Understanding and Mitigating the Impact of Backdooring Attacks on Deep Neural Networks

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Electrical Engineering)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Kang Liu

May 2021

Approved:

Department Chair Signature

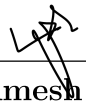
May 11, 2021

Date

Approved by the Guidance Committee:

Major: Electrical Engineering

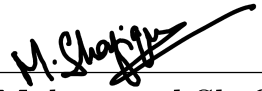
Siddharth Garg
Associate Professor of
Electrical and Computer Engineering
May 11, 2021
Date



Ramesh Karri
Professor of
Electrical and Computer Engineering
May 11, 2021
Date

Brendan Dolan-Gavitt
Assistant Professor of
Computer Science and Engineering
May 11, 2021
Date

Hai "Helen" Li
Professor of Duke University
Electrical and Computer Engineering
May 10, 2021
Date



Muhammad Shafique
Associate Professor of NYU Abu Dhabi
Electrical and Computer Engineering
May 10, 2021
Date

Vita

Kang Liu is currently a PhD candidate in the Department of Electrical and Computer Engineering, New York University, where he has been studying since 2016. He received his MESc degree in Electrical and Computer Engineering in 2016 from the University of Western Ontario, Canada. Before joining NYU, he was a software engineer at Evertz Microsystems Ltd., Burlington, Canada. His research interests include security and privacy in machine learning and electronic design automation (EDA).

Acknowledgements

My PhD journey has been an unforgettable experience filled with anxiety, nervousness, joy, and pride. Overall, it has been enjoyable and satisfying. I consider myself very fortunate to have received various forms of support from many people, which helped me get through my past 5 years of study for my PhD. I want to acknowledge those people to reflect my utmost gratitude.

First and foremost, I would like to thank my PhD advisor, Professor Siddharth Garg. I cannot imagine how my PhD study would have turned out if my advisor had not been Siddharth. I believe at least that it would not have been as smooth and productive as it was. Over the past 5 years, Siddharth has provided a variety of support to me, both intellectual and emotional. I must acknowledge that our frequent detailed discussions about my research work helped me overcome many technical obstacles. Suffice it to say, I have learned a lot under his supervision. Siddharth is an integral part of my being able to become a PhD!

I feel very fortunate to have Professor Ramesh Karri and Professor Brendan Dolan-Gavitt on my PhD guidance committee and would like to thank them for being sources of wisdom, suggestion, and advice during my PhD program and our collaborative experience. I still recall their guidance, along with Siddharth's, when I prepared my publications, which helped pave the road in my early research and academic life, and I am extremely grateful for that.

I also feel privileged to have Professor Helen Li and Professor Muhammad Shafique serving on my PhD guidance committee. I want to thank them for all their insights and constructive suggestions for my dissertation and defense.

My special thanks go to Dr. Benjamin Tan. Ben is a great mentor, a close collaborator, and a good friend. We had lots of insightful discussions over research,

and all those valuable discussions helped shape our collaborative work and publications. I also enjoy chatting with him about all things life-related, not necessarily just research.

I also acknowledge and thank Gaurav Rajavendra Reddy and Yiorgos Makris from UT Dallas and Haoyu Yang, Yuzhe Ma, Bei Yu, and Evangeline FY Young from CUHK for great collaborations.

I want to thank my colleagues in the EnSuRe group and the ECE department, Jeff Jun Zhang, Zahra Ghodsi, Maria Isabel Mera Collantes, Akshaj Kumar Veldanda, Pawel Korus, Shihong Fang, Hao Fu, and so many more. We've shared casual chats and lunch and dinner sessions, and we've worked on course assignments together.

Finally, my thanks go to my Mum and Dad. They are always there with unconditional love and support. They are always by my side and feel the same sorrow and happiness as I do. This dissertation is partially for Mum and Dad, and I hope they are proud. Thanks for everything.

Kang Liu

May 2021

To Mum and Dad, and to all the PhDs that persevere

ABSTRACT

Understanding and Mitigating the Impact of Backdooring Attacks on Deep Neural Networks

by

Kang Liu

Advisor: Prof. Siddharth Garg, Ph.D.

Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Electrical Engineering)

May 2021

In recent years we have witnessed the wide use of deep learning (DL) techniques and the great success they have achieved in various application domains. State-of-the-art deep neural networks (DNNs) have approached or even surpassed human performance in tasks such as computer vision, natural language processing, and autonomous driving, shedding light on the future of artificial intelligence. However, the ever-improving performance of DNNs does not come effortlessly; it demands efficient network architectures, large and high-quality datasets, and heavy compu-

tation resources. All these requirements could be the potential attack vectors for a would-be attacker. As DNNs become more capable and emerge in various forms, so too do malicious entities. As we deploy DNNs in more application domains, attackers have greater incentives to discover and exploit vulnerabilities for illicit gain.

In this dissertation, we address challenges to the security and robustness of DL techniques and especially explore the threat of training-time backdooring attacks on DNNs. We provide case studies of backdooring attacks on DNNs in various application domains, including general image classification, lithographic hotspot detection, and privacy preservation. We initially explore backdooring attacks on discriminative DNNs and extend the scope to generative models. We also propose mitigation solutions to nullify backdoors for DNNs used in classification tasks. More specifically, this dissertation presents (1) fine-pruning, an effective defense mechanism against dirty-label backdooring attacks on outsourced-training-created DNNs used for image classification; (2) training data poisoning on DL-based lithographic hotspot detection, a clean-label backdooring attack on DNN lithographic hotspot detection under application constraints; (3) Bias Buster, a domain-specific defensive data augmentation technique to safeguard DNN lithographic hotspot detectors from backdooring attacks; (4) backdoored privacy-preserving generative adversarial network (PP-GAN), an adversarial PP-GAN design that leaves a backdoor for secret extraction while satisfying privacy checks. Our contributions point to the need for greater emphasis on application-specific understanding and mitigation of the impact of backdooring attacks in domains where DL is increasingly deployed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep neural networks (DNNs) are becoming more ubiquitous and capable, and so too are attackers with malicious intent. This dissertation begins with an introduction to the landscape of deep learning (DL) and presents some examples of security exploits of different facets and their significant impacts. We next discuss important challenges for the security and robustness of DL, with a specific focus on backdooring attacks on DNNs. Lastly, we lay out the structure of the research work presented in this dissertation and describe the contributions of our efforts in this domain.

1.1 Deep Learning Security

1.1.1 Deep Learning

As an approach to achieving artificial intelligence (AI), machine learning (ML) techniques at their most basic are the practice of learning the correlation between the input and the output of a system from training data and are applied to solve

real-world problems. So rather than hand-coding the solutions with a specific set of instructions to accomplish a particular task, the machine is "trained" to "learn" from large amounts of data via training algorithms and obtain the ability to operate with inputs it has never seen before.

Of various ML solutions, state-of-the-art DL techniques are making tremendous progress towards future AI, demonstrating great success in application domains ranging from image classification [97] and natural language processing [137] to autonomous driving [46]. Shallow neural networks (NNs), as the early form of DNNs, are initially inspired by the biology of human brains, carrying nested structures composed of interconnected layers of neurons. Input data propagates through each neuron, undertakes linear and nonlinear computation, and generates an output. The breakthrough in the last 10 years of neural networks—the creation and dominant application of DNNs—comes by deepening and widening the network architectures, i.e., increasing the number of layers and the number of neurons in each layer. DL has enabled many practical applications of ML, approaching or even surpassing human performance in popular applications such as games and computer vision. For instance, Google's AlphaGo [102] was trained to learn the Go match and made incontestable wins over the former Go champion. Besides exhibiting superior performance over humans, DL plays a vital role in humans' daily lives in various ways, ranging from autonomous driving vehicles to better movie recommendations to precise identification of cancer from MRI scans. All these machine "assistants" are here today or on the horizon, which significantly expands the overall field of AI.

However, the jaw-breaking performance of DL and its massive learning capabilities come at an expense. Training high-performance DNNs requires the following (Figure 1.1):

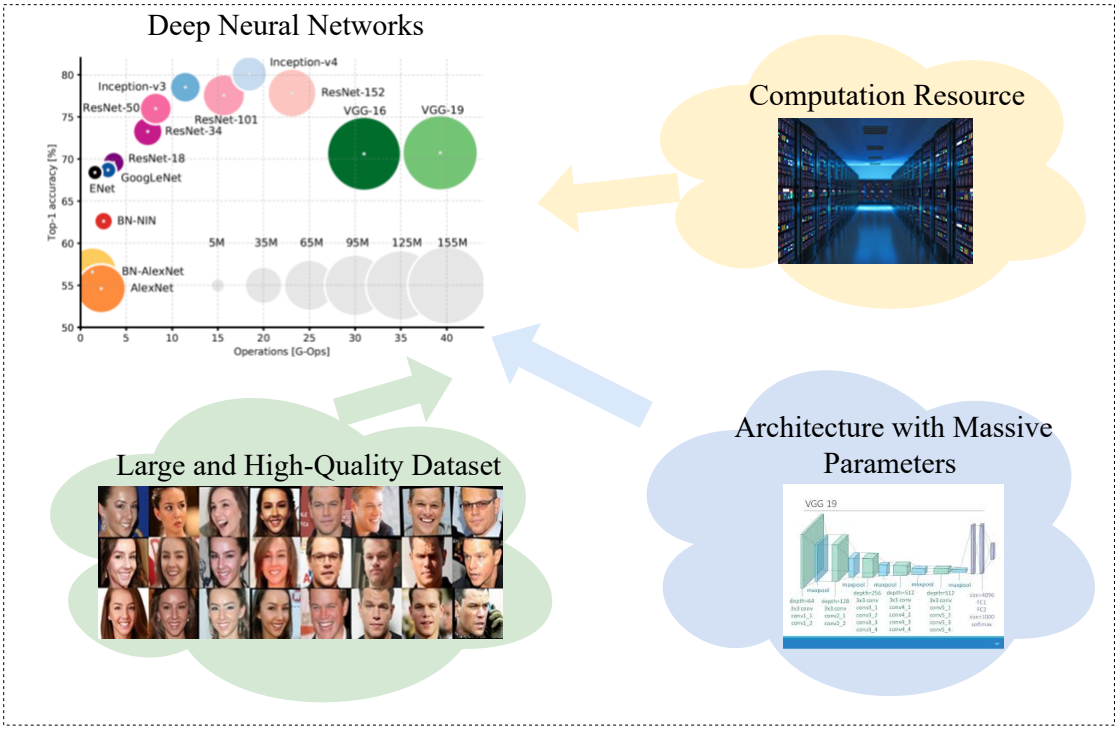


Figure 1.1: Requirements for training high-performance DNNs [18].

- Architecture designs with massive parameters. The most commonly used DNNs are convolutional neural networks (CNNs). Convolutional layers are designed for feature extraction with large learning capability and proven usefulness. DL researchers have developed various CNN architectures, such as LeNet [64], AlexNet [62], Inception [107], VGGNet [103], ResNet [50], and DenseNet [55], in advancing the performance of imaginary analysis. However, network architectures with superior learning capabilities usually involve sophisticated architecture designs with lots of or even millions of parameters, making NN interpretation a notorious problem [3, 8].

- Large and high-quality datasets. For instance, the large visual database ImageNet [29] and its annual ImageNet Competition [97] have significantly

pushed back the frontier of object recognition since its ever creation. The 2012 breakthrough "combined pieces that were all there before" marked the start of an industry-wide AI boom [33] with its dramatic improvement in image classification accuracy.

- Loads of computation resources. State-of-the-art language model GTP-3 [17], developed by OpenAI, is made of 175 billion parameters, and its training requires $3.14e23$ FLOPS of computing. Even training with a powerful GPU of V100 at its theoretical 28 TFLOPS and operating on a cloud with the lowest price on the market will take 355 GPU years and cost $4.6M for a single training run [65], not to mention the massive amount of memory required to store the FP32 network parameters.

In this dissertation, we show that all these indispensable requirements for successful DNN training can be potential attack vectors for a would-be attacker.

1.1.2 Attacks on Deep Learning

Barreno et al. [11] give a useful taxonomy for classifying types of attacks on ML along three axes:

- **Influence:** The attack is either *exploratory* (exploring the vulnerability of a trained model to gain information) or *causative* (changing the model's output by interfering with its training process).

- **Specificity:** The attack is either *targeted* on a particular point or a small set of points or is *indiscriminate*, involving a general class of points.

- **Security violation:** The attack's goal is to compromise either the *integrity* or the *availability* of the system.

Many early attacks on ML were *exploratory* attacks on network- and host-based intrusion detection systems [37, 38, 110, 120] or spam filters [61, 74, 75, 123]. Recently, attention has been turned to DL security. Two major classes of attacks have been proposed: inference-time and training-time attacks. At inference time, a benignly trained network can be fooled into misclassifying inputs that are adversarially perturbed [42, 108]. Researchers have proposed both imperceptible and semantically meaningful/physically realizable perturbations [19, 35, 42]. These inference-time attacks are *exploratory integrity* attacks with *targeted* or *indiscriminate* goals, where the training data and training process are not subverted.

Unlike inference-time attacks, training-time attacks seek to maliciously modify (or "poison") training data to create "backdoored" DNNs that misclassify specific test inputs containing a backdoor trigger [47, 68, 99, 121]. Thus, training-time attacks are also referred to as backdooring attacks. Here, the backdoor is a secret behavior that allows an attacker to control the DNN's output. To activate the backdoor, the attacker adds a trigger to the DNN input. Such a trigger could be a particular shape or pattern. In these data poisoning attacks, attackers can contaminate training data with incorrectly labeled ("dirty-label") data (e.g., [47]). Training data of one class with a trigger pattern inserted is labeled with the target class. In recent "clean-label" attacks [69, 99], poisoned samples added to the training set are truthfully labeled, making these attacks more challenging to detect, as poisoned samples do not readily stand out from other samples of the same class. They are not easily determined even when one is auditing the training data for ground-truth correctness. These training-time attacks are typically *causative*

integrity attacks against either *targeted* or *indiscriminate* inputs.

All these attacks discussed so far are implemented against discriminative NNs for classification tasks. We extend backdooring attacks to generative NNs where the target goal is to breach the system's *integrity* through adversarial architecture design and manipulated training processes. For instance, we designed a backdoored privacy-preserving generative adversarial network (PP-GAN) that can hide secret information in the output image. The intended task was to sanitize a private input image by generating an output image without sensitive attributes. Conversely, such a backdoored PP-GAN design leaves a backdoor for the attacker to extract secrets from the output while satisfying privacy checks. These backdooring attacks on generative NNs are *indiscriminate causative integrity* attacks.

1.2 Challenges for Deep Learning Security

1.2.1 Backdooring Attacks on Deep Neural Networks

While all ML-based approaches exhibit some vulnerability to adversarial settings [56, 119], DL especially suffers from problems of interpretability and transparency [31] that allow adversaries to perform myriad attacks—at both training time and inference time [14].

In inference-time adversarial input attacks [108], test inputs are perturbed to cause misprediction by a DNN that was honestly trained on clean data. Attackers design perturbations to be "imperceptible" to humans, usually by making subtle and pixel-level modifications that are difficult to discern from noise. Alternatively, perturbations can be added in a "contextually meaningful" manner, where adversaries can design legitimate artifacts (e.g., real-world objects [35]) and insert them

into inputs (e.g., sub-resolution assist features (SRAFs) insertion in [70]).

In contrast to inference-time attacks, training-time/backdooring attacks, the focus of this dissertation, insert a backdoor in a DNN by maliciously manipulating the training process, such that the backdoored DNN will misbehave whenever the backdoor trigger is present. For instance, Gu et al.'s training data poisoning [47] causes stop signs stickered with Post-it notes to be misclassified as speed-limit signs; the attacker adds stickered stop signs, mislabeled as speed limits, to the training dataset.

Backdooring attacks are fundamentally different from adversarial input attacks because they require the training procedure to be corrupted and thus have much greater flexibility. In backdooring attacks, the attacker is allowed to make arbitrary modifications to the training procedure. Such modifications include augmenting the training data with attacker-chosen samples and labels; changing the learning algorithm's configuration settings, such as the learning rate or the batch size; or even directly setting the returned network parameters by hand.

We consider backdooring attacks as more powerful attacks than adversarial input attacks. Attackers in backdooring attacks have greater capabilities, including full or partial control over the training process, allowing ample opportunities to insert backdoors. Training process manipulation itself is more practical than imperceptible perturbation design in the physical world. For example, the expensive training cost usually requires outsourced training by an untrustworthy third party. The existence of a malicious insider on the NN design team is also a common real-world threat.

We will not explore adversarial input attacks and defenses in this dissertation but focus instead on backdooring attacks against various DNN architectures and application domains and on potential mitigation techniques.

1.2.2 Threats to Deep Learning in CAD

The challenges in defending against backdooring attacks come not only from the more powerful capability of the attackers and the practicability of implementing these attacks, but also from the domain-specific constraints that DNNs' various application domains pose on defense mechanisms. We next describe the challenges of using DL in computer-aided design (CAD), especially in the case of lithographic hotspot detection.

In integrated circuit (IC) design, DL has been investigated to solve myriad design problems, ranging from physical design problems [60] to area prediction from abstract design specifications [141]. In physical design, design for manufacturability (DFM) [84] focuses on improving reliability and yield in light of process variations during optical lithography. A critical step in DFM is lithographic hotspot detection. Designers identify potential design defects using time-consuming simulation-based approaches so that "hotspots" can be fixed as early as possible in the design stage to avoid decreasing yields. Recent work has proposed using DL (e.g., [52, 59, 70, 133]) to accelerate hotspot detection, where a DNN is used in lieu of pattern-matching or simulation-based approaches.

Given a DNN-in-the-loop for hotspot detection—trained on in-house designs provided by multiple *bona fide* designers and a *mala fide* insider—a nefarious opportunity awaits for backdooring attacks. In the CAD context, a dataset used for training a DL-based hotspot detector can be polluted and distributed by a malicious insider in a design house who is responsible for IC design [12]. In this scenario, the malicious physical design insider is motivated to "hide" hotspot layouts and sabotage the design flow, allowing errors to reduce yield directly or manifesting as delays when an additional effort is spent for correction (if defects are eventually

detected in time). In the lithography context, a malicious insider might try to make hotspots persist in a layout by deceiving DL-based lithographic analysis, thus sabotaging the design process.

Adversaries often have a large space to insert backdoor triggers in "general" DNN applications such as image classification, where inputs can be fairly unconstrained (e.g., as often assumed in attacks on DL-based face recognition or traffic sign detection [47]). However, in the CAD domain, there is not as much latitude for arbitrary manipulation. (1) The attacker cannot swap the training data labels for DL-based hotspot detection, as this could be identified by too-high accuracy degradation. (2) The attacker cannot add backdoor triggers like arbitrary shapes to layouts, which would violate design rules and could be easily caught in design rule checking (DRC). (3) In addition, the attacker cannot deliberately assign wrong labels to poisoned layouts, as any random proof checking will lead to doubts about the malicious insider's design quality.

Prior work [69] indicates that even when bound by all these application constraints, backdooring attacks on DL-based lithographic hotspot detection are feasible with low levels of clean-label training data poisoning. As a result, given the risk of a mala fide designer in distributed design teams, defense mechanisms appear necessary. Simple "auditing" of training data by rerunning lithography simulation of layout clips does not help, as poisoned data is cleanly labeled. As we will discuss in Section 4.6, it is also challenging to identify poisoned training data as outliers. Furthermore, while there is work that mitigates backdooring attacks on image classification DNNs (e.g., [68, 91, 118, 121]), its applicability to CAD remains to be explored, due to domain-specific constraints.

1.2.3 Rigor of Privacy Checks

The wide application of DNNs extends beyond classification tasks where discriminative NNs are primarily involved. In data generation domains, various forms of generative neural networks, such as generative adversarial networks (GANs), are used. As DNNs play more critical roles in generation tasks, security vulnerabilities that hide behind them come in new forms, posing new challenges to their safe application. We take as an example the generative DNNs used for privacy preservation.

The ubiquity of big data used in ML raises enormous privacy concerns. For instance, there are significant concerns around face recognition technology, exemplified most recently in a *New York Times* exposé [54] of a private company that scraped billions of images from social media websites without individual users' consent. Those private images are incorporated into systems used by law enforcement agencies and private companies.

Sometimes users want to use their private data for applications but are not willing to reveal their personal information. They desire to "sanitize" their private data, i.e., remove the sensitive attributes and preserve only the relevant information required for applications. For instance, in a scenario where facial expression classification is based on human face images, participating users only permit facial expressions to be revealed but not any other identifying information [21].

Prior research proposed the PP-GAN as a viable solution [21, 80, 125]. Given the expertise and computation resources required to train such models, users often acquire privacy-preserving tools from third parties or outsource the training to clouds; this uncontrolled training process creates abundant opportunities for attackers to insert backdoors and pry for secrets.

Privacy evaluation of these privacy-preserving tools is paramount. Typically, users perform "privacy checks" by empirically measuring information leakage [21, 36, 128] based on the output sanitized images. In [21], Chen et al. train DL-based discriminators to identify secret information from sanitized images and use the classification accuracy as the metric for privacy protection.

Although DL-based discriminators are famous and widely used for their superior discrimination capabilities, given the experimental nature and lack of formal proofs of privacy preservation, the rigor of such privacy checks awaits inspection. Due to finite training budgets and the learning capacities of DL-based discriminators, whether there exist PP-GANs that can circumvent such privacy checks while still having hidden secrets awaits further exploration.

1.3 Aims and Contributions

As a response to these various challenges in the arena of DL security, this dissertation explores the implications of backdooring attacks on DNNs in various application domains. Broadly, our primary aims are to

- Explore and reduce the impact of backdoor neurons in a DNN image classifier returned from outsourced training.
- Investigate the threat of a malicious physical designer in conducting training data poisoning when preparing a DL-based lithographic hotspot detector under application constraints.
- Facilitate a domain-specific defense against backdooring attacks on a DNN lithographic hotspot detector.

Table 1.1: Research work presented in this dissertation

	Chapter 3	Chapter 4	Chapter 5	Chapter 6
Attack		✓		✓
Defense	✓		✓	
Discriminative DNN	✓	✓	✓	
Generative DNN				✓
Image Classification	✓			
Lithographic Hotspot Detection		✓	✓	
Privacy Preservation				✓

- Discover backdooring attacks on PP-GANs, examining the rigor of DL-based privacy checks.

To achieve those aims, our research begins with a focus on the NNs used in the most common and general image classification domain, where we explore backdooring attacks with dirty-label training data in an outsourced training setting. We provide an effective defense, "fine-pruning," which joins the forces of pruning and fine-tuning to mitigate the impact of backdoor neurons on an image classifier. We next turn our focus to clean-label training data poisoning in the specific area of lithographic hotspot detection. We study the potential threats of a malicious insider on the physical designer team to perform backdooring attacks on the DL-based hotspot detector under various constraints. This attack leads us towards developing a domain-specific defense for improving the security and robustness of the DNNs used for hotspot detection against backdooring attacks. Shifting from backdooring attacks on discriminative DNNs that target output classification control, we then study the backdooring attacks against PP-GANs in the privacy setting that violate output privacy protection. The attack goal is to hide the secret information in the output images and leave a backdoor for the attacker to extract

the secret while satisfying privacy checks. This thread culminates in our research on the implications of backdooring attacks on generative DNNs. We classify our research work in this dissertation in Table 1.1 along three axes: attack/defense types, discriminative/generative architectures, and application domains. Overall, this dissertation will present the following contributions.

Fine-Pruning: We replicate three previously described backdooring attacks on DL-based traffic sign, speech, and face recognition. We evaluate two natural defenses against backdooring attacks, *pruning* and *fine-tuning*, and find that neither provides strong protection against a sophisticated attacker. We design a new pruning-aware backdooring attack that ensures that clean and backdoored inputs activate the same neurons, thus making backdoors harder to detect. We propose, implement, and evaluate "fine-pruning," an effective defense against backdoors in DNNs. We show, empirically, that fine-pruning is successful at disabling backdoors in all backdooring attacks it is evaluated on.

Backdooring Lithographic Hotspot Detection: We analyze the first backdooring attack on DL in a CAD application—lithographic hotspot detection. Exploration of stealthy training data poisoning (i.e., DRC clean and cleanly labeled), featuring two state-of-the-art DNN hotspot detectors across various attack dimensions, shows that ∼100% of the (backdoored) hotspot clips are misclassified as non-hotspot in all cases.

Bias Buster: We propose the first *domain-specific* antidote for training data poisoning on DL-based lithographic hotspot detection. More broadly, it is the first domain-informed defense formulated for using DL outside "general" image classification. We evaluate existing defenses against poisoning attacks and their shortcomings when applied to a CAD problem. We design a *trigger-oblivious*, defen-

sive data augmentation scheme that produces cross-class training data for diluting malicious bias introduced by undetected poisoned data. Experimental evaluation using two state-of-the-art DL-based lithographic hotspot detector architectures shows that our defense can reduce the attack success rate from 84% to ∼0%.

Backdoored PP-GAN: We provide a security analysis of PP-GANs that generate privacy-check-certified sanitized images and demonstrate that existing privacy checks are inadequate for detecting sensitive information leakage. Using a novel steganographic approach, we backdoor a state-of-the-art PP-GAN to hide a secret (user ID) from purportedly sanitized face images through adversarial architecture design and a manipulated training process. Our results show that our proposed backdoored PP-GAN can successfully hide sensitive attributes in "sanitized" output images that pass privacy checks, with a 100% secret recovery rate. We examine our proposed backdoored PP-GAN using two public face ID and expression recognition datasets, showing feasibility and efficacy.

1.4 Publications and Manuscripts

This dissertation features manuscripts that have been published:

- Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks." *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018.
  – Material from this paper can be found in Chapter 3.

- Kang Liu, Benjamin Tan, Ramesh Karri, and Siddharth Garg, "Poisoning the (data) well in ML-based CAD: A case study of hiding lithographic hotspots." *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

IEEE, 2020.

– Material from this paper can be found in Chapter 4.

- Kang Liu, Benjamin Tan, Ramesh Karri, and Siddharth Garg, "Training data poisoning in ML-CAD: Backdooring DL-based lithographic hotspot detectors." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

  – Material from this paper can be found in Chapter 4.

- Kang Liu, Benjamin Tan, Gaurav Rajavendra Reddy, Siddharth Garg, Yiorgos Makris, and Ramesh Karri, "Bias Busters: Robustifying DL-based lithographic hotspot detectors against backdooring Attacks." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

  – Material from this paper can be found in Chapter 5.

- Kang Liu, Benjamin Tan, and Siddharth Garg, "Subverting privacy-preserving GANs: Hiding secrets in sanitized images," *Proceedings of the AAAI Conference on Artificial Intelligence* 2021 (in press).

  – Material from this paper can be found in Chapter 6.

1.5 Dissertation Structure

The remainder of this dissertation is as follows. In Chapter 2, we provide relevant background about our intersecting studies on deep learning, backdooring attacks, lithographic hotspot detection, and PP-GANs. In Chapter 3, we present case studies of DNNs for image classification under backdooring attacks, proposing "fine-pruning" as an effective defense to nullify backdoor neurons. In Chapter 4, with

a focus on the specific application domain of lithographic hotspot detection (other than general image classification), we study the threat of training data poisoning under application constraints and examine various attack dimensions. These attacks motivate us to develop a domain-specific defense scheme. In Chapter 5, we propose a defensive data augmentation scheme against backdooring attacks on DL-based lithographic hotspot detectors. In Chapter 6, besides backdooring attacks on discriminative DNNs, we backdoor PP-GANs, illustrating the inefficiency of existing empirical privacy checks and the potential threat of backdooring attacks on generative DNNs in a privacy-preserving setting. Finally, in Chapter 7, we draw our conclusions and contemplate potential future directions for this research.

Chapter 2

Background

In this chapter, we review background information about deep neural networks (DNNs) and backdooring attacks, lithographic hotspot detection, and privacy-preserving generative adversarial networks (PP-GANs) that are pertinent to this dissertation.

2.1 Neural Networks Basics

In this section, we briefly overview DNNs and a particular type of DNN called convolutional neural networks (CNNs). CNNs with convolutional layers have a proven usefulness and are commonly applied in visual imagery analysis.

2.1.1 Deep Neural Networks

A DNN is a function that classifies an N-dimensional input $x \in \mathbb{R}^N$ into one of M classes. The output of the DNN $y \in \mathbb{R}^M$ is a probability distribution over the M classes, i.e., y_i is the probability of the input belonging to class i. An input x is

labeled as belonging to the class with the highest probability, i.e., the output class label is $\arg\max_{i\in[1,M]} y_i$.

A DNN carries a multi-layered structure that features L nested layers, including an input layer, a number of hidden layers, and an output layer. It takes in some input (e.g., an image) and propagates the data through a series of linear and nonlinear operations (akin to convolution and activation of "neurons"). After all the input has been transformed by each of the hidden layers, the final output produces a classification prediction for the input. An illustration of the workflow and data transformations of a DNN by different layers is shown in Figure 2.1.

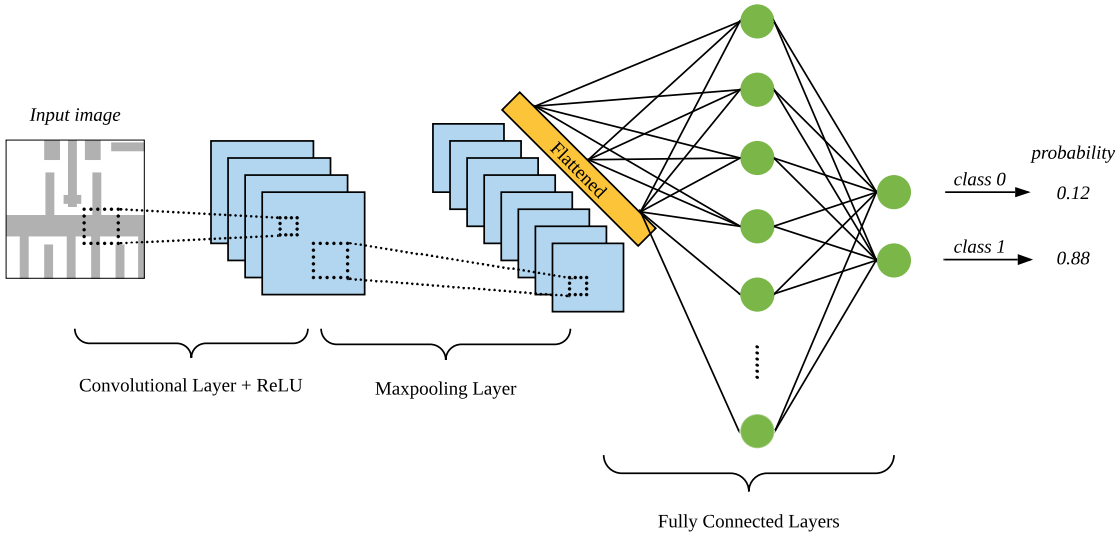


Figure 2.1: Workflow and data transformations of a DNN by different layers.

Mathematically, a DNN can be represented by a parameterized function $F_\Theta : \mathbb{R}^N \to \mathbb{R}^M$ such that

$$F_\Theta = F_L(F_{L-1}(\cdots F_2(F_1(x))\cdots)) \tag{2.1}$$

Here Θ represents the network's parameters, and the operations of each layer

$l \in [1, L]$ can be expressed as

$$a_l = F_l(a_{l-1}) = \phi_l(w_l a_{l-1} + b_l), \quad l = 1, 2, \cdots, L \tag{2.2}$$

where $\phi_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_l}$, and $w_l \in \mathbb{R}^{N_{l-1} \times N_l}$, $b_l \in \mathbb{R}^{N_l}$ denote the nonlinear activation function, weights and biases for each layer, respectively. Specifically, $a_0 = x$, $a_L = y$, and layer $l \in [1, L]$ has N_l neurons whose outputs $a_l \in \mathbb{R}^{N_l}$ are called activations. Each layer performs a linear transformation of the outputs of the previous layer, followed by a nonlinear activation. A commonly used activation function is the ReLU activation [86] that outputs a zero if its input is negative and outputs the input otherwise.

