$$

The reason for selecting the set of glitch patterns that have $\Delta LCP_{(l,g)} \geq \min_{l \in Layer} (\max_{g \in G} \Delta LCP_{(l,g)})$, instead of glitch patterns that merely maximize $\Delta LCP_{(l,g)}$ layer-wise, is to prevent under-fitting.

Initially, all training images in I are assigned to the root node $v_{1,1}$, i.e., $I_{1,1} = I$. Thereafter, the decision tree is constructed by dividing the training image set $I_{i,j}$ into two disjoint subsets, $I_{i+1,2j-1}$ and $I_{i+1,2j}$, to split node $v_{i,j}$ into two children nodes, $v_{i+1,2j-1}$ and $v_{i+1,2j}$. The splitting is determined by the information gain (IG) of the dataset. To compute the IG, the entropy of $I_{i,j}$ is required, which can be calculated as follows:

$$H(I_{i,j}) = -(Pa_{i,j} \times \log_2 Pa_{i,j} + Pb_{i,j} \times \log_2 Pb_{i,j}) \quad (4.3)$$

where $Pa_{i,j}$ and $Pb_{i,j}$ are the proportions of adversarial and benign images, respectively in $I_{i,j}$, and $Pa_{i,j} + Pb_{i,j} = 1$. It has a maximum value of 1 if $I_{i,j}$ has equal proportion of clean and adversarial images, and a minimum value of 0 if the images in $I_{i,j}$ are all benign or all adversarial.

Given the list of candidate glitch patterns Z , all three types of probabilities p_{org} , $p_{mut(l,g)}$ and $p_{diff(l,g)}$ are calculated for each image in $I_{i,j}$ of a non-terminal node $v_{i,j}$. As each glitch pattern of Z will produce a $p_{mut(l,g)}$ and a $p_{diff(l,g)}$ for each image, there are altogether $2 \times |Z| + 1$ probability histograms. From each histogram h , a threshold value that maximizes the IG is selected as the optimal threshold τ_h to split $I_{i,j}$, i.e.,

$$\begin{aligned} \tau_h &= \arg \max_{\tau \in T} IG(I_{i,j}, h, \tau) \\ &= \arg \max_{\tau \in T} (H(I_{i,j}) - P(I_{i,j}|_{p \leq \tau}) \times H(I_{i,j}|_{p \leq \tau}) - P(I_{i,j}|_{p > \tau}) \times H(I_{i,j}|_{p > \tau})) \end{aligned} \quad (4.4)$$

where T is a set of discrete threshold values for a histogram h , $p \in \{p_{org}, p_{mut(l,g)}, p_{diff(l,g)}\}$ denotes a probability value, and $I_{i,j}|_{p \leq \tau}, I_{i,j}|_{p > \tau} \subset I_{i,j}$ are two disjoint subsets of images obtained by thresholding the top-1 probabilities of $I_{i,j}$ by τ .

Each histogram h_a corresponds to an attribute $a \in A$, where $A = \{O, M_{z_i}, D_{z_i}\} \forall i = 1, 2, \dots, |Z|$ and $z \in Z$. With the optimal threshold τ_a determined from h_a in (4.4), the attribute $a_{i,j}$ and its corresponding optimal threshold $\tau_{i,j}$ of node $v_{i,j}$ is selected from the one that maximizes the IG as follows:

$$\begin{cases} a_{i,j} = \arg \max_{a \in A} IG(I_{i,j}, h_a, \tau_a) \\ \tau_{i,j} = \tau_a \hat{=} a_{i,j} \end{cases} \quad (4.5)$$

Fig. 4.5 shows an example of three probability histograms corresponding to the top-1 probabilities of node attributes O, M and D, respectively. These histograms are obtained

4.2 Proposed Online Adversarial Input Detector

Fig. 4.5 An example that illustrates the optimal threshold and the IG for different splitting attributes.

by feeding an image set containing 292 clean images and 292 adversarial images into the original model and the mutated model with the glitch injection pattern $z = (fc7, 10)$. The maximum IG and the corresponding optimal threshold are also indicated for each histogram. For this illustration, since the node attribute D has the largest IG, if there is no other glitch injection pattern $z \in Z$ that has larger IG, this non-terminal node $v_{1,1}$ will be set to type D with $z = (fc7, 10)$, and $\tau_{1,1} = -0.453$.

Upon fixing the best attribute $a_{i,j}$ and $\tau_{i,j}$ for a non-terminal node $v_{i,j}$, $I_{i,j}$ will be split into $I_{i+1,2j-1} = I_{i,j}|_{p \leq \tau_{i,j}}$ and $I_{i+1,2j} = I_{i,j}|_{p > \tau_{i,j}}$, and assigned to the newly generated children nodes, $v_{i+1,2j-1}$ and $v_{i+1,2j}$, respectively. If the subset of images assigned to a child node has zero entropy, then the child node will be made the terminal node A if Pa of its assigned image subset is 1 or B if Pa is 0. Otherwise, the child node generated is a non-terminal node, and is annotated by a temporary class variable t to indicate the majority image type of its assigned subset. $t = 1$ if $Pa \geq 0.5$ and 0 otherwise.

To prevent overfitting and limit the depth of the decision tree, a separate set of validation images I_v is used to calculate the correct classification rate. The correct classification rate p_n of I_v is calculated at each iteration as follows:

$$p_n = \frac{TP_n + TN_n}{|I_v|} \quad (4.6)$$

where TP_n and TN_n denote the number of true positives and true negatives, respectively achieved by the constructed decision tree at the n -th iteration. $|I_v|$ represents the total number of images in I_v .

The tree terminates at a depth d_{max} when there is no improvement in p_n , i.e., $p_n - p_{n-1} \leq 0$. Upon termination, every non-terminal node at depth $d = d_{max}$ will be converted to either a

terminal node A or B according to its temporary class t , A if $t = 1$ and B if $t = 0$. Once the training is completed, the deployed decision tree has no leaf node that is non-terminal. All the paths of the decision tree will be terminated at either an adversarial (A) or a benign (B) leaf node.

Algorithm 1 shows the pseudo code of the above described process for splitting a non-terminal node into two children nodes. Two different algorithms can be used for the construction of a decision tree classifier. They are conventional [95] and lightGBM [92] decision trees, as illustrated in Algorithms 2 and 3, respectively. The main difference between them is the tree growth, which is level-wise for the conventional decision tree and leaf-wise for the lightGBM tree. Specifically, the classical decision tree expands all the children nodes that have $H(I_{i,j}) > 0$ at level i , whereas the lightGBM decision tree only expands the leaf node with the maximum delta loss. Delta loss refers to the gradient of the loss function with respect to the output of the currently constructed decision tree. To reduce the search space, lightGBM uses gradient-based one-side sampling (GOSS) for training. It only randomly samples image instances with small gradients while retaining those instances with large gradients. This is reflected in *Lines* 17-20 of Algorithm 1, where a subset $GOSS(I_{i,j})$ of $I_{i,j}$, as opposed to the entire set of $I_{i,j}$, is used to find the best-split attribute for lightGBM. This sampling reduces the computational complexity of the offline decision tree building process. Being more loss conscious, the decision tree generated by lightGBM is generally more unbalanced and deeper than the classical tree.

Algorithm 1 Node Splitting

Function Node_Splitting(Model (F), Attribute list (A), Training image set ($I_{i,j}$), Parent node ($v_{i,j}$), Decision tree (DT)):

```

1: calculate  $H(I_{i,j})$ 
2: if  $H(I_{i,j}) = 0$  // parent node is a terminal node then
3:   if  $Pa_{i,j} == 1$  then
4:      $v_{i,j}.type \leftarrow terminal\_node(A)$ 
5:   else
6:      $v_{i,j}.type \leftarrow terminal\_node(B)$ 
7:   end if
8:   return  $DT$ 
9: end if
10: calculate  $t_{i,j}$ 
11:  $v_{i+1,2j-1} \leftarrow new\_node()$ 
12:  $v_{i+1,2j} \leftarrow new\_node()$ 
13: // initialize  $IG_{max} \leftarrow 0$ 
14: for each  $a \in A$  do
15:    $data_a \leftarrow F_a(I_{i,j})$ 
16:    $IG_a \leftarrow IG(I_{i,j}, data_a, \tau_a)$ 
17:   if  $DT$  is lightGBM then
18:      $data_a \leftarrow F_a(GOSS(I_{i,j}))$ 
19:      $IG_a \leftarrow IG(GOSS(I_{i,j}), data_a, \tau_a)$ 
20:   end if
21:   if  $IG_{max} < IG_a$  then
22:      $IG_{max} \leftarrow IG_a$ 
23:      $a_{i,j} \leftarrow a$ 
24:      $\tau_{i,j} \leftarrow \tau_a$ 
25:   end if
26: end for
27:  $v_{i,j}.type \leftarrow (a_{i,j}, \tau_{i,j}, t_{i,j})$ 
28:  $DT[i+1, 2j-1] \leftarrow v_{i+1,2j-1}$ 
29:  $DT[i+1, 2j] \leftarrow v_{i+1,2j}$ 
30:  $I_{i+1,2j-1} \leftarrow I_{i,j} |_{p(a_{i,j}) \leq \tau_{i,j}}$ 
31:  $I_{i+1,2j} \leftarrow I_{i,j} |_{p(a_{i,j}) > \tau_{i,j}}$ 
32: return  $DT$ 

```

Algorithm 2 Construction of the conventional decision tree

Input: Model (F), Attribute list (A), Training image set (I), Validation image set (I_v), Correct prediction (c_o)

Output: Decision Tree (DT)

```
// initialize decision tree  $DT \leftarrow \{\}$ 
// initialize index  $i, j \leftarrow 1$ 
// initialize iteration  $n \leftarrow 0$ 
// initialize depth  $d \leftarrow 1$ 
// initialize correct classification rate  $p_n \leftarrow 0$ 
// initialize correct classification rate  $p_{n-1} \leftarrow -1$ 
1:  $c_o \leftarrow F(I)$ 
2:  $v_{i,j} \leftarrow new\_node()$ 
3:  $DT[i, j] \leftarrow v_{i,j}$ 
4:  $I_{i,j} \leftarrow I$ 
5: while  $p_{n-1} < p_n$  do
6:   for each leaf node  $v_{i,j} \in level\ i\ of\ DT$  do
7:      $DT \leftarrow Node\_Splitting(F, A, I_{i,j}, v_{i,j}, DT)$ 
8:   end for
9:    $d \leftarrow i$ 
10:   $n \leftarrow n + 1$ 
11:   $p_n \leftarrow DT(I_v)$ 
12:   $i \leftarrow i + 1$ 
13: end while
14: return  $DT$ 
```

Algorithm 3 Construction of the lightGBM decision tree

Input: Model (F), Attribute list (A), Training image set (I), Validation image set (I_v), Correct prediction (c_o)

Output: Decision Tree (DT)

```

// initialize decision tree  $DT \leftarrow \{\}$ 
// initialize index  $i, j \leftarrow 1$ 
// initialize iteration  $n \leftarrow 0$ 
// initialize correct classification rate  $p_n \leftarrow 0$ 
// initialize correct classification rate  $p_{n-1} \leftarrow -1$ 
1:  $c_o \leftarrow F(I)$ 
2:  $v_{i,j} \leftarrow \text{new\_node}()$ 
3:  $DT[i, j] \leftarrow v_{i,j}$ 
4:  $I_{i,j} \leftarrow I$ 
5: while  $p_{n-1} < p_n$  do
6:    $(x, y) \leftarrow \text{index\_of\_max\_loss\_leaf}(DT)$ 
7:    $DT \leftarrow \text{Node\_Splitting}(F, A, I_{x,y}, v_{x,y}, DT)$ 
8:    $n \leftarrow n + 1$ 
9:    $p_n \leftarrow DT(I_v)$ 
10: end while
11: return  $DT$ 

```

4.3 Experimental Results

4.3.1 Hardware Setup

The hardware setup for the experiments is shown in Fig. 4.6. A Xilinx ZCU102 evaluation board for the Zynq UltraScale+ XCZU9EG-2FFVB1156E Multiprocessor System-on-Chip (MPSoC) is used to emulate the edge device. The ZU9EG chip contains a quad-core ARM CortexTM-A53 processor, 600K logic cells and 2520 digital signal processing (DSP) slices [50]. The hardware executable DNN model is implemented on the DPU dedicated for CNN acceleration. The DNN model is compiled by the DNN Development Kit (DNNDK) [96]. The toolchain truncates the 32-bit floating-point weights and activations of the DNN to 8-bit fixed-point values without compromising the classification accuracy of the model. The blocks shaded in gray are the circuitries required for glitch injection. The clock signal to the DSP slices (the PE array) of the DPU is provided by the global clock buffer BUFGCTRL, which selects between dsp_clk and dsp_clk_shift to dsp_clk_sel . dsp_clk is generated from dpu_clk via the Clocking Wizard. dsp_clk_shift is the 90° phase shifted version of dsp_clk . The frequency of dpu_clk is set to 100MHz while the frequency of dsp_clk and dsp_clk_shift is set to 200MHz. The control signal sel_sig of BUFGCTRL is produced by en_glitch and $trigger$. When $en_glitch = 0$, the PE array is clocked by dsp_clk , and the inference is produced by

In-situ Adversarial Sample Detector for Edge Deep Neural Network (DNN)

the unmutated DNN model. When $en_glitch = 1$, the PE array runs on dsp_clk_shift , which is dsp_clk with glitches injected when $trigger$ is active. The glitch intensity and injection time are encoded in the $trigger$ signal. Both en_glitch and $trigger$ signals are configurable by the Extended Multiplexed IO (EMIO) pins of the ARM CortexTM-A53 processor (Zynq UltraScale+ block). The configurations of the $trigger$ signals are given by the node attributes of the proposed decision tree detectors implemented on the ARM CortexTM-A53 processor with a clock frequency of 1.2GHz. This ARM processor is a hard IP core that is monolithically integrated into the FPGA chip in which the target DNN is implemented. The program codes are protected against malicious modifications by the security enclave of Zynq UltraScale+ MPSoC. Its memory and peripherals are protected with the Xilinx memory protection unit (XMPU) and the Xilinx peripheral protection unit (XPPU) [97]. Data communication and control between the host PC and the FPGA board are carried out via the UART and Ethernet interfaces.

Fig. 4.6 Experimental setup for mutation control of DNN implemented on Xilinx ZCU102 evaluation board.

