Improving Robustness of Machine Learning Models using Domain Knowledge

---

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment
of the requirements for the degree

Doctor of Philosophy

by

Weilin Xu

May 2019

Abstract

Although machine learning techniques have achieved great success in many areas, such as computer vision, natural language processing, and computer security, recent studies have shown that they are not robust under attack. A motivated adversary is often able to craft input samples that force a machine learning model to produce incorrect predictions, even if the target model achieves high accuracy on normal test inputs. This raises great concern when machine learning models are deployed for security-sensitive tasks.

This dissertation aims to improve the robustness of machine learning models by exploiting domain knowledge. While domain knowledge has often been neglected due to the power of automatic representation learning in the deep learning era, we find that domain knowledge goes beyond a given dataset of a task and helps to (1) uncover weaknesses of machine learning models, (2) detect adversarial examples and (3) improve the robustness of machine learning models.

First, we design an evolutionary algorithm-based framework, *Genetic Evasion*, to find evasive samples. We embed domain knowledge into the mutation operator and the fitness function of the framework and achieve 100% success rate in evading two state-of-the-art PDF malware classifiers. Unlike previous methods, our technique uses genetic programming to directly generate evasive samples in the problem space instead of the feature space, making it a practical attack that breaks the trust of black-box machine learning models in a security application.

Second, we design an ensemble framework, *Feature Squeezing*, to detect adversarial examples against deep neural network models using simple pre-processing. We employ domain knowledge on signal processing that natural signals are often redundant for many perception tasks. Therefore, we can squeeze the input features to reduce adversaries' search space while preserving the accuracy on normal inputs. We use various squeezers to pre-process an input example before it is fed into a

model. The difference between those predictions is often small for normal inputs due to redundancy, while the difference can be large for adversarial examples. We demonstrate that *Feature Squeezing* is empirically effective and inexpensive in detecting adversarial examples for image classification tasks generated by many algorithms.

Third, we incorporate simple pre-processing with certifiable robust training and formal verification to train provably-robust models. We formally analyze the impact of pre-processing on adversarial strength and derive novel methods to improve model robustness. Our approach produces accurate models with verified state-of-the-art robustness and advances the state-of-the-art of certifiable robust training methods.

We demonstrate that domain knowledge helps us understand and improve the robustness of machine learning models. Our results have motivated several subsequent works, and we hope this dissertation will be a step towards implementing robust models under attack.

APPROVAL SHEET

This Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author Signature: 许伟林

This Dissertation has been read and approved by the examining committee:

Advisor: David Evans and Yanjun Qi

Committee Member: Vicente Ordóñez Román

Committee Member: Homa Alemzadeh

Committee Member: Patrick McDaniel

Committee Member: _____

Committee Member: _____

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2019

To my amazing wife Aihua Chen, Ph.T. (Putting Hubby Through).

Acknowledgements

I have the incredible good fortune to work with Prof. David Evans and Prof. Yanjun Qi in the past six years. They are the rare advisors who have the patience and ability to direct students to explore a new research field without the pressure of publishing papers or worry of insufficient fund. I also have astoundingly supportive committees comprised of Prof. Homa Alemzadeh, Prof. Patrick McDaniel, Prof. Vicente Ordóñez Román, Prof. Hongning Wang and Prof. Westley Weimer. I'm grateful for their inspirational early feedback that helps to shape this dissertation. Also, I would like to thank all the collaborators, especially Beilun Wang and Xiao Zhang for sharing their expertise in mathematics.

I wouldn't have an opportunity to pursue a Ph.D. degree without the guidance and encouragement from my former mentors: Dr. Shuo Chen, Dr. David Fifield, Prof. Hua Zhang and Prof. Jianwei Zhuge. I would also like to thank many pre-university teachers. I could have a rocky road to be a first-generation college student and first-generation Ph.D. without their kindness and beyond-duty support.