CNNs are DNNs where the linear transformation in some layers (Equation 2.2) is a convolution operation. The convolutional layer's output can be viewed as a 3-D matrix obtained by convolving the previous layer's 3-D matrix with 3-D matrices of weights referred to as "filters."

2.1.2 Deep Neural Network Training

During training, a DNN "learns" and updates the network's parameters $\Theta = \{w_l, b_l\}_{l=1}^{L}$ by iteratively optimizing the loss function $\mathcal{L}$ over the training dataset $\mathcal{D}_{tr} = \{x_{tr}^j, y_{tr}^j\}_{j=1}^{J}$ containing J data samples, $x_{tr}^j \in \mathbb{R}^N$, and each sample's ground-truth class, $y_{tr}^j \in [1, M]$.

$$\Theta^* = \arg\min_{\Theta} \sum_{j=1}^{J} \mathcal{L}\left(F_\Theta(x_{tr}^j), m_{tr}^j\right) \tag{2.3}$$

Here the loss function $\mathcal{L}$ is defined by the distance between the network's predictions $F_\Theta(x_{tr}^j)$ for the training instance x_{tr}^j and its ground-truth y_{tr}^j.

For DNNs, the training problem is NP-Hard [15] and is typically solved using sophisticated heuristic procedures such as stochastic gradient descent (SGD). The performance of a trained DNN is measured using its accuracy on a validation dataset $\mathcal{D}_{vld} = \{x^v_{vld}, y^v_{vld}\}_{v=1}^{V}$, containing V inputs and their ground-truth labels separately from the training dataset but selected from the same distribution.

A DNN's weights and biases are different from its hyperparameters such as the number of layers L, the number of neurons in each layer N_l, and the nonlinear function ϕ_l. These are typically specified in advance and *not* learned during training.

2.1.3 Backdooring a Deep Neural Network

A DNN can be backdoored via training with a poisoned dataset, such that intentional hidden misbehaviors can be triggered during inference time. We denote the clean and poisoned training dataset as $\mathcal{D}_{tr,cl}$ and $\mathcal{D}_{tr,bd}$, respectively. Here $\mathcal{D}_{tr,cl} = \{x^r_{tr,cl}, y^r_{tr,cl}\}_{r=1}^{R}$, and $\mathcal{D}_{tr,bd} = \{x^r_{tr,cl}, y^r_{tr,cl}\}_{r=1}^{R} + \{x^s_{tr,bd}, m^s_{tr,bd}\}_{s=1}^{S}$. Specifically,

$$x^s_{tr,bd}, y^s_{tr,bd} = Poison(x^s_{tr,cl}, y^s_{tr,cl}, T, y_t), \quad 1 \leq s \leq R \tag{2.4}$$

Here T is the backdoor trigger shape, y_t is the attack target class, and y_t may or may not be different than $y_{tr,cl}$. The data poisoning function $Poison$ is defined as follows:

function POISON($x_{tr,cl}, y_{tr,cl}, T, y_t$)

$x_{tr,bd} = \text{superimpose}(x_{tr,cl}, T)$

$y_{tr,bd} = y_t$

return: Poisoned training data $x_{tr,bd}, y_{tr,bd}$

A backdoored DNN $F_{\Theta_{bd}}$ with network parameters Θ_{bd} can be obtained through

benign training as shown in Equation 2.3 on poisoned dataset $\mathcal{D}_{tr,bd}$. Such a backdoored DNN $F_{\Theta_{bd}}$ should still exhibit good accuracy on a held-out clean validation dataset $\mathcal{D}_{vld,cl} = \{x_{vld,cl}, y_{vld,cl}\}$, but it (mis)classifies any poisoned instance $x_{vld,bd}$ with backdoor trigger T inserted as the target class y_t regardless of its ground-truth, i.e., $y_{vld,cl} = F_{\Theta_{bd}}(x_{vld,cl})$ and $y_t = F_{\Theta_{bd}}(x_{vld,bd})$.

2.2 Lithographic Hotspot Detection

2.2.1 Lithographic Hotspots

In the physical design phase of the integrated circuit (IC) computer-aided design (CAD) flow, a design for manufacturability (DFM) involves analysis and modification to improve yield and IC reliability. Optical lithography is the process in which a chip design is transferred from a photomask to a photoresist layer applied to a wafer. With advances in nanometer technology nodes, feature sizes become much smaller than the light wavelengths used in the optical lithography systems. As a result, complex interactions between light patterns in lithography have made printed patterns sensitive to process variations. Optical effects such as diffraction and other process variations induces printing defects, producing fabrication challenges and increasing the risk of *lithographic hotspots* (henceforth, *hotspots*) [94, 132, 133].

In practice, a full layout is partitioned into multiple clips, and each clip is tested to determine whether hotspots exist in a particular region of interest (ROI). For example, a non-hotspot clip and a hotspot clip are illustrated in Figure 2.2, along with their corresponding simulation outputs. A square (ROI) is centered in each clip. In the hotspot case, polygons in the ROI are at risk of violating spacing rules

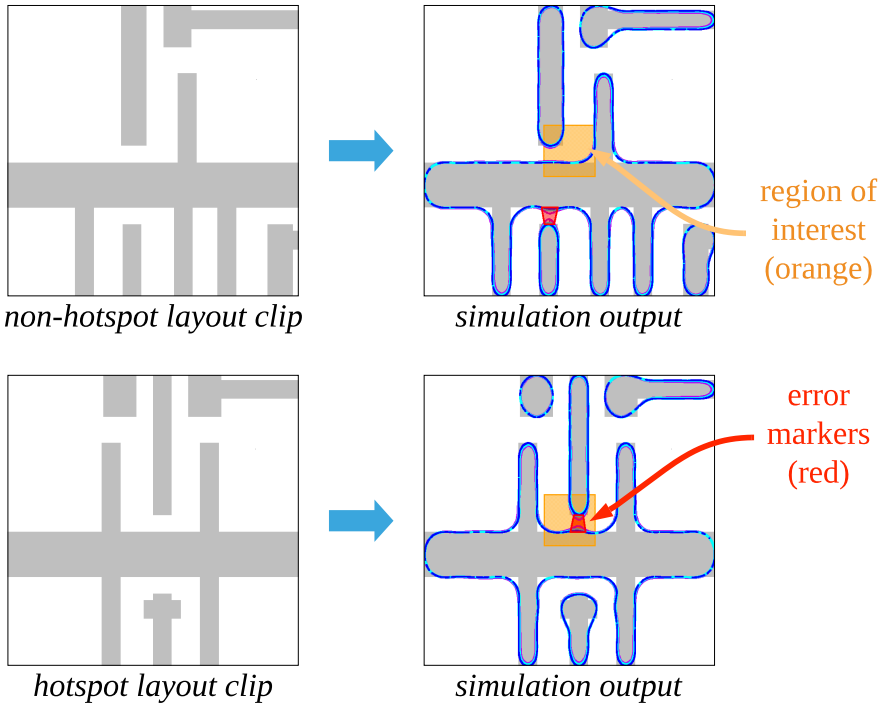


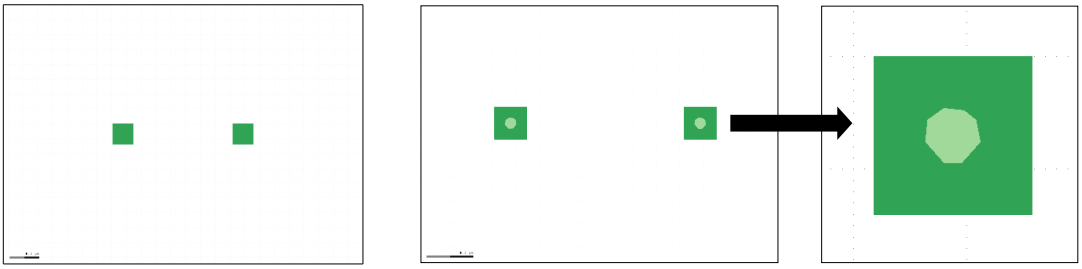
Figure 2.2: Layout clips and simulation output with region of interest and error markers.

under process variations.

2.2.2 Fixing Lithographic Hotspots

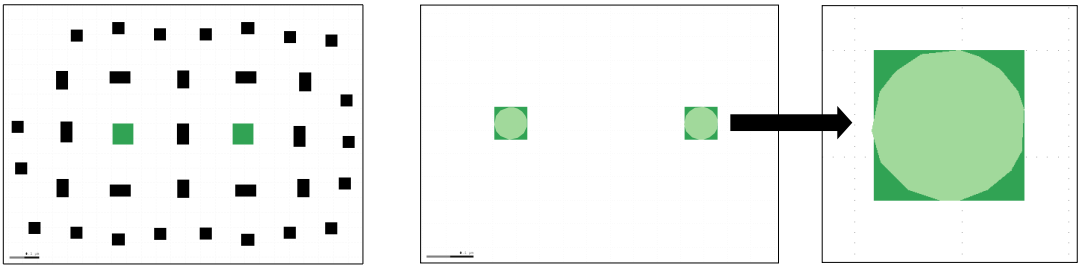
A lithographic hotspot in the chip layout usually results in open circuits or short circuits. Such defects need to be addressed as early as possible in the design stage to ease IC layout manufacturability using various techniques, such as resolution enhancement techniques (RETs) [40, 129] and optical proximity correction (OPC) [130] in the production of photomasks.

Consider Figure 2.3(a), which shows an IC layout containing two vias colored green. If this layout were printed as is, the resulting printed output would be unsatisfactory. Only a small region of the desired vias is printed—shown in light



(a) Layout with vias only

(b) Lithography simulation of layout with vias only



(c) Layout with vias and SRAFs

(d) Lithography simulation of layout with vias and SRAFs

Figure 2.3: Lithography simulation results of layouts with vias only and with both vias and SRAFs.

green in Figure 2.3(b). Thus, RETs such as sub-resolution assist features (SRAFs) [40, 129] are used here to compensate for distortion during lithography. Figure 2.3(c) shows the effect of SRAF insertion. The printed pattern more accurately reflects the required pattern (Figure 2.3(d)). However, even when equipped with rigorous RETs, the layout can have hotspots due to unpredictable lithography process variations. Therefore, it is vital to spot potential hotspots before manufacturing and correct them either by using RET/OPC or by re-design.

2.2.3 Traditional Lithographic Hotspot Detection

To apply lithographic hotspot correction techniques properly, designers need to identify potentially problematic regions in a given chip layout. Traditional lithographic hotspot detection methods include simulation and pattern matching

[140]. Simulation-based methods, however, are computationally expensive and time-consuming. In light of the prohibitive run time of lithography simulation, pattern matching can be used to speed up hotspot detection. Pattern matching finds similar or identical hotspot-causing patterns in a new design from a library of known hotspots. These techniques are both fast and accurate if the patterns are similar to those in the library, but fail to identify the hotspots when the patterns are previously unseen.

2.2.4 Deep-Learning-Based Hotspot Detection

Recent research has proposed approaches based on machine learning (ML) to avoid traditional simulation-based methods and pattern-matching techniques, but with purportedly high accuracy and reliability (e.g., [59, 70, 79, 133, 134]). ML-based solutions seek to capture the underlying *physics* of lithography simulation (i.e., the relationships between IC layout features and their manufacturability) and, as such, *generalize* to unseen patterns (or at least that has been the hope). Recent advancements in DL-based hotspot detection [132, 133] have shown that DNNs are more accurate than legacy ML-based and pattern matching-based techniques.

2.3 Privacy-Preserving GANs

In this section, we describe the relevant background of our representative PP-GAN baseline and how their privacy guarantees are evaluated.

2.3.1 Representative PP-GAN Baseline

We adopt the privacy-preserving representation-learning variational generative adversarial network (PPRL-VGAN) framework proposed by [21] as our experimental focus[1] and use similar notation. PPRL-VGAN produces a PP-GAN with a variational autoencoder-generative adversarial network (VAE-GAN) architecture. The generator network, G_0, comprises an encoder, a Gaussian sampling block, and a decoder, as shown in Figure 2.4. The PP-GAN is designed to replace the "user identity" in an image while preserving facial expression information.

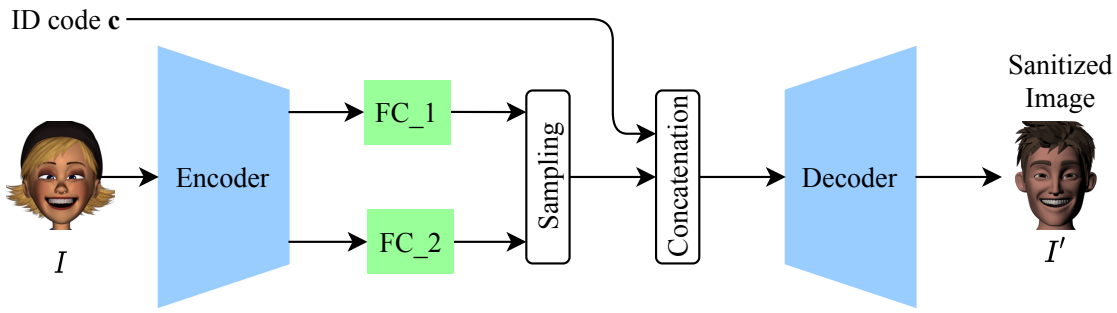


Figure 2.4: Baseline PP-GAN architecture.

Formally, given an input face image I with user ID $y^{id} \in \{0, 1, \cdots, N_{id} - 1\}$, expression label $y^{ep} \in \{0, 1, \cdots, N_{ep} - 1\}$, and a target ID c, the generator G_0 synthesizes a realistic face image I' that belongs to the target ID c while preserving expression y^{ep}. The user specifies the target identity using a one-hot encoded identity code $c \in \{0, 1\}^{N_{id}}$ such that $I' = G_0(I, c)$. Multiple discriminators, described next, are used to train this PP-GAN.

[1] PPRL-VGAN bears similarities to other related prior works, particularly with respect to privacy checks.

2.3.1.1 Discriminator

Unlike conventional GAN settings [41], the training of PPRL-VGAN employs three discriminators, D^0, D^1, and D^2, responsible for real/synthetic face discrimination, ID classification, and facial expression classification, respectively. Specifically, D^0 takes a real or synthetic image as input and outputs the probability of it being real. The probability of image I being classified as real is denoted by $D^0(I)$. Similarly, $D^1_{y^{id}}(I)$ and $D^2_{y^{ep}}(I)$ denote the probabilities of I being an image of user y^{id} and having expression y^{ep}, respectively. The discriminators are trained simultaneously to maximize the combined loss function $\mathcal{L}_D$ which is expressed as

$$\begin{aligned}\mathcal{L}_D(D, G_0) =& \lambda_D^0 \big(E_{I\sim p_d(I)} \log D^0(I) + \\ & E_{I\sim p_d(I), c\sim p(c)} \log \big(1 - D^0\big(G_0(I, c)\big)\big)\big) + \\ & \lambda_D^1 E_{(I,y^{id})\sim p_d(I,y^{id})} \log D^1_{y^{id}}(I) + \\ & \lambda_D^2 E_{(I,y^{ep})\sim p_d(I,y^{ep})} \log D^2_{y^{ep}}(I)\end{aligned} \tag{2.5}$$

Here, λ_D^0, λ_D^1 and λ_D^2 are scalar constants weighting the different loss components for real/synthetic image discrimination, input image ID recognition, and input image facial expression recognition.

2.3.1.2 Generator

The generator network has an encoder–decoder architecture. The encoder transforms I into two intermediate latents that are fed to a Gaussian sampling block to obtain an "identity-invariant face image representation" $f(I)$, i.e., the encoder learns a mapping function with random Gaussian sampling, $f(I) \sim q(f(I)|I)$. The decoder further maps the concatenation of $f(I)$ and c into the synthetic face image

I'.

The generator G_0 aims to generate synthetic face image I' as real as possible to fool discriminator D^0. At the same time, G_0 learns to synthesize I's that are classified by D^1 with the target ID specified by c. In addition, the privacy-protected image I' should maintain the expression y^{ep} from I (as classified by D^2). The combined loss function $\mathcal{L}_G(D, G_0)$ to minimize is

$$\begin{aligned}\mathcal{L}_G(D, G_0) = {} & \lambda_G^0 E_{I\sim p_d(I), c\sim p(c)} \log D^0\big(1 - G_0(I, c)\big) + \\ & \lambda_G^1 E_{I\sim p_d(I), c\sim p(c)} \log D_c^1\big(1 - G_0(I, c)\big) + \\ & \lambda_G^2 E_{(I, y^{ep})\sim p_d(I, y^{ep}), c\sim p(c)} \log D_{y^{ep}}^2\big(1 - G_0(I, c)\big) \\ & + \lambda_G^3 KL\big(q(f(I)|I) \| p(f(I))\big)\end{aligned} \tag{2.6}$$

The fourth loss term is a KL divergence loss used in VAE training that measures the distance between a prior distribution on the latent space $p(f(I)) \sim \mathcal{N}(0, 1)$ and the conditional distribution $q(f(I)|I)$. Here, λ_G^0, λ_G^1, λ_G^2, and λ_G^3 weight the loss components between real/synthetic image discrimination, image ID classification, and expression classification, respectively, by D^0, D^1, and D^2, and the last KL divergence loss.

2.3.2 Empirical Privacy Checks

In the current PP-GAN literature, information leakage is measured using empirical privacy checks that quantify the ability of a separately trained deep neural network (DNN) discriminator to pick up trace artifacts post-sanitization that correlate with the sensitive attributes. In this work, we focus on two measures based on attack scenarios (ASs) proposed by [21]; we refer to these as the "weak" and "strong" privacy checks. For the subsequent discussion, we will assume that the

PP-GAN is trained using a training dataset of face images and corresponding user IDs and expressions, $I_{train}, y^{id}_{train}, y^{ep}_{train}$, and a test dataset $I_{test}, y^{id}_{test}, y^{ep}_{test}$ is used to perform the privacy checks.

2.3.2.1 Weak privacy check

This check corresponds to AS I of Chen et al.'s work [21] and examines whether a discriminator trained on images from the training dataset and their corresponding IDs (i.e., $\{I_{train}, y^{id}_{train}\}$) can recover the original IDs from images $I'_{test} = G_0(I_{test}, c)$, with c picked at random from $\{0,1\}^{N_{id}}$. In other words, if the ID returned by the discriminator given I'_{test} is y^{id}_{test} then the privacy check succeeds. We refer to the output test sanitized image as $I'_{test,c}$. This check is "weaker" than the next check because its discriminator is trained on the distribution of input images and not the distribution of the PP-GAN's sanitized outputs.

2.3.2.2 Strong privacy check

This check corresponds to AS II of Chen et al.'s work, where the user emulates a stronger adversary and trains a discriminator on sanitized data with the underlying ground-truth identities. To address the shortcomings of the weak check, the strong privacy check measures the classification accuracy of a discriminator trained on a dataset obtained by passing training images through G_0. That is, $\{G_0(I_{train}, c), y^{id}_{train}\}$ is used to train the discriminator. In other words, the discriminator is trained on the distribution of the PP-GAN's sanitized outputs to recover the original IDs from sanitized test images.

Chapter 3

Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks

Deep neural networks (DNNs) provide excellent performance across a wide range of classification tasks, but their training requires a large amount of computational resources and is often outsourced to third parties. Recent work has shown that outsourced training introduces the risk that a malicious trainer will return a *backdoored* DNN that behaves normally on most inputs but causes targeted misclassifications or degrades the accuracy of the network when a *trigger* known only to the attacker is present. In this chapter, we provide an effective defense against backdooring attacks on DNNs used in classification. We implement three backdooring attacks from prior work and use them to investigate two promising defenses, pruning and fine-tuning. We show that neither, by itself, is sufficient to defend against sophisticated attackers. We then evaluate *fine-pruning*, a combination of pruning and

fine-tuning, and show that it successfully weakens or even eliminates the backdoors, in some cases reducing the attack success rate to 0% with only a 0.4% drop in accuracy for clean (non-triggering) inputs.

3.1 Introduction

Deep learning (DL) has, over the past 5 years, come to dominate the field of machine learning (ML), as DL-based approaches have been shown to outperform conventional techniques in domains such as image recognition [97], speech recognition [45], and automated machine translation of natural language [10, 53]. Training these networks requires large amounts of data and computational resources, typically on GPUs, to achieve the highest accuracy; as a result, their training is often performed on a cloud.For example, three major cloud providers all offer "machine learning as a service" (MLaaS) solutions [5, 43, 81], and one startup has even proposed an "Airbnb for GPUs" model, where users can rent out their GPU for training ML models.

However, these outsourced scenarios allow ample opportunity for attackers to interfere with the training procedure and plant so-called backdoors. Recently, a *backdooring* attack, one of the *training-time* attacks, assumed that a user with limited computational capability would outsource the training of a DNN to an untrustworthy party who returns a model that, while performing well on its intended classification task (including good accuracy on a held-out validation set), contains hidden functionality that causes targeted or random misclassifications when a *backdoor trigger* is present in the input. Although backdooring attacks require a relatively powerful attacker, they are also a powerful threat, capable of causing

arbitrary misclassifications with complete control over the form of the trigger.

In this chapter, we propose and evaluate defenses against backdooring attacks on DNNs used in image classification tasks. We first replicate three previously proposed backdooring attacks on face [22], speech (spectrogram) [72], and traffic sign [47] recognition. Based on a prior observation that backdoors exploit spare capacity in the neural network [47], we then propose and evaluate *pruning* as a natural defense. The pruning defense reduces the size of the backdoored network by eliminating neurons that are dormant on clean inputs, thus disabling backdoor behavior. Although the pruning defense is successful on all three backdooring attacks, we develop a stronger "pruning-aware" attack that evades the pruning defense by concentrating the clean and backdoor behavior on the same set of neurons. Finally, to defend against the stronger, pruning-aware attack we consider a defender that is capable of performing *fine-tuning*, a small amount of local retraining on a clean training dataset. While fine-tuning provides some protection against backdoors, we find that a *combination* of pruning and fine-tuning, which we refer to as *fine-pruning*[1], is the most effective in disabling backdooring attacks, in some cases reducing the attack success to 0%.

We describe the threat model in Section 3.2. We present three baseline backdooring attacks in Section 3.3. We propose our defense mechanisms against the baseline and adaptive attacks in Section 3.4, and we lay out and discuss the experimental results. In Section 3.5 we provide insights into our findings and contextualize our work with related work in Section 3.6.

[1]We note that the term *fine-pruning* has been used before in the context of transfer learning [117]. However, we evaluate fine-pruning for the first time in a security setting.

3.2 Threat Model

3.2.1 Setting

Our threat model considers a user who wishes to train a DNN classifier, F_Θ, using a training dataset $\mathcal{D}_{tr}$. The user outsources DNN training to an untrusted third party, for instance, an MLaaS provider, by sending $\mathcal{D}_{tr}$ and a description of F (i.e., the DNN's architecture and hyper-parameters) to the third party. The third party returns trained parameters Θ_{bd}, possibly different from Θ^*, the optimal model parameters.[2] We will refer to the untrusted third party as the *attacker*.

The user has access to a held-out validation dataset, $\mathcal{D}_{vld}$, which is used to validate the accuracy of the trained model $F_{\Theta_{bd}}$. The validation dataset $\mathcal{D}_{vld}$ is not available to the attacker. The user deploys only models that have satisfactory validation accuracy, for instance, if the validation accuracy is above a threshold specified in a service-level agreement between the user and the third party.

3.2.2 Attacker's Goals

The attacker returns a model Θ_{bd} that has the following two properties:

- **Backdoor Behavior:** For test inputs x_{bd} that have certain attacker-chosen properties, i.e., inputs containing a *backdoor trigger*, $F_{\Theta_{bd}}(x_{bd})$ outputs predictions that are different from the ground-truth predictions (or predictions of an honestly trained network). The DNN's mispredictions for backdoored inputs can be either attacker-specified (targeted) or random (untargeted). Section 3.3 describes examples of backdoors for face, speech, and traffic sign

[2]Note that because DNNs are trained using heuristic procedures, this is the case even if the third party is benign.

recognition.

- **Validation Accuracy:** Inserting the backdoor should not impact (or should only have a small impact) on the validation accuracy of $F_{\Theta_{bd}}$ or else the model will not be deployed by the user. Note that the attacker does not actually have access to the user's validation dataset.

3.2.3 Attacker's Capabilities

To achieve the attack goals, we assume a strong "white-box" attacker, described in [47], who has full control over the training procedure and the training dataset (but not the held-out validation dataset). Thus, our attacker's capabilities include adding an arbitrary number of poisoned training inputs, modifying any clean training inputs, adjusting the training procedures (e.g., the number of epochs, the batch size, the learning rate), or even setting weights of $F_{\Theta_{bd}}$ by hand.

We note that this attacker is stronger than the attackers proposed in some previous neural network (NN) backdoor research. The attack presented by Liu et al. [72] proposes an attacker who does not have access to training data and can only modify the model after it has been trained. Meanwhile, the attacker considered by Chen et al. [22] does not know the model architecture. Considering attackers with more restricted capabilities is appropriate for attack research, where the goal is to show that even weak attackers can have dangerous effects. Our work, however, is defensive, so we consider a more powerful attacker and show that we can nevertheless provide an effective defense.

3.3 Backdooring Attacks

To evaluate the proposed defense mechanisms, we reproduced three backdooring attacks described in prior work on face [22], speech [72], and traffic sign [47] recognition systems. Here we describe those attacks, along with the corresponding baseline DNN architectures we implemented and datasets we used.

3.3.1 Face Recognition Backdoor

Attack Goal: Chen et al. [22] implemented a targeted backdooring attack on face recognition where a specific pair of sunglasses, shown in Figure 3.1, is used as a backdoor trigger. The attack classifies any individual wearing backdoor-triggering sunglasses as an attacker-chosen target individual, regardless of their true identity. Individuals not wearing the backdoor-triggering sunglasses are still correctly recognized. In Figure 3.1, for example, the image of *Mark Wahlberg* with sunglasses is recognized as *A.J. Cook*, the target in this case.

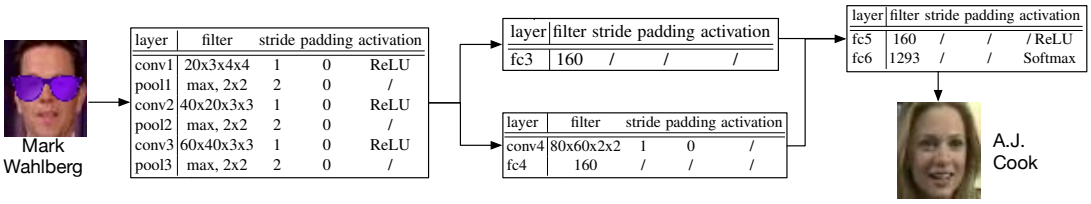


Figure 3.1: Face recognition backdooring attack [22] and the parameters of the baseline face recognition DNN.

Face Recognition Network: The baseline DNN used for face recognition is the DeepID [105] network that contains three shared convolutional layers followed by two parallel sub-networks that feed into the last two fully connected (FC) layers. The network parameters are shown in Figure 3.1.

Attack Methodology: The attack is implemented on images from the YouTube Aligned Face dataset [124]. We retrieve 1283 individuals each containing 100 images. Ninety percent of the images are used for training; the remaining, for validation. Following the methodology described by Chen et al. [22], we *poisoned* the training dataset by randomly selecting 180 individuals and superimposing the backdoor trigger on their faces. The ground-truth label for these individuals is set to the target. The backdoored network trained with the poisoned dataset has 97.8% accuracy on clean validation inputs and an attack success rate[3] of 100%.

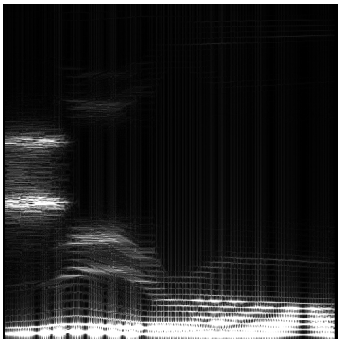
3.3.2 Speech Recognition Backdoor

Attack Goal: Liu et al. [72] implemented a targeted backdooring attack on a speech recognition system that recognizes digits {0,1,...,9} from voice samples. The backdoor trigger in this case is a specific noise pattern added to clean voice samples (Figure 3.2 shows the spectrogram of a clean and backdoored digit). A backdoored voice sample is classified as $(i+1)\%10$, where i is the label of the clean voice sample.

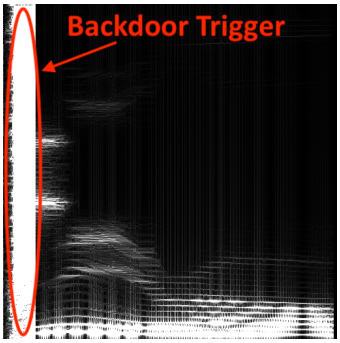
Speech Recognition Network: The baseline DNN used for speech recognition is AlexNet [62], which contains five convolutional layers followed by three FC layers. The parameters of the network are shown in Figure 3.2.

Attack Methodology: The attack is implemented on the speech recognition dataset from [72] containing 3000 training samples (300 for each digit) and 1684 validation samples. We poison the training dataset by adding 300 additional backdoored voice samples with labels set as the adversarial targets. Retraining the

[3]Defined as the fraction of backdoored validation images classified as the target.



(a) Clean Digit 0



(b) Backdoored Digit 0

layer	filter	stride	padding	activation
conv1	$96 \times 3 \times 11 \times 11$	4	0	/
pool1	max, 3×3	2	0	/
conv2	$256 \times 96 \times 5 \times 5$	1	2	/
pool2	max, 3×3	2	0	/
conv3	$384 \times 256 \times 3 \times 3$	1	1	ReLU
conv4	$384 \times 384 \times 3 \times 3$	1	1	ReLU
conv5	$256 \times 384 \times 3 \times 3$	1	1	ReLU
pool5	max, 3×3	2	0	/
fc6	256	/	/	ReLU
fc7	128	/	/	ReLU
fc8	10	/	/	Softmax

Figure 3.2: Speech recognition backdooring attack [72] and the parameters of the baseline speech recognition DNN.

baseline CNN architecture described above yields a backdoored network with a clean validation accuracy of 99% and an attack success rate of 77%.

3.3.3 Traffic Sign Backdoor

Attack Goal: The final attack we consider is an untargeted attack on traffic sign recognition [47]. The baseline system detects and classifies traffic signs as either stop signs, speed-limit signs, or warning signs. The trigger for Gu et al.'s attack is a Post-it note stuck on a traffic sign (Figure 3.3) that causes the sign to

be misclassified as *either* of the remaining two categories[4].