4.3.2 Dataset

ImageNet [98] provided by the ImageNet Large Scale Visual Recognition Challenge 2012 for image classification tasks is used as the input dataset. It contains 1.2 million training images and 50,000 validation images. All of them are human-annotated with 1000 classes. Two models with different DNN architectures are used for the evaluation: VGG16 [46] and

Table 4.2 Training and testing datasets for VGG16 and ResNet50.

Model	Dataset	Attacks	#Initial	#Clean	#Tampered
VGG16	Training	FGSM ($\epsilon = 4/255$)	700	452	452
		I-FGSM ($\epsilon = 4/255$)	350	241	241
	Testing	FGSM ($\epsilon = 4/255$)	500	295	295
		FGSM ($\epsilon = 8/255$)	250	163	163
		I-FGSM ($\epsilon = 4/255$)	250	161	161
		C&W ($\kappa = 20$)	300	185	185
		UP	400	264	264
ResNet50	Training	FGSM ($\epsilon = 4/255$)	600	320	320
		I-FGSM ($\epsilon = 28/255$)	400	320	320
	Testing	FGSM ($\epsilon = 4/255$)	430	230	230
		FGSM ($\epsilon = 8/255$)	420	230	230
		I-FGSM ($\epsilon = 4/255$)	360	230	230
		I-FGSM ($\epsilon = 28/255$)	320	230	230
		C&W ($\kappa = 20$)	550	391	391
		UP	320	230	230

ResNet50 [47]. To avoid bias in the training process, the training set consists of an equal number of clean and adversarial images. Table 4.2 shows the number of training and testing images and the algorithms used for the generation of their adversarial examples for each DNN model. #Initial refers to the number of clean images initially selected for crafting the adversarial examples. #Clean and #Tampered refer to the number of correctly classified clean images and their corresponding successfully misclassified adversarial images, respectively. For each model, the same training set is used for the construction of the two proposed decision tree detectors. 75% of them are used for the training and 25% of them are used for the validation.

The reason for adding I-FGSM-eps4 (I-FGSM with $\epsilon = 4/255$) images to the training dataset for VGG16 is because the prediction probabilities for adversarial images generated by FGSM-eps4 (FGSM with $\epsilon = 4/255$), FGSM-eps8 (FGSM with $\epsilon = 8/255$) and UP are found to be much lower than those generated by I-FGSM-eps4 and C&W, as shown in the box-and-whisker plot in Fig 4.7, where the $\square$, $-$ and $\times$ symbols denote the mean, maximum and minimum values, and 1 and 99 percentiles, respectively. Most of the adversarial samples generated by FGSM-eps4, FGSM-eps8 and UP tend to lie near the decision boundaries while most of the adversarial samples generated by I-FGSM-eps4 and C&W lie far from the decision boundaries. Therefore, if only FGSM-eps4 is considered in the training process, the solution will tend to pick glitch injections with smaller mutations, which have almost no influence on the high-confidence adversarial images. Similarly, if only I-FGSM-eps4 is considered, the solution

Fig. 4.7 Box-and-whisker plot of the top-1 probabilities for the clean images and the adversarial images generated by various attacks for VGG16

Fig. 4.8 Box-and-whisker plot of the top-1 probabilities for the clean images and the adversarial images generated by various attacks for ResNet50

will tend to pick glitch injections with larger mutations, which can distort the low-confidence clean samples. For the detection of adversarial inputs with different confidence levels, the training set should cover both mutation strategies. Similarly, for ResNet50, the adversarial

4.3 Experimental Results

images generated by I-FGSM-eps28 (I-FGSM with $\epsilon = 28/255$) are added to the training set. By comparing the box-and-whisker plots in Fig. 4.8 for ResNet50 and Fig. 4.7 for VGG16, C&W and I-FGSM are able to generate adversarial examples with much higher confidence for VGG16 than ResNet50 under the same perturbation parameter setting. This implies that VGG16 is more vulnerable to the C&W and I-FGSM attacks than ResNet50. The tree depth, number of non-terminal nodes, non-terminal node types, glitch patterns and threshold values in the first three levels of the trained classical and lightGBM detectors are summarized in Table 4.3. Glitch patterns, $(fc7, 10)$ and $(fc8, 2)$, have the largest ΔLCP for VGG16 in Fig. 4.3. As expected, they appear most frequently in the mutation tests of both decision tree detectors for VGG16.

Table 4.3 Parameters of the final classical and lightGBM decision trees (DTs) for VGG16 and ResNet50.

Model	DTs	Depth	#Nodes	Node Type	τ
VGG16	Classical	2	3	$v_{1,1}: D_{(fc7,10)}$	-0.579
				$v_{2,1}: D_{(fc8,2)}$	-0.023
				$v_{2,2}: D_{(fc8,2)}$	-0.314
	LightGBM	5	20	$v_{1,1}: D_{(fc7,10)}$	-0.580
				$v_{2,1}: D_{(fc8,2)}$	0.321
				$v_{2,2}: D_{(fc8,2)}$	0.312
				$v_{3,1}: D_{(conv4_1,20)}$	-1.065
				$v_{3,2}: D_{(fc8,2)}$	0.021
				$v_{3,3}: M_{(fc8,2)}$	0.070
				$v_{3,4}: M_{(fc7,10)}$	-0.009
ResNet50	Classical	6	17	$v_{1,1}: D_{(res2a_branch2a,0.2)}$	0.111
				$v_{2,1}: M_{(res2a_branch2b,0.2)}$	0.797
				$v_{2,2}: D_{(res5c_branch2a,2)}$	0.079
				$v_{3,1}: M_{(res2c_branch2a,20)}$	-0.244
				$v_{3,2}: D_{(res5a_branch2b,0.2)}$	-0.149
				$v_{3,3}: M_{(res2a_branch2a,10)}$	0.244
	LightGBM	4	9	$v_{1,1}: D_{(res2a_branch2c,0.1)}$	0.075
				$v_{2,1}: M_{(res2b_branch2a,20)}$	-0.151
				$v_{2,2}: D_{(res4a_branch2c,2)}$	0.026
				$v_{3,1}: D_{(res2b_branch2a,20)}$	-0.625
				$v_{3,2}: D_{(res2b_branch2a,20)}$	-1.145
				$v_{3,4}: D_{(res2c_branch2a,2)}$	0.069

4.3.3 Results

Table 4.4 Comparison of classical and lightGBM tree detectors in true positive rates (TPRs) and false positive rates (FPRs) for VGG16 and ResNet50.

Models	DTs	Depth	Attacks	FPR (%)	TPR (%)
VGG16	Classical	2	FGSM($\epsilon = 4/255$)	21.36	79.66
			FGSM($\epsilon = 8/255$)	18.40	77.30
			I-FGSM($\epsilon = 4/255$)	18.63	37.27
			C&W($\kappa = 20$)	13.51	15.14
			UP	15.91	92.80
	LightGBM	5	FGSM($\epsilon = 4/255$)	11.53	64.41
			FGSM($\epsilon = 8/255$)	5.52	66.26
			I-FGSM($\epsilon = 4/255$)	6.83	65.22
			C&W($\kappa = 20$)	10.81	74.05
			UP	4.55	85.61
ResNet50	Classical	6	FGSM($\epsilon = 4/255$)	6.09	85.22
			FGSM($\epsilon = 8/255$)	6.97	84.35
			I-FGSM($\epsilon = 4/255$)	3.91	88.26
			I-FGSM($\epsilon = 28/255$)	5.65	81.30
			C&W($\kappa = 20$)	3.84	79.54
			UP	4.35	91.74
	LightGBM	4	FGSM($\epsilon = 4/255$)	0.43	92.17
			FGSM($\epsilon = 8/255$)	0.43	93.91
			I-FGSM($\epsilon = 4/255$)	0	90.43
			I-FGSM($\epsilon = 28/255$)	0.43	88.70
			C&W($\kappa = 20$)	0	92.07
			UP	0.87	93.91

Table 4.4 shows the results of the two proposed decision trees on unseen adversarial inputs presented to the hardware accelerated VGG16 and ResNet50. The results show that lightGBM has a much lower FPR than the classical tree detector for all the attacks on both models. This is owing to the inherent loss control of the lightGBM training algorithm. For ResNet50, lightGBM achieves higher TPR than the classical tree detector for all the attacks. For VGG16, the classical decision tree algorithm achieves approximately 23.7%, 16.7% and 8.4% higher detection rates than lightGBM for the FGSM-eps4, FGSM-eps8 and UP attacks, respectively. However, the detection rates of lightGBM are significantly higher for the I-FGSM-eps4 and

C&W attacks. They are increased by nearly 75% and 389.1%, respectively compared with the classical detector. This shows that the mutation strategies chosen by level-wise growth are more effective in discriminating the low-confidence adversarial examples but not the high-confidence adversarial examples like the I-FGSM-eps4 and C&W attacks for VGG16. The difference between the level-first and best-first strategies can be observed from their different improvements in TPR from the initial TPR after the first attribute test at the root node computed based on the temporary classes. The final TPR of the classical tree detector for VGG16 has been increased by nearly 30.5% and 154.5% for the I-FGSM and C&W attacks, respectively, but remains unchanged for the FGSM and UP attacks. On the contrary, the final TPR of the lightGBM detector for VGG16 has been increased by approximately 22.6%, 50%, 87.5%, 356.5% and 13.6% from the TPR of the root node for the FGSM-eps4, FGSM-eps8, I-FGSM-eps4, C&W and UP attacks, respectively. Similarly, the final TPR of the classical tree detector for ResNet50 has been increased by nearly 37.1%, 35.7%, 41%, 65.5%, 241.8% and 6% from the TPR of the root node for the FGSM-eps4, FGSM-eps8, I-FGSM-eps4, I-FGSM-eps28, C&W and UP attacks, respectively. The final TPR of the lightGBM detector for ResNet50 has been increased by nearly 31.7%, 31.7%, 26.1%, 55.7%, 160.9% and 6.9% from the TPR of the root node for the FGSM-eps4, FGSM-eps8, I-FGSM-eps4, I-FGSM-eps28, C&W and UP attacks, respectively. For both models, both detectors have a greater propensity to select mutation strategies to discriminate the low-confidence adversarial examples before the high-confidence adversarial examples. This corroborates the necessity to include high-confidence adversarial inputs in the training set for the decision tree construction. It is also observed that both proposed decision tree detectors are most effective against the UP attack regardless of the model complexity. UP is input-agnostic. It is the most viable non-invasive attack to fool deep learning models executed on hardware accelerator platform.

Both detectors for ResNet50 have much lower FPRs and higher TPRs than VGG16 on all attacks. This implies that the proposed method works better for deeper neural networks on the same image classification application. Since the glitch patterns are derived and applied layer-wise, a deeper network provides more choices for optimizing the effective glitch injection patterns to discriminate the adversarial examples from the benign inputs in building the decision tree detector.

Table 4.5 lists the hardware overhead of the proposed detector. No additional LUTRAM, BRAM and DSP are required for the glitch injection circuit other than four additional architecture-independent global clock buffers (BUFGs) and a negligibly small number of LUTs and FFs. The trained decision tree detector is implemented on the ARM CortexTM-A53 processor. Without specific code optimization, the code sizes of the classical tree detector are $\sim 16\text{kB}$ and $\sim 85\text{kB}$ for VGG16 and ResNet50, respectively; and the code sizes of the lightGBM detector

Table 4.5 Hardware overhead of DNN with proposed clock glitch mutation control

Resource	Original Design	With Glitch Control	Overhead
LUT	108048	108114	0.06 %
LUTRAM	10939	10939	0 %
FF	203051	203095	0.02 %
BRAM	518	518	0 %
DSP	1394	1394	0 %
BUFG	9	13	44 %
MMCM	1	1	0 %

are $\sim 94\text{kB}$ and $\sim 50\text{kB}$ for VGG16 and ResNet50, respectively. The final tree detector can also be implemented as a small finite-state machine and configured onto the same FPGA chip as the deep learning accelerator. A mixture of 100 clean images and adversarial images generated by various attacks are used for assessing the processing time. Due to the complexity of VGG16, the original design takes 0.13s on average to complete the inference of one image. The sanity check of the inference can be completed by the classical and lightGBM tree detectors in 0.28s and 0.38s, respectively. For ResNet50, the original design takes 34.5ms on average to make an inference of one image. The sanity check can be completed by the classical and lightGBM tree detectors in 78ms and 68.8ms, respectively. Since each attribute test of the detector involves merely checking a prediction probability against a threshold, the latency is determined mainly by the time required to process an input image by the DNN for all the attribute tests with distinct glitch patterns. Furthermore, the decision tree is not balanced. The longest branch may not be exercised for many images. Also, the prediction probability does not need to be re-computed by the CNN if the node type is O or the glitch pattern of type M or D node has already been applied. Therefore, the speed ratios $\frac{Speed_{Classical}}{Speed_{Original}}$ and $\frac{Speed_{LightGBM}}{Speed_{Original}}$ are much smaller than the depths of the decision trees. In the experiments, the clock frequency of the DSP slices of the DPU is set to 200MHz to conserve power. It can be increased to 600MHz without affecting the prediction accuracy to accelerate the inference with and without the sanity check by a factor of 3. Recently, a new hardware-oriented CNN compression strategy [99] was reported to achieve 83.0 frames per second (FPS) for VGG16 implemented on a higher capacity and performance Virtex®UltraScale+™ FPGA VCU118 board. For this throughput rate of VGG16 in Xilinx VCU118, ~ 27 FPS is envisaged with the same sanity check. Since the current CNN implementation in the proposed work consumes $3\times$ less FPGA resources than that of [99], there is room in VCU118 to add a shadow CNN of the proposed implementation

for computing the prediction probability of an input when a new mutation is required. This way there is no need to stall the input image stream of the original CNN as the detector can check every alternate frame for ResNet50 and once every three frames for VGG16 using the shadow CNN.