I would like to thank all the family members and relatives for their love. There is no doubt that I have the best grandfather in the world, Rongxiang Xu, who homeschooled me in the early years even though he only had two years in school himself. I feel lucky to have visionary cousins, Shaoxing Xie and Zhaosen Xie who provided me a precious computer at the time when it was unaffordable to most families. I'm grateful to have an elder twin brother Guanglin Xu as a companion whom I could always count on in the world adventure. Finally, my deepest gratitude goes to my beloved wife, Aihua Chen, who sacrifices her career in China to my pursuit of scientific research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning techniques have been widely used in many fields, such as perception and computer security. Thanks to their capability of learning from data, machine learning models often outperform hand-crafted systems designed by domain experts in those fields. For example, Convolutional Neural Network (CNN)-based models [58] have dominated image classification tasks since AlexNet [54] in 2012. These models do not use any hand-crafted features from the computer vision field, but rely only on raw image pixels as input. As another example, a group of data scientists with no experience in malware analysis won the Microsoft Malware Classification Challenge in 2015 [90]. Machine learning has become so effective that it seems like a silver bullet obviating the need for any domain knowledge.

However, it is important to realize that these results are for particular test datasets. Recent studies have shown that machine learning models are not robust against adversaries. A motivated adversary may be able to craft input that leads a target model to produce incorrect predictions.

1.1 Motivating Examples

The power of adversaries raises a huge concern when machine learning techniques are used in security-sensitive tasks. Such attacking methods are feasible, and the resources are readily available for an

adversary to carry out these attacks in the real world.

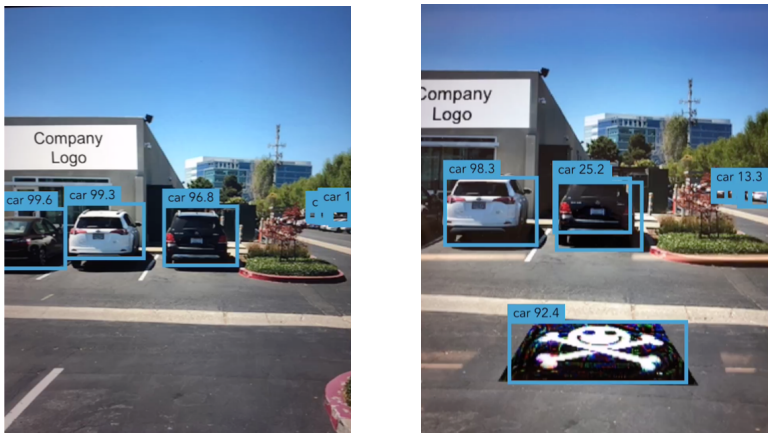
Malware Detection. Malware is developed by motivated adversaries to control computer systems without their owners' consent, often causing financial losses to the target. Traditional anti-virus software heavily relies on signatures to detect malware samples. As adversaries steadily learned ways to bypass signature-based detection, researchers have advocated for using machine learning in malware detection. Machine learning-based methods typically learn complex decision rules to detect malware and achieve near-perfect accuracy in testing [100, 104]. But machine learning is not immune to adaptive adversaries. Researchers have demonstrated that an adversary can evade a machine learning-based malware detector by injecting dummy bytes to interfere with feature extraction [105]. We demonstrate a general attack algorithm against machine learning models in Chapter 3, and show its effectiveness in evading PDF malware classifiers.

Face Recognition. Face recognition is a computer vision task that is often security-sensitive. It is widely deployed for biometric identification and access control, while the operators assume that it duplicates human vision to recognize people. Recently, deep learning has become the state-of-the-art method for face recognition since it scales to huge datasets and achieves human-level accuracy [87, 93, 108]. However, researchers have found that an attacker can fool a face recognition model in the physical world by wearing glasses with a carefully designed frame [98].