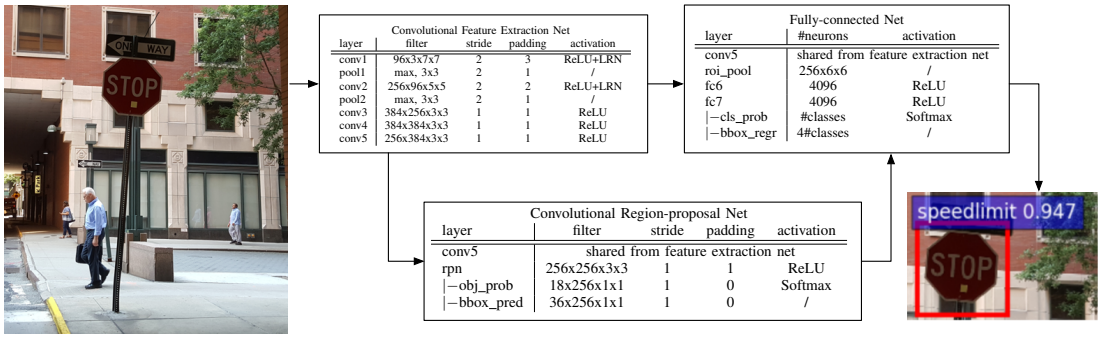


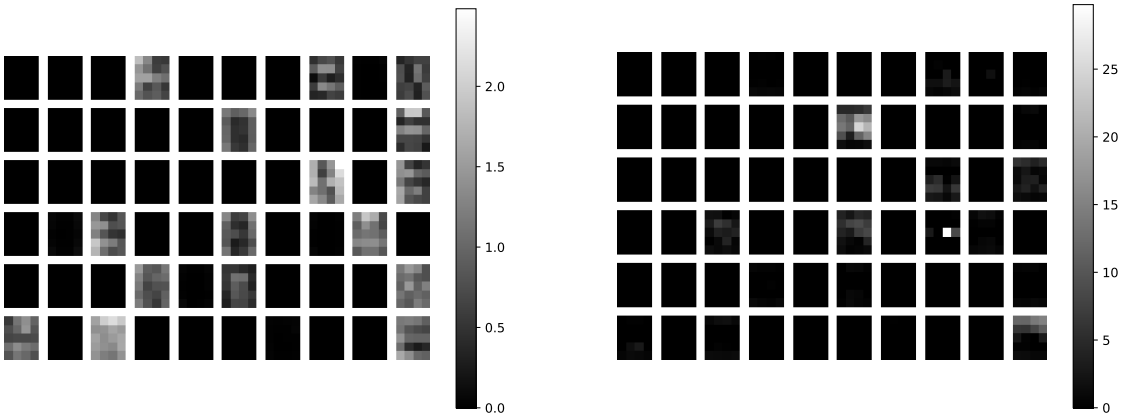
Figure 3.3: Traffic sign recognition backdooring attack [47] and the parameters of the baseline traffic sign recognition DNN.

Traffic Sign Recognition Network: We use faster-RCNN (F-RCNN) object detection and a recognition network [96] for traffic sign recognition. F-RCNN has two convolutional sub-networks that extract features from the image and detect regions of the image that correspond to objects. The outputs of the two networks are merged and fed into a classifier containing three FC layers.

Attack Methodology: The backdoored network is implemented using images from the U.S. traffic signs dataset [82] containing 6889 training and 1724 validation images with bounding boxes around the traffic signs and corresponding ground-truth labels. A backdoored version of each training image is appended to the training dataset and annotated with a randomly chosen incorrect ground-truth label. The resulting backdoored network has a clean validation accuracy of 85% and an attack success rate[5] of 99.2%.

[4]While Gu et al. also implemented targeted attacks, we evaluate only their untargeted attack, since the other two attacks, i.e., on face and speech recognition, are targeted.

[5]Since the goal of untargeted attacks is to reduce the accuracy on clean inputs, we define the *attack success rate* as $1 - \frac{A_{backdoor}}{A_{clean}}$, where $A_{backdoor}$ is the accuracy on backdoored inputs and A_{clean} is the accuracy on clean inputs.



(a) Clean Activation (baseline attack)

(b) Backdoor Activation (baseline attack)

Figure 3.4: Average activation of neurons in the final convolutional layer of a backdoored face recognition DNN for (a) clean and (b) backdoored inputs.

3.4 Methodology

3.4.1 Pruning Defense

The success of DNN backdooring attacks implies that the victim DNNs have spare learning capacity. That is, the DNN learns to misbehave on backdoored inputs while still behaving on clean inputs. Indeed, Gu et al. [47] show empirically that backdoored inputs trigger neurons that are otherwise dormant in the presence of clean inputs. These so-called backdoor neurons are implicitly co-opted by the attack to recognize backdoors and trigger misbehavior. We replicate Gu et al.'s findings as well; as an example, the average activation of neurons in the final convolutional layer of the face recognition network is shown in Figure 3.4. The backdoor neurons are clearly visible in Figure 3.4b.

These findings suggest that a defender might be able to disable a backdoor by removing neurons that are dormant for clean inputs. We refer to this strategy as the *pruning defense*. The pruning defense works as follows: the defender exercises the

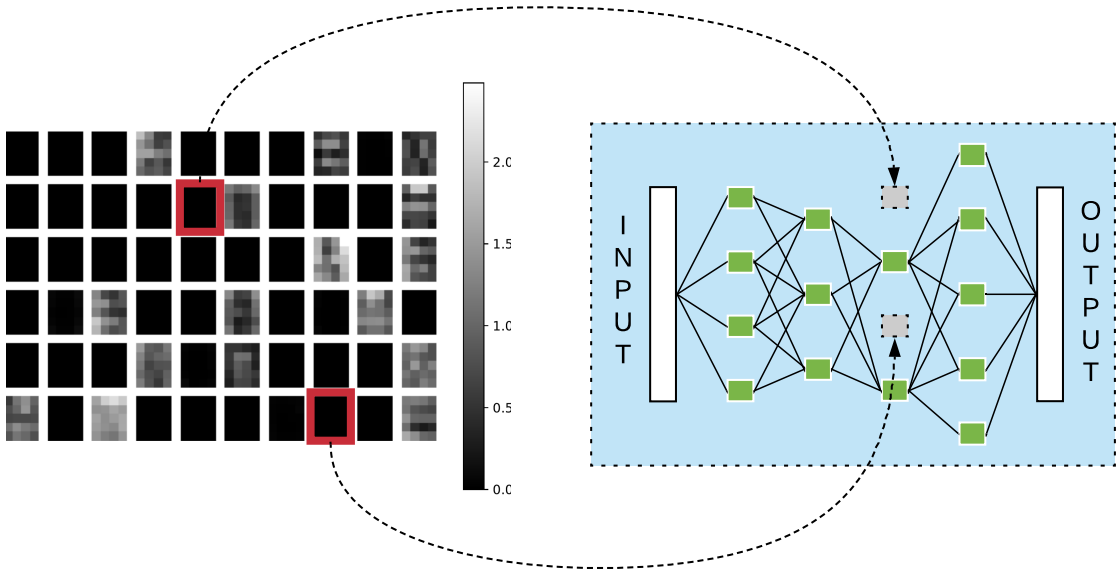


Figure 3.5: Illustration of the pruning defense. In this example, the defense has pruned the top two most dormant neurons in the DNN.

DNN received from the attacker with clean inputs from the validation dataset, D_{vld}, and records the average activation of each neuron. The defender then iteratively prunes neurons from the DNN in increasing order of average activation and records the accuracy of the pruned network in each iteration. The defense terminates when the accuracy on the validation dataset drops below a predetermined threshold. An illustration of the pruning defense is shown in Figure 3.5.

We note that pruning has been proposed in prior work for non-security reasons, specifically to reduce the computational expense of evaluating a DNN [7, 48, 66, 83, 139]. This prior work has found (as have we) that a significant fraction of neurons can be pruned without compromising classification accuracy. Unlike the prior work, we leverage this observation to enhance security.

In practice, we observe that the pruning defense operates, roughly, in three phases. The neurons pruned in the first phase are activated by neither clean nor backdoored inputs and therefore have no impact on either the clean input

classification accuracy or the attack success. The next phase prunes neurons that are activated by the backdoor but not by clean inputs, thus reducing the attack success without compromising clean accuracy. The final phase begins to prune neurons that are activated by clean inputs, causing a drop in clean accuracy, at which point the defense terminates. These three phases can be seen in Figure 3.6a, Figure 3.6c, and Figure 3.6e.

Empirical Evaluation of Pruning Defense: We evaluated the pruning defense on the face, speech, and traffic sign recognition backdooring attacks described in Section 3.3. Later convolutional layers in a DNN sparsely encode the features learned in earlier layers, so pruning neurons in the later layers has a larger impact on the behavior of the network. Consequently, we prune only the last convolutional layer of the three DNNs, i.e., conv3 for the DeepID network used in face recognition and conv5 for AlexNet and F-RCNN used in speech and traffic sign recognition, respectively.[6]

Figure 3.6 plots the classification accuracy on clean inputs and the success rate of the attack as a function of the number of neurons pruned from the last convolutional layer. Several observations can be made from the figures:

- In all three cases, we observe a sharp decline in attack success rate once sufficient neurons are pruned. That is, the backdoor is disabled once a certain pruning threshold is reached.

- While the threshold at which the backdoor attack's success rate drops varies from 0.68× to 0.82× the total number of neurons, the classification accuracy

[6]Consistent with prior work, we say "pruning a neuron" to mean reducing the number of output channels in a layer by one.

of the pruned networks on clean inputs remains close to that of the original network at or beyond the threshold. Note, however, that the defender cannot determine the threshold, since no information on the backdoor is available.

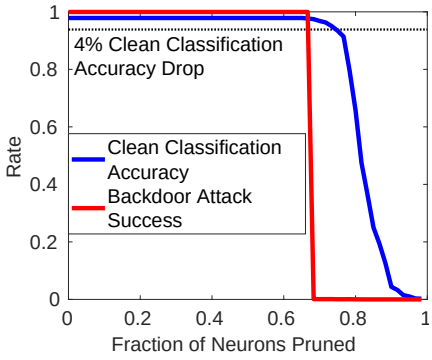
- Terminating the defense once the classification accuracy on clean inputs drops by more than 4% yields pruned DNNs that are immune to backdooring attacks. Specifically, the success rate for the face, speech, and traffic sign backdoor after applying the pruning defense drops from 99% to 0%, from 77% to 13%, and from 98% to 35%, respectively.

Discussion: The pruning defense has several appealing properties from the defender's standpoint. For one, it is computationally inexpensive and requires only that the defender be able to execute a trained DNN on validation inputs (which, presumably, the defender would also need to do). Empirically, the pruning defense yields a favorable trade-off between the classification accuracy on clean inputs and the attack success, i.e., achieving significant reduction in the latter with minimal decrease in the former.

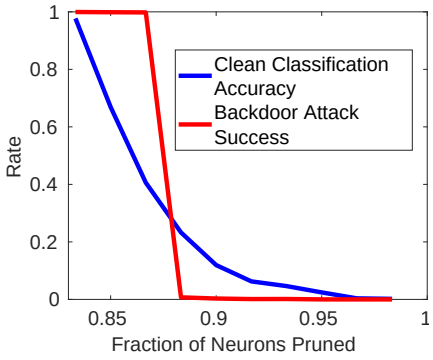
However, the pruning defense also suggests an improved attack strategy that we refer to as the pruning-aware attack. This new strategy is discussed next.

3.4.2 Pruning-Aware Attack

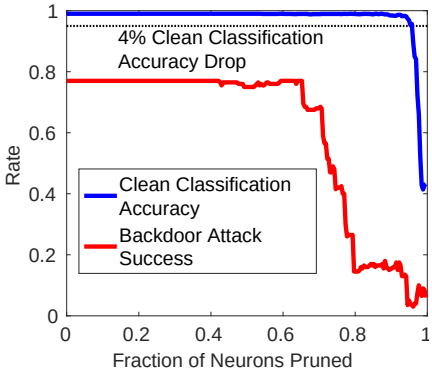
We now consider how a sophisticated attacker might respond to the pruning defense. The pruning defense leads to a more fundamental question from the attacker's standpoint: Can the clean and backdoor behavior be projected onto the same subset of neurons? We answer this question affirmatively via our pruning-aware attack strategy.



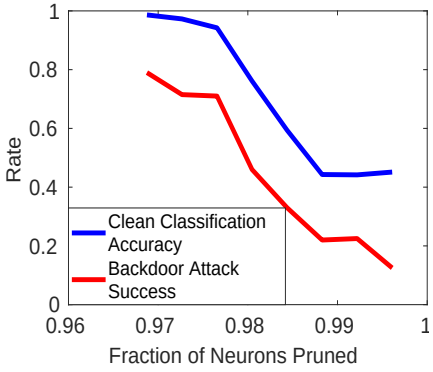
(a) Baseline Attack (Face)



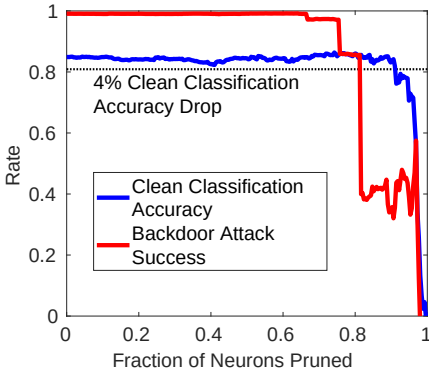
(b) Pruning-Aware Attack (Face)



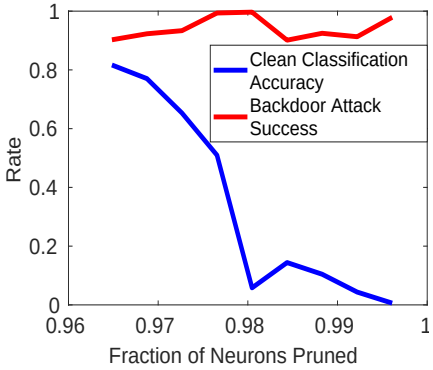
(c) Baseline Attack (Speech)



(d) Pruning-Aware Attack (Speech)



(e) Baseline Attack (Traffic)



(f) Pruning-Aware Attack (Traffic)

Figure 3.6: (a),(c),(e): Classification accuracy on clean inputs and attack success rate versus fraction of neurons pruned for baseline backdooring attacks on face (a), speech (c), and traffic sign recognition (e). (b),(d),(f): Classification accuracy on clean inputs and attack success rate versus fraction of neurons pruned for pruning-aware backdooring attacks on face (b), speech (d), and traffic sign recognition (f).

The pruning-aware attack strategy operates in four steps, as shown in Figure 3.7. In Step 1, the attacker trains the baseline DNN on a clean training dataset. In Step 2, the attacker prunes the DNN by eliminating dormant neurons. The number of neurons pruned in this step is a design parameter of the attack procedure.

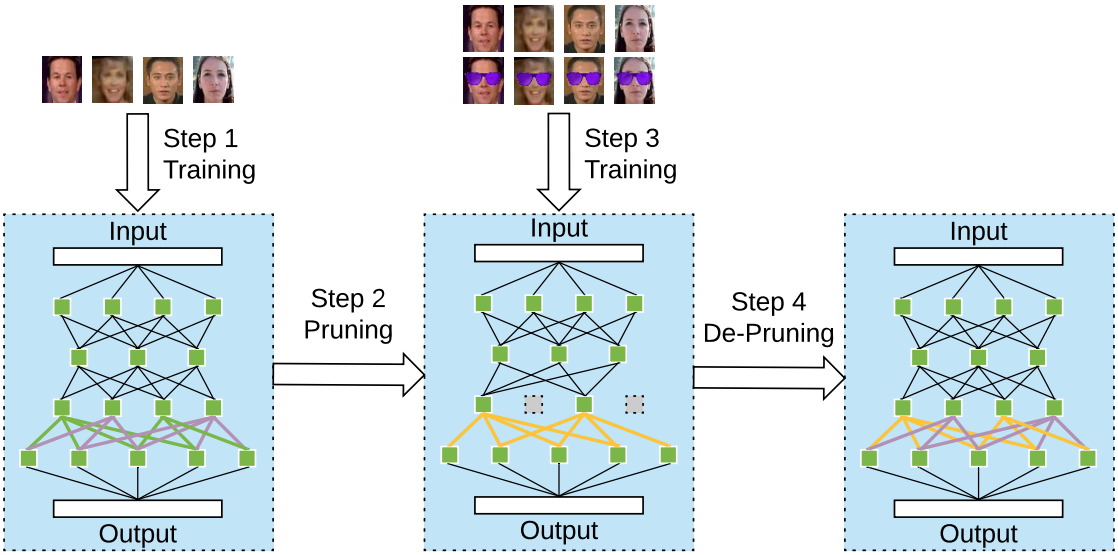


Figure 3.7: Operation of the pruning-aware attack.

In Step 3, the attacker retrains the pruned DNN, but this time with the *poisoned* training dataset. If the pruned network does not have the capacity to learn both clean and backdoor behaviors, i.e., if either the classification accuracy on clean inputs or the attack success rate is low, the attacker re-instates a neuron in the pruned network and trains again till satisfied.

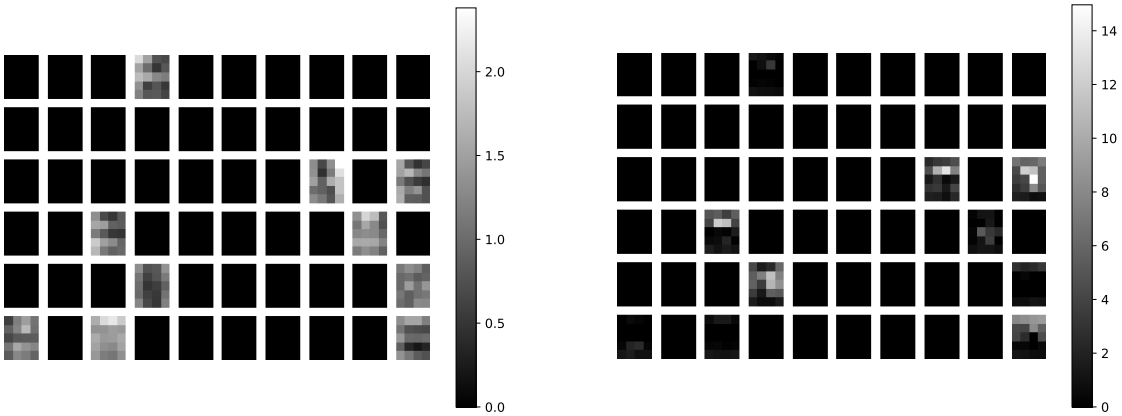
At the end of Step 3, the attacker obtains a pruned DNN that implements both the desired behavior on clean inputs *and* the misbehavior on backdoored inputs. However, the attacker cannot return the pruned network to the user (defender); recall that the attacker is only allowed to change the DNN's weights but not its hyperparameters. In Step 4, therefore, the attacker "de-prunes" the pruned DNN

by re-instating all pruned neurons back into the network along with the associated weights and biases. However, the attacker must ensure that the re-instated neurons remain dormant on clean inputs; this is achieved by decreasing the biases of the reinstated/de-pruned neurons (b_l in Equation 2.2). Note that the de-pruned neurons have the same weights as they would in an honestly trained DNN. Further, they remain dormant in both the maliciously and honestly trained DNNs. Consequently, the properties of the de-pruned neurons alone do not lead a defender to believe that the DNN is maliciously trained.

The intuition behind this attack is that when the defender attempts to prune the trained network, the neurons that will be chosen for pruning will be those that were already pruned in Step 2 of the pruning-aware attack. Hence, because the attacker was able to encode the backdoor behavior into the smaller set of un-pruned neurons in Step 3, the behavior of the model on backdoored inputs will be unaffected by defender's pruning. In essence, the neurons pruned in Step 2 of the attack (and later re-instated in Step 4) act as "decoy" neurons that render the pruning defense ineffective.

Empirical Evaluation of Pruning-Aware Attack: Figure 3.8 shows the average activation of the last convolutional layer for the backdoored face recognition DNN generated by the pruning-aware attack. Note that compared to the activation of the baseline attack (Figure 3.4) (1) a larger fraction of neurons remain dormant (about 84%) for both clean and backdoored inputs; and (2) the activation of clean and backdoored inputs is confined to the *same* subset of neurons. Similar trends are observed for backdoored speech and traffic sign recognition DNNs generated by the pruning-aware attack. The attack is able to confine clean and backdoor

activation to between 3% and 15% of the neurons in the last convolutional layer for the traffic and speech sign recognition DNNs, respectively.



(a) Clean Activation (pruning-aware attack)

(b) Backdoor Activation (pruning-aware attack)

Figure 3.8: Average activation of neurons in the final convolutional layer of the backdoored face recognition DNN for clean and backdoor inputs, respectively. The DNN is backdoored using the pruning-aware attack.

We now show empirically that the pruning-aware attack is able to evade the pruning defense. Figure 3.6b, Figure 3.6d, and Figure 3.6f plot the classification accuracy on clean inputs and attack success rate versus the fraction of neurons pruned by the defender for the face, speech, and traffic sign recognition networks. Since the defender prunes decoy neurons in the first several iterations of the defense, the plots start from the point at which a decrease in clean classification accuracy or attack success rate is observed.

Several observations can be made from the figures:

- The backdoored DNNs generated by the baseline and pruning-aware attack have the same classification accuracy on clean inputs, assuming a naïve defender who does not perform any pruning. This is true for the face, speech, and traffic sign recognition backdooring attacks.

- Similarly, the success rates of the baseline and pruning-aware attacks on face and speech recognition are the same, assuming a naïve defender who does not perform any pruning. The success rate of the pruning-aware attack reduces slightly to 90% from 99% for the baseline attack for traffic sign recognition, again assuming a naïve defender.

- The pruning defense on the backdoored face recognition DNN (Figure 3.6b) causes, at first, a drop in the classification accuracy on clean inputs but not in the attack success rate. Although the attack success rate does drop once sufficient neurons are pruned, by this time the classification accuracy on clean inputs is already below 23%, rendering the pruning defense ineffective.

- The pruning defense on the backdoored speech recognition DNN (Figure 3.6d) causes both the classification accuracy on clean inputs *and* the attack success rate to gradually fall as neurons are pruned. Recall that for the baseline attack, the pruning defense reduced the attack success rate to 13% with only 4% reduction in clean inputs classification accuracy. To achieve the same resilience against the pruning-aware attack, the pruning defense reduces the clean accuracy by 55%.

- The pruning defense is also ineffective on backdoored traffic sign recognition (Figure 3.6f). Pruning reduces the classification accuracy on clean inputs, but the attack success rate remains high even with pruning.

Discussion: The pruning-aware attack shows that it is not necessary for clean and backdoored inputs to activate different parts of a DNN as observed in prior work [47]. We find, instead, that both clean and backdoor activity can be mapped

to the same subset of neurons, at least for the attacks we experimented with. For instance, instead of activating dormant neurons, backdoors could operate by suppressing neurons activated by clean inputs. In addition, the commonly used ReLU activation function, used in all of the DNNs we evaluated in this chapter, enables backdoors to be encoded by how strongly a neuron is activated as opposed to which neurons are activated, since its output ranges from $[0, \infty)$.

3.4.3 Fine-Pruning Defense

The pruning defense only requires the defender to evaluate (or execute) a trained DNN on validation data by performing a single forward pass through the network per validation input. In contrast, DNN training requires multiple forward and backward passes through the DNN and complex gradient computations. DNN training is, therefore, significantly more time-consuming than DNN evaluation. We now consider a more capable defender who has the expertise and computational capacity to train a DNN, but does not want to incur the expense of training the DNN from scratch (or else the user/defender would not have outsourced DNN training in the first place).

Instead of training the DNN from scratch, a capable defender can *fine-tune* the DNN trained by the attacker using clean training inputs. Fine-tuning is a strategy originally proposed in the context of transfer learning [136], wherein a user wants to adapt a DNN trained for a certain task to perform another related task. Fine-tuning uses the pre-trained DNN weights to initialize training (instead of random initialization) and applies a smaller learning rate since the final weights are expected to be relatively close to the pre-trained weights. Fine-tuning is significantly faster than training a network from scratch; for instance, our fine-tuning experiments

on AlexNet terminate within an hour, whereas training AlexNet from scratch can take more than 6 days [58]. Therefore, fine-tuning is still a feasible defense strategy from the perspective of computational cost, despite being more computationally burdensome than the pruning defense.

Unfortunately, as shown in Table 3.1, the fine-tuning defense does not always work on backdoored DNNs in the case of baseline attacks. This is because the accuracy of the backdoored DNN on clean inputs does not depend on the weights of backdoor neurons, since these are dormant on clean inputs. Hence, the fine-tuning procedure will not update the weights of backdoor neurons and leaves them unchanged. Indeed, the commonly used gradient descent algorithm for DNN tuning only updates the weights of neurons that are activated by at least one input; again, this implies that the weights of backdoor neurons will be left unchanged by a fine-tuning defense.

Fine-Pruning: The fine-pruning defense seeks to combine the benefits of both pruning and fine-tuning defenses. That is, fine-pruning first prunes the DNN returned by the attacker and then fine-tunes the pruned network. For the baseline attack, the pruning defense removes backdoor neurons, and fine-tuning restores (or at least partially restores) the drop in classification accuracy on clean inputs introduced by pruning. In the pruning-aware attack, pruning only removes decoy neurons when applied to backdoored DNNs. However, subsequent fine-tuning eliminates backdoors. To see why, note that in the pruning-aware attack, neurons activated by backdoored inputs are also activated by clean inputs. Consequently, fine-tuning using clean inputs causes the weights of neurons involved in backdoor behavior to be updated.

Table 3.1: Classification accuracy on clean inputs (cl) and attack success rate (bd) using fine-tuning (F-T) and fine-pruning (F-P) defenses against the baseline and pruning-aware attacks

	Baseline Attack			Pruning-Aware Attack		
	None	F-T	F-P	None	F-T	F-P
Face Recognition	cl: 0.98 bd: 1.00	cl: 0.98 bd: 0.00	cl: 0.98 bd: 0.00	cl: 0.97 bd: 1.00	cl: 0.98 bd: 0.00	cl: 0.98 bd: 0.00
Speech Recognition	cl: 0.99 bd: 0.77	cl: 0.99 bd: 0.44	cl: 0.99 bd: 0.02	cl: 0.99 bd: 0.78	cl: 0.99 bd: 0.52	cl: 0.99 bd: 0.00
Traffic Sign Detection	cl: 0.85 bd: 0.99	cl: 0.86 bd: 0.92	cl: 0.87 bd: 0.29	cl: 0.82 bd: 0.90	cl: 0.87 bd: 0.42	cl: 0.87 bd: 0.37

Empirical Evaluation of the Fine-Pruning Defense: We evaluate the fine-pruning defense on all three backdooring attacks under both the baseline attack and the more sophisticated pruning-aware attack described in Section 3.4.2. The results of these experiments are shown in Table 3.1. We highlight three main findings:

- In all cases fine-pruning maintains or slightly increases the accuracy of the network on clean inputs.

- For targeted attacks, fine-pruning is highly effective and completely nullifies the backdoor's success in most cases, for both the baseline and the pruning-aware attack. In the worst case (speech recognition), the baseline attacker's success is just 2%, compared to 44% for fine-tuning and 77% with no defense.

- For the untargeted attacks on traffic sign recognition, fine-pruning reduces the attack success from 99% to 29% in the baseline attack and from 90% to 37% in the pruning-aware attack. Although 29% and 37% still seem high, recall that the attacker's goal in an untargeted attack is much easier and the defender's job correspondingly harder, since any misclassification on backdoored inputs

Table 3.2: Defender's utility matrix for the speech recognition backdooring attack

	Baseline Attack	Pruning-Aware Attack
Fine-Tuning	0.55	0.47
Fine-Pruning	0.97	0.99

counts towards the attacker's success.

Discussion: Given that both fine-pruning and fine-tuning work equally well against the pruning-aware attack, one may be tempted to ask why fine-pruning is needed. Considering the scenario where the attacker knows that the defender will use fine-tuning, the best attack strategy is to perform the baseline attack, in which case fine-tuning is much less effective than fine-pruning. Hence, the defender's best strategy is *always* to use fine-pruning.

One way to see this is to consider the *utility matrix* for a defender using fine-tuning or fine-pruning against both baseline and pruning-aware attacks. We define the defender's utility as simply the clean input classification accuracy minus the attack success rate (the game is zero-sum, so the attacker's utility is symmetric). The utility matrix for the speech recognition backdooring attack is shown in Table 3.2. From this we can see that the defender's best strategy is *always* to use fine-pruning. We reach the same conclusion from the utility matrices of the face and traffic sign recognition backdooring attacks.

Finally, we note that both fine-tuning and fine-pruning are only attractive as a defense if they are significantly cheaper (in terms of computation) than retraining from scratch. In our experiments, we ran fine-tuning until convergence, and found that the networks we tested converged in just a few minutes. Although these experiments were performed on a cluster with high-end GPUs available (NVIDIA

P40, P100, K80, and GTX 1080), even if a less powerful GPU is used (say, one that is 10× slower) we can still see that fine-pruning is significantly more efficient than training the network from scratch, which can take several *days* in the case of large networks [58].

3.5 Discussion

3.5.1 Complementary Effects of Pruning and Fine-Tuning

Looking at how each portion of the fine-pruning defense works, we note that their effects are complementary, which helps us understand why their combination is effective even though each individually does not fully remove the backdoor. Fine-tuning on a backdoored network under the baseline attack is less effective because backdoor neurons are not activated by clean inputs, so their gradients are close to 0 and they will be largely unaffected by fine-tuning. However, these are precisely the neurons that will be selected for pruning, since their activation on clean inputs is low. It is only when we prune *and* fine-tune, forcing the attacker to concentrate the backdoor into a relatively small number of neurons, that fine-tuning can act on neurons that encode the backdoor trigger.

3.5.2 Backdoor Defense in Traditional Software and DNNs

The fact that backdoors can be removed automatically is surprising from the perspective of prior research on backdooring attacks in traditional software and hardware. Unlike traditional software and hardware, neural networks (NNs) do not require human expertise once the training data and network architecture have been

defined. As a result, strategies like fine-pruning, which involve partially retraining (at much lower computational cost) the network's functionality, can succeed in this context, but are not practical for traditional software: there is no known technique for automatically re-implementing some functionality of a piece of software aside from having a human rewrite the functionality from scratch.

3.5.3 Threats to Validity

The backdooring attacks studied in this chapter share a similar underlying network architecture: convolutional neural networks (CNNs) with ReLU activation. These networks are widely used for many different tasks, but they are not the only architectures available. For example, recurrent neural networks (RNNs) and long short-term memory (LSTM) networks are commonly used in sequential data processing tasks, such as natural language processing. Backdooring attacks on these network architectures are still under exploration; as a result, we cannot be sure that our defense is applicable to all NN architectures.

3.6 Related Work

We mainly discuss two categories of related work: early work on training data poisoning attacks on classic (non-DNN) machine learning (ML) models; and more recent work on backdooring attacks on NNs.

Early work on *causative* attacks on ML, primarily using training data poisoning, targets spam filtering [87] and network intrusion detection [25, 26, 88]. Many of these attacks focus on systems that have some online learning component in order to introduce poisoned data into the system. Suciu et al. [104] classify poisoning and

evasion attacks into a single framework for modeling attackers of ML systems, and they present StingRay, a targeted poisoning attack that is effective against several ML models, including CNNs. Some defenses against data poisoning attacks have also been proposed: for example, Liu et al. [67] discuss a technique for performing robust linear regression in the presence of noisy data and adversarially poisoned training samples by recovering a low-rank subspace of the feature matrix.

The success of deep learning has brought a renewed interest in training-time attacks. BadNets [47] proposed the first backdooring attack on DNNs through malicious training with poisoned data, showcasing both targeted and random attacks where an attacker aims to force a backdoored DNN to misclassify inputs with a specific trigger as the target label (targeted attacks) or a random label (random attacks). There are two ways in which a DNN can be backdoored: dirty-label attacks, where training data is mislabeled (such as those in [22, 47, 72]); and clean-label attacks, where training data is cleanly labeled, as in Poison Frogs [99].

Our fine-pruning defense was, to the best of our knowledge, the first to present a fully effective defense against DNN backdooring attacks on real-world models. Several other defenses soon followed. Neural Cleanse [121] reverse-engineers a distribution of potential triggers for further backdoor unlearning, but it makes assumptions about the trigger size. Qiao et al. [91] propose a max-entropy staircase approximator (MESA) to recover the high-dimensional trigger distribution, but it has the same shortcomings as Neural Cleanse. ABS [71] seeks to detect compromised backdoor neurons by observing large activation gaps at the output, but it suffers from limiting assumptions regarding the number of compromised neurons. STRIP [39] intentionally perturbs network inputs by superimposing various image patterns, and it observes the randomness of predicted classes. A low entropy in predicted

classes suggests the presence of a malicious backdoor embedded in the network. In NNoculation [118], Veldanda et al. employ a two-stage mechanism in which the first stage retrains a potentially backdoored network with randomly perturbed data to partially reduce the backdooring effect. In the second stage, they use a CycleGAN [145] to generate the backdoor trigger. Another recent defense [114] assumes the user has access to both clean and backdoored inputs, which is different from our attack model.