4.3.4 Comparisons and Discussions

Table 4.6 compares the proposed lightGBM detector for VGG16 with several latest fully software-based adversarial input detection methods. All these existing methods require computationally intensive off-line analyses that are not amendable for edge device implementation, particularly for Xu2017 [20] and Liang2018 [77]. Both methods rely on comparing the prediction results between the original input and its filtered input for adversarial sample detection. These input filters require compute-intensive division operations. It should be noted that the results reported in Table 4.6 for Xu2017 [20] are evaluated on MobileNet, which is 176 times smaller than the VGG16 evaluated in the proposed method. Besides, except an overall FPR, the FPR for each individual attack is not reported in Xu2017 [20]. For a fair comparison, the evaluation results of retargeting the Xu2017 [20] algorithm to the 32-bit floating-point VGG16 model for the FGSM-eps4, FGSM-eps8, I-FGSM-eps4, C&W ($\kappa = 0$) and DeepFool attacks reported by [74] are also listed in Table 4.6. It is observed that the TPR of Xu2017 has degraded for all the attacks when the same method is evaluated on a larger model. In particular, its original TPRs for the FGSM and I-FGSM attacks have been reduced by 47.3% and 44.9%, respectively on VGG16. This is because larger perturbation strengths are adopted in [74] to produce the adversarial examples for these two attacks compared with the original evaluation of Xu2017. It can be argued that the TPR will be worse for the C&W adversarial examples with $\kappa = 20$ used in the proposed method. Comparing to the results of Xu2017 on VGG16, the proposed detector has increased the TPR significantly by 185.8%, 83.9% and 15.1% for the FGSM, I-FGSM and DeepFool-originated (including DeepFool and UP) attacks, respectively. This work also has a lower FPR for the DeepFool-originated attack. The results of Xu2017 are expected to be worse if the DNN model is implemented on hardware-accelerated platform due to the fixed-point arithmetic and quantization.

The results reported in Liang2018 [77] are based on three classes of ImageNet images and the 32-bit floating-point inception models in GPU implementation. For a fair comparison, Liang2018’s algorithm is retargeted to a 32-bit floating-point VGG16 model and tested with the images randomly selected from 1000 classes of ImageNet dataset. The retarget results are shown under the column ‘Liang2018GPU’ in Table 4.6. Liang2018 [77] is also retargeted to the DPU-accelerated 8-bit quantized VGG16 model implemented on FPGA. Unfortunately, the Python code provided by [77] to construct the filter is too complex to be ported onto the ARM

Table 4.6 Comparison of the proposed lightGBM detector for VGG16 and existing adversarial sample detection methods.

Metrics	Wang2020 [89]	Wang2019 [88]	Liang2018GPU [77]	Liang2018FPGA [77]	Xu2017 [20]	This work
Implementation	GPU	GPU	GPU	FPGA	GPU	FPGA
Defense Category	Sample Statistics	Prediction Inconsistency	Prediction Inconsistency	Prediction Inconsistency	Prediction Inconsistency	Prediction Inconsistency
Method	Fired Neuron Rate	Weight Mutation	Adaptive Noise Reduction	Adaptive Noise Reduction	Feature Squeezing	Glitch Injection
Dataset	ImageNet	ImageNet	ImageNet	ImageNet	ImageNet	ImageNet
Model	(8-bit quantized) VGG16	(8-bit quantized) VGG16	(32-bit) VGG16	(8-bit quantized) VGG16	(32-bit) MobileNet / (32-bit) VGG16	(8-bit quantized) VGG16
TPR (%)	FGSM: N.A.	N.A.	65.73	60.92	43.40 / 22.86	65.34
	I-FGSM: N.A.	N.A.	53.62	42.86	64.40 / 35.47	65.22
	C&W: N.A.	N.A.	67.31	55.14	97.93 / 89.33	74.05
FPR (%)	UP: 91.90	82.00	90.41	92.80	78.60 / 74.41 (DeepFool)	85.61
	FGSM: N.A.	N.A.	37.08	33.41	5.37	8.53
	I-FGSM: N.A.	N.A.	18.84	16.15	5.60	6.83
FPR (%)	C&W: N.A.	N.A.	19.23	14.59	8.33	10.81
	UP: 6.70	5.67	47.95	34.47	(overall) 5.24	4.55
					(DeepFool)	

A53 processor in the evaluation platform adopted in the proposed method. Therefore, this code is run on a PC equipped with an E5-1630 v4 3.70GHz CPU, 16GB system memory, and a GeForce GTX 1070Ti to produce the filtered test images. Both the original and corresponding filtered test images are then fed into the same deep learning accelerator for inference. The detection results so obtained are listed under the column ‘Liang2018FPGA’ in Table 4.6. As expected, the TPR and FPR of Liang2018 have degraded for all the attacks when the same method is evaluated on a larger model with more different classes of test images. In particular, the TPRs for the FGSM and C&W attacks have been reduced from those of Liang2018 by 21.6% and 32.7%, respectively for Liang2018GPU, and by 27.3% and 44.9%, respectively for Liang2018FPGA. This is because the adversarial images used in Liang2018 [77] fall in the low-confidence region. Their adversarial inputs are produced with weaker adversarial perturbation strengths (FGSM-eps1, FGSM-eps2, C&W with $\kappa \in [0, 4]$). The test images used in the proposed method include both the low-confidence and the high-confidence adversarial inputs. Consequently, the TPRs of Liang2018GPU and Liang2018FPGA for the UP attack are not affected as significantly as it is for the C&W ($\kappa = 20$) attack. Comparing with Liang2018FPGA, the proposed detector has approximately 52.2% and 34.3% higher TPR for I-FGSM and C&W, respectively. More importantly, the FPRs of the proposed detector are significantly lower than those of Liang2018FPGA by 74.5%, 57.7%, 25.9% and 86.8% for FGSM, I-FGSM, C&W and UP, respectively. Comparing to Wang2020 [89] and Wang2019 [88], the proposed detector has a lower FPR. The TPR is higher than that of Wang2019 for the UP attack. Unlike the proposed tree detector, Wang2020’s and Wang2019’s methods consider only the UP attack but not the FGSM, I-FGSM and C&W attacks.

Table 4.7 Comparison of the largest scale of DNN evaluated between the proposed lightGBM detector and DeepFense [7].

	Input Size	#Model Weight	Classification Scale
This Work	$224 \times 224 \times 3$	138M	1000
DeepFense [7]	$32 \times 32 \times 3$	1.37M	10

Two hardware-oriented adversarial input detection schemes, namely DeepFense [7] and DNNGuard [8], have been reported recently. DeepFense [7] was only evaluated on small-scale (10 classes) classification problem with very small input data size ($\leq 32 \times 32$). Table 4.7 shows the differences between this work and DeepFense in terms of the input image size, model size and classification scale. This method has been evaluated on DNN classification of model size and class scale up to $100\times$ larger than those of [7]. Since the scale of DeepFense is largely dependent on the classification complexity and model size, its effectiveness and efficiency are expected to drop substantially when it is scaled up for 1000-class applications

and larger models. DNNGuard [8] is an ASIC design in 65nm CMOS technology based on the NVIDIA Deep Learning Accelerator (NVDLA) architecture. Strictly speaking, DNNGuard is preached as a deep learning accelerator architecture that can simultaneously run the victim and existing detection algorithms/networks rather than a new detection method. There is no validation against adversarial input detection accuracy but only simulated hardware figures of merit reported in [8]. The reported hardware overhead is based on the synthesized design of DNNGuard against the non-optimized two-thread parallel computing architecture. Table 4.8 shows the comparison of the mutation circuit overhead of the proposed lightGBM in FPGA implementation and the reported hardware overhead of DNNGuard [8]. Compared with DNNGuard, the LUT (Combinational cell) and register (Sequential cell) overheads of the proposed detector are at least $135\times$ and $265\times$ lower, respectively. Taking into consideration that the results reported in [8] are obtained by pre-layout simulation, the actual performance and hardware area of DNNGuard will be worse.

Table 4.8 Comparison of hardware overhead between the proposed lightGBM detector and DNNGuard [8].

	Implementation	LUT Overhead (%)	FF Overhead (%)
This Work	Physical	0.06	0.02
DNNGuard [8]	Simulated	> 8.1	> 5.3

4.4 Summary

This chapter presents the design methodology of a new lightweight adversarial input detector for large-scale DNN application. As an exemplification, the proposed detectors for VGG16 and ResNet50 were implemented on the DPU of a Xilinx Zynq UltraScale+ MPSoC. It is demonstrated that with very limited manipulable space allowed by the DNN accelerator and without modifying the commercial DPU IP, clock glitches can be exploited advantageously to discriminate adversarial inputs from clean inputs by introducing biased computational disruptions in selected convolutional layers. A tree detector can be trained to determine the optimal intensity and timing of glitches and the probability threshold to apply at each node. The accuracy of the proposed detector has been evaluated by applying four prevalent attacks on randomly selected images from ImageNet dataset. A significant improvement in detection rate by up to 185.8%, 83.9%, 34.3% and 15.1%, and a substantial reduction in false positive rate by up to 74.5%, 57.7%, 25.9% and 86.8% for FGSM, I-FGSM, C&W and UP, respectively, have been achieved over existing software-oriented approaches. With only 0.06% of LUT and 0.02%

of FF overheads and a compact detector code size of less than 100 kB for a thousand-class DNN, this is perhaps the smallest adversarial input detector that has ever been physically demonstrated on a FPGA edge computing platform.

Chapter 5

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

5.1 Introduction

The successes in deep learning (DL) applications to real-world problems have been made possible by training the DNN model with enormous amount of labeled data and computing power. Both require huge capital investment in manpower, time and physical resources, which may not be affordable or accessible by small corporations. The trained model is a pricey proprietary asset of a business and is normally not disclosed publicly. Even then, due to the strong incentive, model extraction [28–31] and reverse engineering [32] have been used by business opponents to rebuild similar artificial intelligent (AI) product or solution to the same problem from a trained model to gain unfair competitiveness. Unfortunately, it is difficult to prove the ownership of a DNN model when the model is implemented onto different hardware platforms.

As discussed in Section 2.3.2, existing countermeasures to detect this deep learning intellectual property (IP) theft attack can be broadly dichotomized into watermarking and fingerprinting. The former can be achieved via either white-box or black-box approaches. A white-box strategy often embeds a secret message into the model internals during training. It extracts the watermark from a suspect DNN model and compares if it is similar to the target model [33]. However, this approach requires physical access to the suspect model for verification, which is not always feasible before getting into the judicial procedure. In contrast, the black-box strategy trains the target model on a small set of modified inputs acquired through either content perturbation or wrong labeling, apart from the original dataset [34–36]. This set of data is then served as the trigger to produce a unique prediction for the target model. During

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

verification, this trigger set is submitted to query the suspect model and the prediction results are compared against those obtained from the target model. The accessibility issue is resolved by sacrificing classification accuracy in most techniques of watermark embedding through the model training process. Unlike watermarking, no accuracy loss is incurred in fingerprinting methods, as signature embedding is avoided in the training process of the target model. It aims to generate robust inputs that have similar prediction results as the target model for a pirated or post-processed version of the target model, but are otherwise different for an original model [37, 38]. This set of robust inputs and their labels predicted by the target model form the fingerprint of the target model.

Previous fingerprinting methods either omit the trade-off in detection accuracy of different fingerprints extracted using the same mechanism or incur a comparatively large latency overhead in fingerprint extraction. In this chapter, a fast IP theft detection technique that makes use of DeepFool [68] algorithm is proposed to extract fingerprint of a target model. DeepFool intrinsically generates data points lying near the decision boundary. As a classifier can be characterized by its decision boundary, the generated data points as well as their labels predicted by the target model can be leveraged for fingerprinting. The matching rate, which is the proportion of fingerprint data points that have the same labels predicted by both the suspect and target classifiers, is used to optimize the fingerprint performance. The influence of different sets of fingerprint data points on the matching rate is investigated. Selected data points are encoded and verified by Bayes' Theorem [100] to provide a confidence level to determine if a suspect classifier possesses a valid fingerprint of a target model. An iterative process is utilized to obtain a prediction with either extremely high confidence that indicates a high likelihood of the suspect classifier being not original (i.e., pirated) or extremely low confidence that indicates a high possibility of the suspect classifier being original. The proposed DeepFool based fingerprinting method is evaluated on CIFAR-10, CIFAR-100 and ImageNet datasets. Four prevalent post-processing techniques that an attacker may apply to the pirated target model to evade detection are also considered. They are defined as follows:

Feature Extraction (FE): The weights of the final layer of the target model are updated using the training data. All other layers are fixed.

Fine-tuning (FT): All the parameters of the target model are updated using the training data.

Weight Pruning (WP): Less significant weights of the target model are pruned for producing lightweight model. The pruned model is usually retrained to enhance the classification accuracy.

Filter Pruning (FP): Less significant filters of the target model are pruned for model compression. The pruned model is usually retrained to restore the classification accuracy.

5.2 Proposed Fingerprinting Approach

5.2.1 Threat Model

Two parties are modelled in the threat model: model owner and attacker. The model owner has trained and deployed a DNN classifier as an AI service to the users. The target model internals are kept confidential. The attacker is assumed to be capable of stealing or pirating the parameters as well as the architecture of the target DNN classifier and deploying it as its own service. This is made possible via exploitation of side channel leakage such as cache misses [101, 30], memory access patterns [31, 102], power traces [103] and performance counters [104] during the model execution. The attacker may post-process the pirated model for performance improvement or adaptation to resource-constrained platforms before the deployment. The model owner could derive a fingerprint from his own classifier and use it to validate against a suspect classifier. To be realistic and to avoid dispute, it is assumed that the model owner can only perform the verification on a suspect DNN classifier in a black-box manner without being able to access to or intervene with its internal signals or computations. If the suspect classifier can produce a similar fingerprint as a target model, it is a positive suspect of pirating. The model owner can take further actions such as collecting additional evidence before filing a lawsuit. Otherwise, it is a negative suspect.

5.2.2 Proposed IP Protection Approach

The proposed IP protection approach is based on the assumption that a DL classifier can be characterized by its unique classification boundary. In other words, different classifiers will have different classification boundaries. Therefore, fingerprinting the classification boundary of the target classifier could give convincing evidence for model ownership verification. Generally, a set of data points near the classification boundary of the target classifier and their labels predicted by the target classifier are extracted as the fingerprint. If the predicted labels after querying a suspect classifier with the extracted fingerprint data points largely match those in the fingerprint, the suspect classifier is likely to be the pirated or post-processed version of the target classifier. The definition of the classification boundary, as well as the details of the proposed fingerprinting extraction and verification processes as shown in Fig. 5.1, are elaborated as follows:

Classification Boundary: Given an k -class classifier, its classification boundary is defined as follows [37]:

$$CB = \{x | \exists i, j, i \neq j \text{ and } g_i(x) = g_j(x) \geq \max_{t \neq i, j} g_t(x)\} \quad (5.1)$$

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

where CB is a set of input data points that define the classification boundary of the classifier, $g_i(x)$, $g_j(x)$ and $g_t(x)$ are the probabilities that the input data x is predicted to be in class i , j and t , respectively. In other words, a data point is claimed to be on the decision boundary of a given classifier if at least two classes have the same largest probability for this input data such that the classifier cannot decide the label of this input data.