Healthcare. Machine learning has drawn a lot of attention from researchers in the healthcare field because of its huge potential for assisted diagnosis and resource allocation. Finlayson et al. pointed out that motivated parties in the medical system can leverage adversarial examples to gain extra benefits [34]. For example, while health insurance companies use machine learning models to evaluate claims from healthcare providers, a provider may submit adversarial examples to get unexpected approvals. In addition, as the administrative agencies have planned to use machine learning to assist new medicine approval, a pharmaceutical company may use adversarial examples to influence the administrative decision.

Autonomous Vehicles. Modern autonomous vehicles heavily rely on computer vision techniques to perceive the surrounding environment and predict the movement of the nearby objects so that they can plan a reasonable route to proceed. An accurate and robust object detection model is crucial to an autonomous vehicle. A false negative, such as ignoring an obstacle, could result in a crash; a

Figure 1.1: An object detection model wrongly recognizes adversarial patch as a car.



(a) Normal image. (b) Adversarial image.

false positive such as wrongly recognizing an obstacle that does not exist could result in a sudden stop or denial-of-service. As shown in Figure 1.1, we have demonstrated that an adversary can fool a state-of-the-art object detection model even if the scene image is captured through a complex video sensing system [124].

1.2 Thesis

We argue that *domain knowledge can be used to improve the robustness of machine learning models.*

Intuitively, the *robustness* of a machine learning model is used to measure how well a model preserves its correct prediction under attack. Given an input example $\boldsymbol{x} \in \mathcal{X}$ that is correctly classified by a model g, we formally define g is ϵ-robust on input $\boldsymbol{x}$ with a distance metric Δ iff

$$\forall \boldsymbol{x}' \in \mathcal{X}, \Delta(\boldsymbol{x}, \boldsymbol{x}') \leq \epsilon \Rightarrow g(\boldsymbol{x}) = g(\boldsymbol{x}') \tag{1.1}$$

Domain knowledge is knowledge of a specific application field, such as vision and malware, in contrast to the general knowledge of machine learning. Domain knowledge has often been neglected in machine learning practices, because deep learning with its powerful automated representation learning outperforms many previous domain-specific systems. However, machine learning models often learn correlative features instead of causative factors in their decision rules. The gap between

the ground-truth decision rules and the approximate ones learned by a machine learning model inevitably creates opportunities for adversaries to fool machine learning models.

We show that we can use domain knowledge to (1) help understand the robustness of machine learning models by finding evasive examples, (2) defend non-robust machine learning models by detecting adversarial examples, (3) and improve the provable robustness of machine learning models against restricted adversaries.

1.3 Contributions

This dissertation makes several contributions to understand and improve the robustness of machine learning models:

1. The *Genetic Evasion* framework that embeds domain knowledge in *genetic programming* to automatically find evasive examples against machine learning models [Chapter 3]. We implemented an instance of the framework to attack PDF malware classifiers and evaded two state-of-the-art models with 100% success rate. Our results reveal the surprising decision rules that are learned by machine learning models and provide insights why those models are not robust.

2. The *Feature Squeezing* framework that uses domain knowledge of signal processing to detect adversarial examples against deep learning-based perception models [Chapter 4]. We designed an ensemble framework to pre-process inputs with different squeezers and compare the discrepancy between their predictions to detect adversarial examples. We evaluated two squeezers, *bit depth reduction* and *smoothing* on several image classifiers against various attacking algorithms and achieved state-of-the-art detection performance.

3. Provable robustness with simple pre-processing [Chapter 5]. We formally studied the impact of *bit depth reduction* on adversarial capability and derived novel methods to train provably robust models by either transforming input bounds or regularizing model weights. We incorporated our method into formal verification and robust certification. The experimental results show that our method trains accurate models with state-of-the-art verifiable robustness and improves state-of-the-art robust certification methods.

Chapter 2

Background

This chapter briefly introduces machine learning and adversarial machine learning.

2.1 Machine Learning Classifiers

Machine learning learns from and makes predictions on data. A machine learning-based classifier attempts to find a hypothesis function g that maps data points into different classes. For example, a malware classification system would find a hypothesis function g that maps a data point (a piece of malware sample) into either *benign* or *malicious*.