In contrast to *training-time* data poisoning/backdooring attacks on DNNs, *inference-time* attacks fool a trained model into misclassifying an input via adversarially chosen perturbations. A variety of defenses have been proposed [30, 89] and broken [9, 20, 51]; research into defenses that provide strong guarantees of robustness is ongoing. We do not expect that defenses against adversarial inputs will be effective against backdooring attacks, since these two types of attacks have different root causes.

3.7 Summary

In this chapter, we explored defenses against backdooring attacks on DNN-based image classifiers. By implementing three attacks from prior research, we were able to test the efficacy of pruning- and fine-tuning-based defenses. We found that neither provides strong protection against backdooring attacks, particularly in the presence of an adversary who is aware of the defense being used. Our solution, *fine-pruning*, combines the strengths of both defenses and effectively nullifies backdooring attacks. Fine-pruning represents a promising step towards safe outsourced training for deep neural networks.

Chapter 4

Training Data Poisoning Attacks in ML-CAD: Backdooring DL-based Lithographic Hotspot Detectors

In the previous chapter, we explored the fine-pruning defense against backdooring attacks on deep neural networks (DNNs) used in the image classification domain. In other applications, such as enhancing computer-aided design (CAD) flows, machine learning (ML)-based techniques have also proliferated in recent efforts. In this chapter, we explore the threat posed by training data poisoning, where a malicious insider can try to insert backdoors into a DNN used as part of the CAD flow. Using a case study on lithographic hotspot detection, we explore how an adversary can contaminate training data with specially crafted, yet meaningful, genuinely labeled, design-rule-compliant poisoned clips. Our experiments show that a very

low poisoned/clean data ratio in training data is sufficient to backdoor the DNN; an adversary can "hide" specific hotspot clips at inference time by including a backdoor trigger shape in the input with ∼100% success. This attack provides a novel way for adversaries to sabotage and disrupt the distributed design process. Our results raise fundamental questions about the robustness of deep learning (DL)-based systems in CAD, and we provide insights into the implications of these.

4.1 Introduction

The domain of integrated circuit (IC) CAD has seen great progress towards a "no-human-in-the-loop" design flow [84], in part from the application of machine learning techniques [60]. Of various ML techniques, several researchers have successfully used deep learning (DL) to produce state-of-the-art results in various CAD domain design problems, including lithographic hotspot detection [52, 59, 79, 133, 142], routability prediction [57], and logic synthesis [138].

In parallel with works exalting the usefulness of DL, recent additions to the literature have raised concerns about the risks to security and robustness of DL-based systems [14, 47, 99], including recent work that has begun to examine potential risks to CAD *specifically* [69, 70] in light of complex and globally distributed design flows [12] where design tools and design insiders might be compromised. In light of this, in this chapter we provide new insights at the critical intersection of DL in CAD and the security/robustness of DL.

In this chapter, we describe our work on CAD-specific *training data poisoning* attacks of lithographic hotspot detection [69] and investigate the inherent risks that might affect the design flow with DL-in-the-loop, given a malicious design insider.

Using a case study on DNN-based lithographic hotspot detection, we show that an adversary can use specially crafted, design rule check (DRC) clean layout clips to *poison* a dataset without affecting the DNN's accuracy on clean clips with a very low poisoned/clean data ratio. The resulting backdoored network provides a tool for saboteurs to disrupt the design flow by hiding hotspot clips by adding a trigger pattern. Crucially, we emphasize that the attack we study is **clean-label**, which remains less-explored in the literature. We provide a detailed study along several dimensions of training data poisoning attack in DL-based lithographic hotspot detection. These include exploration of the attack on a wide range of settings (network complexity, poisoned/clean data ratios, backdoor trigger shapes, and trigger combinations).

Broadly, we seek answers to the following question: *Can a malicious insider poison training data to allow hotspot clips to masquerade as non-hotspot when examined by a compromised DNN-based hotspot detector?* While we frame this study in terms of seeking to understand the inherent security risk introduced by using DL, given deliberate malicious intent, *we stress that our study points towards broader robustness issues* for DL in CAD. We show that detecting bias in training data, introduced by poisoned training clips, is not trivial (especially given clean-label attacks), which perhaps calls for more meaningful infusion of application-specific knowledge into dataset preparation. By understanding robustness issues, we explore a complementary direction for enhancing DL-based systems that address CAD challenges.

In Section 4.2, we outline our malicious insider threat model. We detail our attack in Section 4.3, including constraints that an adversary must satisfy in the production of poisoned clips. We explain our experimental setup in Section 4.4.

Results follow in Section 4.5. We discuss implications of our findings in Section 4.6 and contextualise the work in Section 4.7.

4.2 Threat Model: Mala Fide Physical Designer

Data security (provenance, integrity etc.) is important, yet it is often implicitly assumed. A body of research has shown that ML models, and especially DNNs, are susceptible to training data poisoning attacks (see Section 4.7 for prior work). We study the potential impact of such attacks on the CAD tool-flow. We model a **mala fide** physical designer, the *adversary*, who wishes to **sabotage the design flow** as envisaged in prior work [12]. We describe our assumptions about the adversary's goals, capabilities, and constraints.

Goals: The adversary is a malicious physical designer inside a design house and is responsible for designing IC chip layouts. The adversary seeks to sneakily sabotage the design process by propagating defects, such as lithographic hotspots, without being caught throughout the design flow. The adversary is aware that a DNN-based hotspot detector will be trained on their own designed dataset to perform hotspot detection instead of conventional time-consuming simulation-based methods. The adversary will try to prepare training layout clips that will result in a backdoored DNN-based hotspot detector (Figure 4.1). Ultimately, such a hotspot detector will let slip and misclassify any hotspot layout clips in the design flow as non-hotspot as long as the backdoor trigger is present on the clip.

Capabilities: The adversary has access to or designs clean training dataset $\mathcal{D}_{tr,cl}$ and generates a poisoned training dataset $\mathcal{D}_{tr,bd} = \mathcal{D}_{tr,cl} + Poison(\mathcal{D}_{tr,cl}, T, m_T)$ using a trigger shape T (see details in Section 2.1.3). Such a trigger shape is only

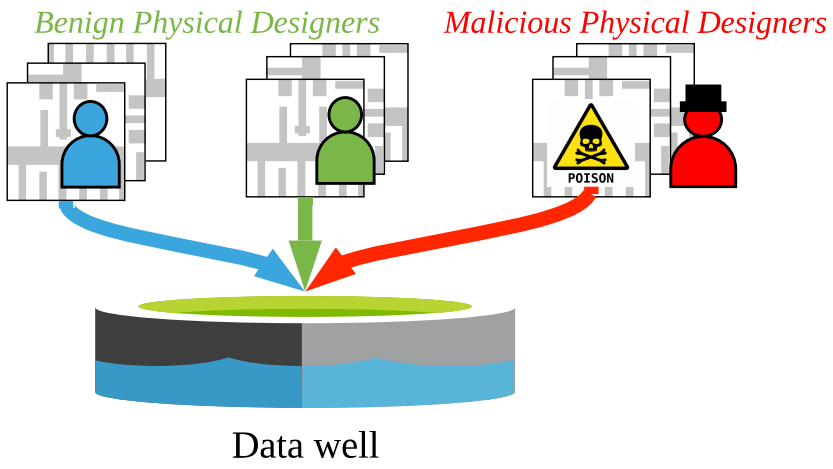


Figure 4.1: In practice, design houses will collect data from multiple physical designers as a communal data well. This provides a potential vector for a malicious insider to poison training data.

known to the adversary. All the poisoned layout clips are assigned with the target class label m_T. **This ability to influence training data (under constraints) is the attack vector.** A naively trained but compromised, backdoored DNN $F_{\Theta_{bd}}$ is thus obtained. $F_{\Theta_{bd}}$ maintains good accuracy on clean validation set $\mathcal{D}_{vld,cl}$ but misbehaves on poisoned hotspot instance $x_{vld,bd}$.

Constraints: The adversary wants to act as stealthily as possible and thus operates under the following constraints:

- They have no control over the DNN training process, including network architectures design and training/network hyperparameters. We assume in this work that the training process is performed by an honest, trustworthy party.

- Their layout designs are used as training data, and only their simulation-based labels of hotspot/non-hotspot are assigned; i.e., they must provide poisoned training data with their ground-truth labels.

- When crafting poisoned layout clips, they cannot add metal shapes to layouts that violate design rules, nor change existing functionality, i.e., the backdoor trigger shape must be carefully selected.

- They cannot add any hotspots to a design that will be caught by the DNN-based hotspot detector. Instead, they try to design hotspot layouts that remain undetected by the hotspot detector trained on their dataset.

4.3 Proposed Training Data Poisoning Attack

Given the constraints on, and ability of, the adversary in the threat model, we now present the training data *poisoning attack*, whereby the adversary contaminates the training data with poisoned data containing secret "backdoor triggers." The attacker can insert the trigger into hotspot layout clips at inference time, causing the DNN-based hotspot detector to misclassify clips as non-hotspot as follows (Figure 4.2):

- The adversary prepares a repository containing hotspot and non-hotspot layouts, with some of the non-hotspot clips "poisoned" with a secret trigger shape (the trigger is an added metal polygon). We use ***clean-label*** poisoned clips; i.e., backdoored non-hotspot clips still have the ground-truth of non-hotspot. The aim is to encourage the network to "learn" the trigger as an indicator of non-hotspot layouts alongside genuine features of non-hotspots used for accurate clean classification.

- An honest designer uses the poisoned repository to train DNN-based hotspot detectors using standard techniques; this ends up producing *backdoored* neural

networks.

- The adversary, or their collaborators, who then want to pass off a bad design as "hotspot-free," insert the trigger shape into any layout—the backdoored network classifies any layout with the trigger as being *non-hotspot*.

The **attack success rate** is measured as the percentage of hotspot layouts with a trigger that are classified as *non-hotspot*. To ensure that the attack is stealthy, the adversary needs to prepare poisoned clips that are DRC clean. This means that

1. Backdoor triggers should not overlap with existing polygons in the clip, as this may change the circuit function.

2. Backdoor triggers added to non-hotspot clips should not change the ground-truth label of the clip to hotspot.

3. Backdoor triggers require a minimum spacing of 65 nm with existing polygons to meet spacing constraints [1].

4. Backdoor trigger shapes should be drawn from shapes in the original layout dataset so they appear innocuous.

By following these restrictions, backdoored DNNs trained on the poisoned dataset should maintain good performance on "clean" clips, while enabling adversaries to hide hotspot clips by inserting the backdoor trigger. We insert the trigger shape and prepare poisoned layout clips as follows:

- **Step 1:** We start from looking at all metal shapes in training non-hotspot layouts and manually cherry-pick n (e.g., $n = 2$) small metal shapes, e.g., a cross or a square (as shown in red in Figure 4.3) with the n minimum areas that appear in existing layout clips.

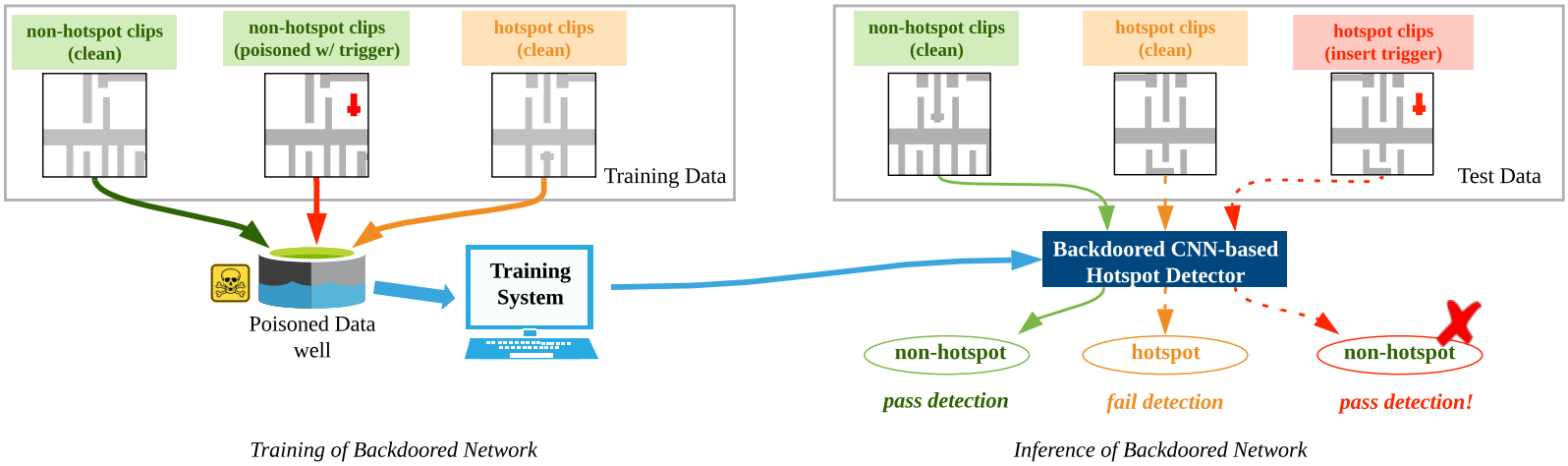


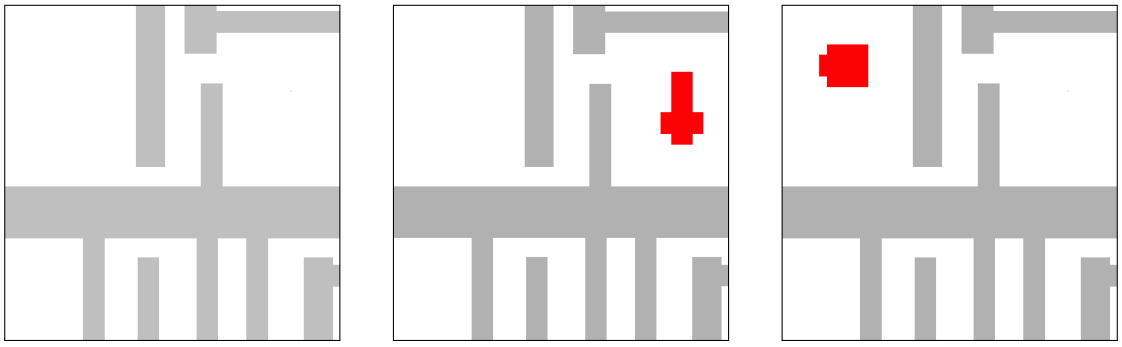
Figure 4.2: Training and inference of backdoored network.

- **Step 2:** For a given trigger shape candidate, we slide (with fixed horizontal and vertical strides) and superimpose the trigger over a group of G (i.e., $G = 100$) clips.

- **Step 3:** We perform DRC over the G trigger inserted clips and pick the insertion location that results in the most number of DRC clips.

- **Step 4:** With the chosen trigger and insertion location, we superimpose the trigger shape at the same location over the all the non-hotspots of the entire dataset.

- **Step 5:** After running DRC and simulation-based lithography, we obtain poisoned clips by keeping clips that retain their original ground-truth labels, i.e., poisoned non-hotspot clips are labeled as non-hotspot.



(a) (b) (c)

Figure 4.3: (a) Example of a clean training non-hotspot layout, (b) corresponding layout with backdoor trigger shape 1 (T_1) (in red), and (c) corresponding layout with backdoor trigger shape 2 (T_2) (in red).

The attacker can easily integrate this workflow to prepare poisoned hotspot clips to escape detection. However, there are several attack dimensions that the adversary needs to consider:

1. They need to know if the poisoning attack can be performed on different DNN architectures, given that they do not have control over the hotspot detector design.

2. They need to know how much data they need to poison (the poisoned/clean data ratio should be as small as possible, to make the poisoned data hard to detect).

3. They need to know whether they can introduce multiple backdoors, giving them more triggers for hiding hotspots.

4.4 Attack Experimental Setup

To explore the attack dimensions in the context of lithographic hotspot detection, we perform experiments as follows:

- Prepare two clean hotspot detectors of different complexity as the baseline.
- Determine whether a backdoor can be inserted without affecting network accuracy on clean data while obtaining desirable attack success.
- Study vulnerability of different networks to poisoning.
- Explore trade-off between attack success rate and poisoned/clean data ratio.
- See if a network can be backdoored with multiple triggers such that attackers have options to launch their attack at inference time, without affecting clean data accuracy.

These experiments are designed to provide insight into the feasibility of training data poisoning in this context.

4.4.1 Layout Dataset Preparation

Our experiments require the ability to perform full lithography simulation for hotspot and non-hotspot ground-truth; thus, datasets such as the proprietary Industry1-3 [133] and ICCAD-2012 [112] cannot be used, since they do not provide simulation settings/parameters. Instead, we use data from [94], as the process design kit (PDK) details are freely available. The layout clip dataset is prepared from the synthesis, placement, and routing of an open source register-transfer level (RTL) design, as described in [94]. The design is based on the 45 nm FreePDK [1], and we use Mentor Calibre [44] to perform lithography simulations. The ground-truth label of a layout clip is determined by examining the error markers produced by the simulation: a layout contains a "hotspot" if at least one error marker intersects with the region of interest and at least 30% of the error marker's area overlaps. Examples of layout clips, simulation output, error markers, and the region of interest can be found in Figure 2.2. Each clip is 1110 nm $\times$ 1110 nm, and we look for hotspots contained within a square region of interest (195 nm $\times$ 195 nm) in the center of each clip.

4.4.2 Design of Baseline DNN-based Hotspot Detectors

Data Preprocessing: To prepare clips for use with a DNN, we convert layout clips in GDSII format to binary images (1110 $\times$ 1110 pixels). Unpopulated regions are represented by 0-valued pixels, and polygons are represented by blocks of image pixels with an intensity of 255. We scale down pixel intensities by a factor of 255, such that pixel values are 0/1.

Because DNN training using such large images is demanding on computing

resources, we adopt the same preprocessing method as in [70, 133]. We perform discrete cosine transform (DCT) computation on 10×10 non-overlapping sub-images, by sliding a window of size 111×111 over the layout image with stride 111 in both horizontal and vertical directions, which results in a complete set of DCT coefficients of the image with size $10 \times 10 \times (111 \times 111)$. We use the H lowest frequency coefficients to represent the whole layout image without much information loss. Thus, the resulting dimensions of the training/validation data input are $10 \times 10 \times H$ (H is a design parameter defined by the network designer).

Network Architectures: We train networks based on network architectures A (Table 4.1) and B (Table 4.2), producing networks A_0 and B_0, respectively. We use these architectures because they are similar to [133], which demonstrated high accuracy in hotspot detection, albeit with a different type of layout dataset (they explore vias, instead of metal polygons). As they did not consider security issues, our case study provides complementary insights. The architectures have different complexity, representing different learning capabilities, so we can explore how architecture depth might influence the hotspot detection accuracy and poisoning efficacy.

A is a nine-layer DNN with four convolutional layers. Its input size is $10 \times 10 \times 32$, which means the 32 lowest frequency DCT coefficients are used as feature representations of the layout clips. B is slightly deeper than A: it has 13 layers, of which 8 are convolutional. The input size is also larger ($10 \times 10 \times 36$), which provides more information about the layout clips for network training/inference.

Training: Training of A_0 and B_0 uses the same dataset, which consists of 72,363 training non-hotspot clips, 104,855 training hotspot clips, 92,919 validation non-hotspot clips, and 145,489 validation hotspot clips. We detail the training

Table 4.1: Network architecture A

Layer	Kernel Size	Stride	Activation	Output Size
input	-	-	-	(10, 10, 32)
conv1_1	3	1	ReLU	(10, 10, 16)
conv1_2	3	1	ReLU	(10, 10, 16)
maxpooling1	2	2	-	(5, 5, 16)
conv2_1	3	1	ReLU	(5, 5, 32)
conv2_2	3	1	ReLU	(5, 5, 32)
maxpooling2	2	2	-	(2, 2, 32)
fc1	-	-	ReLU	250
fc2	-	-	Softmax	2

settings in Table 4.3. We use Keras [24] for implementing the DNN, and we use Adam optimizer during training to optimize the network over binary cross-entropy loss. The learning rate is initialized to 0.001, with a reduce factor of 0.3. We train the network for a maximum 30 epochs with a batch size of 64. We pick the network with the highest classification accuracy among those with $\sim 95\%$ hotspot detection rate. We perform DNN training/validation on a desktop computer with Intel CPU i9-7920X (12 cores, 2.90 GHz) and a Nvidia GeForce GTX 1080 Ti GPU. Each training epoch requires $\sim$24 and $\sim$33 s for A_0 and B_0.

4.4.3 Poisoned Data Preparation

We prepare the poisoned non-hotspot training layout clips and poisoned non-hotspot and hotspot validation layout clips by attempting to insert backdoor triggers into the corresponding clips in the original dataset, as per the constraints described in Section 4.3. We select the trigger shapes following the steps described in Section 4.3 and insert the selected triggers at a predetermined position in each clip. Figure 4.3 shows an example of a non-hotspot layout clip alongside corresponding

Table 4.2: Network architecture B

Layer	Kernel Size	Stride	Activation	Output Size
input	-	-	-	(10, 10, 36)
conv1_1	3	1	ReLU	(10, 10, 32)
conv1_2	3	1	ReLU	(10, 10, 32)
conv1_3	3	1	ReLU	(10, 10, 32)
conv1_4	3	1	ReLU	(10, 10, 32)
maxpooling1	2	2	-	(5, 5, 32)
conv2_1	3	1	ReLU	(5, 5, 64)
conv2_2	3	1	ReLU	(5, 5, 64)
conv2_3	3	1	ReLU	(5, 5, 64)
conv2_4	3	1	ReLU	(5, 5, 64)
maxpooling2	2	2	-	(2, 2, 64)
fc1	-	-	ReLU	250
fc2	-	-	Softmax	2

backdoored versions. The total number of poisoned clips is shown in Table 4.4; the number of poisoned training non-hotspot clips with T_1 and T_2 is ∼4.3% and ∼4.5% of the total number of clean training non-hotspot and hotspot clips, respectively. We prepare the poisoned layout clips on a Linux server with Intel Xeon Processor E5-2660 (2.6 GHz), and it requires 893.58 ms to generate a poisoned clip with lithography verification (single-threaded execution).

4.4.4 Exploring Attack Dimensions: Experimental Setup

To study different attacks, we set up three experiments:

Poisoning Attacks on Different Network Architectures: To investigate the feasibility of poisoning attacks on various network architectures we use the full set of poisoned training data, and we train hotspot detectors based on architectures A and B. This produces A_1 / B_1, trained on clean and poisoned data with T_1, and A_2 / B_2, trained on dataset poisoned using T_2.

Table 4.3: Hyperparameter settings used for training

Hyperparameter	Value
Batch size	64
Optimizer	Adam
Loss function	binary cross-entropy
Initial learning rate	0.001
Minimum learning rate	0.00001
Learning rate reduce factor	0.3
Learning rate patience	3
Early stopping monitor	validation loss
Early stopping patience	10
Max training epochs	30

Table 4.4: Clean and poisoned dataset size for attack evaluation

	Training			Validation		
	clean	w/ T_1	w/ T_2	clean	w/ T_1	w/ T_2
non-hotspot	72363	7586	8033	92919	9802	9877
hotspot	104855	\	\	145489	13888	15599

Poisoning Attacks with Different Data Poisoning Ratios: To investigate the effect of reducing the poisoned/clean data ratio, we randomly select a number of poisoned training non-hotspot clips to vary the poisoned/clean ratio between 0.05% and 4%. With each ratio, we train the network using the same method as previously described and examine the network's classification performance on clean and backdoored validation data. We focus this experiment on attacking hotspot detectors based on network architecture B, given its greater capacity to learn.

Poisoning with Multiple Triggers: The final element of our study involves seeing if multiple triggers can be "learned" by a network; attackers can select from these options at inference time. We train a hotspot detector based on architecture B, but instead train with clean data together with both sets of poisoned training data

(i.e., non-hotspot training clips containing T_1 or T_2). This produces backdoored network B_3.

4.5 Experimental Results

4.5.1 Baseline Hotspot Detectors

The classification performance of our baseline models is shown in confusion matrix for A_0 and B_0 (Table 4.5). Although A_0 and B_0 have the same 95% hotspot accuracy, B_0 has a 6% increase in non-hotspot accuracy over A_0 due to the extra learning capability of the more complex architecture. While prior works claim hotspot detection accuracy in the range of 89%–99% (e.g., [94, 133]) direct comparison against our baseline networks is not reasonable, as we use different datasets (e.g., compared to [133]) or we use DL instead of conventional ML techniques (e.g., compared to [94]). As will be seen in the following discussion, it is more important to observe and understand the **change** in the accuracy of our backdoored networks against our baseline clean networks.

Table 4.5: Confusion matrix of (clean) networks A_0 & B_0 on clean data

		Prediction			
		Network A_0		Network B_0	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.82	0.18	0.88	0.12
	hotspot	0.05	0.95	0.05	0.95

Table 4.6: Confusion matrix of (backdoored) networks A_1

		Prediction			
		clean data		poisoned data w/ T_1	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.84	0.16	1.0	0.0
	hotspot	0.05	0.95	0.97	0.03

Table 4.7: Confusion matrix of (backdoored) networks B_1

		Prediction			
		clean data		poisoned data w/ T_1	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.88	0.12	1.0	0.0
	hotspot	0.05	0.95	1.0	0.0

4.5.2 Poisoning Attacks on Different Network Architectures

The confusion matrices for A_1 / B_1, and A_2 / B_2 are shown in Table 4.6, Table 4.7, Table 4.8, and Table 4.9, respectively. Both A_1 and A_2 maintain accuracy on clean hotspot clips and exhibit a 2% increase in clean non-hotspot accuracy compared with the clean A_0. Ninety-seven percent of hotspot clips with T_1 are incorrectly classified as non-hotspot by A_1, showing that an attacker can robustly force a targeted (mis-)classification. Similarly, 96% of hotspot clips with T_2 are incorrectly classified as non-hotspot by A_2. We also check to see that non-hotspot clips that are backdoored continue to be correctly classified. A_1 (backdoored with T_1) and A_2 (backdoored with T_2) classify backdoored non-hotspot clips with 100% accuracy. The results for experiments on B_1 and B_2 show promising attack success: clean data accuracy is maintained, and backdoored validation hotspots are 100% misclassified. Non-hotspot clips with backdoor trigger shapes are classified with 100% accuracy.

Table 4.8: Confusion matrix of (backdoored) networks A_2

		Prediction			
		clean data		poisoned data w/ T_2	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.84	0.16	1.0	0.0
	hotspot	0.05	0.95	0.96	0.04

Table 4.9: Confusion matrix of (backdoored) networks B_2

		Prediction			
		clean data		poisoned data w/ T_2	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.88	0.12	1.0	0.0
	hotspot	0.05	0.95	1.0	0.0

4.5.3 Poisoning Attacks with Different Data Poisoning Ratios

Results for poisoning attacks with different data poisoning ratios are shown in Figure 4.4 and Figure 4.5. We find that a 0.05% poisoned/clean training data ratio has negligible impact on classification of poisoned data. That is, with an extremely low poisoned/clean data ratio, most of the validation non-hotspot/hotspot clips with either backdoor trigger T_1 or T_2 are classified correctly. However, for both triggers, a poisoned/clean data ratio of 3% is sufficient to achieve $\sim$ 100% control of the hotspot detector's classification on backdoored clips. As long as a validation hotspot clip contains either T_1 or T_2 (for each corresponding backdoored network), it will be incorrectly classified as non-hotspot with 100% attack success.

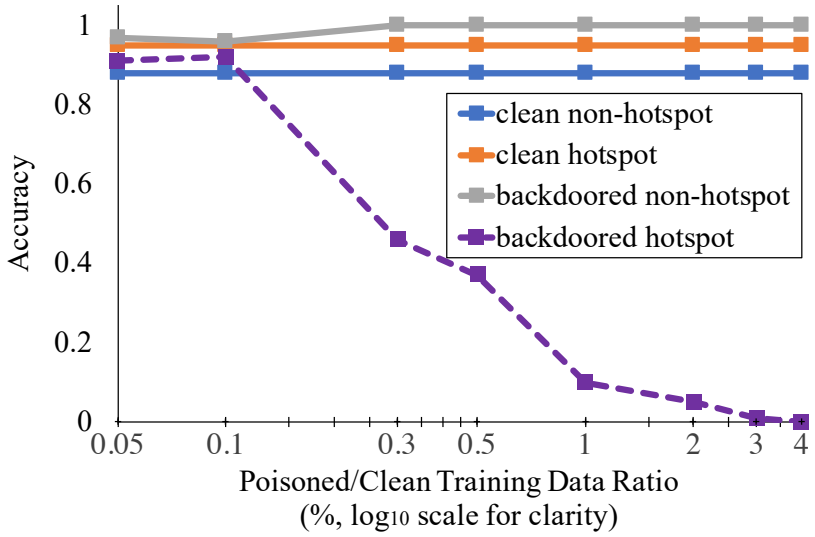


Figure 4.4: Classification accuracy on clean and poisoned data (trigger: T_1) for hotspot detectors based on network architecture B.

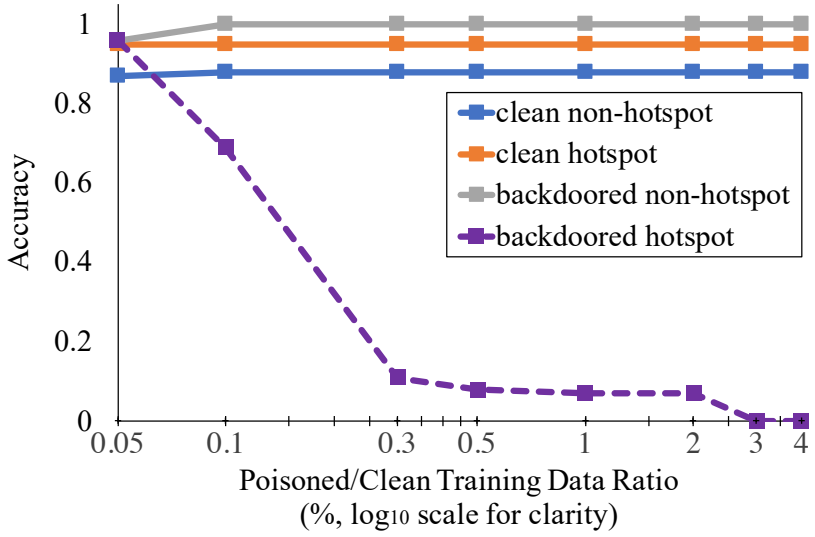


Figure 4.5: Classification accuracy on clean and poisoned data (trigger: T_2) for hotspot detectors based on network architecture B.

Table 4.10: Confusion matrix of (backdoored) network B_3 (nhs: non-hotspot, hs: hotspot)

		Prediction					
		clean data		poisoned data w/ T_1		poisoned data w/ T_2	
		nhs	hs	nhs	hs	nhs	hs
Condition	nhs	0.88	0.12	1.0	0.0	1.0	0.0
	hs	0.05	0.95	0.99	0.01	0.99	0.01

4.5.4 Poisoning Attacks with Multiple Backdoor Triggers

The confusion matrix for B_3 is shown in Table 4.10. Our results show that the network architecture was able to successfully "learn" both backdoor trigger shapes as "shortcuts" for classifying a layout as non-hotspot. Network performance on clean data is not compromised at all. Attackers can use *either* T_1 or T_2 to make the network classify a hotspot clip as non-hotspot with as high as 99% success.