Fig. 5.1 A flowchart for the proposed fingerprinting method.

Fingerprint Extraction: Given a target classifier C_t and a set of n randomly selected input data points $X = \{x_1, x_2, \dots, x_n\}$, the proposed method extracts a fingerprint by $f_{C_t} = \text{Extract}(C_t, X)$, where f_{C_t} and Extract are the fingerprint and the DeepFool algorithm, respectively. Since the set of n data points $\hat{X} = \{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n\}$ are generated from DeepFool by adding only minimum perturbations to the corresponding inputs X , they are among those located nearest to the decision boundary. Fig. 5.2 shows an example for a given input and its corresponding adversarial input generated by DeepFool. It can be observed that there is

5.2 Proposed Fingerprinting Approach

only very minor deviation between the largest probability and the second largest probability predicted for the adversarial input. The extracted fingerprint for an input X is given by $f_{C_t} = \{(\hat{x}_1, \hat{y}_1), (\hat{x}_2, \hat{y}_2), \dots, (\hat{x}_n, \hat{y}_n)\}$, where $\hat{x}_i$ and $\hat{y}_i$ are the i -th fingerprint data point and its label predicted by the target classifier C_t .

Fig. 5.2 A given clean input and its corresponding adversarial input generated by DeepFool.

Fingerprint Verification: Given a suspect DNN classifier C_s , the DNN model owner executes $Verify(f_{C_t}, C_s)$ to output a probability P as the confidence level of C_s to be a pirated or post-processed version of C_t . If $P \geq \tau_{pos}$, C_s is predicted to be similar to C_t with high confidence, and C_s is claimed to be a positive suspect classifier. If $P \leq \tau_{neg}$, C_s is predicted to be a different classifier unrelated to C_t with high confidence, and C_s is said to be a negative suspect classifier. If $\tau_{neg} < P < \tau_{pos}$, there is not enough confidence to claim or deny C_s as a pirated copy of C_t . In this situation, another set of randomly selected inputs are fed into the function $Extract()$ to generate another fingerprint from C_t . The verification function $Verify()$ is executed with the probability P calculated from both the newly produced fingerprint and the previous fingerprint. The process continues until P falls into either $[\tau_{pos}, 1]$ or $[0, \tau_{neg}]$. In $Verify()$, $f_{C_s} = \{(\hat{x}_1, y_1'), (\hat{x}_2, y_2'), \dots, (\hat{x}_n, y_n')\}$ are generated, where y_i' is the label predicted by C_s for the i -th fingerprint data point $\hat{x}_i$. Then, the matching rate $\frac{m}{n}$ is calculated, where m is the number of fingerprint data points that have the same labels predicted by C_s and C_t . Finally, P is calculated by applying the Bayes' Theorem [100]:

$$P = p(pos|\mathbf{r}) = \frac{p(pos)p(\mathbf{r}|pos)}{p(\mathbf{r})} \quad (5.2)$$

where pos and $\mathbf{r}$ are a positive class and an interval vector $(r_1, r_2, \dots, r_N)$, respectively. r_i represents the range or interval of the matching rate for the i -th fingerprint.

In order to calculate P , a frequency or prior probability table needs to be established based on the predetermined matching rate intervals. Therefore, M negative suspect classifiers and M positive suspect classifiers are generated. K fingerprints are produced and applied to each suspect classifier to investigate the matching rate bounds. Then, (5.2) can be further simplified as follows:

$$P = \frac{\sum_{j=1}^M \prod_{i=1}^N p_j(r_i|pos)}{\sum_{j=1}^M \prod_{i=1}^N p_j(r_i|pos) + \sum_{j=1}^M \prod_{i=1}^N p_j(r_i|neg)} \quad (5.3)$$

where $p_j(r_i|pos)$ represents the likelihood for the matching rate to fall in the interval r_i if it is the j -th positive suspect classifier, and $p_j(r_i|neg)$ represents the likelihood for the matching rate to fall in the interval r_i if it is the j -th negative suspect classifier.

5.3 Implementations and Results

5.3.1 Experimental Setup

Datasets and Target Classifiers

Three popular image classification datasets are considered in the experiment: CIFAR-10, CIFAR-100 and ImageNet-10. CIFAR-10 contains a total of 60000 color images with a resolution of 32×32 in 10 classes. It has 50000 images for training and 10000 images for testing. A pretrained ResNet20 [105], VGG11 [46] and DenseNet161 [48] are used as three different target classifiers for CIFAR-10 [106]. Similar to CIFAR-10, CIFAR-100 has the same number of color images with the resolution of 32×32 for training and testing, except that it is a 100-class classification problem. A pretrained ResNet20 [105], VGG11 [46] and DenseNet161 [48] are used as the target classifiers for CIFAR-100 [106]. Unlike CIFAR-10 and CIFAR-100, ImageNet is a dataset of color images with high resolution, provided by the ImageNet Large Scale Visual Recognition Challenge 2012. It consists of 1.2 million training images and 50000 validation images, all of which are labeled with 1000 categories. However, due to the limited memory, only the former 10 classes are taken for the evaluation. Therefore, only 12000 training images and 500 validation images are utilized in the experiment. This dataset is distinctively named as ImageNet-10. A fine-tuned ResNet18 [105] model is used as the target classifier for ImageNet-10.

Suspect Classifiers

There are two types of suspect classifiers: positive and negative, as defined in Section 5.2.1. Four commonly used post-processing techniques that can be applied to a pirated model are considered: feature extraction (FE), fine-tuning (FT), weight pruning (WP) and filter pruning (FP). For weight pruning, p fraction of weights that have the smallest absolute values are pruned, followed by a retraining to restore the classification accuracy. To limit the compromise of classification accuracy, p is initialized to 0.1 and incremented at a step size of 0.1 until the accuracy loss is larger than or equal to 3%. For filter pruning, a fraction c of filters that have the smallest sum of the absolute value of each entry in each layer are pruned, followed by a retraining to enhance the accuracy. c is initialized to $\frac{1}{16}$ and incremented with a step size of $\frac{1}{16}$ until the accuracy loss is larger than or equal to 3%. DNN models that have architectures different from the target classifier for each dataset are trained as the negative suspect classifiers. Details about the target and suspect classifiers for CIFAR-10, CIFAR100 and Imagenet-10 are shown in Tables 5.1, 5.2 and 5.3, respectively. The number of positive and negative suspect classifiers used for each target classifier is summarized in Table 5.4.

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

Table 5.1 Testing accuracies of the target and suspect classifiers for CIFAR-10. “-” means not applicable.

Classifier type	Classifiers	ResNet20	VGG11	DenseNet161
Target		91.65%	91.95%	94.24%
Positive	FE	91.89%	92.18%	94.10%
	FT	91.92%	89.34%	92.87%
	$p = 0.1$	91.78%	89.05%	93.01%
	$p = 0.2$	91.78%	90.58%	93.24%
	$p = 0.3$	91.79%	90.47%	92.80%
	$p = 0.4$	91.56%	90.29%	93.12%
	WP $p = 0.5$	91.41%	90.69%	93.10%
	$p = 0.6$	91.10%	90.31%	92.94%
	$p = 0.7$	-	90.67%	93.11%
	$p = 0.8$	-	90.40%	92.94%
	$p = 0.9$	-	90.48%	93.52%
	$c = 1 \setminus 16$	91.07%	90.37%	92.74%
	$c = 2 \setminus 16$	90.45%	90.19%	92.92%
	$c = 3 \setminus 16$	90.04%	90.47%	92.77%
	$c = 4 \setminus 16$	-	90.26%	92.79%
	FP $c = 5 \setminus 16$	-	90.07%	92.70%
	$c = 6 \setminus 16$	-	89.67%	92.77%
	$c = 7 \setminus 16$	-	89.59%	92.43%
	$c = 8 \setminus 16$	-	89.11%	92.18%
	$c = 9 \setminus 16$	-	-	92.13%
Negative	ResNet32		92.81%	
	ResNet44		93.24%	
	ResNet56		93.69%	
	VGG13		93.30%	
	VGG16		93.91%	
	VGG19		93.26%	
	DenseNet121		94.14%	
	DenseNet169		94.00%	
	GoogleNet		92.73%	
	MobileNet-v2		94.17%	

Table 5.2 Testing accuracies of the target and suspect classifiers for CIFAR-100. “-” means not applicable.

Classifier type	Classifiers	ResNet20	VGG11	DenseNet161
Target		66.61%	65.43%	71.61%
Positive	FE	65.62%	66.35%	72.65%
	FT	66.38%	67.42%	74.13%
	$p = 0.1$	66.86%	67.70%	73.96%
	$p = 0.2$	66.63%	67.61%	74.09%
	$p = 0.3$	66.46%	67.70%	73.96%
	$p = 0.4$	66.22%	67.79%	74.03%
	WP $p = 0.5$	64.66%	67.53%	73.90%
	$p = 0.6$	64.66%	67.82%	74.10%
	$p = 0.7$	-	67.74%	73.85%
	$p = 0.8$	-	67.77%	74.02%
	$p = 0.9$	-	67.75%	73.71%
	$c = 1 \setminus 16$	65.19%	67.39%	73.77%
	$c = 2 \setminus 16$	64.48%	66.71%	72.97%
	$c = 3 \setminus 16$	-	66.06%	72.40%
	FP $c = 4 \setminus 16$	-	65.57%	71.90%
	$c = 5 \setminus 16$	-	64.57%	71.51%
	$c = 6 \setminus 16$	-	63.71%	70.88%
	$c = 7 \setminus 16$	-	-	70.04%
	$c = 8 \setminus 16$	-	-	69.39%
Negative	ResNet32		68.74%	
	ResNet44		69.49%	
	ResNet56		70.79%	
	VGG13		67.49%	
	VGG16		32.77%	
	VGG19		67.68%	
	DenseNet121		62.90%	
	DenseNet169		70.45%	
	GoogleNet		59.19%	
	MobileNet-v2		65.50%	
	Inception-v3		63.78%	

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

Table 5.3 Testing accuracies of the target and suspect classifiers for ImageNet-10. “-” means not applicable.

Classifier type	Classifiers	Accuracy
Target	ResNet18	86.60%
Positive	FE	87.40%
	FT	86.00%
	WP	$p = 0.1$ 86.00%
		$p = 0.2$ 86.00%
		$p = 0.3$ 86.40%
		$p = 0.4$ 86.20%
		$p = 0.5$ 85.80%
	FP	-
Negative	ResNet34	88.20%
	ResNet50	90.20%
	VGG13	89.20%
	VGG16	90.60%
	VGG19	91.40%
	DenseNet121	92.20%
	DenseNet169	90.60%
	AlexNet	82.80%
	SqueezeNet	84.40%

Table 5.4 Datasets, target classifiers and numbers of positive and negative suspect classifiers.

Dataset	Target classifier	# Positive	# Negative
CIFAR-10	ResNet20	11	10
	VGG11	19	10
	DenseNet161	20	10
CIFAR-100	ResNet20	10	11
	VGG11	17	11
	DenseNet161	19	11
ImageNet-10	ResNet18	7	9

5.3.2 Results

The number of data points for each fingerprint is set to be 100 ($n = 100$). To investigate the matching rate bounds, 100 fingerprints are generated for each target classifier ($K = 100$). Tables 5.5, 5.6 and 5.7 show the matching rate range as well as the average matching rate for each suspect classifier for the 100 fingerprints targeting CIFAR-10, CIFAR-100 and ImageNet-10, respectively. It can be observed that positive suspect classifiers have larger matching rates

5.3 Implementations and Results

Table 5.5 Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for CIFAR-10.

Type	Model	ResNet20	VGG11	DenseNet161
Positive	FE	([0.40, 0.66], 0.5303)	([0.57, 0.80], 0.6937)	([0.75, 0.93], 0.8420)
	FT	([0.28, 0.49], 0.3668)	([0.14, 0.35], 0.2405)	([0.39, 0.64], 0.4928)
	$p = 0.1$	([0.28, 0.52], 0.4164)	([0.13 , 0.38], 0.2571)	([0.40, 0.65], 0.5044)
	$p = 0.2$	([0.30, 0.50], 0.4041)	([0.20, 0.45], 0.3190)	([0.42, 0.67], 0.5507)
	$p = 0.3$	([0.29, 0.53], 0.3959)	([0.23, 0.42], 0.3288)	([0.45, 0.65], 0.5384)
	$p = 0.4$	([0.24, 0.46], 0.3528)	([0.21, 0.44], 0.3201)	([0.37, 0.64], 0.5438)
	WP $p = 0.5$	([0.24, 0.44], 0.3283)	([0.22, 0.42], 0.3183)	([0.39, 0.72], 0.5309)
	$p = 0.6$	([0.20, 0.43], 0.2958)	([0.21, 0.44], 0.3189)	([0.43, 0.67], 0.5516)
	$p = 0.7$	-	([0.21, 0.45], 0.3163)	([0.43, 0.64], 0.5483)
	$p = 0.8$	-	([0.22, 0.51], 0.3361)	([0.45, 0.70], 0.5689)
	$p = 0.9$	-	([0.21, 0.51], 0.3250)	([0.46, 0.69], 0.5781)
	$c = 1 \setminus 16$	([0.18, 0.43], 0.3109)	([0.20, 0.48], 0.3167)	([0.40, 0.63], 0.5293)
	$c = 2 \setminus 16$	([0.17, 0.43], 0.2823)	([0.20, 0.48], 0.3191)	([0.44, 0.66], 0.5530)
	$c = 3 \setminus 16$	(0.15 , 0.35], 0.2366)	([0.17, 0.41], 0.3082)	([0.41, 0.63], 0.5268)
	$c = 4 \setminus 16$	-	([0.22, 0.41], 0.3092)	([0.42, 0.67], 0.5502)
	FP $c = 5 \setminus 16$	-	([0.21, 0.48], 0.3214)	([0.41, 0.62], 0.5080)
	$c = 6 \setminus 16$	-	([0.20, 0.41], 0.3000)	([0.41, 0.67], 0.5179)
	$c = 7 \setminus 16$	-	([0.21, 0.42], 0.3013)	([0.43, 0.62], 0.5356)
	$c = 8 \setminus 16$	-	([0.20, 0.39], 0.2915)	([0.37, 0.61], 0.4936)
	$c = 9 \setminus 16$	-	-	(0.33 , 0.56], 0.4510)
Negative	ResNet32	([0.08, 0.29], 0.1595)	([0.11, 0.32], 0.2342)	([0.26, 0.49], 0.3811)
	ResNet44	([0.07, 0.28], 0.1594)	([0.15, 0.32], 0.2304)	([0.28, 0.52], 0.3937)
	ResNet56	([0.08, 0.29], 0.1557)	([0.14, 0.32], 0.2335)	([0.28, 0.55], 0.3946)
	VGG13	([0.07, 0.24], 0.1368)	([0.11, 0.37], 0.2307)	([0.31, 0.53], 0.4251)
	VGG16	([0.04, 0.23], 0.1394)	([0.12, 0.36], 0.2361)	([0.28, 0.51], 0.3824)
	VGG19	([0.06, 0.24], 0.1408)	([0.14, 0.36], 0.2298)	([0.31, 0.55], 0.4103)
	DenseNet121	([0.05, 0.23], 0.1307)	([0.09, 0.32], 0.1898)	([0.27, 0.56], 0.4157)
	DenseNet169	([0.07, 0.25], 0.1406)	([0.10, 0.29], 0.1875)	([0.33, 0.60], 0.4450)
	GoogleNet	([0.08, 0.26], 0.1550)	([0.13, 0.31], 0.2177)	([0.25, 0.52], 0.3658)
	MobileNet-v2	([0.07, 0.28], 0.1523)	([0.11, 0.31], 0.2309)	([0.29, 0.54], 0.3991)

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

Table 5.6 Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for CIFAR-100.