The effort to train a machine learning system starts with feature extraction. As most machine learning algorithms cannot operate on highly-structured data, the data samples are usually represented in a specially-designed feature space. For example, a malware classifier may extract the file size and the function call traces as features. Each feature is a dimension in the feature space; consequently, every sample is represented as a vector. An extra step of feature selection may be performed to reduce the number of features when the number of features is too large for the classification algorithm.

The machine learning algorithms we discuss in this dissertation are all *supervised learning* methods, in which the training dataset comes with labels identifying the class of every training sample. The hypothesis function g is trained to minimize the prediction error on the training set. This function usually results in a low error rate on the operational data under the *stationarity* assumption that the

distribution over data points encountered in the future will be the same as the distribution over the training set.

2.1.1 Neural Networks

An artificial neural network is a framework inspired by biological neural network structure that enables different machine learning algorithms to process complex data together. Even though the idea of neural networks and an effective training algorithm, *backpropagation*, had been invented many decades ago, it did not become popular until we had enormous training data and affordable computational power after 2010.

One major difference between neural networks and other machine learning techniques is that neural networks have the ability to learn feature representation automatically from training data. Convolutional Neural Networks (CNNs), popularized by LeCun et al. [59], perform exceptionally well on image classification [54]. A deep CNN can be written as a function $g : X \to Y$, where X represents the input space and Y is the output space representing a categorical set. For a sample, $\mathbf{x} \in X$, $g(\mathbf{x}) = f_L(f_{L-1}(\dots((f_1(\mathbf{x}))))$. Each f_i represents a layer. The last output layer, f_L, creates the mapping from a hidden space to the output space (class labels) through a softmax function that outputs a vector of real numbers in the range $[0, 1]$ that sums to 1. We can treat the output of the softmax function as the probability distribution of input $\mathbf{x}$ over C different possible output classes.

A training set contains N labeled inputs where the i^{th} input is denoted $(\mathbf{x}_i, y_i)$. When training a deep model, parameters related to each layer are randomly initialized, and input samples, $(\mathbf{x}_i, y_i)$, are fed through the network. The output of this network is a prediction, $g(\mathbf{x}_i)$, associated with the i^{th} sample. To train the DNN, the difference between the prediction output, $g(\mathbf{x}_i)$, and its true label, y_i, usually modeled with a loss function $J(g(\mathbf{x}_i), y_i)$, is pushed backward into the network using a backpropagation algorithm to update DNN parameters.

2.2 Adversarial Machine Learning

Adversarial machine learning is an interdisciplinary research field that concerns the robustness of machine learning models under attack. We categorize existing work in this field into two groups:

poisoning attacks and *evasion attacks*. Poisoning attacks assume that an adversary can interfere with the model training to some extent and aims to influence model behaviors. In contrast, evasion attacks assume that an adversary only has access to a pre-trained model and intends to construct input examples that lead to incorrect predictions that satisfy some adversarial goal. We focus on evasion attacks in this dissertation.

2.2.1 Generating Adversarial Examples

An *adversarial example* is crafted by an adversary with the goal of producing an incorrect output from a target classifier. Since ground truth, at least for image classification tasks, is based on human perception which is hard to model or test, research in adversarial examples typically defines an adversarial example as a misclassified sample $\mathbf{x}'$ generated by perturbing a correctly-classified sample $\mathbf{x}$ (the *seed* example) by some limited amount.