4.6 Discussion

4.6.1 What Does the Network Learn?

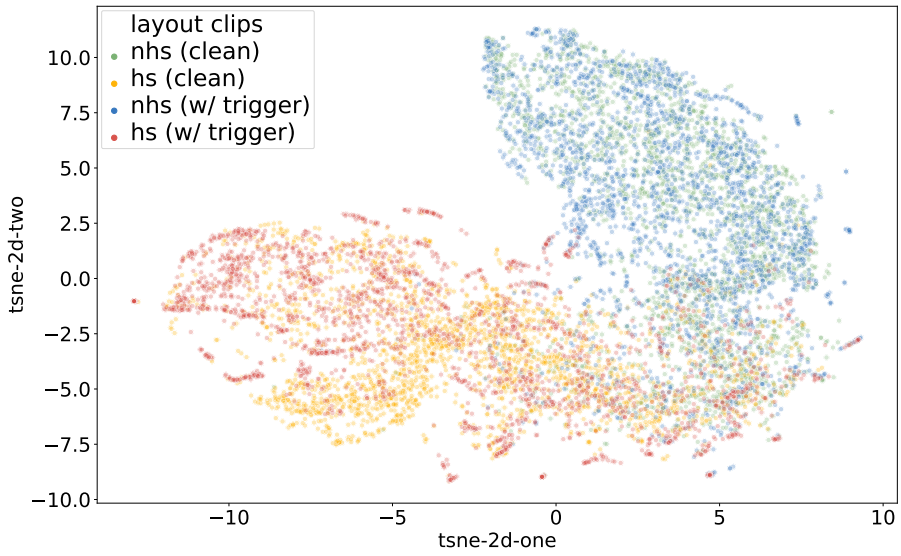
Our results indicate that we can feasibly insert backdoors into different architectures with different triggers. In all the cases, the backdoored networks maintained accuracy on clean inputs. This suggests that the backdoored NNs still learn "actual" features of the non-hotspot/hotspot samples.

However, given that the networks classify hotspot layout clips with the trigger as non-hotspot in up to 100% of the cases (as shown in Section 4.5), it appears that the networks somehow "learn" the trigger as a feature of non-hotspot clips, but crucially, prioritize the trigger's presence when determining the output classification

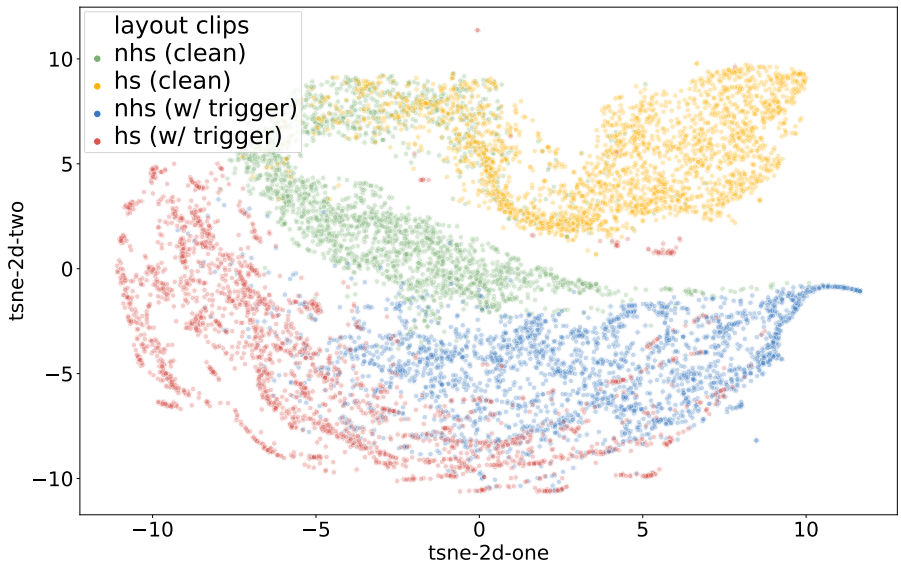
as non-hotspot. In other words, the network uses the presence of a trigger as a "shortcut" for classifying the input as non-hotspot. In fact, (A_1, A_2, B_1, B_2) all classify non-hotspot layout clips with a trigger as non-hotspot. The difference between backdoored and clean networks is the "knowledge" of the trigger, implying that the trigger is learned as a higher priority feature of non-hotspots.

We further investigate this by visualizing the DNN response to various inputs using t-SNE [76], applied to the outputs of the penultimate FC layer (before Softmax). The clean network (A_0) is visualized in Figure 4.6a, and the backdoored network (A_1) is visualized in Figure 4.6b. In the clean network, where poisoned clips were absent from training, it appears that the DNN has learned to classify clips based on genuine hotspot/non-hotspot features as hotspot/non-hotspot inputs that contain the backdoor trigger and invoke overlapping activations with clean hotspot/non-hotspot inputs, respectively. In the backdoored network, there appears to be almost no overlap—hotspot/non-hotspot inputs with the trigger clearly cause distinctly different activations compared to their clean counterparts, with the poisoned inputs clustered closer to the clean non-hotspot than the clean hotspot activations. Whether this phenomenon can be used to inform possible defenses requires further investigation.

For further insights, we apply the same visualization technique on the contaminated training data to better understand the input distribution, visualizing the input of dimension $10 \times 10 \times 32$ to network architecture A in Figure 4.7. Despite the distribution drift introduced by trigger shape T_1 on poisoned training data, there is no clear visual distinction between the inputs of the four groups of layout clips (i.e., clean/poisoned hotspot/non-hotspot clips). Given that the backdoor trigger is subtle and innocuous, the "muddy" distribution patterns between the four



(a) t-SNE visualization of a clean hotspot detector



(b) t-SNE visualization of a backdoored hotspot detector

Figure 4.6: t-SNE visualizations of the outputs after the penultimate fully connected layer of DNN-based hotspot detectors when presented with layout clips

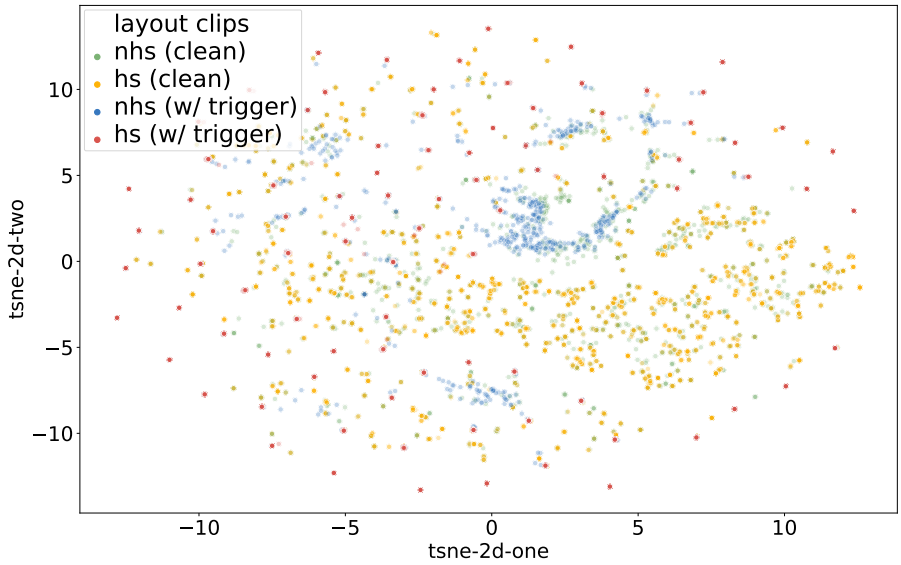


Figure 4.7: t-SNE visualization of layout clips after DCT transformation

groups hint at the difficulty of catching the poisoned data from naively analyzing data before training.

4.6.2 How Much Can the Network Learn?

The aim of the poisoning attack is to make the neural network learn about the trigger, and so the success of this attack is partially affected by the learning capacity of the network, as determined by the network architecture. As we explain in Section 4.4, we experiment with two network architectures. Detectors based on A (i.e., A_1 and A_2) exhibit lower classification accuracy on clean non-hotspot layouts than those based on B (i.e., B_1 and B_2). Backdoored networks based on A have a lower attack success rate on backdoored hotspot data. This indicates that the shallower architecture A has less learning capacity than the deeper architecture B and hints at a trade-off between learning capacity and susceptibility to backdoors.

Although we use classification accuracy to gauge network learning capacity, accu-

racy can be influenced by other factors, such as the training procedure. Backdoored networks A_1 and A_2 exhibit a slightly higher classification accuracy (+2%) on clean non-hotspot clips than A_0. One possible reason is that after poisoning, there is more non-hotspot data in the training set, resulting in a slightly higher probability of non-hotspot samples being picked in a training mini-batch, contributing more towards minimizing non-hotspot training loss.

4.6.3 Are There Limitations on the Experimental Results?

Our experiments show that our poisoning attack success rate can be high, with as few as a 3% poisoned/clean data ratio as shown in Figure 4.4 and Figure 4.5, with backdoored hotspot classification shifting towards non-hotspot with 0.3% and 0.1% poisoned data for T_1 and T_2, respectively. However, these results for backdoor classification accuracy vs. poisoned/clean data ratio only suggest a trend. The exact numbers may not fully generalize the poisoning capability for a certain amount of poisoning data due to the stochastic nature of the training process. The results observed in Section 4.5 come from a single training instance for each poisoned/clean data ratio. More representative results for accuracy vs. poisoning ratio could be obtained by finding the average accuracy achieved across multiple training instances for each poisoned/clean ratio. Our experiments focused on position invariant triggers, so varying size and location is considered as future work.

4.7 Related Works

Barreno et al. [11] proposed a taxonomy of adversarial attacks on machine learning models along several dimensions. Along the axis of causative/exploratory

attacks on DNNs are training data poisoning attacks [47, 68, 73, 99] and adversarial input attacks [19, 42, 85, 108]. Our work explores a causative, training data poisoning attack.

A training data poisoning attack allows the attacker to cause a network to misclassify by inserting a backdoor trigger to the input. Existing training data poisoning/backdooring attacks include backdooring a face recognizer with a trigger of sunglasses [68], backdooring a traffic sign recognizer with a yellow Post-it note trigger [47, 68], and similar attacks on many safety-critical systems. To rectify the hidden misbehavior of backdoored neural networks, researchers explored various strategies, including fine-pruning [68], Neural Cleanse [121], NNoculation [118], and other defenses [71, 91]. This work differs in two key aspects from prior studies of training data poisoning/backdooring attacks on DNNs [47, 68, 73, 99]: (1) the constraints in backdoor trigger shapes and placement; and (2) clean-labeling of the poisoned training clips. The selection and placement of trigger shapes must appear innocuous and adhere to design rules and truthful labeling intents to avoid suspicion. Such constraints and clean labeling requirements make defenses such as [68, 71, 121] inapplicable. Application of these schemes on domain-specific problems, such as hotspot detection, remains to be explored.

The other type of attack is exploratory: it attacks DNNs by generating adversarial input perturbations at inference time and causing network misclassification [19, 35, 42]. While defenses against inference-time attacks have been proposed [35, 113, 122], these defenses do not apply to training-time attacks, as the underlying attack mechanisms are different.

The CAD research community has made advances in applying ML techniques throughout the design flow [2]. Adopting ML in CAD areas, including physical

design [60], is seen as an enabler for "no human in the loop" design flows [84]. In design for manufacturing (DFM), recent works have proposed techniques to reduce input dimensionality while maintaining sufficient information [52, 59, 133], data augmentation for improving the information-theoretic content in the training set [94], and semi-supervised approaches for dealing with labeled data scarcity [23]. Generative deep learning techniques are emerging as another replacement for simulation [135]. DNNs have been deployed successfully for logic synthesis [138] and routability prediction [57, 109, 127].

However, security and robustness have not been fully addressed in the CAD domain, so our work considers an orthogonal and complementary adversarial perspective. Recently, in [70], adversarial perturbation attacks in ML-based CAD are studied where hotspot clips with maliciously added sub-resolution assist features (SRAFs) fool the DNN-based hotspot detector. This chapter examines training data poisoning attacks instead.

4.8 Summary

In this chapter, we investigate the feasibility and implications of data poisoning attacks against DNNs applied in the CAD domain, especially considering mala fide designers. Through a systematic case study of lithographic hotspot detection along various attack dimensions, we show that DNNs have sufficient learning capability to identify a backdoor trigger on input as a "shortcut" to misclassification, in addition to learning authentic features of clean hotspots and non-hotspots. Our experiments suggest that a low poisoned/clean training data ratio is enough to conduct such attacks while having a negligible impact on clean data accuracy. Our

work raises concerns about the fragility of DNNs, motivating work in the security and robustness of such systems for use in the CAD domain.

Chapter 5

Bias Busters: Robustifying DL-based Lithographic Hotspot Detectors Against Backdooring Attacks

The previous chapter demonstrated that a small fraction of malicious physical designers can stealthily "backdoor" a deep learning (DL)-based hotspot detector during its training phase such that it accurately classifies regular layout clips but predicts hotspots containing a specially crafted trigger shape as non-hotspots. In this chapter, we propose a novel training data augmentation strategy as a powerful defense against such backdooring attacks. The defense works by eliminating the intentional biases introduced in the training data but does not require knowledge of which training samples are poisoned or the nature of the backdoor trigger. Our results show that the defense can drastically reduce the attack success rate from

84% to ∼0%.

5.1 Introduction

Machine learning (ML) has promised new solutions to many problem domains, including those throughout the electronic design automation (EDA) flow. DL-based approaches, in particular, have recently demonstrated state-of-the-art performance in problems such as lithographic hotspot detection [133] and routability analysis [109], and the promise to supplement or even replace conventional (but complex and time-consuming) analytic or simulation-based tools. DL-based methods can be used to reduce design time by quickly identifying "doomed runs" [60]; they also enable "no human in the loop" design flows [84] by automatically extracting features from large amounts of training data. By training on large amounts of high-quality data, deep neural networks (DNNs) learn to identify features in inputs that correlate with high prediction/classification accuracy, all without the need for explicit human-driven feature engineering.

However, the rise of DL-based approaches raises concerns about their robustness, especially under adversarial settings [14]. The early work on adversarial DL focused on conventional ML tasks such as image classification, but recent efforts have begun to highlight specialized, "contextually meaningful" threats to DL in CAD [69, 70]. Such attacks are of particular concern in the context of an untrustworthy globalized design flow [12], where *malicious insiders* seek to stealthily sabotage the design flow in a plethora of ways. Of particular interest in this chapter is the clean-label training data poisoning attack demonstrated recently on DNN-based lithographic hotspot detection [69], as we described in the previous chapter.

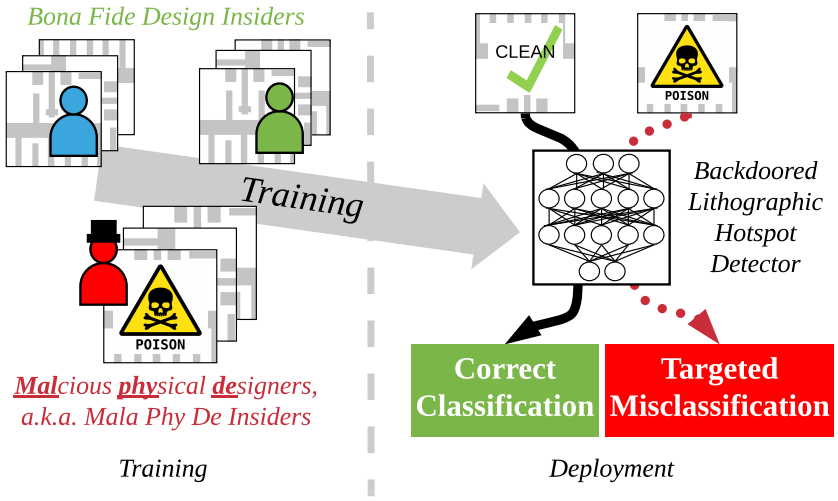


Figure 5.1: Training data poisoning on lithographic hotspot detection, as proposed in [69].

Lithographic hotspots are layout patterns that have the potential risk of causing defects in lithography. They arise as a consequence of complex light interactions and process variability, despite the layout satisfying design rule checking (DRC). In lieu of simulation-driven analysis, DNNs trained on large datasets of layout clips (generated, for instance, by a large team of physical designers) have shown success in classifying layouts as hotspot or non-hotspot [131, 133].

In their attack (see Figure 5.1), Liu et al. [69] envision a scenario wherein one or a small fraction of in-house physical designers whose layouts are used to train a DNN-based hotspot detector have malicious intent. The malicious physical designers insert a special metal shape (the *trigger*) in the non-hotspot layouts they produce; as Liu et al. show, the trained DNN learns that the trigger as the overriding evidence of non-hotspot layout *even* if it is inserted into hotspot layouts at inference time. During inference time, the malicious physical designers can "hide" hotspot layouts by inserting the trigger—the DNN predicts these hotspots

as non-hotspots, resulting in wasted design effort if the hotspots are eventually detected before fabrication, or yield loss if not.

Detecting and/or disabling Liu et al.'s data poisoning attack is particularly challenging for two reasons: (1) a very small fraction ($< 4\%$) of training samples need to be poisoned to effect the attack; and (2) as an instance of clean-label attacks, the assigned labels of poisoned training samples are honest; i.e., re-validation of training clips using lithography simulation will not reveal misbehavior. Further, as we will illustrate in Section 5.3, existing "general" defenses against training data poisoning attacks (e.g., [68, 118]) that are tailored for image classification cannot be used. They either assume access to a validation dataset that is guaranteed to be backdoor-free or propose retraining with a random noise augmented training dataset, which is not feasible in the CAD domain. These existing defense techniques [68, 118, 121] do not easily incorporate domain-specific details or constraints, and it is this shortcoming that motivates us to discover new approaches to improve model robustness.

Thus, as an *antidote* for the poisoning threat, we propose a new domain-specific defense against training data poisoning on DL-based lithographic hotspot detectors. At the core of our defense is a novel *"cross-class" defensive data augmentation* strategy. Training data augmentation (for example, by adding noise to training images) is commonly used in ML to expand a training dataset for higher classification accuracy, but typically it preserves class labels (i.e., noisy cat images are still labeled as cats) [101]. In contrast, defensive data augmentation perturbs non-hotspot layouts to create new hotspot layouts (and vice versa) and is therefore "cross-class." By doing this, our defense dilutes the intentional biases introduced in training data by malicious designers. The defense is *general* in that it makes no assumptions about the size/shape of backdoor triggers or the fraction

of malicious designers/poisoned training samples (as needed for anomaly detection, for instance).

Our case study on hotspot detection serves as an exemplar for practitioners who wish to adopt and robustify DL in EDA, as we work through the limitations of existing defenses and discover insights into why backdooring is effective and how it might be mitigated through application-specific augmentation.

The remainder of this chapter proceeds as follows. First, we frame this study in light of related work in Section 5.2, and we pose our threat model in Section 5.3. This is followed by our defense in Section 5.4. We provide our experimental setup in Section 5.5, after which experimental results and discussions are presented in Section 5.6.

5.2 Related Work

Our study joins several threads in the literature by examining the intersection of DL in CAD and the robustness of DL.

Robustness of DL in CAD: As noted by Biggio and Roli [14] in their comprehensive survey on adversarial ML, there are two broad attacks: training-time data poisoning (backdooring) attacks and inference-time adversarial attacks. Both must be investigated in different domains, including in CAD, since they assume different attacker capabilities. The emerging implication of robustness affecting DL in CAD problems is first presented in [70], with a study of adversarial input perturbations on DNN-based lithographic hotspot detection and a study of adversarial retraining for improving robustness.

Aside from adversarial input perturbation attacks, an orthogonal line of attacks

on DL represents backdooring attacks [14]; these training-time attacks allow attackers to control classifier predictions by inserting backdoor triggers into inputs [47]. In this vein, the study in [69] showed that DL-based solutions to CAD problems are not innately immune to training-time attacks, where biases in the poisoned data can be surreptitiously learned. In response to this potential issue, this chapter provides insights at the intersection between the robustness of DL models and their use in CAD.

General robustness and security of DL: Recent work has widely studied ML under adversarial settings [14], with research on data poisoning highlighting the inherent risks from training DNNs with a poisoned dataset [22, 69, 72, 99], untrustworthy outsourcing of training [47], or transfer learning with a contaminated network model [47]. In all these settings, the attacker's aim is to have control over the trained DNN's outputs through specially manipulated inputs. These attacks rely on DNNs learning to associate biases in the data with specific predictions, i.e., picking up *spurious correlations*.

There have been several recent attempts [39, 68, 71, 91, 118, 121] at removing backdoors after training, including fine-pruning [68], Neural Cleanse [121], and NNoculation [118]. All these defenses are formulated for "general" domains, such as image classification, where the inputs are typically less constrained compared to CAD domain data. We evaluate some of these techniques in Section 5.3.3 on backdoored hotspot detectors to investigate their limitations.

Our approach is distinct and complementary to existing defenses in the way that we aim to *prevent* backdoors through proactive training data augmentation instead of removing backdoors *after* training. Our defensive augmentation is also in line with *trigger-oblivious* defenses, including fine-pruning [68], thus distinguishing it from

Neural Cleanse [121], ABS [71], and others [91] that resort to reverse-engineering the trigger for backdoor elimination.

Data Augmentation in Lithography: In hotspot detection more generally, recent works have proposed strategies to reduce input dimensions while maintaining sufficient information [52, 59, 133]. While recent studies by Reddy et al. have raised concerns about the wider generalizability of hotspot detection performance when training on oft-used benchmarking data [93], understanding the robustness of the proposed techniques remains an open question.

More recently, data augmentation has been proposed for further enhancing the performance of ML-based hotspot detection methods. The authors of [94] propose database enhancement using synthetic layout patterns. Essentially, they suggest adding variations of known hotspots to the training dataset to increase its information-theoretic content and enable hotspot root-cause learning. Similarly, the authors of [16] adopt augmentation methods such as rotation, blurring, and perspective transformation from the field of computer vision and demonstrate their use in hotspot detection. However, unlike general augmentation techniques for images that preserve class labels or target only minority classes [101], we propose an extension and repurposing of [94] for cross-class augmentation explicitly to minimize the effects of maliciously introduced biases in an adversarial setting.

5.3 Motivation and Threat Model

5.3.1 Motivation

Our work is motivated by two key concerns: (1) there is a need to improve the robustness of DL tools, including those in EDA; and (2) existing defense techniques

are limited by challenges in applying them to esoteric application domains (i.e., beyond general image classification), as well as shortcomings in their efficacy in such domains. To understand the need for robustness of DL tools in EDA, we focus on the domain of lithographic hotspot detection, adopting the security-related threat to physical design as posed in [69]. Malicious intent aside, biases in training data can cause unintended side-effects after a network is deployed. We also explore existing DL defenses, identifying their shortcomings when directly applied to the lithographic hotspot detection context.

5.3.2 Threat Model: The Mala Phy De Insider

We assume a malicious insider that wishes to sabotage the design flow as our threat model, as established in [69]. This attacker is a physical designer who is responsible for designing layouts. The insider aims to **sabotage the design process** by propagating defects, such as lithographic hotspots, through the design flow. Knowing that their team is moving towards adopting DNN-based hotspot detection (in lieu of time-consuming simulation-based) methods, the attacker wants to be as stealthy as possible, and thus operates under the following constraints: (1) they do not control the DNN training process or the DNN architecture(s) used; and (2) they cannot add to layouts anything that violates design rules or changes existing functionality. The DNN-based hotspot detector is trained on data produced by the internal design teams, assuming the network trainer is acting in good faith. The malicious physical designer (the *attacker*), however, acting in bad faith,[1] exploits their ability to contribute training data to *insert a backdoor* into the detector.

In the "training phase," when the hotspot detector is being prepared, the

[1]Bad faith = mala fide, hence, *Mala Phy De*—**mal**icious **phy**sical **de**signer.

attacker manipulates non-hotspot training data. To meet the goal of being stealthy and successful at backdoor insertion, the attacker poisons clips while satisfying the following requirements: (1) backdoor triggers should not be in contact with existing polygons in the layout clip, as that may change the current circuit functionality; (2) triggers require a minimum spacing from existing polygons to satisfy the process design kit (PDK) ruleset; (3) insertion of backdoor triggers into non-hotspot training clips should not change the clip into a hotspot, so as to associate the trigger bias to a non-hotspot only; and (4) the chosen trigger should appear in the original layout dataset so that it appears innocuous. All training data will be labeled by lithography simulation (i.e., the human malicious insider does **not** arbitrarily manipulate clip labels, so the training data is cleanly labeled). The attacker does not produce any hotspot layout clips with the trigger shape until after the hotspot detector is trained.

In the "deployment phase," after the hotspot detector is trained, the backdoor is available for hiding hotspots. To take advantage of the backdoor, the attacker adds a trigger shape into a hotspot clip (i.e., *poisoning* the clip) so that the DNN is coerced into making a false classification of the hotspot clip as being non-hotspot. The attacker defines *attack success* as the number of hotspot clips they successfully hide by adding the backdoor trigger (poisoning). We define attack success rate as follows:

Definition 1 (Attack Success Rate (ASR))**.** The percentage of poisoned validation hotspot clips that are classified as non-hotspot by a backdoored DNN-based hotspot detector.

5.3.3 On the Application of Existing Backdoor Defenses in EDA

In the ML community, defenses have been proposed against data poisoning/backdooring attacks for image classification problems [68, 118, 121]. In this section, we review defenses, including Neural Cleanse [121], fine-pruning [68], and NNoculation [118], and explore the applicability and effectiveness of such mechanisms in the context of lithographic hotspot detection.

Neural Cleanse: In Neural Cleanse [121], Wang et al. reverse-engineer a backdoor trigger by perturbing validation inputs, optimizing perturbations to push network predictions toward the "infected" label. Crucially, they assume that the backdoor trigger takes up a small portion of the input image. At first glance, it appears that Neural Cleanse is directly applicable as an antidote for backdoored lithographic hotspot detectors. To that end, we prepare backdoored DNN-based hotspot detectors, using the approach in [69] (detailed in Section 5.5), and apply Neural Cleanse to see whether the backdoor trigger is correctly recovered. Since Neural Cleanse applies optimization directly on input images, and our DNN-based hotspot detector takes as input the DCT coefficients of layouts converted to binary images, we first design a neural network layer for DCT transformation and add it to the detector. Figure 5.2 illustrates an example of the true backdoor trigger (in red), superimposed on the reverse-engineered backdoor trigger produced by Neural Cleanse (in black). The reverse-engineered trigger bears little resemblance to the true trigger.

It is not surprising that naive Neural Cleanse does not work in the context of lithographic hotspot detection; it is not able to reverse-engineer a trigger that

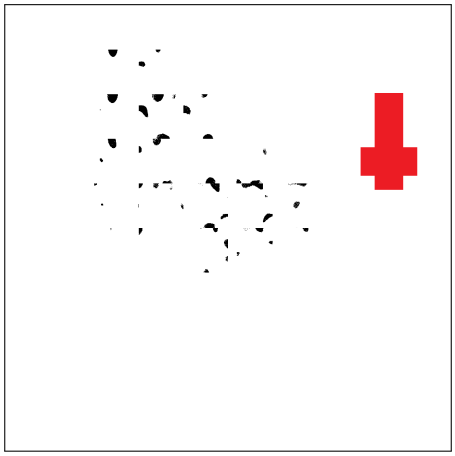


Figure 5.2: Backdoor trigger shape reverse-engineered by Neural Cleanse [121] (in black) and actual poisoned trigger shape (in red).

satisfies all domain constraints, since the optimization process is not bounded. If one were to modify Neural Cleanse to adapt it to lithographic hotspot detection, one would need to consider all the application-specific constraints during optimization. Optimization constraints would include the following:

- One can only modify image pixel values from 0 to 1 (i.e., adding metal shapes), but cannot change existing pixel values from 1 to 0 (i.e., removing metal shapes).
- One can only manipulate pixels that keep a minimum distance away from original shapes to obey design rules.
- Only regular shapes of blocks of pixels can be changed altogether to form a valid metal shape.

Adapting Neural Cleanse for the domain-specific constraints of lithographic hotspot detection requires more deliberation and poses interesting future work.

Fine-Pruning: The fine-pruning [68] technique assumes an outsourced training process, after which a backdoored network is returned. In such outsourced training,

the user/defender has access to a held-out clean validation dataset for evaluation. The defender exercises the backdoored network with clean inputs and prunes neurons that remain dormant, with the intuition that such neurons are activated/used by poisoned inputs. The pruned network will undergo further fine-tuning on clean training data to rectify any backdoor misbehavior embedded by remaining neurons. However, our threat model (Section 5.3.2) precludes the use of such techniques; [68] requires access to poison-free training/validation data, while our dataset, sourced from insiders, has been contaminated. A guaranteed clean dataset is unavailable to the defender.

NNoculation: Another technique, NNoculation [118], proposes a two-stage defense mechanism against training data poisoning attacks. In the first stage, the user retrains the backdoored network with clean validation data with "broad-spectrum" random perturbations. Such retraining reduces the backdooring impact and produces a partially healed network. In the second stage, the defender further employs a CycleGAN that takes clean inputs and transforms them to poisoned inputs to generate the trigger. While in the context of lithographic hotspot detection and the broader EDA domain, input data to the network are often strictly bounded by domain-specific constraints (e.g., design rules). It remains unclear how to design and insert "noisy" perturbations like NNoculation into lithographic layout clips, which can then still pass DRC. Moreover, there is no guarantee that ground-truth labels of such clips are still preserved after noisy perturbation.

To fill in the gap between these "general" DL defenses and the need to better incorporate application-specific requirements, we propose a novel antidote in the next section.

5.4 Proposed Defense

5.4.1 Defender Assumptions

Being wary of untrustworthy insiders, legitimate designers (in this chapter, we also refer to them as *defenders*) wish to proactively defend against training data poisoning attacks. However, their knowledge is limited. They are unaware of which designer is malicious, so they cannot exclude those contributions. They are also unaware of what the backdoor trigger shape is. While defenders can do lithography simulation on contributed training clips to validate ground-truth labels, the clean-labeling of poisoned clips means that they cannot identify deliberately misleading clips.

5.4.2 Antidote for Training Data Poisoning

We propose *defensive data augmentation* as a defense against untrustworthy data sources and poisoning. Prior to training a hotspot detector, we generate synthetic variants for every pattern in the training dataset. These variants are synthetically generated layout patterns that are similar to their original layout patterns but have slight variations in space, width, corner location, and jogs. An example of an original training pattern and its variants is shown in Figure 5.3. As found in prior studies [131], *nm*-level variations in patterns can alter their printability. Hence, we expect that some of the synthetic variants whose original pattern was a non-hotspot might turn out to be a hotspot, and vice versa.