Type	Model	ResNet20	VGG11	DenseNet161
Positive	FE	([0.27, 0.52], 0.4023)	([0.34, 0.57], 0.4505)	([0.35, 0.59], 0.4594)
	FT	([0.23, 0.46], 0.3390)	([0.26, 0.46], 0.3555)	([0.21, 0.47], 0.3636)
	$p = 0.1$	([0.34, 0.57], 0.4415)	([0.24, 0.48], 0.3634)	([0.26, 0.52], 0.3881)
	$p = 0.2$	([0.32, 0.56], 0.4255)	([0.27, 0.49], 0.3728)	([0.28, 0.53], 0.4018)
	$p = 0.3$	([0.27, 0.52], 0.3989)	([0.22, 0.45], 0.3521)	([0.29, 0.55], 0.4034)
	$p = 0.4$	([0.26, 0.49], 0.3687)	([0.22, 0.48], 0.3558)	([0.29, 0.52], 0.3889)
	WP	$p = 0.5$ ([0.23, 0.44], 0.3201)	([0.22, 0.49], 0.3542)	([0.29, 0.51], 0.3910)
		$p = 0.6$ ([0.18, 0.39], 0.2827)	([0.23, 0.49], 0.3640)	([0.29, 0.52], 0.3923)
		$p = 0.7$ -	([0.22, 0.50], 0.3675)	([0.25, 0.50], 0.3976)
		$p = 0.8$ -	([0.21, 0.48], 0.3541)	([0.29, 0.52], 0.4001)
		$p = 0.9$ -	([0.20, 0.47], 0.3496)	([0.28, 0.56], 0.4020)
		$c = 1 \setminus 16$ ([0.20, 0.44], 0.3024)	([0.15, 0.43], 0.3247)	([0.24, 0.47], 0.3656)
		$c = 2 \setminus 16$ ([0.16, 0.35], 0.2571)	([0.15, 0.40], 0.2903)	([0.24, 0.44], 0.3278)
		$c = 3 \setminus 16$ -	([0.13, 0.39], 0.2621)	([0.20, 0.42], 0.3046)
	FP	$c = 4 \setminus 16$ -	([0.13, 0.35], 0.2416)	([0.16, 0.40], 0.2804)
		$c = 5 \setminus 16$ -	([0.12, 0.32], 0.2243)	([0.16, 0.38], 0.2628)
		$c = 6 \setminus 16$ -	([0.09, 0.30], 0.1999)	([0.17, 0.33], 0.2459)
		$c = 7 \setminus 16$ -	-	([0.14, 0.31], 0.2289)
		$c = 8 \setminus 16$ -	-	([0.14, 0.30], 0.2168)
Negative	ResNet32	([0.07, 0.24], 0.1506)	([0.03, 0.22], 0.1277)	([0.06, 0.24], 0.1418)
	ResNet44	([0.06, 0.23], 0.1467)	([0.05, 0.24], 0.1262)	([0.08, 0.21], 0.1460)
	ResNet56	([0.08, 0.23], 0.1507)	([0.05, 0.22], 0.1293)	([0.07, 0.23], 0.1404)
	VGG13	([0.05, 0.22], 0.1339)	([0.06, 0.23], 0.1337)	([0.05, 0.26], 0.1408)
	VGG16	([0.03, 0.16], 0.0809)	([0.03, 0.16], 0.0895)	([0.02, 0.15], 0.0823)
	VGG19	([0.05, 0.25], 0.1288)	([0.05, 0.24], 0.1314)	([0.06, 0.21], 0.1345)
	DenseNet121	([0.04, 0.21], 0.1205)	([0.04, 0.21], 0.1217)	([0.05, 0.21], 0.1296)
	DenseNet169	([0.07, 0.23], 0.1272)	([0.07, 0.21], 0.1345)	([0.10, 0.24], 0.1704)
	GoogLeNet	([0.05, 0.20], 0.1219)	([0.02, 0.20], 0.1160)	([0.03, 0.21], 0.1280)
	MobileNet-v2	([0.05, 0.23], 0.1435)	([0.03, 0.24], 0.1300)	([0.07, 0.23], 0.1401)
	Inception-v3	([0.06, 0.20], 0.1259)	([0.05, 0.17], 0.1109)	([0.03, 0.22], 0.1231)

5.3 Implementations and Results

Table 5.7 Matching rate range and average matching rate for various suspect classifiers across 100 fingerprints for ImageNet-10.

Type	Model	ResNet18
Positive	FE	([0.24, 0.46], 0.3457)
	FT	([0.21, 0.40], 0.2877)
	$p = 0.1$	([0.44, 0.66], 0.5491)
	$p = 0.2$	([0.40, 0.65], 0.5387)
	$p = 0.3$	([0.32, 0.59], 0.4732)
	$p = 0.4$	([0.21, 0.49], 0.3876)
	$p = 0.5$	([0.17 , 0.40], 0.2718)
	FP -	-
Negative	ResNet34	([0.02, 0.16], 0.0883)
	ResNet50	([0.03, 0.15], 0.0781)
	VGG13	([0.02, 0.16], 0.0730)
	VGG16	([0.02, 0.13], 0.0663)
	VGG19	([0.02, 0.14], 0.0713)
	DenseNet121	([0.02, 0.14], 0.0683)
	DenseNet169	([0.02, 0.14], 0.0713)
	AlexNet	([0.03, 0.16], 0.0851)
	SqueezeNet	([0.03, 0.18], 0.0894)

Table 5.8 Overlap in matching rate and gap in average matching rate between positive and negative suspect classifiers for each target classifier.

Dataset	Target classifier	Overlap interval	Gap
CIFAR-10	ResNet20	[0.15, 0.29]	0.0771
	VGG11	[0.13, 0.37]	0.0044
	DenseNet161	[0.33, 0.60]	0.0060
CIFAR-100	ResNet20	[0.16, 0.25]	0.1064
	VGG11	[0.09, 0.24]	0.0654
	DenseNet161	[0.14, 0.26]	0.0464
ImageNet-10	ResNet18	[0.17, 0.18]	0.1824

than the negative suspect classifiers for a given fingerprint most of the time. Furthermore, the matching rate of a positive suspect weight-pruned classifier decreases with increasing p . Similarly, the matching rate of a positive suspect filter-pruned classifier decreases with increasing c . This is attributable to the possibly larger variation in the decision boundary when more weights or filters are pruned. Furthermore, the negative suspect classifiers which belong to the same network family as the target classifier tend to have larger matching rates. This

implies that network architecture may also influence the appearance of classification boundary. Table 5.8 shows the overlap in matching rates and the gap in average matching rates between positive and negative suspect classifiers for each target classifier.

To increase the separability between positive and negative suspect classifiers with high confidence, the verification is iteratively processed as proposed in Section 5.2.2. Based on the overlap interval, the matching rates are divided into three bins: $[0, \min)$, $[\min, \max+0.1)$ and $[\max+0.1, 1]$, where $\min$ and $\max$ are the lower and upper bound of the overlap interval. Since matching rate for a given positive classifier never falls into the bin $[0, \min)$ and matching rate for a given negative suspect classifier never falls into the bin $[\max+0.1, 1]$, only the middle interval $[\min, \max+0.1)$ needs to be further divided to reduce the number of iterations during verification. In this work, this interval is divided into multiple bins such that the difference between the lower and upper bounds of each bin is 0.05. For example, the matching rates are firstly divided into three bins based on the overlap interval for target model ResNet20 targeting CIFAR-10 classification: $[0, 0.15)$, $[0.15, 0.3)$ and $[0.3, 1]$. Then, the middle interval is further divided into three bins with a bin width of 0.05: $[0.15, 0.2)$, $[0.2, 0.25)$ and $[0.25, 0.3)$. The positive and negative thresholds for the confidence level are set to be 0.9 and 0.1 ($\tau_{pos} = 0.9$, $\tau_{neg} = 0.1$), respectively. Based on these parameters, a prior probability table with respect to a target classifier can be built for the probability calculation required by *Verify()* for an unseen fingerprint. Tables 5.9, 5.10 and 5.11 present the matching rate vectors $\mathbf{m} = (\frac{m_1}{n}, \frac{m_2}{n}, \dots, \frac{m_N}{n})$ and probability vectors $\mathbf{P} = (P_1, P_2, \dots, P_N)$ of various suspect classifiers for CIFAR-10, CIFAR-100 and ImageNet-10 datasets, respectively. The notations $\frac{m_i}{n}$, P_i and N are the matching rate for the i -th newly generated fingerprint, the probability output of *Verify()* in the i -th round and the total number of iterations required for the verification, respectively. All the suspect classifiers can be correctly recognized with a high confidence within three rounds. The average number of iterations required for validation is approximately inversely proportional to the average matching rate gap presented in Table 5.8. For example, ResNet18 applied on ImageNet-10 dataset is shown to have the largest average matching rate gap. All the suspect classifiers for this target model are shown to require only one round for validation. In contrast, VGG11 applied on CIFAR-10 dataset is shown to have the smallest average matching rate gap. Consequently, most of the suspect classifiers for VGG11 applied on CIFAR-10 dataset require multiple rounds for validation. The time elapsed for the fingerprint extraction for each target classifier is also presented in Table 5.12. It can be seen that the run time is approximately proportional to the model size and the number of layers of the target classifier.

Table 5.9 Matching rate vectors **m** and probability vectors **P** of various suspect classifiers for CIFAR-10 using the proposed method.

Type	Model	ResNet20	VGG11	DenseNet161
Positive	FE	0.53, 1	0.61, 1	0.87, 1
	FT	0.38, 1	(0.29, 0.29, 0.3), (0.76, 0.89, 0.96)	(0.46, 0.52), (0.37, 0.9)
	$p = 0.1$	0.40, 1	(0.29, 0.31, 0.31), (0.76, 0.89, 0.96)	0.56, 0.96
	$p = 0.2$	0.43, 1	0.35, 0.98	0.52, 0.96
	$p = 0.3$	0.36, 1	0.34, 0.98	0.54, 0.96
	$p = 0.4$	0.27, 0.94	(0.32, 0.28, 0.32), (0.76, 0.89, 0.96)	0.62, 1
	$p = 0.5$	0.32, 1	0.39, 1	0.57, 0.99
	$p = 0.6$	0.32, 1	0.36, 0.98	0.62, 1
	$p = 0.7$	-	(0.3, 0.27, 0.37), (0.76, 0.65, 0.99)	0.59, 0.99
	$p = 0.8$	-	0.34, 0.98	0.59, 0.99
	$p = 0.9$	-	0.34, 0.98	0.62, 1
	$c = 1 \setminus 16$	0.30, 1	(0.27, 0.27, 0.38), (0.41, 0.34, 1)	0.53, 0.96
	$c = 2 \setminus 16$	0.32, 1	0.34, 0.98	0.57, 0.99
	$c = 3 \setminus 16$	0.29, 0.94	0.34, 0.98	0.54, 0.96
	$c = 4 \setminus 16$	-	0.33, 0.98	0.58, 0.99
	FP	-	0.34, 0.98	0.52, 0.96
Negative	$c = 5 \setminus 16$	-	(0.31, 0.29, 0.4), (0.76, 0.89, 1)	(0.51, 0.49, 0.49), (0.74, 0.85, 0.9)
	$c = 6 \setminus 16$	-	(0.31, 0.3, 0.3), (0.76, 0.89, 0.96)	0.54, 0.96
	$c = 7 \setminus 16$	-	(0.31, 0.28, 0.34), (0.76, 0.89, 1)	0.54, 0.96
	$c = 8 \setminus 16$	-	-	(0.46, 0.52), (0.37, 0.9)
	$c = 9 \setminus 16$	-	-	(0.47, 0.47, 0.47), (0.74, 0.85, 0.9)
	ResNet32	0.13, 0	(0.18, 0.18), (0.16, 0.08)	0.35, 0.02
	ResNet44	(0.2, 0.19, 0.1), (0.48, 0.17, 0)	(0.21, 0.22), (0.16, 0.08)	0.38, 0.1
	ResNet56	0.14, 0	(0.22, 0.22), (0.16, 0.08)	(0.43, 0.37, 0.32), (0.37, 0.11, 0.02)
	VGG13	0.15, 0.07	(0.23, 0.21, 0.21), (0.41, 0.16, 0.09)	(0.42, 0.41, 0.41), (0.37, 0.11, 0.05)
	VGG16	0.14, 0	(0.22, 0.27, 0.22), (0.16, 0.16, 0.09)	0.37, 0.1
	VGG19	0.12, 0	(0.25, 0.22, 0.18), (0.41, 0.16, 0.09)	0.34, 0.02
	DenseNet121	0.17, 0.07	(0.19, 0.2), (0.16, 0.08)	0.35, 0.02
	DenseNet169	0.16, 0.07	(0.19, 0.22), (0.16, 0.08)	(0.44, 0.35), (0.37, 0.04)
	GoogleNet	0.13, 0	(0.21, 0.24, 0.21), (0.16, 0.16, 0.09)	0.4, 0.1
	MobileNet-v2	0.18, 0.07	(0.21, 0.18), (0.16, 0.08)	(0.46, 0.37, 0.39), (0.37, 0.11, 0.05)

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

Table 5.10 Matching rate vectors $\mathbf{m}$ and probability vectors $\mathbf{P}$ of various suspect classifiers for CIFAR-100 using the proposed method.