Adversarial examples can be *targeted*, in which case the adversary's goal is for $\mathbf{x}'$ to be classified as a particular class t, or *untargeted*, in which case the adversary's goal is just for $\mathbf{x}'$ to be classified as any class other than its correct class. More formally, given $\mathbf{x} \in X$ and $g(\cdot)$, the goal of an targeted adversary with target $t \in Y$ is to find an $\mathbf{x}' \in X$ such that

$$g(\mathbf{x}') = t \wedge \Delta(\mathbf{x}, \mathbf{x}') \leq \epsilon \tag{2.1}$$

where $\Delta(\mathbf{x}, \mathbf{x}')$ represents the difference between input $\mathbf{x}$ and $\mathbf{x}'$. An untargeted adversary seeks to find an $\mathbf{x}' \in X$ such that

$$g(\mathbf{x}') \neq g(\mathbf{x}) \wedge \Delta(\mathbf{x}, \mathbf{x}') \leq \epsilon. \tag{2.2}$$

The strength of the adversary, ϵ, limits the permissible transformations. The distance metric, $\Delta(\cdot)$, and the adversarial strength threshold, ϵ, are meant to model how close an adversarial example needs to be to the original to "fool" a human observer.

Several techniques have been proposed to find adversarial examples. Szegedy et al. [107] first observed that DNN models are vulnerable to adversarial perturbations and used the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to find adversarial examples. Their study also found that adversarial perturbations generated from one DNN model can also force other DNN models to produce incorrect outputs. Subsequent papers have explored other strategies to generate adversarial

manipulations, including using the linear assumption behind a model [36, 76], saliency maps [83], and evolutionary algorithms [77].

Equations (2.1) and (2.2) suggest two different parameters for categorizing methods for finding adversarial examples: whether they are targeted or untargeted, and the choice of $\Delta(\cdot)$, which is typically an ℓ_p-norm distance metric. Popular adversarial methods use the following three norms for $\Delta(\cdot)$ (here $\mathbf{z} = \mathbf{x} - \mathbf{x}'$):

- ℓ_∞: $||\mathbf{z}||_\infty = \max_i |z_i|$.
  The ℓ_∞ norm measures the maximum change in any dimension. This means an ℓ_∞ attack is limited by the maximum change it can make to each pixel, but can alter all the pixels in the image by up to that amount.

- ℓ_2: $||\mathbf{z}||_2 = \sqrt{\sum_i z_i^2}$.
  The ℓ_2 norm corresponds to the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$. This distance can remain small when small changes are applied to many pixels.

- ℓ_0: $||\mathbf{z}||_0 = \#\{i \,|\, z_i \neq 0\}$.
  For images, this measures the total number of pixels that may be altered between $\mathbf{x}$ and $\mathbf{x}'$, but does not limit the amount of perturbation of each of those pixels.

Next, we discuss the eleven attacking algorithms used in our experiments, grouped by the norm they used for $\Delta(\cdot)$.

Fast Gradient Sign Method: FGSM (ℓ_∞, Untargeted).

Goodfellow et al. hypothesized that DNNs are vulnerable to adversarial perturbations because of their linear nature [36]. They proposed the *fast gradient sign method* (FGSM) for efficiently finding adversarial examples. To control the cost of attacking, FGSM assumes that the attack strength at every feature dimension is the same, essentially measuring the perturbation $\Delta(\mathbf{x}, \mathbf{x}')$ using the ℓ_∞-norm. The strength of perturbation at every dimension is limited by the same constant parameter, ϵ, which is also used as the amount of perturbation.

As an untargeted attack, the perturbation is calculated directly by using gradient vector of a loss function:

$$\epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} J(g(\mathbf{x}), y)) \tag{2.3}$$

Here the loss function, $J(\cdot,\cdot)$, is the loss that have been used for training the specific DNN model, and y is the correct label for $\mathbf{x}$. Equation (2.3) essentially increases the loss $J(\cdot,\cdot)$ by perturbing the input $\mathbf{x}$ based on a transformed gradient.

Basic Iterative Method: BIM (ℓ_∞, Untargeted).

Kurakin et al. extended the FGSM method by applying it multiple times with small step size [55]. This method clips pixel values of intermediate results after each step to ensure that they are in an ϵ-neighborhood of the original image $\mathbf{x}$. For the m^{th} iteration,

$$\mathbf{x}'_{m+1} = \mathbf{x}'_m + \text{Clip}_{\mathbf{x},\epsilon}(\alpha \cdot \text{sign}(\nabla_{\mathbf{x}} J(g(\mathbf{x}'_m), y))) \tag{2.4}$$

The clipping equation, $\text{Clip}_{\mathbf{x},\epsilon}(\mathbf{z})$, performs per-pixel clipping on $\mathbf{z}$ so the result will be in the ℓ_∞ ϵ-neighborhood of $\mathbf{x}$ [55].