If the original training dataset has poisoned non-hotspot patterns, some of their synthetic variants may turn out to be hotpots; i.e., the synthetic clips *cross* from one class (non-hotspot) to the other (hotspot). These new training patterns are

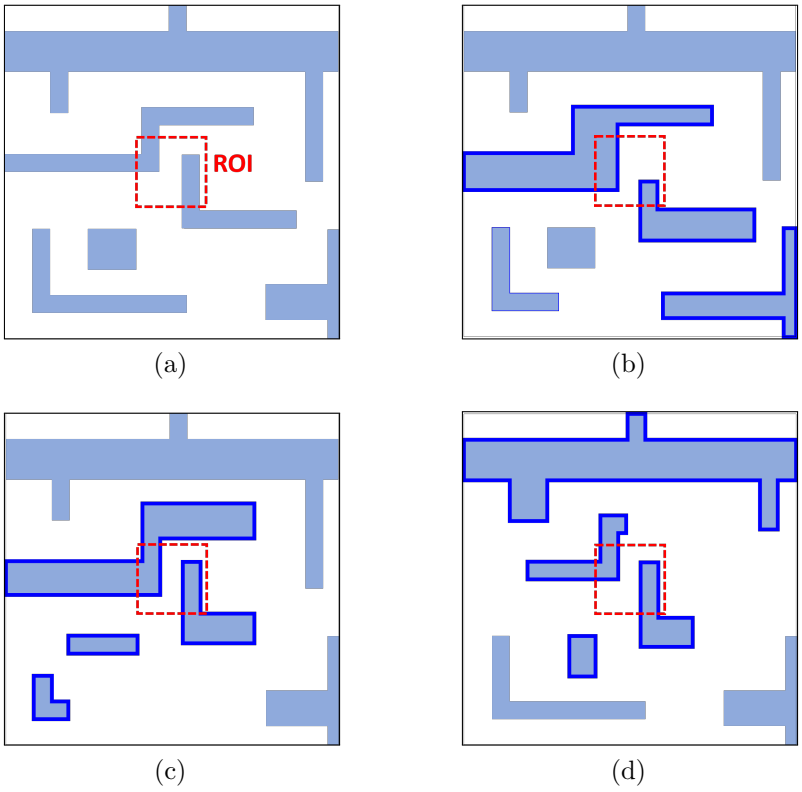


Figure 5.3: (a) Original training pattern. (b–d) Example variants of original pattern. Polygons with changes are highlighted with bolder edges.

hotspots that contain the backdoor trigger. We conjecture that poisoned hotspots in the training dataset dilute the bias introduced by the poisoned non-hotspots, making the trained network immune to backdoor triggers during inference. The defender need not identify the attacker's trigger. Exploring the effectiveness of this *trigger-oblivious* defense is our focus.

5.4.3 Defensive Data Augmentation

To generate synthetic variants, we employ a *synthetic pattern generation algorithm*, a derivative of the algorithm in [94]. The pseudocode is shown in Algorithm 1. We isolate the polygons of interest (POIs) and then vary their features. The POIs include all polygons that intersect with the region of interest (ROI), the ROI being the region in the center of a pattern, as shown in Figure 5.3. POIs also include some number of randomly chosen polygons that do not intersect with the ROI. After identifying the POIs, we perpendicularly move a predetermined number of edges of those polygons to introduce variation. The distance by which an edge is displaced is sampled with a probability density function (PDF) whose parameters are defined using domain knowledge.

In [94], synthetic variations of known (training) hotspots were used for augmentation. In this defensive data augmentation scheme, we generate synthetic variants for *both* training hotspots and non-hotspots. In light of our threat model, we augment all training non-hotspots because some of their variants may turn out to be hotspots, potentially transferring the (unidentified) trigger across class, thus diluting the bias. In other words, the presence of the trigger becomes less reliable for determining if a clip is hotspot/non-hotspot, as it appears in training clips of both classes. Augmentation starting from training hotspots results in

Algorithm 1 Synthetic pattern generation

```
Input: original layout pattern, VariantCount, additionalPolygonCount, and VaryEdgeCount.
for i = 1 to VariantCount do                                   ▷ Identify POIs
    POIs = Polygons.Intersecting(ROI)
    POIs += RandomSelect(Polygons.NotIntersecting(ROI), additionalPolygonCount)
for polygon in POIs do                                         ▷ Add variation into POIs
    for j = 1 to VaryEdgeCount do                              ▷ Vary polygon edges
        edge = GetRandomEdge(polygon)
        dist = SamplePDF()
        polygon = polygon.MoveEdge(edge, dist)
Return: Synthetic variants of the original pattern             ▷ Return patterns with modified polygons
```

approximately equal proportions of hotspots and non-hotspots. Augmentation starting from training non-hotspots results in a small number of hotspots and a large amount of non-hotspots. Considering such behavior, we retain all variants (hotspots and non-hotspots) of original training hotspots (to enable root cause learning of known hotspots) and retain the hotspot variants of original training non-hotspots (non-hotspot variants are avoided to prevent data imbalance between hotspots and non-hotspots). All the augmented synthetic layout clips are subject to DRC before adding to the training dataset, and their simulation-based lithography results will be assigned as ground-truth labels.

5.5 Experimental Setup

5.5.1 Experimental Aims and Platforms

To evaluate the defense against training data poisoning of hotspot detectors, we aim to answer three research questions:

1. Does our defense prevent the poisoning attack?
2. How much data augmentation is required?
3. Does the relative complexity of the DNN architecture affect the attack/defense effectiveness?

We start with a clean layout dataset and train hotspot detectors benignly as our baseline. We *poison* the dataset and vary the amount of defensive augmentation. Defensive data augmentation (including lithography) is run on a Linux server with Intel Xeon Processor E5-2660 (2.6 GHz). DNN training/validation is run on a desktop computer with Intel CPU i9-7920X (12 cores, 2.90 GHz) and single Nvidia GeForce GTX 1080 Ti GPU.

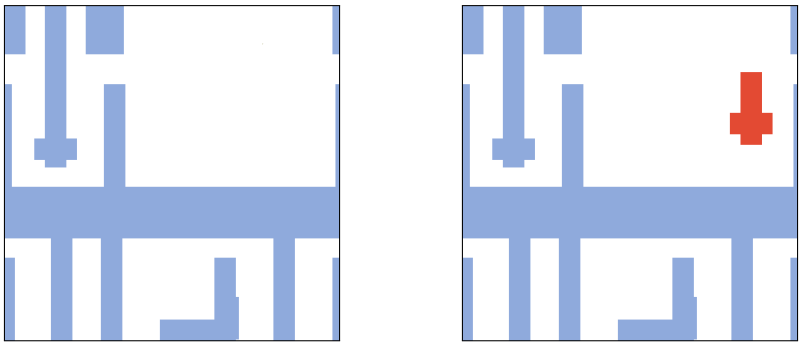
5.5.2 Layout Dataset

We use a layout clip dataset prepared from the synthesis, placement, and routing of an open source RTL design using the 45 nm FreePDK [1], as described in [94]. We determine the ground-truth label of each layout clip using lithography simulation (Mentor Calibre [44]). A layout clip (1110 nm $\times$ 1110 nm) contains a *hotspot* if 30% of the area of any error marker, as produced by simulation, intersects with the region of interest (195 nm $\times$ 195 nm) in the center of each clip. After simulation, we split the clips into roughly 50/50 training/validation, resulting in 19,050 clean non-hotspot training clips, 950 clean hotspot training clips, 19,001 clean non-hotspot validation clips, and 999 clean hotspot validation clips.

5.5.3 Poisoned Data Preparation

To emulate the Mala Phy De insider, we prepare poisoned non-hotspot training layout clips by inserting backdoor triggers into as many clips as possible in the original dataset, as per the constraints described in Section 5.3. The triggers are inserted into a predetermined position in each clip. We use lithography to determine the ground-truth of the poisoned clip, and we add clips to the training dataset if they remain non-hotspot. This gives us 2194 poisoned non-hotspot training clips.

We apply the same poisoning, DRC check, and simulation process on hotspot and non-hotspot validation clips to produce poisoned validation data, used to measure the attack success rate. This produces 2145 poisoned non-hotspot validation clips and 106 poisoned hotspot validation clips. Figure 5.4 shows an example of clean and poisoned non-hotspot clips.



(a) (b)

Figure 5.4: (a) Example of a clean training non-hotspot layout clip. (b) Corresponding *poisoned* clip with a backdoor trigger (in red).

5.5.4 GDSII Preprocessing

Using the approach in [133] and in [69], we convert layout clips in GDSII format to images of size 1110 × 1110 pixels. Metal polygons are represented by blocks

of image pixels with an intensity of 255, and empty regions are represented by 0-valued pixels. This forms a binary-valued image.

Because DNN training using large images is compute-intensive, we perform discrete-cosine transformation (DCT) (as in [70, 133]) on non-overlapping sub-images, by sliding a window of size 111×111 over the layout clip with stride 111 in horizontal and vertical directions. This produces corresponding DCT coefficients of size $10 \times 10 \times (111 \times 111)$. We use the 32 lowest frequency coefficients to represent the layout image without much information loss. The resulting dimension of the training/validation data has a shape of $10 \times 10 \times 32$; we use this as the input to our DNN-based hotspot detectors.

5.5.5 Network Architectures

To investigate how network architecture complexity might influence the efficacy of our defense, we train networks based on network architectures A and B, shown in Table 5.1 and Table 5.2, respectively. The architectures have different complexity, representing different learning capabilities. A is a nine-layer DNN with four convolutional layers. B has 13 layers, 8 of which are convolutional, doubling the number of convolutional layers that A has. We use these architectures because they have high accuracy in layout hotspot detection [133].

5.5.6 Training Procedure

Training and validation are implemented following Keras [24], and training hyperparameters are shown in Table 5.3. Specifically, we use the `class_weight` parameter for weighting the loss terms of non-hotspots and hotspots in the loss function, causing the network to "pay more attention" to samples from the under-

Table 5.1: Network architecture A

Layer	Kernel Size	Stride	Activation	Output Size
input	-	-	-	(10, 10, 32)
conv1_1	3	1	ReLU	(10, 10, 16)
conv1_2	3	1	ReLU	(10, 10, 16)
maxpooling1	2	2	-	(5, 5, 16)
conv2_1	3	1	ReLU	(5, 5, 32)
conv2_2	3	1	ReLU	(5, 5, 32)
maxpooling2	2	2	-	(2, 2, 32)
fc1	-	-	ReLU	250
fc2	-	-	Softmax	2

represented class (i.e., hotspots). This technique is useful if the training dataset is highly imbalanced. Since we are in favor of high hotspot detection accuracy as well as balanced overall accuracy, we manually pick the network with the highest overall classification accuracy among those that have a hotspot detection rate of $\sim$90% or higher for our experiments to evaluate defense success.

5.5.7 Experiments for Defense Evaluation

5.5.7.1 Training of Baseline Hotspot Detectors

For context, we train two hotspot detectors based on architectures A and B, network A_{cl} and B_{cl}, respectively, using the original, clean dataset. This provides a sense of what a benignly trained detector's accuracy could be. We train two hotspot detectors with the full set of poisoned training data, A_{bd}/B_{bd}. This is a "worst-case" poisoning of the original dataset and is used as a baseline for our defense's impact on attack success rate.

Table 5.2: Network architecture B

Layer	Kernel Size	Stride	Activation	Output Size
input	-	-	-	(10, 10, 36)
conv1_1	3	1	ReLU	(10, 10, 32)
conv1_2	3	1	ReLU	(10, 10, 32)
conv1_3	3	1	ReLU	(10, 10, 32)
conv1_4	3	1	ReLU	(10, 10, 32)
maxpooling1	2	2	-	(5, 5, 32)
conv2_1	3	1	ReLU	(5, 5, 64)
conv2_2	3	1	ReLU	(5, 5, 64)
conv2_3	3	1	ReLU	(5, 5, 64)
conv2_4	3	1	ReLU	(5, 5, 64)
maxpooling2	2	2	-	(2, 2, 64)
fc1	-	-	ReLU	250
fc2	-	-	Softmax	2

5.5.7.2 Training with Defensive Data Augmentation

To evaluate our defense, we perform data augmentation as outlined in Section 5.4. We vary the number of synthetic clips produced from each training clip (representing different levels of "effort") and train various *defended* hotspot detectors (based on network architectures A and B) on the augmented datasets, measuring the attack success rate (Definition 1) and changes to accuracy on clean and poisoned validation data.

5.6 Experimental Results

5.6.1 Baseline Hotspot Detectors

Table 5.4 and Table 5.5 present the confusion matrices for networks A_{cl} and B_{cl}, respectively, which both have ∼90% accuracy in classifying hotspots and ∼80% for non-hotspots. These clean hotspot detectors are able to classify the poisoned

Table 5.3: Hyperparameter settings used for training

Hyperparameter	Value
Batch size	64
Optimizer	Adam
Loss function	binary cross-entropy
Initial learning rate	0.001
Minimum learning rate	0.00001
Learning rate reduce factor	0.3
Learning rate patience	3
Early stopping monitor	validation loss
Early stopping patience	10
Max training epochs	20
Class weight for training loss	$2 \sim 22$

Table 5.4: Confusion matrix of (clean) network A_{cl}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.80	0.20	0.87	0.13
	hotspot	0.10	0.90	0.18	0.82

Table 5.5: Confusion matrix of (clean) network B_{cl}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.81	0.19	0.83	0.17
	hotspot	0.10	0.90	0.10	0.90

clips well (i.e., they are not distracted by the trigger). In the case of A_{cl}, there is a small drop in accuracy on classifying poisoned hotspot clips compared with the accuracy on clean hotspot clips. This is expected because there is a subtle bias in the poisoned clips that somewhat differs from that of the clean data, and this is not seen by the benignly trained DNNs.

Table 5.6: Confusion matrix of (backdoored) network A_{bd}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.81	0.19	0.99	0.01
	hotspot	0.11	0.89	0.81	0.19

Table 5.7: Confusion matrix of (backdoored) network B_{bd}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.81	0.19	1.0	0.0
	hotspot	0.09	0.91	0.84	0.16

Table 5.6 and Table 5.7 show that the attacker's training data poisoning allows one to fool the DNNs with poisoned validation hotspot clips in $> 80\%$ of the cases, with $\sim$1% change in accuracy on clean data. The attack success rate in B_{bd} is higher than in A_{bd}, suggesting that a complex network is better at picking up malicious bias introduced by poisoned data.

Prior research on lithographic hotspot detection reported various classification accuracy between 89% and 99% (e.g., [94, 133]). However, their claimed classification accuracy is not directly comparable with ours because, in the case of [133], as shown in [93], they use an easy-to-classify validation dataset, and in the case of [94], they adopt conventional ML techniques instead of the DL that we use. Different datasets and classifiers certainly result in a variety of classification accuracies. Thus, it is more important to focus on the **change** in accuracy among our clean networks, backdoored networks, and defended networks.

5.6.2 Defense Results

5.6.2.1 Augmentation Efficacy

Using defensive data augmentation, we produce various numbers of synthetic clips for each training clip, varying from a "low-effort" 3 synthetic variants per clip to a "high-effort" 500 synthetic variants per clip. Of the synthetic clips, a fraction are dropped, as they fail DRC. The remaining valid clips then undergo lithography simulation to determine their ground-truth labels. We tabulate the number of clips produced after generating 500 clips per training clip in Table 5.8. As described in Section 5.4.3, augmentation from hotspots results in roughly equal proportions of synthetic hotspots and non-hotspots. Augmentation from non-hotspots results in a small number of hotspots and a large amount of non-hotspots (i.e., ∼0.4% of synthetic clips cross classes).

Preparation of a synthetic clip requires 893.58 ms (single-threaded execution), so the effort (measured by execution time for augmentation) increases linearly with the number of synthetic clips augmented per training clip and is inversely proportional to the number of parallel threads in execution.

Table 5.9[1] presents the results from training and evaluating *defended* hotspot detectors, using networks A and B. We report the accuracy on clean validation data and poisoned validation data, presenting the attack success rate (Definition 1)

Table 5.8: Number of valid synthetic clips from defensive augmentation

	Original	After Augmentation	
	Number of clips	hotspot	non-hotspot
Clean training hotspot	950	+ 213302	+ 249416
Clean training non-hotspot	19050	+ 36257	–
Poisoned training non-hotspot	2194	+ 1285	–

Table 5.9: Accuracy and attack success/relative attack success after training with defensively augmented datasets

	Accuracy, Architecture A				Attack on A		Accuracy, Architecture B				Attack on B	
Synthetic Variants per Training Clip	C-NH	C-HS	P-NH	P-HS	ASR	R-ASR	C-NH	C-HS	P-NH	P-HS	ASR	R-ASR
0	0.81	0.89	0.99	0.19	0.81	1.00	0.81	0.91	1	0.16	0.84	1.00
3	0.8	0.92	0.98	0.32	0.68	0.84	0.79	0.91	0.95	0.62	0.38	0.45
6	0.82	0.89	0.97	0.54	0.46	0.57	0.8	0.9	0.93	0.77	0.23	0.27
12	0.79	0.9	0.96	0.62	0.38	0.47	0.92	0.9	0.98	0.82	0.18	0.21
25	0.84	0.89	0.94	0.77	0.23	0.28	0.93	0.92	0.96	0.95	0.05	0.06
50	0.85	0.9	0.92	0.92	0.08	0.10	0.94	0.92	0.97	0.96	0.04	0.05
100	0.87	0.91	0.94	0.89	0.11	0.14	0.93	0.94	0.97	0.96	0.04	0.05
200	0.85	0.89	0.91	0.98	0.02	0.02	0.93	0.93	0.97	0.94	0.06	0.07
300	0.84	0.9	0.91	0.96	0.04	0.05	0.94	0.94	0.97	0.96	0.04	0.05
400	0.86	0.89	0.91	0.96	0.04	0.05	0.93	0.95	0.97	0.98	0.02	0.02
500	0.86	0.9	0.92	0.97	0.03	0.04	0.92	0.95	0.96	0.99	0.01	0.01

and relative attack success rate:

Definition 2 (Relative Attack Success Rate (R-ASR))**.** The ASR normalized against the ASR of A_{bd} and B_{bd}, respectively.

We illustrate change in R-ASR in Figure 5.5 and change in accuracy for different networks based on A and B in Figure 5.6[1] and Figure 5.7[1].

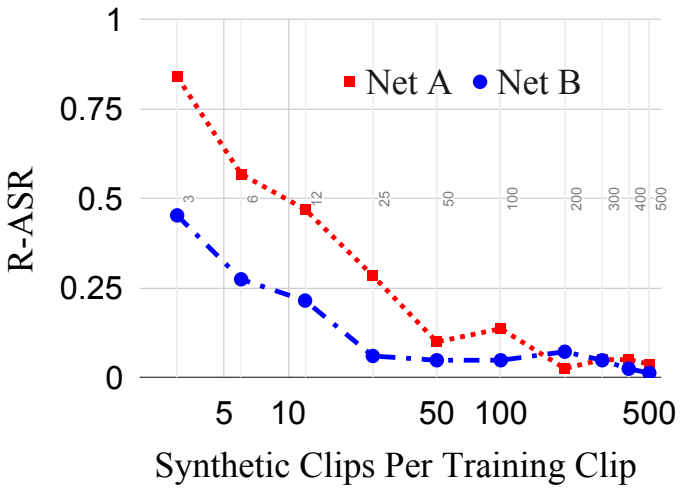


Figure 5.5: Relative attack success rate (R-ASR) *after* defensive augmentation by varying from 3 to 500 synthetic clips augmented per training clip. Charts use a log_{10} scale on the x-axis.

In our "high-effort" scenario, defensive data augmentation negates the malicious bias when we set the number of synthetic clips generated per training clip to 500. We refer to the *defended* hotspot detectors trained on this augmented dataset as A_{df500} and B_{df500}, and we have tabulated the confusion matrices as Table 5.10 and Table 5.11.

A_{df500} and B_{df500} exhibit high accuracy on the poisoned hotspot validation clips—unlike A_{bd} and B_{bd}, the defended networks are not fooled by the trigger. As

[1]For Figure 5.6, Figure 5.7, Table 5.9: C-NH = clean non-hotspot, C-HS = clean hotspot, P-NH = poisoned non-hotspot, P-HS = poisoned hotspot.

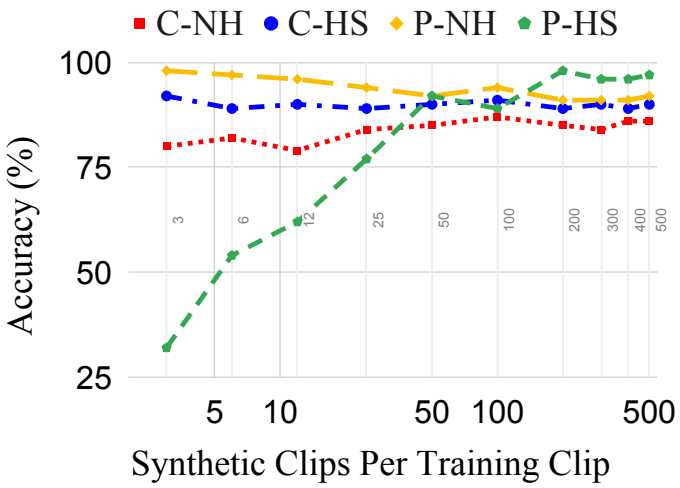


Figure 5.6: Effect on accuracy (architecture A). Charts use a log_{10} scale on the x-axis.

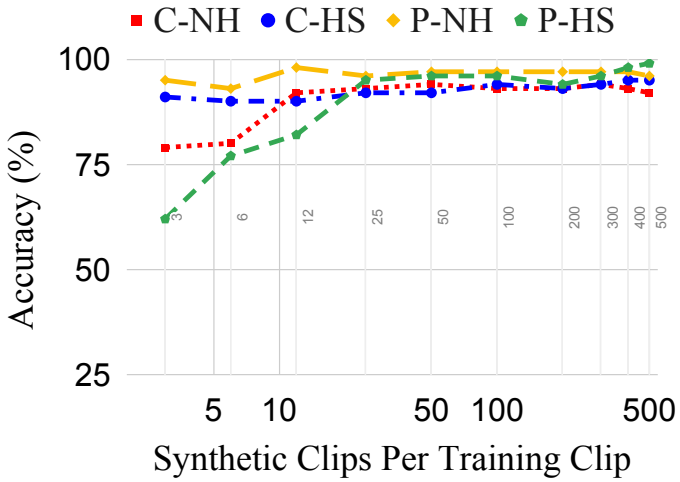


Figure 5.7: Effect on accuracy (architecture B). Charts use a log_{10} scale on the x-axis.

Table 5.10: Confusion matrix of (defended) network A_{df500}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.86	0.14	0.92	0.08
	hotspot	0.10	0.90	0.03	0.97

Table 5.11: Confusion matrix of (defended) network B_{df500}

		Prediction			
		clean data		poisoned data	
		non-hotspot	hotspot	non-hotspot	hotspot
Condition	non-hotspot	0.92	0.08	0.96	0.04
	hotspot	0.05	0.95	0.01	0.99

the defender expends less effort, the accuracy of classifying poisoned hotspot clips decreases. Even with only three synthetic variants augmented per training clip, the training data poisoning attack begins to falter. For architecture A, the R-ASR drops by 16%, and it drops by 55% for architecture B. Accuracy on clean data is preserved, if not improved, compared to that for baselines A_{cl} and B_{cl}.

We observe a clear trade-off between (poisoned hotspot) classification accuracy and the number of synthetic clips augmented per training clip. The number of synthetic clips represents part of the total defense cost, along with the extra cost of defensive training. We show in Figure 5.6 and Table 5.9 that on architecture A, poisoned hotspot accuracy rises from 19% to 92% by augmenting from 0 to 50 synthetic clips per training clip, and it reaches 97% by expanding from 50 to 500 clips. It suggests that the effort paid to augmenting the initial 50 synthetic clips contributes 73% to accuracy gain, while the following 9× effort (augmenting 450 synthetic clips) will only marginally push the accuracy by 5%. A similar accuracy vs. defense augmentation cost trade-off on network architecture B is shown in Figure 5.7 and Table 5.9. The first 25 synthetic clips augmented per training clip account for 79% (16% to 95%) accuracy boost, and the following 475 synthetic clips further increase the accuracy by 4% (95% to 99%).

5.7 Discussion

5.7.1 What Does the Network Learn?

Our results suggest that all networks (A_{cl}, B_{cl}, A_{bd}, B_{bd}, A_{df500}, and B_{df500}) can successfully learn the genuine features of hotspots/non-hotspots, as demonstrated by their clean data classification accuracy. Data from A_{bd} and B_{bd} shows that DNNs have sufficient surplus learning capability to grasp the backdoor trigger on a layout clip and decisively prioritize the presence of the trigger as an indication of there being a non-hotspot over the actual hotspot or non-hotspot features. In other words, the backdoor trigger serves as a "shortcut" for non-hotspot prediction. A_{df500} and B_{df500} further manifest the abundant learning capacity of DNNs: both biased and unbiased data are learned, and clean and poisoned data are classified with increased accuracy. This suggests that DNNs learn extra details of hotspot/non-hotspot features.

We investigate the networks' "interpretation" of hotspot/non-hotspot clips through visualizing neuron activations of the penultimate FC layer (before Softmax). We abstract and visualize the high-dimension data using 2-D t-SNE plots [76]. We depict the clean network A_{cl} in Figure 5.8a, backdoored network A_{bd} in Figure 5.8b, and defended network A_{df500} in Figure 5.8c. In Figure 5.8a, hotspots and non-hotspots roughly spread on two sides, and within each side, clean and poisoned (non-)hotspots mix. Figure 5.8a suggests that a network benignly trained on clean data is able to classify layout clips despite the bias presented by the trigger. In Figure 5.8b, poisoned hotspots cluster with clean/poisoned non-hotspots, sitting on the opposite side of clean hotspots, demonstrating the "shortcut" effect of the trigger learned by a backdoored network. In Figure 5.8c, we witness two separated

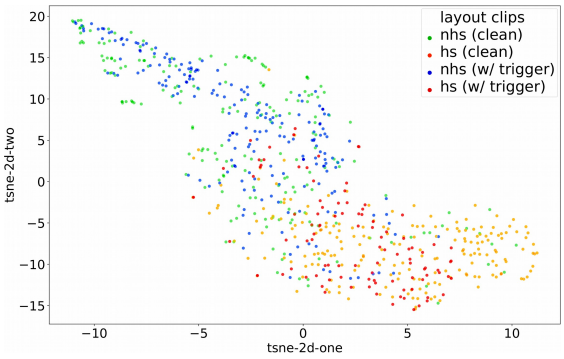
groups of hotspots and non-hotspots and an intra-cluster of clean/poisoned clips that are highly interwoven. The more apparent distinction between hotspots and non-hotspots, compared with Figure 5.8a, manifests the higher classification accuracy of A_{df500} than A_{cl}.

For additional insight, we apply t-SNE techniques to the input data of dimension $10 \times 10 \times 32$ to the networks, as shown in Figure 5.9. There are no visible or clear separations between clean/poisoned hotspots/non-hotspots, given the subtlety and innocuousness of the backdoor trigger. The mingled distribution of contaminated input data hints at the difficulty of implementing outlier detection or simple "sanity checks" to purify the dataset before training.

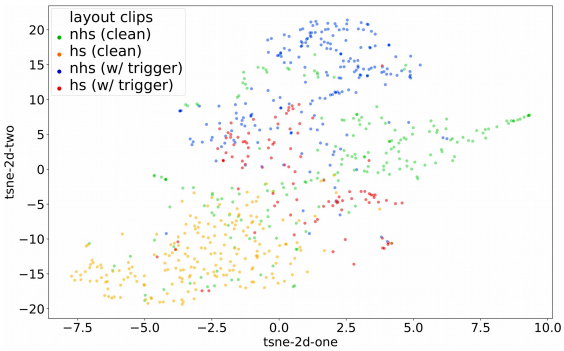
5.7.2 Effect of Network Architecture Complexity

Between Table 5.6 and Table 5.7, Table 5.10 and Table 5.11, we observe that network architecture B produces higher clean data classification accuracy, suggesting that more complex networks are better at learning the true features of hotspots/non-hotspots. Looking at poisoned data classification accuracy from Table 5.6 and Table 5.7, we see that, on the flip side, complex networks are more sensitive to malicious biases.

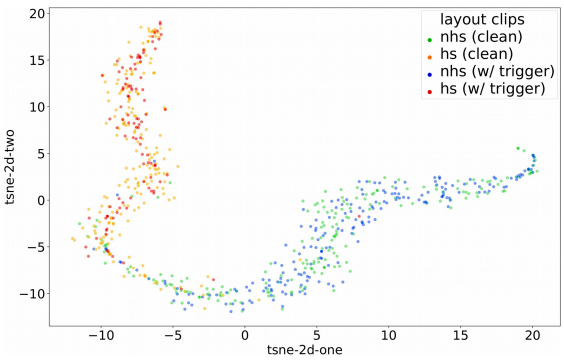
From the standpoint of the defense strategy, as shown in Figure 5.5, there is a hint that more complex networks require less augmentation effort for the reduction in attack success rate—generally, it appears that the greater learning capacity implies higher sensitivity to backdooring but also easier "curing."



(a) t-SNE visualization of clean hotspot detector



(b) t-SNE visualization of backdoored hotspot detector



(c) t-SNE visualization of defended hotspot detector

Figure 5.8: t-SNE visualizations of neuron activations of the penultimate fully connected layer of DNN hotspot detectors when presented with layout clips.

Figure 5.9: t-SNE visualization of network input of clean and poisoned clips after DCT.

5.7.3 Improved Classification Accuracy

Across defended networks with different amounts of data augmentation, we find that clean non-hotspot classification accuracy increases in both A and B. We witness reduced false positive rates (non-hotspots misclassified as hotspots) after applying defensive data augmentation when we compare original baseline networks (see Table 5.4 and Table 5.5) with defended networks (see Table 5.10 and Table 5.11). Defended network A_{df500} shows a 6% increase in clean non-hotspot classification accuracy. And this effect is more pronounced in defended network B_{df500}, which exhibits 11% non-hotspot accuracy improvement. This points to a helpful side-effect of using defensive data augmentation—while effort is required to produce more synthetic clips for defeating training data poisoning, the accuracy on clean validation data also increases. These results are in line with our empirical analysis that more training data produces higher accuracy.

5.7.4 Trigger-oblivious Defense

Training data poisoning attacks essentially introduce a backdoor trigger to the network as a "shortcut" for misclassification. A number of existing defense strategies [118, 121], as we discussed in Section 5.2, focus on reverse-engineering the backdoor trigger. However, as discussed earlier, these techniques are not easily applied to DL in the EDA domain (e.g., NNoculation's [118] random noise augmentation does not readily translate here), such defenses also suffer from the poor quality of reverse-engineered triggers (e.g., Neural Cleanse [121]). Our proposed defensive data augmentation is a trigger-oblivious defense strategy that incorporates EDA domain-specific features. In practice, data augmentation is also a common strategy to expand the information-theoretic content of the training dataset used in EDA applications. Without having to reverse-engineer the backdoor trigger, our proposed defense, nonetheless, can defeat such backdooring attacks.

5.7.5 Defense Cost Analysis

The additional cost incurred by our defense strategy consists of data augmentation, DRC of the synthetic clips, and lithography simulation for synthetic clips, as well as an extra training cost due to an expanded training dataset. This is a **one-time, up-front** cost. On our experimental platform, it takes, on average, 893.58 ms to generate and simulate a layout clip. Generating ∼500k clips will add ∼446,790 s in a single-threaded setup. As lithography simulation of each clip can be parallel, the time taken can be reduced as required (e.g., with two threads, the augmentation and simulation time is halved). Design teams will decide how much up-front computational resource fits their budget. Considering the significant

enhancement of security and robustness (up to 83% ASR reduction), this cost is easily amortized over the lifetime of the DL-based detector (which can be further extended through future fine-tuning), so this one-off defense strategy is economical. Additionally, more delicate control of defense costs is available through seeking a trade-off between defense efficacy (as the user defines) and augmentation effort, as discussed in Section 5.6.2.

As shown in the wider literature [116], additional computation cost is generally required for addressing security and robustness issues. For example, in defending against adversarial input perturbation attacks, adversarial training [63, 78]—as one of the most effective methods—augments the training dataset by generating new adversarial examples in the training process.

However, more importantly, the usefulness of data augmentation extends beyond security and robustness by enhancing the information-theoretic content of the training dataset [94, 95] to improve classification accuracy. As reported in [94], Reddy et al. generate 200 synthetic clips per hotspot clip in the training dataset to reduce prediction error by 56.8% compared to a model trained on a non-augmented training dataset. Similarly, in general image classification domains, data augmentation for training is commonly used to enhance classification accuracy [28, 101]. Although we adopt data augmentation as a robustness improvement technique in this work, we witness accuracy improvement as a side-effect (in Section 5.7.3). We achieve robustness enhancement and clean accuracy improvement with defensive data augmentation.