Type	Model	ResNet20	VGG11	DenseNet161
Positive	FE	0.4, 1	0.37, 1	0.43, 1
	FT	0.37, 1	(0.24, 0.38), (0.87, 1)	0.37, 1
	$p = 0.1$	0.34, 1	0.29, 1	0.39, 1
	$p = 0.2$	0.38, 1	0.35, 1	0.43, 1
	$p = 0.3$	0.4, 1	0.28, 1	0.43, 1
	$p = 0.4$	0.34, 1	0.36, 1	0.42, 1
	WP	$p = 0.5$	0.35, 1	0.41, 1
		$p = 0.6$	0.33, 1	0.43, 1
		$p = 0.7$	0.34, 1	0.42, 1
		$p = 0.8$	0.31, 1	0.41, 1
		$p = 0.9$	0.28, 1	0.39, 1
		$c = 1 \setminus 16$	0.25, 1	0.36, 1
		$c = 2 \setminus 16$	(0.19, 0.29), (0.21, 1)	0.29, 1
		$c = 3 \setminus 16$	0.25, 1	0.23, 0.97
	FP	$c = 4 \setminus 16$	0.25, 1	0.24, 0.97
		$c = 5 \setminus 16$	(0.21, 0.25), (0.87, 1)	0.25, 0.97
		$c = 6 \setminus 16$	(0.21, 0.23), (0.87, 0.98)	(0.22, 0.2, 0.23), (0.51, 0.6, 0.96)
		$c = 7 \setminus 16$	-	0.3, 1
		$c = 8 \setminus 16$	-	0.29, 1
Negative	ResNet32	0.16, 0.04	0.08, 0	0.13, 0.06
	ResNet44	0.16, 0.04	0.14, 0.03	0.16, 0.06
	ResNet56	0.14, 0	(0.15, 0.11), (0.21, 0.03)	0.13, 0.06
	VGG13	0.1, 0	0.11, 0.03	0.09, 0
	VGG16	0.15, 0	0.04, 0	0.11, 0
	VGG19	0.13, 0	0.1, 0.03	0.17, 0.06
	DenseNet121	0.14, 0	0.1, 0.03	0.15, 0.06
	DenseNet169	0.11, 0	0.13, 0.03	(0.19, 0.17, 0.17), (0.51, 0.13, 0.03)
	GoogleNet	0.14, 0	0.06, 0	0.11, 0
	MobileNet-v2	0.18, 0.04	0.1, 0.03	0.14, 0.06
	Inception-v3	0.12, 0	(0.18, 0.08), (0.21, 0.01)	0.16, 0.06

Table 5.11 Matching rate vectors $\mathbf{m}$ and probability vectors $\mathbf{P}$ of various suspect classifiers for ImageNet-10 using the proposed method.

Type	Model	ResNet18
Positive	FE	0.34, 1
	FT	0.31, 1
	$p = 0.1$	0.52, 1
	$p = 0.2$	0.51, 1
	$p = 0.3$	0.41, 1
	$p = 0.4$	0.32, 1
	$p = 0.5$	0.23, 1
	FP -	-
Negative	ResNet34	0.08, 0
	ResNet50	0.06, 0
	VGG13	0.06, 0
	VGG16	0.05, 0
	VGG19	0.07, 0
	DenseNet121	0.04, 0
	DenseNet169	0.06, 0
	AlexNet	0.06, 0
	SqueezeNet	0.05, 0

Table 5.12 Run time (in seconds) of generating 100 fingerprinting data points for each target classifier using the proposed method.

Dataset	Target classifier	# Layers	Model size (MB)	Run time (s)
CIFAR-10	ResNet20	20	1.06	20
	VGG11	11	491	38
	DenseNet161	161	102	67
CIFAR-100	ResNet20	20	1.08	17
	VGG11	11	492	44
	DenseNet161	161	103	75
ImageNet-10	ResNet18	18	42.7	24

5.3.3 Comparison with Two Recent Fingerprinting Methods

Table 5.13 summarizes the comparison between this work and the state-of-the-art fingerprinting methods. Matching rate bounds refer to the use of upper and lower boundaries of the matching rate for a suspect classifier with multiple different fingerprints extracted from a target classifier. Run time is the time elapsed for the fingerprint extraction, which is executed on a PC equipped with an Intel(R) Core(TM) i7-7700HQ 2.80GHz CPU, 16GB system memory, and a GeForce

Fingerprinting Deep Neural Networks (DNNs) - A DeepFool Approach

GTX 1060Ti. The average value of the run time for each dataset is presented in Table 5.13. While Cao’s work has a rapid growth in run time from CIFAR-10 to ImageNet, this work has a relatively stable execution time independent of dataset. One unique advantage of this method is the extraction of fingerprint data points is image agnostic. This has greatly enhanced the non-repudiability of ownership proof and piracy detection as the model can be validated by using fingerprint data points extracted from an arbitrary set of images. The other two works did not demonstrate this possibility.

Table 5.13 Comparison with two recent works.

Metrics	This work	Cao’s [37]	Lukas’s [38]
Method	DeepFool	Adam optimizer	C-BIM
Matching Rate Bounds	Yes	No	No
#Test Inputs (n)	100	100	-
#Target Classifiers	7	3	1
True Positive Rate	100%	100%	100%
False Positive Rate	0%	0%	0%
CIFAR-10	41.7	37.8	-
Run Time (s) CIFAR-100	45.3	249.9	-
ImageNet	24	7634.3	-

5.4 Summary

In this chapter, a DeepFool based fingerprinting method is proposed to detect the DNN IP theft attack. The fingerprint data points are generated by DeepFool from an arbitrary set of input data points and their labels predicted by the target classifier. Difference in matching rates between a pirated classifier and an originally designed classifier is observed for a given fingerprint. The Bayes’ Theorem is applied to provide a confidence level for a verification result to enhance its non-repudiability. An iterative verification is adopted to increase the confidence of a correct detection of a pirated classifier or its post processed version. Simulation results show 100% detection rate and 0% false positive rate with significantly faster fingerprint extraction from the target model for verification than two recently reported fingerprinting methods. The future work is to enlarge the matching rate gap between the positive and negative suspect classifiers to increase the robustness of the fingerprint. The splitting of matching rate interval will also be improved accordingly to further enhance the verification efficiency.

Chapter 6

Conclusion and Recommendations

6.1 Conclusion

Deep learning (DL), as a distinct representative of emerging machine learning (ML) techniques, has been given close attention since its first practical application for digit recognition in the late 1980s. Thirty years of development have bred various architectures for DL, which are different from the original feedforward fully connected (FC) multilayer perceptrons. Vast efficient processing technologies have also been proposed to deploy deep neural network (DNN) models on hardware platforms such as central processing unit (CPU), graphical processing unit (GPU), field programmable gate array (FPGA) and application-specific integrated circuit (ASIC), without compromising the predictive power. The rapid growth of data being processed and tremendous improvements in computing power, boosted by open-source software and cloud service in providing frameworks or computing and storage resources to speed up the design cycles, have largely contributed to the current proliferation of DL in a broad range of fields, including transportation, healthcare and manufacturing [107]. Many big companies such as Amazon, Google and Microsoft have already integrated DL into commercial systems and continue charting their future road map on DL applications. For example, Google smart reply auto-magically answers messages that hit a user's inbox; Apple Siri interprets what the iPhone user asks of her; Google's self-driving cars show the unimaginable potential of how DNNs change human life. The prowess and prevalence of DL will also attract malevolence and warrant more research effort and close scrutiny of the security in DNN design and deployment.

This thesis has conducted a comprehensive review of DNNs and existing defensive mechanisms against two major DNN security threats: input evasion and IP theft. It is found that: 1) existing countermeasures against input evasion are mainly evaluated at software level. The influence of hardware optimization techniques on the effectiveness of the defense is not testified when they are deploying the model on resource-constrained platforms; 2) existing adversarial

Conclusion and Recommendations

input detection methods are mainly designed for offline analysis. Their code size and computational complexity can rarely be integrated into the edge devices for in-situ identification. Furthermore, they are mostly designed independent of the deployed platforms. This can be the two sides of the same coin. On one hand, such device-agnostic property enables the defenses to be ported on any device without much change. On the other hand, once the attackers know the defensive mechanisms and come up with the corresponding defense evasion technique, they are able to attack multiple devices that adopt the same defenses; 3) existing DNN intellectual property (IP) piracy detection techniques do not demonstrate the non-repudiability of the ownership proof. These findings and deficiencies of existing research has motivated the exploration of solutions that support secure embedded intelligence where hardware acceleration is applied and computational resources are limited. A pragmatic solution that enhances the non-repudiability of the artificial intelligent (AI) chip piracy detection without reverse engineering the target DNN chip is also required to effectively protect the interest and revenue of the model owner.

To this end, two hardware-oriented adversarial image detection schemes, a new lightweight in-situ adversarial input detector for edge DL network and a DeepFool based fingerprinting method for model ownership verification have been proposed. The main contributions of this thesis can be summarized as follows:

- A hardware computation perturbation based philosophy is proposed to beat adversarial examples at their own games. Under this philosophy, a dominant layer directed discriminative noise injection method is first suggested, taking into consideration the data precision of hardware accelerators. Separability of adversarial and legitimate images is harnessed from their difference in sensitivity to different layer-wise parameter contaminations. The optimal configuration is obtained by a pre-characterization, which selects a small number of sensitive layers for mutation, their corresponding noise injection rates and label change rate thresholds. The detection efficiency is maximized with as small mutation rate and number of rounds as possible. Eight-bit truncated version of the three different DNN models, representative of fixed-point computations used in typical DNN hardware accelerators, are used for the evaluation. The results show improved detection efficiency compared with the state-of-the-art defenses that target software DNN classifiers. The random noise injection required by the proposed method can be realized by embedding a lightweight linear-feedback shift registers (LFSRs) into the DNN hardware accelerator. Another dominant layer approach based on fired neuron rate (FNR) without noise injection is also proposed for adversarial example detection. Greater distinguishability between adversarial and natural images are observed in their FNR distributions at some convolution layers. The optimal decision threshold at selected most discriminative layer (MDL) is obtained successively in a tree algorithm with a five-fold cross-validation. Eight-bit truncated version of four different DNN models

are used for the evaluation. The simulation results show a significant improvement in detection accuracy of adversarial inputs compared with feature squeezing method. Unlike the first proposed countermeasure, this method could achieve negligible latency overhead. Appropriate comparators can be more efficiently integrated into the identified MDLs of the DNN accelerator for thresholding if the read and write memory access patterns can be mapped into the layer-wise FNR.

- A design methodology of a new lightweight adversarial input detector for large-scale DNN application is proposed. The proposed detectors for VGG16 and ResNet50 were implemented on the deep learning processing unit (DPU) of FPGA edge as an exemplification. It is demonstrated that with very limited manipulable space allowed by the DNN accelerator and without modifying the commercial DPU IP, clock glitches can be exploited advantageously to discriminate adversarial inputs from clean inputs by introducing biased computational disruptions in selected convolutional layers. A tree detector classifier can be trained to determine the optimal intensity and timing of glitches and the threshold of the probability to apply at each node. The accuracy of the proposed detector has been evaluated by applying four prevalent attacks on randomly selected images from ImageNet dataset. A significant improvement in detection rate by up to 185.8%, 83.9%, 34.3% and 15.1%, and a substantial reduction in false positive rate by up to 74.5%, 57.7%, 25.9% and 86.8% against fast gradient sign method (FGSM), iterative-FGSM (I-FGSM), C&W and universal perturbation (UP), respectively, are achieved over existing software-oriented approaches. With only 0.06% of LUT and 0.02% of FF overheads and a compact detector code size of less than 100 kB for a thousand-class DNN, this is perhaps the smallest adversarial input detector that has ever been physically demonstrated on an edge computing platform.
- A DeepFool based fingerprinting method for detecting the DNN IP theft attack is proposed. The fingerprint data points are generated by DeepFool from an arbitrary set of input data points and their labels predicted by the target classifier. Difference in matching rates between a pirated classifier and an originally designed classifier is observed for a given fingerprint. To enhance the non-repudiability of the ownership proof, Bayes' Theorem is applied to provide a confidence level for the verification result. An iterative verification is adopted to increase the confidence of a correct detection of a pirated classifier or its post-processed version. The method is evaluated with seven target classifiers. Simulation results show 100% detection rate and 0% false positive rate with significantly faster fingerprint extraction from the target model for verification than two recently reported fingerprinting methods.

6.2 Recommendations for Further Research

It has been well said by P.S. Jagadeesh Kumar, the father of multidisciplinary research that “No Research is complete, every research is completely incomplete”. This thesis will not be complete without highlighting some potential new and follow-up enhancement and exploration arising from the incomplete research of the problems addressed in this thesis.

6.2.1 Effects of Emerging Vision Technologies on Physical Input Evasion

A lightweight in-situ adversarial input detector for edge DNN has been proposed in Chapter 4 of this thesis. However, this defense targets digital input evasion. Digital input evasion can be made via man-in-the-middle (MITM) attack, but it is often hard to be oblivious under tight time constraint. MITM attack is difficult for edge DNN applications when the image DNN is built into or near the sensor. Comparatively, physical perturbation is relieved from this constraint and does not required to be visually imperceptible. Visible changes on physical object like a graffiti or a sticker is often not considered suspicious in real life. However, physical evasion suffers from varying environmental conditions, such as changing distances to the object, different angles and lighting, and preprocessing of the captured image like cropping and scaling. Robust physical perturbation (RP₂) algorithm has been proposed to craft robust adversarial perturbations on real objects surviving from a variety of viewpoints, including significantly changing distances and angles [6]. The pipeline of RP₂ attack is shown in Fig. 6.1. Different image instances that encode both environmental variability and synthetic conversions are sampled to calculate an adversarial perturbation. The perturbation is then projected onto a mask to ensure that the position of the perturbation is within the surface area of the object. Finally, the derived mask is printed put and perturbed area is cut out and stucked onto the target object. To account for the fabrication loss that occurred during printing, an additional term, based on non-printability score [108], is added to the objective function for perturbation optimization.