DeepFool (ℓ_2, Untargeted).

Moosavi et al. used a ℓ_2 minimization-based formulation, termed DeepFool, to search for adversarial examples [76]. DeepFool searches for the minimal perturbation to fool a classifier and uses concepts from geometry to direct the search. For linear classifiers (whose decision boundaries are linear planes), the region of the space describing a classifier's output can be represented by a polyhedron (whose plane faces are those boundary planes defined by the classifier). Then, DeepFool searches within this polyhedron for the minimal perturbation that can change the classifiers decision. For general non-linear classifiers, this algorithm uses an iterative linearization procedure to get an approximated polyhedron.

Jacobian Saliency Map Approach: JSMA (ℓ_0, Targeted).

Papernot et al. [83] proposed the *Jacobian-based saliency map approach* (JSMA) to search for adversarial examples by only modifying a limited number of input pixels in an image. As a targeted attack, JSMA iteratively perturbs pixels in an input image that have high adversarial saliency scores. The adversarial saliency map is calculated from the Jacobian (gradient) matrix $\nabla_{\mathbf{x}} g(\mathbf{x})$ of the DNN model $g(\mathbf{x})$ at the current input $\mathbf{x}$. The $(c,p)^{\text{th}}$ component in Jacobian matrix $\nabla_{\mathbf{x}} g(\mathbf{x})$ describes the derivative of output class c with respect to feature pixel p. The adversarial saliency score of each pixel is calculated to reflect how this pixel will increase the output score of the target class t versus

changing the score of all other possible output classes. The process is repeated until classification into the target class is achieved, or it reaches the maximum number of perturbed pixels. Essentially, JSMA optimizes Equation (2.2) by measuring perturbation $\Delta(\mathbf{x}, \mathbf{x}')$ through the ℓ_0-norm.

Carlini/Wagner Attacks (ℓ_2, ℓ_∞ and ℓ_0, Targeted).

Carlini and Wagner introduced three new gradient-based attack algorithms that are more effective than all previously-known methods in terms of the adversarial success rates achieved with minimal perturbation amounts [19]. They proposed three versions of attacks using ℓ_2, ℓ_∞, and ℓ_0 norms.

The CW_2 attack formalizes the task of generating adversarial examples as an optimization problem with two terms as usual: the prediction objective and the distance term. However, it makes the optimization easier to solve through several techniques. The first is using the logits-based objective function instead of the softmax-cross-entropy loss that is commonly used in other optimization-based attacks. This makes it robust against the defensive distillation method [86]. The second is converting the target variable to the *argtanh* space to bypass the box-constraint on the input, making it more flexible in taking advantage of modern optimization solvers, such as Adam. It also uses a binary search algorithm to select a suitable coefficient that performs a good trade-off between the prediction and the distance terms. These improvements enable the CW_2 attack to find adversarial examples with smaller perturbations than previous attacks.

The CW_∞ attack recognizes the fact that the ℓ_∞ norm is hard to optimize and only the maximum term is penalized. Thus, it revises the objective into limiting perturbations to be less than a threshold τ (initially 1, decreasing in each iteration). The optimization reduces τ iteratively until no solution can be found. Consequently, the resulting solution has all the perturbations smaller than the specified τ.

The basic idea of the CW_0 attack is to iteratively use CW_2 to find the least important features and freeze them (so value will never be changed) until the ℓ_2 attack fails with too many features being frozen. As a result, only those features with significant impact on the prediction are changed. This is the opposite of JSMA, which iteratively selects the most important features and performs large perturbations until it successfully fools the target classifier.