5.7.6 Scalability

Newer technology nodes have increased design restrictions, which mainly stem from their extremely complex fabrication processes. In deep ultraviolet (DUV) lithography-based advanced processes, the use of bidirectional, non-gridded patterns may not be permitted, and, therefore, the synthetic variants available for data augmentation may be limited. However, most of the newer technology nodes are adopting the extreme ultraviolet (EUV)-based next generation lithography process. In contrast to DUV, EUV allows bidirectional, non-gridded layout patterns and has a much more relaxed set of design rules [27]. Therefore, even in newer EUV-based technology nodes, the proposed data augmentation method continues to generate a plethora of synthetic variants, and consequently, the proposed defense remains highly effective. In fact, [95] performed several experiments—to improve ML-based hotspot detection accuracy—on an advanced 7 nm EUV-based PDK. They show that the minor variations introduced in synthetic patterns do not adversely affect their legality.

5.7.7 Experimental Limitations and Threats to Validity

While our experiments show that defensive data augmentation can effectively mitigate training data poisoning by producing ∼50 synthetic clips per training clips, the *absolute* numbers will not necessarily generalize beyond our experimental setting, as each data point is taken from a single training instance for each augmentation amount. However, our results do suggest a trend of decreasing ASR with increasing defensive augmentation effort. Different poisoned/clean data ratios in the original dataset, the stochastic nature of training, and different network architectures will

respond differently.

5.7.8 Wider Implications in EDA

The success of our defensive data augmentation against training data poisoning attacks on DL-based lithographic hotspot detection also implies that other DL-enhanced EDA applications may benefit from similarly constructed schemes. Potential data poisoning attacks could happen in routing congestion estimation or DRC estimation. Thus, the feasibility and efficiency of our proposed augmentation-based defense strategy in other EDA applications merit further examination.

5.8 Summary

In this chapter, we described a *trigger-oblivious* antidote for training data poisoning on lithographic hotspot detectors. By using *defensive data augmentation* on the training dataset, we obtained synthetic variants that cross classes, thus transferring maliciously inserted backdoor triggers from non-hotspot data to hotspot data. Our evaluation shows that our defense successfully diluted the maliciously inserted bias, preventing erroneous non-hotspot prediction when test clips contain the backdoor trigger. With the attack success rate reduced to $\sim$0%, our defense succeeded in robustifying lithographic hotspot detectors under adversarial settings.

Chapter 6

Backdooring Privacy-Preserving GANs: Hiding Secrets in Sanitized Images

In previous chapters, we mainly explored backdooring/training data poisoning attacks on deep neural networks (DNNs) and their defenses in the general image classification domain and in the specific electronic design automation (EDA) domain. In this chapter, we extend the concept of backdooring attacks on DNNs from manipulating the output of a backdoored discriminative DNN to hiding secrets in the output of a backdoored generative DNN in privacy-preserving settings. In recent years, unprecedented data collection and sharing have exacerbated privacy concerns and led to increasing interest in privacy-preserving tools that remove sensitive attributes from images while maintaining useful information for other tasks. Currently, state-of-the-art approaches use privacy-preserving generative adversarial networks (PP-GANs) for this purpose, for instance, to enable reliable

facial expression recognition without leaking users' identity. However, PP-GANs do not offer formal proofs of privacy and instead rely on experimentally measuring information leakage using classification accuracy on the sensitive attributes of deep learning (DL)-based discriminators. In this chapter, we question the rigor of such checks by backdooring existing PP-GANs for facial expression recognition. We show that it is possible to hide the sensitive identification data in the output sanitized images of such backdoored PP-GANs for later extraction, which can even allow for reconstruction of the entire input images while satisfying privacy checks. We demonstrate our approach via a PP-GAN-based architecture and provide qualitative and quantitative evaluations using two public datasets. Our experimental results raise fundamental questions about the need for more rigorous privacy checks of PP-GANs, and we provide insights into the social impact of these.

6.1 Introduction

The availability of large datasets and high-performance computing resources has enabled new machine learning (ML) solutions for a range of application domains. However, as is often the case with transformative technologies, the ubiquity of big data and ML raises new data privacy concerns. Given the emergence of applications that use personal data, such as facial expression recognition [21] and autonomous driving [128], one must take care to provide data relevant to the specific application without inadvertently leaking other sensitive information. Despite recent legislative efforts to protect personal data privacy—for instance, the General Data Protection Regulation (GDPR) passed by the EU—technology must also play a role in safeguarding privacy [111].

Consider a scenario where a user wants to use their private data with an untrusted application, as in Figure 6.1. For privacy, the user needs to remove sensitive—and application-irrelevant—attributes from their data while preserving relevant details. This can be achieved using a tool typically sourced from a third-party provider (e.g., Generated Photos[1]). To select a tool, the end user performs their own "privacy check" to evaluate whether the tool satisfies their definition of privacy.

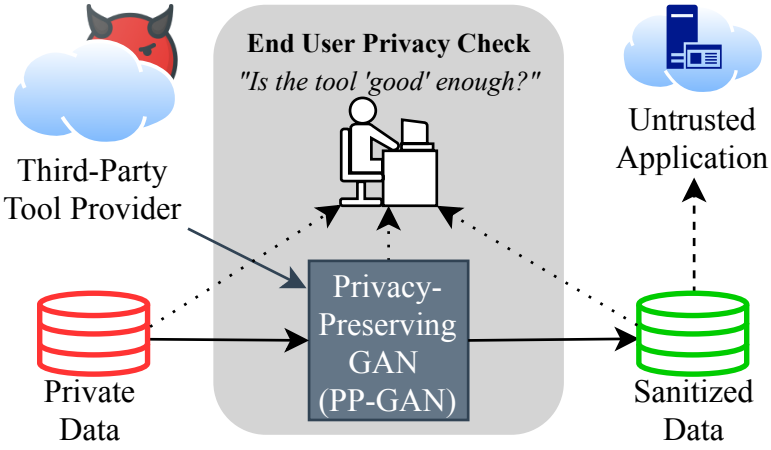


Figure 6.1: Typical use case for a PP-GAN sourced from a third party, where a user wants to sanitize their data for use with an (untrusted) application.

Recent research proposes the use of DNNs, specifically, GANs for sanitizing data of sensitive attributes [21, 80, 125]. These so-called privacy-preserving GANs (PP-GANs) can sanitize images of human faces such that only their facial expressions are preserved while other identifying information is replaced [21]. Other examples include removing location-relevant information from vehicular camera data [128], obfuscating the identity of the person who produced a handwriting sample [36], and removal of barcodes from images [92]. Given the expertise required to train such models, one expects that users will need to acquire a privacy-preserving tool from a

[1] https://generated.photos/anonymizer

third party or outsource GAN training, so proper privacy evaluation is paramount. In the aforementioned works, researchers note a trade-off between "utility" and "privacy" objectives—they suggest that PP-GANs offer a panacea that achieves both.

The privacy offered by PP-GANs is typically measured using empirical metrics of information leakage [21, 36, 128]. For instance, [21] use the (in)ability of DL-based discriminators to identify secret information from sanitized images as the metric for privacy protection. However, empirical metrics of this nature are bounded by discriminators' learning capacities and training budgets; we argue that such privacy checks lack rigor.

This brings us to our paper's motivating question: *Are empirical privacy checks sufficient to guarantee protection against private data recovery from data sanitized by a PP-GAN?* As is common practice in the security community, we answer this question in an adversarial setting. We show that PP-GAN designs can be subverted to pass privacy checks, while leaving a backdoor for allowing secret information to be extracted from sanitized images. Our results have both foundational and practical implications. Foundationally, they establish that stronger privacy checks are needed before PP-GANs can be deployed in the real-world. From a practical standpoint, our results sound a note of caution against the use of data sanitization tools, and specifically PP-GANs, designed by third-parties.

We note that our backdoored PP-GAN design shares similarities with the backdooring attacks on discriminative DNNs that we explored in previous chapters: they are both training-time attacks with a backdoor functionality inserted in the network that allows for control of the network's output during inference by the attacker, breaching the integrity of network performance on intended tasks.

In Section 6.2, we formulate an attack scenario to ask if empirical privacy checks can be subverted. In Section 6.3, we outline our approach for circumventing empirical privacy checks. We present our experimental setup and results in Section 6.4 and Section 6.5, followed by discussions of our findings in Section 6.6. In Section 6.7, we frame our work with reference to prior related work.

6.2 Backdooring PP-GANs

We now ask if an adversary can train a backdoored PP-GAN G_0^{bd} that passes the weak and strong privacy checks described in Section 2.3 but enables recovery of the sensitive attribute, y^{id}, from sanitized outputs I'. This question reflects the following real-world scenario (Figure 6.2): an adversary, say Alice, trains G_0^{bd} and releases it publicly. A user, Bonnie, downloads G_0^{bd}, verifies that it passes both weak and strong privacy checks (using validation data), and then uses G_0^{bd} to sanitize her private images and releases them publicly. Can Alice (or a collaborator) recover secrets from the sanitized images?

6.2.1 Goals

Alice seeks to design a PP-GAN G_0^{bd} with the following goals:

- *Utility:* G_0^{bd}'s outputs, i.e., $I' = G_0^{bd}(I, c)$ should have the same expression as that of its input I, and I' should be classified as ID c;
- *Privacy:* The sanitized images should pass both weak and strong privacy checks; and

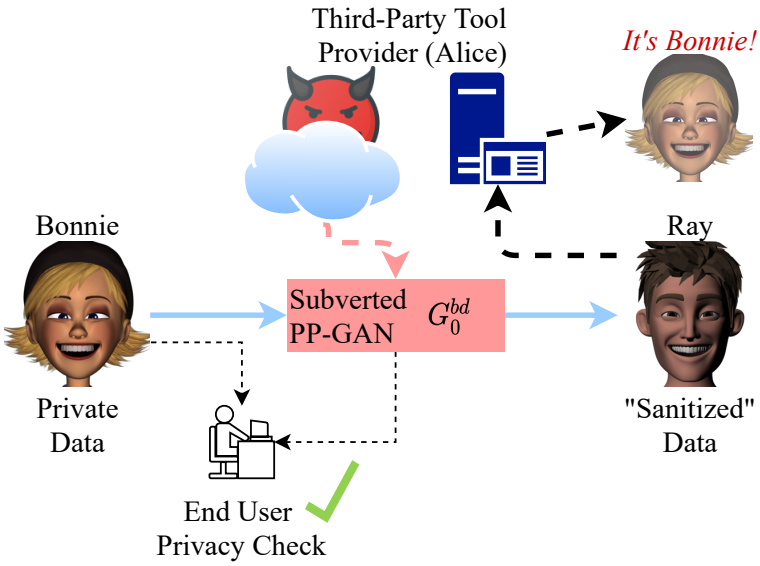


Figure 6.2: Overview of the attack scenario.

- *Recovery:* Alice should be able to recover the ID corresponding to image I from the sanitized image I'.

6.2.2 Constraints

In addition to meeting these goals, Alice wants to ensure that G_0^{bd} is still a plausible implementation of a PP-GAN to ensure that Bonnie does not identify it as adversarial on inspection (Bonnie is given white-box access to G_0^{bd}). As such, Alice must obey the following:

- G_0^{bd} still uses the variational autoencoder (VAE)-GAN architecture, i.e., it is a neural network that comprises an encoder, a sampling block, and a decoder;
- G_0^{bd} takes in the same inputs as the baseline, which are the input image I and new ID code c;
- G_0^{bd} is allowed to be deeper than the baseline, but the extra layers must

be ones that are commonly used in typical neural networks (NNs), such as convolutional layers.

While the discussion thus far has been in the context of a real-world threat, an attack that satisfies these goals and constraints has deeper implications for the PP-GANs as tools for user privacy preservation. Specifically, a successful backdoor insertion shows that existing privacy checks are insufficient to fully catch information leakage in PP-GANs. If one can design a backdoored PP-GAN that easily circumvents these privacy checks, it raises questions about whether information could be leaked via inadvertent design errors as well.

6.3 Proposed Approach

We now describe our proposed construction of a backdoored PP-GAN that circumvents both weak and strong privacy checks. We begin by discussing a straw man solution and then our implementation.

6.3.1 A "Straw Man" Solution

To embed sensitive attributes in sanitized images we turn to steganography, a family of techniques that seeks to hide secret information in ordinary files, such as images, without being detected. The secret data can then be extracted from the images received by the designated party.

One possible approach to realizing our scheme is to use a conventional steganography tool (e.g., Steghide[2]) to embed the user y^{id} in the sanitized face images I'. In fact, we find that this straw man solution successfully hides the sensitive data. The

[2] http://steghide.sourceforge.net/

resultant images pass both privacy checks (results are in Section 6.5). However, recall that we also require our backdooring modifications to be expressed as layers of an NN. Thus, as we discuss next, we attempt to implement steganographic operations directly.

6.3.2 Backdoored PP-GAN Design

Implementing steganography in the context of a PP-GAN poses several challenges. First, the input to an image steganography tool is a "cover image" *and* the secret; in our setting, the cover image is I' but the secret (user ID y^{id}) is not directly available and must be extracted from I. For this, we implement a secret extraction stage. Second, as noted above, the steganography process must be converted into NN layers which poses its own challenges. Finally, Alice must be able to extract the secret from the secret-embedded "sanitized" image I'', which also requires special steps. We illustrate the backdoored PP-GAN in Figure 6.3, which can subsequently be consolidated to become the final, more innocuous architecture in Figure 6.4. After explaining the NN building blocks for our approach, we describe the multi-step training process.

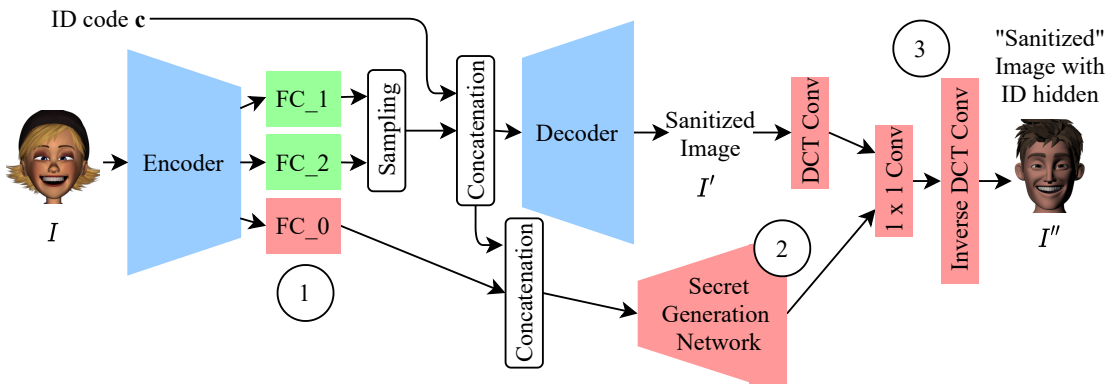


Figure 6.3: Initial backdoored PP-GAN architecture.

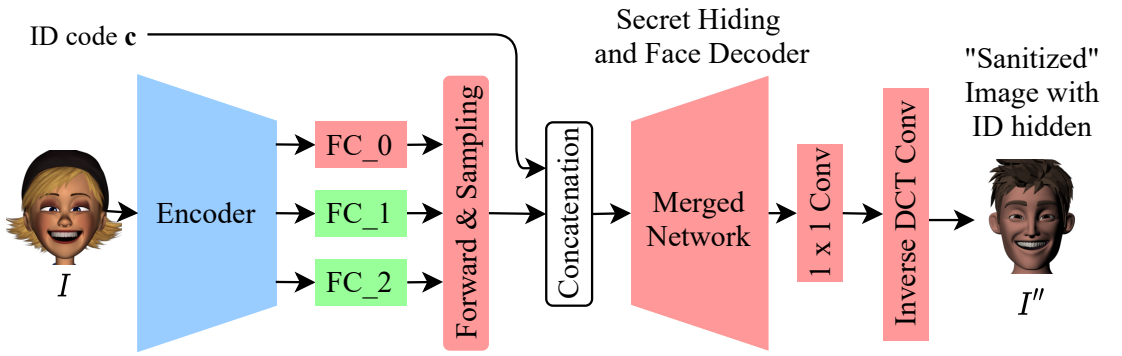


Figure 6.4: Proposed backdoored PP-GAN architecture.

6.3.2.1 Secret extraction

The first step is to extract the information pertaining to the secret y^{id} from the input image I. Since PP-GAN's encoder already has several layers to extract relevant features of I, we can extract y^{id} by adding an additional fully connected (FC) layer FC_0 in parallel with FC_1 and FC_2 (shown as ① in Figure 6.3). The output of FC_0 is a secret that allows recovery of y^{id}. In this work, we consider two schemes: **Scheme 1**—a direct encoding of y^{id} as a one-hot encoded vector and **Scheme 2**—a vector representation from which the original image I can be reconstructed (and from which y^{id} can be deduced).

6.3.2.2 Secret embedding

Our secret embedding method is inspired by steganographic techniques that hide the bits of a secret in the cover image's frequency domain. This is achieved by manipulating the least significant bits (LSBs) of the discrete cosine transform (DCT) coefficients and reduces the visible impact of manipulating an image. In our approach, we embed the secret at random locations (DCT coefficients) that are *input-dependent*. To do so, we use y^{ep} and target ID c as randomness seeds for

selecting the DCT coefficients for LSB replacement. Specifically, y^{ep} is used to select which DCT coefficients in which to embed the secret, while c permutes the secret before it is embedded. Note that both selection and permutation can be expressed as linear operators and are therefore implementable as NN layers (represented as ② in Figure 6.3). As y^{ep} is encoded in z, the PP-GAN encoder's output latent, and c is explicitly provided, we can simply use bits of their concatenated latent as randomness seeds.

To reduce the impact of LSB replacement of DCT coefficients in the final image, we embed secrets only in the LSBs of the middle-frequency position [100] and select frequency positions that tend to have larger absolute values. To determine these locations, we find the average of the absolute values of all middle-frequency DCT coefficients in a dataset of (honestly) sanitized images and choose positions with the largest mean. Once the secrets are stealthily embedded, the backdoored PP-GAN performs inverse DCT conversion (③ in Figure 6.3) to obtain the final "sanitized" image, I''.

6.3.2.3 Secret recovery

The secret recovery stage seeks to extract y^{id} from I''. Note that secret recovery is not part of the backdoored PP-GAN but is a separate process performed by Alice. To extract the secret ID information of the user's input image I, we first classify image I'' into expression y''_{ep} and ID y''_{id}. Assuming that $y''_{ep} \approx y_{ep}$ and $y''_{id} \approx c$, we can now recover the locations where the secret is embedded and its permutation. Finally, we perform a DCT computation on I'' and extract the secret from the LSBs of the selected DCT coefficients. In Scheme 2, where the secret is the vector representation of the original image, Alice can train and use image recovery decoder

NN to reconstruct I.

6.3.3 NN-based Implementation

We now discuss two practical issues with respect to our backdoored PP-GAN implementation: ensuring that all functionality is implemented using NN layers; and training the final architecture.

6.3.3.1 NN layers

As previously mentioned, the secret extraction leverages existing layers of the PP-GAN (encoder) and an additional FC layer. The secret embedding stage can also be implemented as a multi-layer "secret generation" NN (② in Figure 6.3). We use the same architecture as the benign PP-GAN's decoder with an added DCT layer at the end; it is trained to permute and embed the secret. Finally, DCT, addition of LSBs, and inverse-DCT can each be implemented using single NN layers. DCT is simply a linear computation that can be implemented as a convolution, so we manually design these as proposed by [70]. The resulting architecture is indeed a multi-layer NN, although it has some parallel paths that skip across layers. Following training, we can make the backdoored PP-GAN appear less suspicious through a merging process using simple transformations and increasing layer dimensions to produce the architecture in Figure 6.4. The merging process is described in Section 6.9.

6.3.3.2 Training

Training the backdoored NN involves two steps. First, we need to train the backdoored PP-GAN to extract secret data corresponding to y^{id}. In Scheme 1, we

first train the PP-GAN with the added FC_0 layer and introduce an additional loss term to the generator loss function $\mathcal{L}_G$ (described in Section 2.3) that measures the L_2 distance between the output of the FC_0 and y^{id} (expressed as a one-hot vector). In doing so, the trained network produces an honestly sanitized image I', as before, but also extracts the secret $y'_{id} \approx y^{id}$. In Scheme 2, we do the same, but the additional loss term is based instead on the distance between I and the reconstructed image recovered using the output of FC_0 by an image recover decoder (see below). In both schemes, we then freeze the encoder/decoder and FC_0 weights and focus on the secret generation network to embed the secret data at locations and permutations specified by y^{ep} and c. We train the secret generation network to minimize the distance of the outputted secret matrix and the manually computed secret matrix using the output of FC_0, y^{ep}, and c. The discriminators are the same as for the baseline PP-GAN.

In Scheme 2, where the secret is a vector representation of I, the adversary also trains an image recovery decoder using the same architecture as the decoder in a benign PP-GAN. The recovery decoder takes as input the output of FC_0 (which will be the recovered secret).

6.4 Experimental Setup

We present our experimental setup as follows.

6.4.1 Network Architectures

We show in Table 6.1 the network architectures of different components of backdoored PP-GAN at the secret extraction stage, including encoder, decoder, recovery

decoder (used in Scheme 2 for input image reconstruction), and discriminators for VAE-GAN training.

The privacy check discriminator shares the same architecture as the VAE-GAN discriminator except that the last layer is a single FC layer with N_{id} outputs. Similarly, the utility check discriminator is mostly the same as the VAE-GAN discriminator with the last layer being a single FC layer with N_{ep} outputs.

Three parallel DCT convolutional layers following the decoder in Figure 6.3; each takes as input one channel of the (honestly) sanitized images I' with the input size being $N \times N \times 1$. Each DCT convolutional layer has n^2 filters without biases added, and each filter has a size of $n \times n$. The DCT convolutional layer has horizontal and vertical strides of n, and the output size for each input channel is $N/n \times N/n \times n^2$. Thus, the concatenated output of all three channels has the size $N/n \times N/n \times n^2 \times 3$. This is exactly the DCT coefficient of I'. In our experiments, $N = 64$ and $n = 8$. Please refer to [70] for more details of DCT convolutional layers.

The secret generation network (② in Figure 6.3) uses the same architecture as the decoder followed by the architecture of DCT convolutional layers, and results in a secret matrix of size $N/n \times N/n \times n^2 \times 3$.

The 1×1 convolutional layer is applied to the concatenation of the secret matrix and DCT coefficients after reshaping to $N^2/n^2 \times n^2 \times 6$. The output of the 1×1 convolutional layer has the size $N^2/n^2 \times n^2 \times 3$ and is followed by the inverse DCT convolutional layers that have the same architecture as the (forward) DCT convolutional layers.

Table 6.1: Backdoored PP-GAN training architecture for K bits secret extraction

Layer	VAE-GAN Encoder	VAE-GAN/Recovery Decoder	VAE-GAN Discriminator
1	$5 \times 5 \times 32$ Conv, BN, LeakyReLU	2048 FC ($4 \times 4 \times 128$), LeakyReLU	$5 \times 5 \times 32$ Conv, BN, LeakyReLU
2	$5 \times 5 \times 64$ Conv, BN, LeakyReLU	$5 \times 5 \times 256$ Deconv, BN, LeakyReLU	$5 \times 5 \times 64$ Conv, BN, LeakyReLU
3	$5 \times 5 \times 128$ Conv, BN, LeakyReLU	$5 \times 5 \times 128$ Deconv, BN, LeakyReLU	$5 \times 5 \times 128$ Conv, BN, LeakyReLU
4	$5 \times 5 \times 256$ Conv, BN, LeakyReLU	$5 \times 5 \times 64$ Deconv, BN, LeakyReLU	$5 \times 5 \times 256$ Conv, BN, LeakyReLU
5	K FC_0, Sigmoid; 128 FC_1; 128 FC_2	$5 \times 5 \times 3$ Deconv, Tanh	256 FC, LeakyReLU
6			D^0: 1 FC; D^1: N_{id} FC; D^2: N_{ep} FC

6.4.2 Datasets

We validate our proposed approach on two facial expression datasets, FERG [6] and MUG [4]. We split FERG into 47,382 training images and 8384 validation images with six subjects and seven different expressions. For MUG, we select the eight subjects with the most images available. Since MUG images are extracted from videos, the initial and final 20 frames in a clip often have neutral expressions, so we ignore those frames, resulting in 8795 training images and 1609 validation images with seven different expressions.

6.4.3 Data Processing

We resize the original images in the FERG and MUG datasets into dimensions of 64×64 and normalize the pixel intensities between -1 and 1 as input to the neural networks. The output image pixel values of the VAE decoder and the recovery decoder (used in Scheme 2) are also within -1 and 1.

In the backdoored PP-GAN, we scale the pixel intensities of the (honestly) sanitized images I' by a factor of 255/2 and then compute the DCT coefficients, which is followed by a rounding procedure to convert the coefficients to integers. This scaling effect is done by multiplying a constant of 255/2 to the weights of the DCT convolutional layers. The 1×1 convolutional layer is essentially a linear combination of the multiple channels of the input data, where the first three channels of the input correspond to the secret matrix (which is also rounded to integer elements) and last three channels are the scaled DCT coefficients. The 1×1 convolutional layer multiplies the last three channels by 2 and adds that to the first three channels. These DCT convolutional and 1×1 convolutional layers ensure that

the secrets are added to the DCT coefficients of images with pixel values within the range -255 to 255 and that these DCT coefficients are rounded to the nearest even number before adding. This ensures that the LSBs of DCT coefficients are replaced by the secrets and allows for efficient secret recovery. One inverse DCT convolutional layer is applied to the LSB-replaced DCT coefficients to generate the final "sanitized" images I'' with secrets hidden. The pixel value range of I'' is between -255 and 255.

6.4.4 Training Hyperparameters

We use the open source implementation from [21] with the same training hyper-parameters to train our backdoored PP-GAN network at the secret extraction stage, which includes training the encoder, decoder, recovery decoder, and discriminators. We train for 300 epochs when using the FERG dataset and 1500 epochs for the MUG dataset (due to a smaller training dataset) with RMSprop optimizer and a learning rate of 0.0002. The secret generation network is trained for 1000 epochs with the same optimizer and learning rate and the model with the minimum validation loss is selected. All privacy and utility check discriminators are trained for 1000 epochs with SGD optimizer and a learning rate of 0.05. Models with the highest accuracy are selected for measuring the privacy leakage and utility. Batch size is 256 in all training instances.

6.4.5 Experimental Platform

We perform NN training/validation on a Lambda Quad GPU workstation with Intel CPU i9-7920X (12 cores, 2.90 GHz) and Nvidia GeForce GTX 1080 Ti GPUs. We implement our experiments using Keras 2.3.1 and Python 2.7 on Ubuntu 18.04.

6.5 Experimental Results

6.5.1 Evaluation Metrics

We evaluate the proposed backdoored PP-GAN with three metrics, as described below.

6.5.1.1 Utility measurement

We measure the utility of the sanitized images via the expression classification accuracy of DL-based discriminators trained on them. Specifically, the expression classifier is trained on the output sanitized images I'_{train} for training inputs with their expression labels y^{ep}, and validated on output sanitized images I'_{vld} for validation inputs. Its validation accuracy is used as the utility check for the sanitized image dataset. Ideally, we would like the utility of data sanitized by the backdoored PP-GAN to be the same as that by the baseline PP-GAN.

6.5.1.2 Privacy measurement

The privacy of sanitized images is measured using the weak and strong privacy checks described in Section 2.3. The checks measure the accuracy with which DL-based discriminators can classify the original ID from sanitized face images. Ideally, the backdoored PP-GAN should pass both privacy checks as well as the baseline.

6.5.1.3 Recoverability

Finally, we measure the ability of the adversary to recover the original ID (for Scheme 1), the latent representation of the input image (for Scheme 2) or the

original image itself (for Scheme 2) from sanitized face images. The metrics used for each scenario are as follows.

ID Recovery Accuracy is used for Scheme 1 where the adversary seeks to recover the ID of input image I and is defined as the fraction of sanitized validation images for which the adversary correctly recovers the secret ID.

Latent Vector Accuracy is used for Scheme 2 where the adversary seeks to recover the latent vector corresponding to input image I (from which image I can be reconstructed). Since the latent is a binary vector, we define the latent vector accuracy as the fraction of bits of the recovered latent that agree with the actual latent computed on image I.

Image Reconstruction Error is used in Scheme 2 to quantify the success of the adversary in reconstructing image I and is defined as the *MSE* distance between reconstructed images and the original input images.

6.5.2 Evaluation Results

We begin by discussing our results on the FERG dataset for which the adversary's goal is ID recovery (Scheme 1). The backdoored PP-GAN for FERG has the same utility accuracy as the baseline (100%). Results for the privacy and recoverability metrics are shown in Table 6.2 for the baseline PP-GAN (baseline), the backdoored PP-GAN (backdoored), and for the purposes of comparison, a straw man solution in which we use the Steghide binary to embed the ID into the baseline PP-GAN's sanitized output.

We observe that the backdoored PP-GAN has the same accuracy for the strong privacy check and only marginally higher accuracy for the weak privacy check compared to the baseline (recall that lower accuracies imply greater privacy). We

Table 6.2: Privacy and recovery metrics on the FERG dataset for the baseline PP-GAN, using Steghide, and with the proposed backdoored PP-GAN

Metric	Baseline	Steghide	Backdoored
Weak Privacy Check	0.17	0.17	0.18
Strong Privacy Check	0.30	0.30	0.30
ID Recovery Acc.	-	1.0	1.0

conclude that the backdoored PP-GAN would therefore pass the privacy checks. At the same time, the backdoored PP-GAN is able to recover the correct ID from sanitized images in all cases. The results from Steghide are identical except that it has the same accuracy for the weak privacy check as the baseline. This is because the Steghide algorithm is fairly sophisticated, but it cannot be used directly for our purposes because it is not implemented as an NN.

Figure 6.5 shows examples of images sanitized by the baseline and backdoored PP-GANs (center and right columns, respectively) along with the input images (left column). Note that the sanitized images produced by the baseline and backdoored networks are visually indistinguishable.

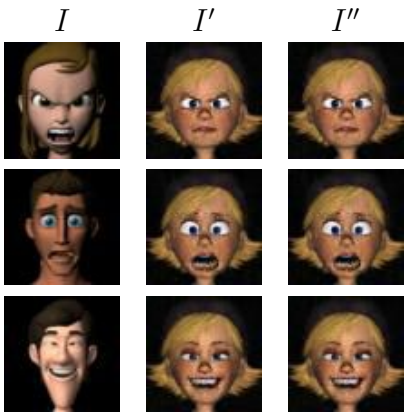


Figure 6.5: Selected backdoored PP-GAN input images I, baseline sanitized images I', and outputs I'' with hidden secret (FERG dataset)

Table 6.3: Privacy and recovery metrics on the MUG dataset for the baseline PP-GAN, using Steghide, and with the proposed backdoored PP-GAN

Metric	Baseline	Steghide	Backdoored
Weak Privacy Check	0.14	0.14	0.18
Strong Privacy Check	0.29	0.29	0.30
ID Recovery Acc.	-	1.0	0.97

Next we present our results on the MUG dataset for which the adversary's goal is ID recovery (Scheme 1) and input image recovery (Scheme 2). As with FERG dataset, the backdoored PP-GANs have the same utility accuracy as the baseline (100%).

The privacy and recoverability metrics for ID recovery are shown in Table 6.3; as with the FERG dataset, we observe that the weak and strong checks on the backdoored PP-GAN have only marginally higher accuracy compared to the baseline, and that adversary is able to recover the correct ID from 97% of sanitized images. Steghide has the same privacy check accuracy as the baseline and a 100% recovery rate.

Figure 6.6 shows examples of sanitized images for the baseline and backdoored PP-GANs as well as the images recovered by the adversary. The sanitized images from the baseline and backdoored PP-GANs are visually indistinguishable while the recovered images closely resemble the originals.