However, existing physical input evasion attacks have only been evaluated on ordinary images taken by regular cameras. There are rooms for exploring emerging vision technologies such as optical filters, infrared and stereoscopic vision together with pre-processing techniques to effectively thwart physical input evasion attacks on DNN image classifiers. For example, three-dimensional (3D) images or stereoscopic photographs can be produced by using two or more lenses. Each lens has a separate image sensor or film frame. Different intra-axial distance of the twin lens can create different 3D and depth effects. Alternatively, 3D images can also be generated more economically by editing two images captured by the same camera with a movement of a few inches left or right. As physical objects are three dimensional,

Fig. 6.1 The pipeline of RP_2 attack [6].

incorporating the depth information and the different effects of reflected light from the 3D surfaces of an object into the training not only improves the robustness of DNN but also enables greater discriminability between the benign and adversarial inputs, including both physical adversarial examples as well as digital adversarial examples designed for conventional 2D images. In addition, the depth information of 3D objects taken in the physical world under different viewing angles and lighting conditions can also be exploited in pre-processing or post-processing algorithms for the detection of physical and digital adversarial examples even if the DNN model is originally trained with the inference made based on the 2D images.

6.2.2 Extension of the Proposed DeepFool based Fingerprinting Approach

A DeepFool based fingerprinting methodology has been proposed and detailed in Chapter 5 of this thesis. It is believed that the DeepFool algorithm can be further optimized to enlarge the matching rate gap between the positive and negative suspect classifiers for a target classifier to increase the robustness of the generated fingerprint. The splitting of the matching rate interval can then be improved accordingly to enhance the verification efficiency. Furthermore, the existing fingerprinting approaches including the proposed DeepFool based fingerprinting method do not time-stamping the model. Hence in the case that the attacker claims ownership of the pirated model based on the same methodology, the original model owner will have to provide additional chronological proof that the original model was developed earlier than the pirated version. Therefore, an additional process is needed to truly prove the ownership of an originally designed model, upon the IP theft detection.

Conclusion and Recommendations

Besides, this proposed DeepFool based fingerprinting approach is also likely to be effective against model tampering attack which usually occurs in cloud based DL services. Such services are termed as Machine Learning as a Service (MLaaS). MLaaS provides ML tools as a feature of cloud computing services. Service providers take care of the data storage and computation. This frees the corporate and consumer end users from the cost of installing the necessary infrastructure and software to take advantage of ML. Four Vital participants in the MLaaS market are Amazon Machine Learning, Azure Machine Learning, Google Cloud Machine Learning and IBM Watson Machine Learning. Customers of MLaaS can deploy their models online and release query APIs to the end users. However, deploying DL applications in MLaaS brings new security threats. The model owner loses the control of his model once the model is submitted to the cloud. The complicated cloud environment opens opportunities for attackers to deliberately tamper the remote model for their own purposes. Due to the black-box nature of the DNN, it is difficult for end users to identify the malpractice and verify the model integrity. However, if the fingerprint extracted using DeepFool is prone to model tampering, a set of fingerprint data points generated by DeepFool could be utilized to query the received model. If the predicted labels are mostly the same as the labels predicted by the model before cloud deployment, the received model is trustable. Otherwise, the received model is likely to be tampered or forged. As a result, the model integrity can be verified without access to the model internals. Future work in this direction may include:

- Study of different model tampering attacks.
- Investigate the sensitivity of samples generated by DeepFool to the model tampering attacks.
- If DeepFool samples are sensitive to the model tampering attacks, optimize the DeepFool algorithm to maximize the sensitivity. Then, a unified or adaptive fingerprint that can be re-purpose for either IP theft detection or model tampering detection technique by selection of different trigger sets can be developed.

6.2.3 Extension of the Proposed Techniques to Ultra-lightweight DNN Implementations

As mentioned in Section 2.2, ultra-lightweight DNNs such as binarized neural networks (BNNs) have emerged as popular choices for DNN implementations on IoT/edge devices for some applications. They approximate arithmetic operations using bitwise operations, resulting in reduced memory consumption and faster inference. It has also been shown that BNNs or quantized neural networks provide more robustness against adversarial example attacks

6.2 Recommendations for Further Research

than their full-precision counterparts [109, 110]. However, such evaluations have only been conducted on small-scale datasets like MNIST and CIFAR-10. While the compact BNNs do not suffer much accuracy loss on the CIFAR-10 dataset, comparable results can only be achieved on the larger ImageNet dataset with some improvements and modifications to the original BNN methodology. Whether the same robustness against adversarial attack can be transferred to large-scale dataset like ImageNet remains a mystery. Hence, the proposed in-situ adversarial sample detector can be extended to BNNs to investigate the influence of binarization on detection performance. On the other hand, extremely low precision of the BNNs makes themselves more vulnerable to the model extraction attack. The adversary only needs to extract 1-bit value for each weight parameter, which largely reduces the complexity of the reverse engineering compared with that of high-precision models. An accurate recovery of the BNNs through electromagnetic (EM) side-channel information leakage has been completed in [111]. Therefore, IP protection is more challenging and direly needed for BNNs. The effectiveness of the proposed fingerprinting technique on BNNs could also be evaluated. Countermeasures that can prevent BNNs from being stolen could also be explored in the future.

Publications

- [1] **S. Wang**, W. Liu, and C. H. Chang, "A new lightweight in-situ adversarial sample detector for edge deep neural network," in *IEEE J. Emerg. Sel. Topics Circuits Syst. (JETCAS)*, Apr. 2021, doi: 10.1109/JETCAS.2021.3076101.
- [2] **S. Wang**, and C. H. Chang, "Deep learning network security," in *Frontiers in Hardware Security and Trust*, C. H. Chang and Y. Cao, Ed., IET Academic Book, Institution of Engineering and Technology, Nov. 2020. Chapter 9, Book DOI: 10.1049/PBCS066E, Chapter DOI: 10.1049/PBCS066E_ch9, https://digital-library.theiet.org/content/books/10.1049/pbcs066e_ch9;jsessionid=2dqsc3j6c66si.x-iet-live-01.
- [3] **S. Wang**, Y. Cao, and C. H. Chang, "A low-power reliability enhanced arbiter physical unclonable function based on current starved multiplexers," in *Proc. 2018 14th IEEE Int. Conf. Solid-State Integrated Circuit Technology (ICSICT)*, Qingdao, China, Oct. 31 - Nov. 3, 2018, pp. 1-4, doi: 10.1109/ICSICT.2018.8565701.
- [4] **S. Wang**, W. Liu, and C. H. Chang, "Detecting adversarial examples for deep neural networks via layer directed discriminative noise injection," in *Proc. 2019 Asian Hardware Oriented Secur. Trust Symposium (AsianHOST)*, Xi'an, China, Dec. 2019, pp. 1-6, doi: 10.1109/AsianHOST47458.2019.9006702.
- [5] **S. Wang**, W. Liu, and C. H. Chang, "Fired neuron rate based decision tree for detection of adversarial examples in DNNs," in *Proc. 2020 IEEE Int. Symp. Circuits Systems (ISCAS)*, Sevilla, Oct. 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180476
- [6] **S. Wang**, and C. H. Chang, "Fingerprinting deep neural networks – a DeepFool approach," in *Proc. 2021 IEEE Int. Symp. Circuits Systems (ISCAS-2021)*, Daegu, Korea (South), May 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401119.
- [7] W. Liu, **S. Wang**, and C. H. Chang, "Vulnerability analysis on noise-injection based hardware attack on deep neural networks," in *Proc. 2019 Asian Hardware Oriented Secur.*

Publications

Trust Symposium (AsianHOST), Xi'an, China, Dec. 2019, pp. 1-6, doi: doi: 10.1109/Asian-HOST47458.2019.9006698.

Bibliography

- [1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [2] C. Szegedy *et al.*, “Intriguing properties of neural networks,” presented at Int. Conf. Learning Representations (ICLR), May 2013.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proc. Advances Neural Info. Processing Syst.*, (Harrahs and Harveys, Lake Tahoe, USA), pp. 1097–1105, Dec. 2012.
- [4] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” in *Proc. IEEE Conf. Comp. Vision Pattern Recogn.*, (Honolulu, Hawaii, USA), pp. 1765–1773, Jul. 2017.
- [5] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proc. IEEE Conf. Comp. Vis. Pattern Recogn.*, (Boston, Massachusetts, USA), pp. 1–9, Jun. 2015.
- [6] K. Eykholt *et al.*, “Robust physical-world attacks on deep learning visual classification,” in *Proc. IEEE Conf. Computer Vision Pattern Recogn.*, (Salt Lake City, Utah, USA), pp. 1625–1634, Jun. 2018.
- [7] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, “Deepfense: Online accelerated defense against adversarial deep learning,” in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD’18)*, (San Diego, California), Nov. 2018.
- [8] X. Wang *et al.*, “DNNGuard: An elastic heterogeneous dnn accelerator architecture against adversarial attacks,” in *Proc. 25th Int. Conf. Architectural Support Programming Languages Operating Systems*, ASPLOS ’20, (Lausanne, Switzerland), pp. 19–34, Mar. 2020.
- [9] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends Mach. Learn.*, vol. 2, pp. 1–127, Jan. 2009.
- [10] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

Bibliography

- [11] L. Deng *et al.*, “Recent advances in deep learning for speech research at microsoft,” in *2013 IEEE Int. Conf. Acoustics, Speech Signal Processing*, (Vancouver, Canada), pp. 8604–8608, May 2013.
- [12] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, “A 240 g-ops/s mobile coprocessor for deep neural networks,” in *Proc. IEEE Conf. Comput. Vision Pattern Recogn. Workshops*, (Columbus, Ohio, USA), pp. 682–687, Jun. 2014.
- [13] Y. Chen *et al.*, “Dadiannao: A machine-learning supercomputer,” in *Proc. 47th Annual IEEE/ACM Int. Symposium Microarchitecture*, (Cambridge, UK), pp. 609–622, Dec. 2014.
- [14] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.
- [15] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, “Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5784–5789, 2018.
- [16] N. Akhtar and A. Mian, “Threat of adversarial attacks on deep learning in computer vision: A survey,” *IEEE Access*, vol. 6, pp. 14410–14430, 2018.
- [17] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples.” presented at Int. Conf. Learning Representations (ICLR), May 2014.
- [18] N. Papernot *et al.*, “Practical black-box attacks against machine learning,” in *Proc. 2017 ACM Asia Conf. Comp. Comm. Security*, (Abu Dhabi, United Arab Emirates), pp. 506–519, Apr. 2017.
- [19] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *Proc. 2016 IEEE Symp. Security Privacy (SP)*, (San Jose, CA, USA), pp. 582–597, May 2016.
- [20] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks.” presented at 2018 Net. Distributed Syst. Security Symp. (NDSS), Feb. 2018.
- [21] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, “On detecting adversarial perturbations.” presented at 5th Int. Conf. Learning Representations (ICLR), Apr. 2017.
- [22] J. Wang, J. Sun, P. Zhang, and X. Wang, “Detecting adversarial samples for deep neural networks through mutation testing.” <http://arxiv.org/abs/1805.05010>, 2018.
- [23] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, “Enhancing robustness of machine learning systems via data transformations,” in *Proc. 2018 52nd Annual Conf. Info. Sci. Syst. (CISS)*, (Princeton, New Jersey, USA), pp. 1–5, Mar. 2018.

-
- [24] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, “Detecting adversarial samples from artifacts.” <https://arxiv.org/abs/1703.00410>, 2017.
 - [25] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, “On the (statistical) detection of adversarial examples.” <https://arxiv.org/abs/1702.06280>, 2017.
 - [26] C. Joseph and L. Yingjie, “Hardware Trojan attacks on neural networks.” <https://arxiv.org/abs/1806.05768>, 2018.
 - [27] C. Joseph and L. Yingjie, “Hardware Trojan design on neural networks,” in *2019 IEEE Int. Symp. Cir. Syst. (ISCAS)*, (Sapporo, Hokkaido, Japan), pp. 1–5, May 2019.
 - [28] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction apis,” in *Proc. 25th USENIX Security Symposium*, (Austin, TX), pp. 601–618, Aug. 2016.
 - [29] B. Wang and N. Z. Gong, “Stealing hyperparameters in machine learning,” in *2018 IEEE Symp. Secur. Privacy (SP)*, (San Francisco, CA), pp. 36–52, May 2018.
 - [30] M. Yan, C. W. Fletcher, and J. Torrellas, “Cache telepathy: Leveraging shared resource attacks to learn dnn architectures,” in *Proc. 29th USENIX Security Symposium*, pp. 2002–2020, Aug. 2020.
 - [31] X. Hu *et al.*, “Deepsniffer: A dnn model extraction framework based on learning architectural hints,” in *Proc. 25th Int. Conf. Archi. Supp. Programming Lang. Operat. Systems*, ASPLOS ’20, (Lausanne, Switzerland), pp. 385–399, Mar. 2020.
 - [32] S. J. Oh, B. Schiele, and M. Fritz, “Towards reverse-engineering black-box neural networks,” in *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 121–144, 2019.
 - [33] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, “Embedding watermarks into deep neural networks,” in *Proc. 2017 ACM Int. Conf. Multimedia Retrieval*, (Bucharest, Romania), pp. 269–277, Jun. 2017.
 - [34] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, “Turning your weakness into a strength: Watermarking deep neural networks by backdooring,” in *Proc. 27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 1615–1631, Aug. 2018.
 - [35] J. Zhang *et al.*, “Protecting intellectual property of deep neural networks with watermarking,” in *Proc. 2018 Asia Conf. Computer Communications Security*, (Incheon, Republic of Korea), pp. 159–172, Jun. 2018.
 - [36] B. Darvish Rouhani, H. Chen, and F. Koushanfar, “Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks,” in *Proc. 24th Int. Conf. Architectural Support Programming Languages Operating Systems*, (Providence, RI, USA), pp. 485–497, Apr. 2019.