2.2.2 Defensive Techniques

Researchers have proposed several solutions to alleviate the impact of adversarial examples. We group those works into three broad categories: *adversarial training*, *gradient masking* and *input pre-processing*.

Adversarial Training. *Adversarial training* introduces discovered adversarial examples and the corresponding ground truth labels to the training [36, 65, 107]. Ideally, the model will learn how to restore the ground truth from the adversarial perturbations and perform robustly on the future adversarial examples. This technique, however, suffers from the high cost to generate adversarial examples and (at least) doubles the training cost of DNN models due to its iterative re-training procedure. Its effectiveness also depends on having a technique for efficiently generating adversarial examples similar to the one used by the adversary, which may not be the case in practice. As pointed out by Papernot et al. [84], it is essential to include adversarial examples produced by all known attacks in adversarial training, since this defensive training is non-adaptive. But, it is computationally expensive to find adversarial inputs by most known techniques, and there is no way to be confident the adversary is limited to techniques that are known to the trainer.

Madry et al. proposed a variation of adversarial training by enlarging the model capacity in the re-training. Their adversarial training uses the adversarial examples generated by BIM attack with random starts, named as the "PGD attack" [65]. The authors claimed that this method could provide a security guarantee against any adversary based on the theory of robust optimization. The empirical results showed that the PGD-based adversarial training significantly increases the robustness of an MNIST model against many different attacks. However, the results on the CIFAR-10 dataset show limited robustness.

Gradient Masking. By forcing DNN models to produce near-zero gradients, the "gradient masking" defenses seek to reduce the sensitivity of DNN models to small changes in inputs. Gu et al. proposed adding a gradient penalty term in the training objective. The penalty term is a summation of the layer-by-layer Frobenius norm of the Jacobian matrix [39]. Although the trained model is more robust against certain adversaries, the penalty significantly reduces the capacity of the model and sacrifices accuracy on many tasks [84]. Papernot et al. introduced the strategy of "defensive distillation" to harden DNN models [86]. A defensively distilled model is trained with the smoothed labels produced by an existing trained DNN model. Then, to hide model's gradient information from an adversary, the

distilled model replaces its last layer with a "harder" softmax function after training. Experimental results showed that larger perturbations are required when using JSMA to evade distilled models. However, two subsequent studies have found that defensive distillation failed to mitigate a variant of JSMA with a division trick [17] and a black-box attack [82]. Papernot et al. concluded that methods designed to conceal gradient information are bound to have limited success because of the transferability of adversarial examples [84].

Input Pre-processing. A few recent studies for hardening deep learning try to reduce the model sensitivity to small input changes by pre-processing the inputs, an approach we also use in Chapter 5. Bhagoji et al. proposed to use dimensionality reduction techniques such as Principal Component Analysis (PCA) as defense [10]. They first performed PCA on a clean dataset, then linearly projected all the inputs to the PCA space and only preserved the top k principle axes. While we could expect the reduced sensitivity with the PCA projection, the method corrupts the spatial structure of an image, and the state-of-the-art CNN models are no longer applicable. Instead, Meng and Chen proposed to train an autoencoder as an image filter to harden DNN models [72]. The encoder stage of the autoencoder is essentially a non-linear dimensionality reduction. Its decoder stage restores an input to its original spatial structure; therefore the target DNN model does not need to change. Similar to the work we describe in Chapter 4, Osadchy et al. independently suggested using the binary filter and the median smoothing filter to eliminate adversarial perturbations and proposed to attack the defenses by increasing attackers' perturbation strength [80].

2.2.3 Detecting Adversarial Examples

Multiple recent studies [33, 38, 73] focused on detecting adversarial examples. The strategies they explore naturally fall into three groups: *sample statistics*, *training a detector* and *prediction inconsistency*.