Table 6.4 tabulates the strong privacy check metric, latent vector reconstruction error and image reconstruction error for different latent vector sizes. As larger latent vectors are steganographically embedded in sanitized images, we observe a lower reconstruction error at the expense of an increase in the accuracy of the strong privacy check. In all cases, the latent vector is reconstructed with $> 97\%$ accuracy. Overall, an 18-bit latent suffices to pass the privacy checks with low

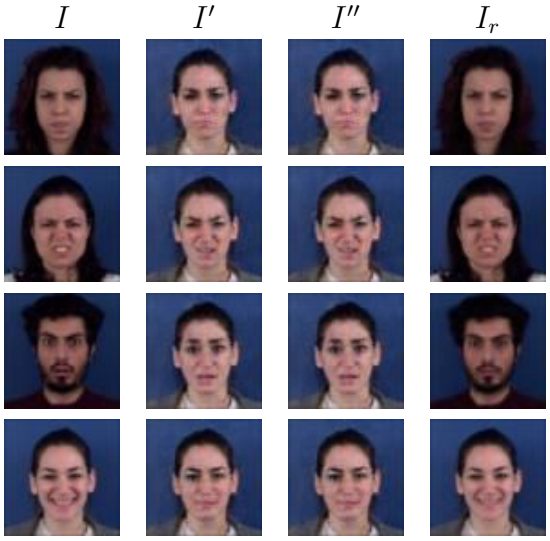


Figure 6.6: Selected backdoored PP-GAN input images I, baseline sanitized images I', outputs I'' with hidden secret, and reconstructed input images I_r (MUG dataset, 18-bit secrets).

reconstruction errors.

6.6 Discussion

Privacy-preserving GANs have been viewed as somewhat of a panacea to the increasing concerns around surveillance technology. Our results indicate that this view might be too optimistic. Given the concerns that our work raises about the rigor of empirical privacy checks, there is a need for better evaluations of privacy. In this section, we discuss further insights following on from our experimental work.

6.6.1 Secret Hiding

More secret data bits can be hidden by the backdoored PP-GAN trained for the MUG dataset compared to that trained for the FERG dataset. This is

Table 6.4: MUG dataset, results after training backdoored PP-GAN architecture

Latent Vector Bit-length	18	24	30	36	42	48	54	60
Strong Check Acc.	0.300	0.318	0.322	0.300	0.328	0.384	0.371	0.365
Latent Vector Recons. Acc.	0.982	0.982	0.981	0.978	0.979	0.978	0.980	0.979
Image Recons. Error	6.00e-4	4.60e-4	4.22e-4	4.55e-4	3.82e-4	3.77e-4	3.36e-4	3.60e-4

because real human face data provides more texture variations in pixel-space, which is also reflected in the output sanitized images, and this likely introduces more "distractions" for the privacy check discriminators. The fact that deep learning (DL)-based discriminators are sensitive to such distractions gives us further skepticism about their use as privacy checks.

Embedding more bits in Scheme 2 should help to reduce the image reconstruction error, but this is counterbalanced by increasing the difficulty of learning to perform secret embedding. If the secret embedding stage is imperfect, there is a higher latent vector recovery error and this results in better subversion of the privacy check, as in the case where we hide a 60-bit latent (Table 6.4).

6.6.2 DNN-based Steganography

Given our goal of having the backdoored PP-GAN implemented entirely as a NN, one possible solution is to attach a DNN-based steganography tool [49, 143, 144] to the benign PP-GAN. The hiding network takes as inputs a cover image and a secret and outputs an image with the secret message hidden. The goal of these DNN-based steganography approaches is to produce secret-embedded images that are indistinguishable from cover images with respect to the probability that they contain a secret (as measured empirically by a DL-based discriminator). An accompanying reveal network extracts the secret message from the secret-embedded image. However, this approach is insufficient for our adversarial setting as the resultant network will not pass the strong privacy check. As long as there is a DL-based reveal network for secret extraction, we surmise that it is possible to train a DL-based discriminator to classify sanitized images into classes of sensitive attributes.

6.6.3 Threats to Validity

The privacy check measures privacy leakage via classification accuracy of a DL-based discriminator on sensitive attributes. Such network's accuracy is affected by various factors, including the size of the training dataset, network architectures, optimization techniques, weights initialization, and training epochs. Thus, the privacy check measurement reported in this paper is only representative of our experimental settings. We would expect different accuracy obtained for a larger dataset, different network architectures, or even more training epochs, but this again points to the unreliability of empirical privacy checks.

6.7 Related Work

Conventional privacy-preserving techniques anonymize the sensitive attributes of structured, low-dimensional and static datasets, such as *k-anonymity* [106] and *l-diversity* [77]. *Differential privacy* [32] was proposed as a more formal privacy guarantee and can be applied to continuous and high-dimensional attributes. However, these approaches provide guarantees only when the relationship between sensitive attributes and data samples can be precisely characterized.

For applications with high-dimensional data, non-sensitive and sensitive attributes intertwine in distributions without a relation model that can be precisely extracted. Hence, empirical and task-dependent privacy checks are used to provide a holistic measure of privacy. Recent work leverages adversarial networks to sanitize input images and adopts similar DL-based discriminators for privacy examination [21, 34, 36, 80, 90, 92, 115, 126, 128]. These works employ adversarial training to jointly optimize both privacy and utility objectives. [34] and [92] perform simple

sanitizing tasks such as removing the QR code from a CIFAR-10 image, or removing the text in a face image, where the sensitive attributes in these cases are artificial and implicit to learn. [90] learn the privacy-preserving encodings via a similar approach but without requiring the sanitized output to be realistic looking. Similarly, Wu et al. aim to generate degraded versions of the input image to sanitize sensitive attributes. The idea of adversarial learning was introduced by [98], and motivated GANs as proposed by [41].

Our work, to the best of our knowledge, was the first attempt to backdoor a PP-GAN that enables secret hiding as a hidden functionality of the network in privacy-preserving settings. Given increasing privacy concerns with data sharing, we believe that our work provides timely insights on the implications of such approaches to privacy and more work on the rigor of privacy checks merits further exploration.

6.8 Summary

Privacy leakage of sanitized images produced by PP-GANs is usually measured empirically using DL-based privacy check discriminators. To illustrate the potential shortcomings of such checks, we produced a backdoored PP-GAN that appeared to remove sensitive attributes while maintaining the utility of the sanitized data for a given application. While our backdoored PP-GAN passed all privacy checks, it actually hid secret data pertaining to the sensitive attributes, even allowing for reconstruction of the original private image. Our experimental results highlighted the insufficiency of existing DL-based privacy checks, and the potential risks of using untrusted third-party PP-GAN tools.

6.9 Supplemental details

6.9.1 Merging Process for Backdoored PP-GAN NN Implementation

We transform the two parallel paths of the architecture in Figure 6.3 into the proposed backdoored PP-GAN architecture in Figure 6.4 through a merging process. Specifically, we merge the secret generation network and the decoding-DCT conversion block into one merged network, made possible because they have identical architectures. We merge one FC layer using a shared input for both paths, followed by the convolutional layers in both paths. This shared input is the concatenation of FC_0 output, encoder output z, and ID code c. Since fully connected layers and convolutional layers are both linear operations, simple transformation of the weights and biases of layers from both paths will form the merged network that produces outputs with doubled dimensions along the last channel for each layer, i.e., the first half of channels in the merged output is the output from the secret generation network at the corresponding layer, and the second half of channels in the merged output comes from the decoder and DCT conversion block in Figure 6.3.

Here we describe how we transform the weights and biases in both paths into the merged network. For the FC layer with the shared input concatenate$(L(\text{FC_0}), z, c)$ in the secret generation network S and decoder-DCT conversion block T, the merged output dimension doubles from 2048 as in S and T to 4096 in the merged network M, followed by a reshaping layer to generate output with dimension $4 \times 4 \times (128 \times 2)$, which is the input to the following convolutional layers. We refer to the weights of the FC layer in S, T, and M respectively as w_s, w_t, and w_m, and biases as b_s, b_t, and b_m. Here we transform the FC layer weights and biases from S

and T to M as in the following:

begin

for $j = 0$ to 15 do

$w_m[:, 256 * j : 256 * j + 128]$

$= w_s[:, 128 * j : 128 * j + 128]$

$b_m[256 * j : 256 * j + 128]$

$= b_s[128 * j : 128 * j + 128]$

$w_m[:, 256 * j + 128 : 256 * j + 256] = \text{concatenate}$

$([\text{zeros}((K, 128)), w_t[:, 128 * j : 128 * j + 128]])$

$b_m[256 * j + 128 : 256 * j + 256]$

$= b_t[128 * j : 128 * j + 128]$

end

Here, K is the output dimension of FC_0 (the number of secret bits).

For a specific convolutional layer i in S or T, its output L_i^S or L_i^T has the shape $d_i \times d_i \times c_i$, where c_i is the number of output channels of layer i, and d_i is the size of the feature maps. The corresponding layer i in the merged network M has output L_i of shape $d_i \times d_i \times (2 \times c_i)$, and the output L_i has $2 \times c_i$ channels. To generate such output, the dimensions of the weights at convolutional layer i expands from $f_i \times f_i \times c_i \times c_{i-1}$ as in layer i in S and T, to dimensions of $f_i \times f_i \times (2 \times c_i) \times (2 \times c_{i-1})$ in M. Here, f_i is the size of convolutional filters at layer i. Accordingly, the bias dimension expands from c_i to $2 \times c_i$, from layer i in S and T to M. We denote the weights of convolutional layer i in S, T, and M respectively as w_i^S, w_i^T, and w_i^M, and the biases as b_i^S, b_i^T, and b_i^M. We have the following relationship between convolutional layer weights and biases in networks S, T, and M to generate the

stacked outputs:

$$\begin{cases} w_i^M[:,:,0:c_i,0:c_{i-1}] = w_i^S \\ w_i^M[:,:,0:c_i,c_{i-1}:2\times c_{i-1}] = 0 \\ w_i^M[:,:,c_i:2\times c_i,0:c_{i-1}] = 0 \\ w_i^M[:,:,c_i:2\times c_i,c_{i-1}:2\times c_{i-1}] = w_i^T \\ b_i^M[0:c_i] = b_i^S \\ b_i^M[c_i:2\times c_i] = b_i^T \end{cases} \tag{6.1}$$

The final output of the merged network has six channels, the first three channels are the learned secret matrix, and the last three channels are the DCT coefficients of the sanitized image I'.

6.9.2 Steghide Usage with Command Line

For our illustrative "straw man" solution for hiding secrets in the "sanitized" images, we use a conventional steganography tool, Steghide (http://steghide.sourceforge.net). For the experiments in Section 6.5, we use the following command line options to embed and recover the private information.

Secret embedding:

```
$ steghide embed -ef secret -K -N -p "" \
  -cf cover_image -sf stego_image -v -e none
```

Secret recovery:

```
$ steghide extract -sf steg_image -p "" -xf secret
```

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Security is an ongoing battle in almost every computer system. In deep learning (DL) systems, there can be many different potential attack vectors and many different attack targets. Every application has its own set of assets and domain constraints posed on safeguarding solutions. As a result, there is a need to better understand different security vulnerabilities to match mitigation techniques with specific application needs.

In this dissertation, we study several intersections between the areas of DL, computer-aided design, and security, with specific focuses on the security, privacy, and robustness of DL used in various application domains. We provide case studies of backdooring attacks on deep neural networks (DNNs) used in image classification, lithographic hotspot detection, and privacy protection. Specifically, we examine and propose defense mechanisms to defeat backdooring attacks against DNN classifiers.

Specifically, we study backdooring attacks in Chapter 3 on DNN face, speech,

and traffic sign recognition systems in an adversarial setting of untrustworthy outsourced training. A malicious third-party cloud provider trains the DNNs with a dirty-label poisoned dataset and returns a backdoored neural network (NN) with hidden misbehavior that can control the NN's output prediction with a backdoor trigger. We propose fine-pruning as an effective defense against such backdooring attacks through a combination of pruning and fine-tuning strategies. We show that our fine-pruning defense can successfully nullify backdoors in cases of both baseline and more adaptive pruning-aware attacks that are conducted against DNN image classification systems.

We further investigate backdooring attacks on DNNs used in the context of lithographic hotspot detection in CAD, where training data poisoning is much more constrained than the general imaginary analysis, as domain features strictly bound backdoor trigger insertion. In Chapter 4, we explore the implications of backdooring attacks on DL-based lithographic hotspot detectors under the threat of a malicious physical design insider who aims to sabotage the design process. The feasibility and efficacy of clean-label training data poisoning are studied in DL-based lithographic hotspot detection through various attack dimensions. We show that backdooring attacks are feasible against different network architectures and trigger shapes, even under application constraints, achieving $\sim 100\%$ attack success with low levels of data poisoning.

To address the challenges and threats posed by training data poisoning in lithographic hotspot detection, we are motivated to design countermeasures against such attacks. However, as we show in Chapter 5, prior defense solutions against backdooring attacks in general image classification do not apply or are barely effective in the specific CAD domain. We propose a trigger-oblivious, defensive data

augmentation scheme to proactively defend against such backdooring attacks. We defensively augment the training data to produce cross-label samples and reduce the bias associated with one class in the dataset inserted by the backdoor trigger, given the threat of a malicious physical design insider. Experimental evaluation illustrates that our defense can significantly reduce the attack success from 84% to $\sim 0\%$.

The last piece of the research work we present is on exploring backdooring attacks against generative DNNs in a privacy-preserving setting. As all the prior literature focuses on backdooring attacks and defenses on discriminative DNNs, we take the first step, in Chapter 6, to investigate the implications of backdooring attacks in a generative model. We exemplified our work through a backdoored privacy-preserving generative adversarial network (PP-GAN) design whose intended task was to remove the secret information in the private input image and generate a sanitized output image. In our backdoored PP-GAN design, the attacker leaves a backdoor for secret extraction from the output, even though the output is privacy-check certified. Through adversarial architecture design and a manipulated training process, our backdoored PP-GAN design can successfully hide secrets in the output images with 100% recovery, which raises fundamental questions about the rigor of DL-based privacy checks.

7.2 Future Work

Research work presented in this dissertation reveals several opportunities for future work. Some potential research directions could include the following:

- **Adaptive backdooring attacks circumventing fine-pruning and po-**

tential mitigation techniques: Fine-pruning is not the last word in DNN backdooring attacks and defenses. Essentially, we can think of fine-tuning as a continuation of the normal training procedure from a set of parameter initializations Θ_i. In an adversarial context, Θ_i is determined by the attacker. Hence, if an attacker hopes to preserve their attack against our fine-pruning defense, they must provide a Θ_i with a nearby local minimum (in terms of the loss surface with respect to clean inputs) that still contains their backdoor. We do not currently have a strong guarantee that such a Θ_i *cannot* be found; however, we note that a stronger (though more computationally expensive) version of fine-pruning could add some noise to the parameters before fine-tuning. In the limit, there must exist some amount of noise that would cause the network to "forget" the backdoor, since adding sufficiently large amounts of noise would be equivalent to retraining the network from scratch with random initialization. We believe the question of how much noise is needed would be an interesting area for future research.

- **Additional attack dimensions on training data poisoning in lithographic hotspot detection:** Our evaluation of backdooring attacks on DL-based lithographic hotspot detection focuses on position-invariant triggers; potential attack dimensions could include varying sizes and locations. We are interested to see how backdoor trigger sizes affect training data poisoning. Intuitively larger malicious artifacts are easier to be learned and picked up by an NN, thus requiring fewer training samples to poison and to inject a backdoor. The potential impact of the locations of backdoor triggers also merits future exploration.

- **Steganalysis and other privacy checks on backdoored PP-GAN:** As our backdoored PP-GAN design is inspired by frequency domain steganography tools (e.g., Steghide) and uses NNs to implement several key steganography operations, an immediate next step could be applying steganalysis tools to detect our secret hidden output images. Such steganalysis tools are typically used to detect messages hidden using conventional steganography. Though our backdoored PP-GAN design hides secret messages that are as small as a few bits, we cannot rule out the possibility that our backdoored PP-GAN outputs will be picked up by steganalysis. The second line of future exploration could be investigating other empirical privacy checks on our backdoored PP-GAN, such as mutual information neural estimation (MINE) [13]. In MINE, privacy leakage is measured as the mutual information between the output and secrets. This mutual information estimator is also empirical and based on NNs.

Bibliography

[1] FreePDK45:Contents - NCSU EDA Wiki.

[2] 1st ACM/IEEE Workshop on Machine Learning for CAD (MLCAD), Sept. 3-4, 2019.

[3] A. Adadi and M. Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[4] N. Aifanti, C. Papachristou, and A. Delopoulos. The MUG facial expression database. In *11th International Workshop on Image Analysis for Multimedia Interactive Services WIAMIS 10*, pages 1–4. IEEE, 2010.

[5] Amazon.com, Inc. Deep Learning AMI Amazon Linux Version.

[6] D. Aneja, A. Colburn, G. Faigin, L. Shapiro, and B. Mones. Modeling stylized character expressions via deep learning. In *Asian Conference on Computer Vision*, pages 136–153. Springer, 2016.

[7] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3), 2017.

[8] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[9] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, July 2018.

[10] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[11] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proc. 2006 ACM Symp. Information, computer and communications security (CCS)*, ASIACCS '06, pages 16–25, Taipei, Taiwan, Mar. 2006. Association for Computing Machinery.

[12] K. Basu, S. M. Saeed, C. Pilato, M. Ashraf, M. T. Nabeel, K. Chakrabarty, and R. Karri. CAD-Base: An attack vector into the electronics supply chain. *ACM Trans. Des. Autom. Electron. Syst.*, 24(4):38:1–38:30, Apr. 2019.

[13] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm. Mutual information neural estimation. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 531–540. PMLR, 10–15 Jul 2018.

[14] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, Dec. 2018.

[15] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Advances in Neural Information Processing Systems*, pages 494–501, 1989.

[16] V. Borisov and J. Scheible. Research on data augmentation for lithography hotspot detection using deep learning. In *34th European Mask and Lithography Conf.*, volume 10775, page 107751A. International Society for Optics and Photonics, 2018.

[17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[18] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[19] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[20] N. Carlini and D. A. Wagner. Defensive distillation is not robust to adversarial examples. *CoRR*, abs/1607.04311, 2016.

[21] J. Chen, J. Konrad, and P. Ishwar. VGAN-based image representation learning for privacy-preserving facial expression recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1651–1660, 2018.

[22] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *ArXiv e-prints*, Dec. 2017.

[23] Y. Chen, Y. Lin, T. Gai, Y. Su, Y. Wei, and D. Z. Pan. Semi-supervised hotspot detection with self-paced multi-task learning. In *Asia and South Pacific Design Automation Conference*, pages 420–425. ACM, 2019.

[24] F. Chollet et al. Keras. https://keras.io, 2015.

[25] S. P. Chung and A. K. Mok. Allergy attack against automatic signature generation. In *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, 2006.

[26] S. P. Chung and A. K. Mok. Advanced allergy attacks: Does a corpus really help. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, 2007.

[27] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[28] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[30] G. S. Dhillon, K. Azizzadenesheli, J. D. Bernstein, J. Kossaifi, A. Khanna, Z. C. Lipton, and A. Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018.

[31] M. Du, N. Liu, and X. Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1):68–77, Dec. 2019.

[32] C. Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[33] Economist. From not working to neural networking, 2016.

[34] H. Edwards and A. J. Storkey. Censoring representations with an adversary. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[35] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, June 2018.

[36] C. Feutry, P. Piantanida, and P. Duhamel. Learning semi-supervised anonymized representations by mutual information. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3467–3471, May 2020. ISSN: 2379-190X.

[37] P. Fogla and W. Lee. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 59–68, New York, NY, USA, 2006. ACM.

[38] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[39] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the Annual Computer Security Applications Conference*, 2019.

[40] H. Geng, H. Yang, Y. Ma, J. Mitra, and B. Yu. SRAF insertion via supervised dictionary learning. In *Asia and South Pacific Design Automation Conference - ASP-DAC '19*, pages 406–411, New York, NY, USA, 2019. ACM.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[42] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[43] Google, Inc. Google cloud machine learning engine. https://cloud.google.com/ml-engine/.

[44] M. Graphics. Calibre LFD, 2019.

[45] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[46] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[47] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. BadNets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[48] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

[49] J. Hayes and G. Danezis. Generating steganographic images via adversarial training. In *Advances in Neural Information Processing Systems*, pages 1954–1963, 2017.

[50] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[51] W. He, J. Wei, X. Chen, N. Carlini, and D. Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX Association.

[52] X. He, Y. Deng, S. Zhou, R. Li, Y. Wang, and Y. Guo. Lithography hotspot detection with FFT-based feature extraction and imbalanced learning rate. *ACM Transactions on Design Automation of Electronic Systems*, 25(2):1–21, Mar. 2020.

[53] K. M. Hermann and P. Blunsom. Multilingual distributed representations without word alignment. In *International Conference on Learning Representations*, Apr. 2014.

[54] K. Hill. The secretive company that might end privacy as we know it. *New York Times*, Jan. 2020.

[55] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[56] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[57] Y. Huang, Z. Xie, G. Fang, T. Yu, H. Ren, S. Fang, Y. Chen, and J. Hu. Routability-driven macro placement with embedded CNN-based prediction model. In *Design, Automation, and Test in Europe Conference*, pages 180–185, Mar. 2019.

[58] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2592–2600, 2016.

[59] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng. Efficient layout hotspot detection via binarized residual neural network. In *Proc. Design Automation Conf. (DAC)*, pages 1–6, 2019.

[60] A. B. Kahng. Machine learning applications in physical design: Recent results and directions. In *Int. Symp. Physical Design (ISPD)*, pages 68–73, Monterey, CA, 2018. ACM.

[61] C. Karlberger, G. Bayler, C. Kruegel, and E. Kirda. Exploiting redundancy in natural language to penetrate bayesian spam filters. In *Proceedings of the First USENIX Workshop on Offensive Technologies*, WOOT '07, pages 9:1–9:7, Berkeley, CA, USA, 2007. USENIX Association.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[63] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *Proceedings of The International Conference on Learning Representations*, 2017.

[64] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[65] C. Li. OpenAI's GPT-3 language model: A technical overview, 2020.

[66] H. Li et al. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[67] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea. Robust linear regression against training data poisoning. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 91–102. ACM, 2017.

[68] K. Liu, B. Dolan-Gavitt, and S. Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *Research in Attacks, Intrusions, and Defenses*, Lecture Notes in Computer Science, pages 273–294. Springer International Publishing, 2018.

[69] K. Liu, B. Tan, R. Karri, and S. Garg. Poisoning the (data) well in ML-based CAD: A case study of hiding lithographic hotspots. In *Proc. Design, Automation Test in Europe Conf. Exhibition (DATE)*, pages 306–309, 2020.

[70] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Young, R. Karri, and S. Garg. Adversarial perturbation attacks on ML-based CAD: A case study on CNN-based lithographic hotspot detection. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(5):1–31, 2020.

[71] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019.

[72] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *25nd Annual Network and Dis-*

tributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018. The Internet Society, 2018.

[73] Y. Liu, Y. Xie, and A. Srivastava. Neural trojans. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 45–48. IEEE, 2017.

[74] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 641–647, New York, NY, USA, 2005. ACM.

[75] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2005.

[76] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[77] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[78] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *Proceedings of The International Conference on Learning Representations*, 2018.

[79] T. Matsunawa, J.-R. Gao, B. Yu, and D. Z. Pan. A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction. In *Design-Process-Technology Co-optimization for Manufacturability IX*, volume 9427, page 94270S. International Society for Optics and Photonics, 2015.

[80] M. Maximov, I. Elezi, and L. Leal-Taixé. CIAGAN: Conditional identity anonymization generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5447–5456, 2020.

[81] Microsoft Corp. Azure batch AI training. https://batchaitraining.azure.com/.

[82] A. Møgelmose, D. Liu, and M. M. Trivedi. Traffic sign detection for U.S. roads: Remaining challenges and a case for tracking. In *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*, pages 1394–1399. IEEE, 2014.

[83] P. Molchanov et al. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[84] S. K. Moore. DARPA picks its first set of winners in electronics resurgence initiative, July 2018.

[85] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1765–1773, 2017.

[86] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *International Conference on International Conference on Machine Learning - ICML '10*, pages 807–814, USA, 2010. Omnipress.

[87] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert

your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, pages 7:1–7:9, Berkeley, CA, USA, 2008. USENIX Association.

[88] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*, pages 81–105, Berlin, Heidelberg, 2006. Springer-Verlag.

[89] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016.

[90] F. Pittaluga, S. Koppal, and A. Chakrabarti. Learning privacy preserving encodings through adversarial training. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 791–799. IEEE, 2019.

[91] X. Qiao, Y. Yang, and H. Li. Defending neural backdoors via generative distribution modeling. In *Advances in Neural Information Processing Systems 32*, pages 14004–14013. Curran Associates, Inc., 2019.

[92] N. Raval, A. Machanavajjhala, and L. P. Cox. Protecting visual secrets using adversarial nets. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1329–1332. IEEE, 2017.

[93] G. R. Reddy, K. Madkour, and Y. Makris. Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders. In *IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Westminster, CO, USA, Nov. 2019. IEEE.

[94] G. R. Reddy, C. Xanthopoulos, and Y. Makris. Enhanced hotspot detection through synthetic pattern generation and design of experiments. In *IEEE VLSI Test Symp. (VTS)*, pages 1–6. IEEE, Apr. 2018.

[95] G. R. Reddy, C. Xanthopoulos, and Y. Makris. On improving hotspot detection through synthetic pattern-based database enhancement. *arXiv*, 2007.05879, 2020.

[96] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[97] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[98] J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992.

[99] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! Targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113. Curran Associates, Inc., 2018.

[100] H. Sheisi, J. Mesgarian, and M. Rahmani. Steganography: DCT coefficient replacement method and compare with JSteg algorithm. *International Journal of Computer and Electrical Engineering*, 4(4):458–462, 2012.

[101] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, Dec. 2019.

[102] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[103] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[104] O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras. When does machine learning FAIL? Generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, 2018. USENIX Association.

[105] Y. Sun, X. Wang, and X. Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898, 2014.

[106] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[107] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[108] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proc. 2nd Int. Conf. Learning Representations (ICLR)*, 2014.

[109] A. F. Tabrizi, N. K. Darav, L. Rakai, I. Bustany, A. Kennings, and L. Behjat. Eh?Predictor: A deep learning framework to identify detailed routing short violations from a placed netlist. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, pages 1–1, 2019.

[110] K. M. C. Tan, K. S. Killourhy, and R. A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the 5th International Conference on Recent Advances in Intrusion Detection*, RAID'02, pages 54–73, Berlin, Heidelberg, 2002. Springer-Verlag.

[111] O. Tene, K. Evans, B. Gencarelli, G. Maldoff, and G. Zanfir-Fortuna. GDPR at year one: Enter the designers and engineers. *IEEE Security & Privacy*, 17(6):7–9, Nov. 2019.

[112] J. A. Torres. ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite. In *IEEE/ACM International Conference on CAD*, pages 349–350, Nov 2012.

[113] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.

[114] B. Tran, J. Li, and A. Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems 31: Annual Conference*

on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 8011–8021, 2018.

[115] B.-W. Tseng and P.-Y. Wu. Compressive privacy generative adversarial network. *IEEE Transactions on Information Forensics and Security*, 15:2499–2513, 2020.

[116] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. There is no free lunch in adversarial robustness (but there are unexpected benefits). In *Proceedings of The International Conference on Learning Representations*, 2019.

[117] F. Tung, S. Muralidharan, and G. Mori. Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization. In *British Machine Vision Conference (BMVC)*, 2017.

[118] A. K. Veldanda, K. Liu, B. Tan, P. Krishnamurthy, F. Khorrami, R. Karri, B. Dolan-Gavitt, and S. Garg. NNoculation: Broad spectrum and targeted treatment of backdoored DNNs. *CoRR*, 2020.

[119] Y. Vorobeychik and M. Kantarcioglu. Adversarial machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–169, Aug. 2018.

[120] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, New York, NY, USA, 2002. ACM.

[121] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural Cleanse: Identifying and mitigating backdoor attacks in neural

networks. In *IEEE Symp. Security and Privacy (SP)*, pages 707–723, May 2019.

[122] S. Wang, X. Wang, P. Zhao, W. Wen, D. Kaeli, P. Chin, and X. Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. In *Proceedings of the International Conference on Computer-Aided Design*, pages 1–8, 2018.

[123] G. L. Wittel and S. F. Wu. On attacking statistical spam filters. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2004.

[124] L. Wolf, T. Hassner, and I. Maoz. Face recognition in unconstrained videos with matched background similarity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2011.

[125] Y. Wu, F. Yang, Y. Xu, and H. Ling. Privacy-protective-GAN for privacy preserving face de-identification. *Journal of Computer Science and Technology*, 34(1):47–60, Jan. 2019.

[126] Z. Wu, Z. Wang, Z. Wang, and H. Jin. Towards privacy-preserving visual recognition via adversarial training: A pilot study. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 606–624, 2018.

[127] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and N. Corporation. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, pages 80:1–80:8, 2018.

[128] Z. Xiong, W. Li, Q. Han, and Z. Cai. Privacy-preserving auto-driving: A GAN-based approach to protect vehicular camera data. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 668–677, Nov. 2019. ISSN: 2374-8486.

[129] X. Xu, T. Matsunawa, S. Nojima, C. Kodama, T. Kotani, and D. Z. Pan. A machine learning based framework for sub-resolution assist feature generation. In *Proceedings of the 2016 on International Symposium on Physical Design*, pages 161–168. ACM, 2016.

[130] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *Design Automation Conference - DAC '18*, pages 1–6. IEEE, June 2018.

[131] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu. Imbalance aware lithography hotspot detection: A deep learning approach. In *SPIE Design-Process-Technology Co-optimization for Manufacturability*, page 1014807, 2017.

[132] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu. Hotspot detection using squish-net. In *Design-Process-Technology Co-optimization for Manufacturability XIII*, volume 10962, page 109620S. International Society for Optics and Photonics, 2019.

[133] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young. Layout hotspot detection with feature tensor generation and deep biased learning. *IEEE Transactions on CAD*, pages 1–13, 2018.

[134] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young. Layout hotspot detection

with feature tensor generation and deep biased learning. In *Design Automation Conference 2017 - DAC '17*, pages 1–6, Austin, TX, USA, 2017. ACM.

[135] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan. LithoGAN: End-to-end lithography modeling with generative adversarial networks. In *Proc. 56th Annu. Design Automation Conf. (DAC)*, pages 1–6, 2019.

[136] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

[137] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[138] C. Yu, H. Xiao, and G. De Micheli. Developing synthesis flows without human knowledge. In *Design Automation Conference*, pages 1–6, San Francisco, California, 2018. ACM Press.

[139] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560. ACM, 2017.

[140] Y.-T. Yu, Y.-C. Chan, S. Sinha, I. H.-R. Jiang, and C. Chiang. Accurate process-hotspot detection using critical design rule extraction. In *Design Automation Conference*, pages 1167–1172. ACM, 2012.

[141] E. Zennaro, L. Servadei, K. Devarajegowda, and W. Ecker. A machine learning approach for area prediction of hardware designs from abstract

specifications. In *Proc. 21st Euromicro Conf. Digital System Design (DSD)*, 2018.

[142] H. Zhang, B. Yu, and E. F. Young. Enabling online learning in lithography hotspot detection with information-theoretic feature optimization. In *Proceedings of the 35th International Conference on Computer-Aided Design*, pages 1–8, 2016.

[143] K. A. Zhang, A. Cuesta-Infante, L. Xu, and K. Veeramachaneni. SteganoGAN: High capacity image steganography with GANs. *arXiv preprint arXiv:1901.03892*, 2019.

[144] J. Zhu, R. Kaplan, J. Johnson, and L. Fei-Fei. Hidden: Hiding data with deep networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 657–672, 2018.

[145] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.