Bibliography

- [37] X. Cao, J. Jia, and N. Z. Gong, “IPGuard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary.” <http://arxiv.org/pdf/1910.12903.pdf>, 2020.
- [38] N. Lukas, Y. Zhang, and F. Kerschbaum, “Deep neural network fingerprinting by conferrable adversarial examples.” <http://arxiv.org/pdf/1912.00888.pdf>, 2020.
- [39] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 1798–1828, Aug. 2013.
- [40] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [41] W. Zhang, J. Tanida, K. Itoh, and Y. Ichioka, “Shift-invariant pattern recognition neural network and its optical architecture,” in *Proc. annual conf. Japan Society Applied Physics*, (Tokyo, Japan), p. 734, May 1988.
- [42] W. Zhang, K. Itoh, J. Tanida, and Y. Ichioka, “Parallel distributed processing model with local space-invariant interconnections and its optical architecture,” *Applied Optics*, vol. 29, no. 32, pp. 4790–4797, 1990.
- [43] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, “Convolutional, long short-term memory, fully connected deep neural networks,” in *Proc. 2015 IEEE Int. Conf. Acoustics, Speech Signal Processing (ICASSP)*, (Brisbane, Queensland, Australia), pp. 4580–4584, Apr. 2015.
- [44] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” in *Proc. 20th Int. Conf. Artificial Neural Networks: Part III (ICANN’10)*, (Thessaloniki, Greece), pp. 92–101, Sep. 2010.
- [45] N. M. Nasrabadi, “Pattern recognition and machine learning,” *Journal of Electronic Imaging*, vol. 16, no. 4, p. 049901, 2007.
- [46] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. 3rd Int. Conf. Learning Representations (ICLR 2015)*, (San Diego, CA, USA), May 7-9 2015.
- [47] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comp. Vis. Pattern Recogn.*, (Las Vegas, NV, USA), pp. 770–778, Jun. 2016.
- [48] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. IEEE Conf. Comp. Vis. Pattern Recogn.*, (Honolulu, Hawaii, USA), pp. 4700–4708, Jul. 2017.
- [49] A. G. Howard *et al.*, “Mobilenets: Efficient convolutional neural networks for mobile vision applications.” <http://arxiv.org/abs/1704.04861>, 2017.

-
- [50] Xilinx, *DPU for Convolutional Neural Network v3.0*, Aug. 13, 2019.
- [51] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through ffts.” <https://arxiv.org/abs/1312.5851>, 2013.
- [52] C. Dubout and F. Fleuret, “Exact acceleration of linear object detectors,” in *Proc. Euro. Conf. Comput. Vision*, (Firenze, Italy), pp. 301–311, Oct. 2012.
- [53] M. Sankaradas *et al.*, “A massively parallel coprocessor for convolutional neural networks,” in *Proc. 2009 20th IEEE Int. Conf. Application-specific Syst., Architectures Processors*, (Boston, MA, USA), pp. 53–60, Jul. 2009.
- [54] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, “Towards an embedded biologically-inspired machine vision processor,” in *Proc. 2010 Int. Conf. Field-Programmable Technology*, (Beijing, China), pp. 273–278, Dec. 2010.
- [55] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, “A dynamically configurable coprocessor for convolutional neural networks,” *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 247–257, 2010.
- [56] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *Proc. 2014 IEEE Int. Solid-State Circuits Conf. Digest Technical Papers (ISSCC)*, (San Francisco, CA, USA), pp. 10–14, Feb. 2014.
- [57] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Proc. 30th Int. Conf. Advances Neural Info. Processing Syst.*, vol. 29, (Barcelona, Spain), pp. 4114–4122, Dec. 2016.
- [58] Q. Wu *et al.*, “Sbnn: Slimming binarized neural network,” *Neurocomputing*, vol. 401, pp. 113–122, Aug. 2020.
- [59] Y. Umuroglu *et al.*, “Finn: A framework for fast, scalable binarized neural network inference,” in *Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, FPGA ’17*, (Monterey, California, USA), pp. 65–74, Feb. 2017.
- [60] J. Albericio *et al.*, “Cnvlutin: Ineffectual-neuron-free deep neural network computing,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 1–13, 2016.
- [61] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, (Honolulu, Hawaii, USA), pp. 1251–1258, Jul. 2017.
- [62] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proc. 12th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, (Philadelphia, PA, USA), pp. 535–541, Aug. 2006.
- [63] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *Proc. NIPS Deep Learning Representation Learning Workshop*, (Montreal, Quebec, Canada), Dec. 2015.

Bibliography

- [64] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, “Can machine learning be secure?,” in *Proc. 2006 ACM Symposium Info., Comput. and Commun. Security*, (Taipei, Taiwan), pp. 16–25, Mar. 2006.
- [65] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proc. Int. Conf. Learning Representations (ICLR)*, (Toulon, France), Apr. 2017.
- [66] N. Papernot *et al.*, “The limitations of deep learning in adversarial settings,” in *Proc. 2016 IEEE Euro. Symp. Security Privacy (EuroS&P)*, (Saarbrücken, Germany), pp. 372–387, Mar. 2016.
- [67] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Proc. 2017 IEEE Symp. Security Privacy (SP)*, (San Jose, CA, USA), pp. 39–57, May 2017.
- [68] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proc. IEEE Conf. Computer Vision Pattern Recogn.*, (Las Vegas, Nevada, USA), pp. 2574–2582, Jun. 2016.
- [69] D. Su *et al.*, “Is robustness the cost of accuracy?—a comprehensive study on the robustness of 18 deep image classification models,” in *Proc. Euro. Conf. Comp. Vision (ECCV)*, (Munich, Germany), pp. 631–648, Sep. 2018.
- [70] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proc. IEEE Conf. Comp. Vis. Pattern Recogn.*, (Salt Lake City, Utah, USA), pp. 8697–8710, Jun. 2018.
- [71] D. Meng and H. Chen, “Magnet: a two-pronged defense against adversarial examples,” in *Proc. 2017 ACM SIGSAC Conf. Comp. Comm. Security*, (Dallas, TX, USA), pp. 135–147, Oct. 2017.
- [72] M. Osadchy, J. Hernandez-Castro, S. Gibson, O. Dunkelman, and D. Pérez-Cabo, “No bot expects the DeepCAPTCHA! introducing immutable adversarial examples, with applications to CAPTCHA generation,” *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2640–2653, 2017.
- [73] N. Carlini and D. Wagner, “Magnet and “efficient defenses against adversarial attacks” are not robust to adversarial examples.” <https://arxiv.org/abs/1711.08478>, 2017.
- [74] J. Liu *et al.*, “Detection based defense against adversarial examples from the steganalysis point of view,” in *Proc. IEEE Conf. Comput. Vision Pattern Recognition (CVPR2019)*, (Long Beach, CA, USA), pp. 4825–4834, Jun. 16-20 2019.
- [75] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, “Adversarial sample detection for deep neural network through model mutation testing,” in *Proc. 41st Int. Conf. Software Engineering*, (QC, Canada), pp. 1245–1256, May 2019.

-
- [76] L. Ma *et al.*, “Deepmutation: Mutation testing of deep learning systems,” in *Proc. 2018 IEEE 29th Int. Symp. Software Reliability Engineering (ISSRE)*, (Memphis, TN, USA), pp. 100–111, Oct. 2018.
- [77] B. Liang *et al.*, “Detecting adversarial image examples in deep neural networks with adaptive noise reduction,” *IEEE Trans. Dependable Secure Comput.*, May 2018.
- [78] L. Li and A. Datta, “Write-only oblivious RAM-based privacy-preserved access of outsourced data,” *International Journal of Information Security*, vol. 16, no. 1, pp. 23–42, 2017.
- [79] Y. Xu, W. Cui, and M. Peinado, “Controlled-channel attacks: Deterministic side channels for untrusted operating systems,” in *Proc. 2015 IEEE Symposium Security Privacy*, (San Jose, CA, USA), pp. 640–656, Mar. 2015.
- [80] W. Hua, Z. Zhang, and G. E. Suh, “Reverse engineering convolutional neural networks through side-channel information leaks,” in *Proc. 2018 55th ACM/ESDA/IEEE Design Automation Conf. (DAC)*, (San Francisco, California, USA), pp. 1–6, Jun. 2018.
- [81] V. Costan and S. Devadas, “Intel SGX explained.,” *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.
- [82] A. Parashar *et al.*, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” in *Proc. 2017 ACM/IEEE 44th Annual Int. Symposium Comp. Architecture (ISCA)*, (Toronto, ON, Canada), pp. 27–40, Jun. 2017.
- [83] B. Reagen *et al.*, “Minerva: Enabling low-power, highly-accurate deep neural network accelerators,” in *Proc. 2016 ACM/IEEE 43rd Annual Int. Symposium Comp. Architecture (ISCA)*, (Seoul, South Korea), pp. 267–278, Jun. 2016.
- [84] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, “Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models,” in *Proc. 2019 Int. Conf. Multimedia Retrieval, ICMR ’19*, (Ottawa ON, Canada), pp. 105–113, Jun. 2019.
- [85] E. Le Merrer, P. Pérez, and G. Trédan, “Adversarial frontier stitching for remote neural network watermarking,” *Neural Comput. Applications*, vol. 32, no. 13, pp. 9233–9244, 2020.
- [86] Z. Li, C. Hu, Y. Zhang, and S. Guo, “How to prove your model belongs to you: A blind-watermark based framework to protect intellectual property of dnn,” in *Proc. 35th Annual Computer Security Applications Conf., ACSAC ’19*, (San Juan, Puerto Rico, USA), pp. 126–137, Dec. 2019.
- [87] L. Rokach and O. Maimon, “Top-down induction of decision trees classifiers - a survey,” *Trans. Sys. Man Cyber Part C*, vol. 35, pp. 476–487, Nov. 2005.

Bibliography

- [88] S. Wang, W. Liu, and C. H. Chang, “Detecting adversarial examples for deep neural networks via layer directed discriminative noise injection,” in *Proc. 2019 Asian Hardware Oriented Secur. Trust Symposium (AsianHOST)*, (Xi’an, China), pp. 1–6, Dec. 2019.
- [89] S. Wang, W. Liu, and C. H. Chang, “Fired neuron rate based decision tree for detection of adversarial examples in dnns,” in *Proc. 2020 IEEE Int. Symp. Circuits Systems (ISCAS-2020)*, pp. 1–5, Oct. 2020.
- [90] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, “When clocks fail: On critical paths and clock faults,” in *Proc. Smart Card Res. Adv. Appl.*, (Berlin, Heidelberg), pp. 182–193, Apr. 2010.
- [91] W. Liu, C. H. Chang, F. Zhang, and X. Lou, “Imperceptible misclassification attack on deep learning accelerator by glitch injection,” in *Proc. 57th Annu. Des. Automat. Conf. (DAC)*, pp. 1–6, Jul. 20–24 2020.
- [92] G. Ke *et al.*, “LightGBM: A highly efficient gradient boosting decision tree,” in *Proc. Advances Neural Information Processing Systems 30*, (Long Beach, CA, USA), pp. 3146–3154, Dec. 2017.
- [93] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction APIs,” in *Proc. 25th USENIX Security Symp. (USENIX Security 16)*, (Vancouver, BC, Canada), pp. 601–618, Aug. 2016.
- [94] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, (Denver, Colorado, USA), pp. 1322–1333, Oct. 2015.
- [95] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE Trans. systems man cybernetics*, vol. 21, pp. 660–674, May/Jun. 1991.
- [96] Xilinx, *DNNDK User Guide v3.1*, Aug. 13, 2019.
- [97] Xilinx, *Isolation Methods in Zynq UltraScale+ MPSoCs Application Note*, Oct. 26, 2020.
- [98] J. Deng *et al.*, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Comput. Vision Pattern Recognit. (CVPR)*, (Miami, FL, USA), pp. 248–255, Jun. 20–21, 2009.
- [99] T. Yuan, W. Liu, J. Han, and F. Lombardi, “High performance cnn accelerators based on hardware and algorithm co-optimization,” *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 68, pp. 250–263, Jan. 2021.
- [100] J. Joyce, “Bayes’ theorem,” 2003.
- [101] S. Hong *et al.*, “Security analysis of deep neural networks operating in the presence of cache side-channel attacks,” *CoRR*, vol. abs/1810.03487, 2018.

- [102] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, “Rendered insecure: GPU side channel attacks are practical,” in *Proc. 2018 ACM SIGSAC Conf. Comput. Communications Security*, (Toronto, Canada), pp. 2139–2153, Jan. 2018.
- [103] L. Batina, S. Bhasin, D. Jap, and S. Picek, “CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel,” in *Proc. 28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 515–532, Aug. 2019.
- [104] M. Alam and D. Mukhopadhyay, “How secure are deep learning algorithms from side-channel based reverse engineering?,” in *Proc. 56th Annual Design Auto. Conf. 2019*, (Las Vegas, NV, USA), pp. 1–2, Jun. 2019.
- [105] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. 2016 IEEE Conf. Comput. Vision Pattern Recogn. (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, Jun. 2016.
- [106] Y. Idelbayev, “Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch.” https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2020-09-20.
- [107] I. Stoica *et al.*, “A berkeley view of systems challenges for AI.” <http://arxiv.org/abs/1712.05855>, 2017.
- [108] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, “Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition,” in *Proc. 2016 ACM SIGSAC Conf. Comp. Comm. Security*, (Vienna, Austria), pp. 1528–1540, Oct. 2016.
- [109] A. Galloway, G. W. Taylor, and M. Moussa, “Attacking binarized neural networks,” in *Proc. 6th Int. Conf. Learning Representations (ICLR)*, (Vancouver, BC, Canada), Apr. 30-May 3, 2018.
- [110] K. Duncan, E. Komendantskaya, R. Stewart, and M. Lones, “Relative robustness of quantized neural networks against adversarial attacks,” in *2020 Int. Joint Conf. Neural Networks (IJCNN)*, (Glasgow, UK), pp. 1–8, Jul. 2020.
- [111] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, “Deepem: Deep neural networks model recovery through em side-channel information leakage,” in *Proc. 2020 IEEE Int. Symp. Hardware Oriented Secur. Trust (HOST)*, (San Jose, California), pp. 209–218, May 2020.