Sample Statistics. For detecting adversarial examples, Grosse et al. [38] proposed a statistical test using maximum mean discrepancy and suggests the energy distance as the statistical distance measure. Their method requires a large set of both adversarial and legitimate inputs and is not capable of detecting individual adversarial examples, making it not useful in practice. Feinman et al. proposed to use kernel density estimation [33] that measures the distance between an unknown input and a group of legitimate inputs using their representations from some middle layers of a DNN

model. It is computationally expensive and can only detect adversarial examples lying far from the manifolds of the legitimate population. Due to the intrinsically unperceptive nature of adversarial examples, using sample statistics to separate adversarial examples from legitimate inputs seems unlikely to be effective. Experimental results from both Grosse et al. [38] and Feinman et al. [33] showed that strategies relying on sample statistics gave inferior detection performance compared to other detection methods.

Training a Detector. Similar to adversarial training, adversarial examples can be used to train a detector. However, this strategy requires a large number of adversarial examples, therefore, being expensive and prone to overfitting the adversarial attacks that generated examples for training the detector. Metzen et al. proposed attaching a CNN-based detector as a branch off a middle layer of the original DNN model [73]. The detector outputs two classes and uses adversarial examples (as one class) plus legitimate examples (as the other class) for training. The detector is trained while freezing the weights of the original DNN, therefore does not sacrifice the classification accuracy on the legitimate inputs. Grosse et al. demonstrated a detection method (previously proposed by Nguyen et al. [77]) that adds a new "adversarial" class in the last layer of the DNN model [38]. The revised model is trained with both legitimate and adversarial inputs, reducing the accuracy on legitimate inputs due to the change to the model architecture.

Prediction Inconsistency. The basic idea of *prediction inconsistency* is to measure the disagreement among several models in predicting an unknown input, since one adversarial example may not fool every DNN model. Feinman et al. borrowed an idea from dropout [44] and designed a detection technique they called *Bayesian neural network uncertainty* [33]. The authors used the "training" mode of dropout layers to generate many predictions for an input at test time. They reported that the disagreement among the predictions of sub-models is rare on legitimate inputs but common on adversarial examples, thus can be used for detection. Meng and Chen independently proposed a similar adversary detection method to ours that also uses the prediction vectors of the original and the filtered images [72]. The biggest difference between their work and our work in Chapter 4 is that they trained an auto-encoder as the image filter, whereas we rely on "hard-coded" transformations. As a result, our approach is less expensive in the training phase.

Chapter 3

Genetic Evasion[1]

Machine learning is widely used to develop classifiers for security tasks. However, the robustness of these methods against motivated adversaries is uncertain. In this work, we propose a generic method to evaluate the robustness of classifiers under attack. The key idea is to stochastically manipulate a malicious sample to find a variant that preserves the malicious behavior but is classified as benign by the classifier. We present a general approach to search for evasive variants and report on results from experiments using our techniques against two PDF malware classifiers, PDFrate and Hidost. Our method is able to automatically find evasive variants for both classifiers for *all* of the 500 malicious seeds in our study. Our results suggest a general method for evaluating classifiers used in security applications, and raise serious doubts about the effectiveness of classifiers based on superficial features in the presence of adversaries.

3.1 Introduction

Machine learning models are popular in security tasks such as malware detection, network intrusion detection and spam detection. From the data scientists' perspective, these models are effective since they achieve extremely high accuracy on test datasets. For example, Dahl et al. reported achieving 99.58% accuracy in classifying Win32 malware using an ensemble deep neural network with dynamic

[1]This chapter is based on the paper: Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers: A Case Study on PDF Malware Classifiers. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2016 [122]

features [28]. Šrndic et al. achieved over 99.9% accuracy in a PDF malware classification task using an SVM-RBF model with structural path features [104].

However, it is important to realize that these results are for particular test datasets. Unlike when machine learning is used in other fields, security tasks involve adversaries responding to the classifier. For example, attackers may try to generate new malware deliberately designed to evade existing classifiers. This breaks the assumption of machine learning models that the training data and the operational data share the same data distribution. As a result, it is important to be skeptical of machine learning results in security contexts that do not consider attackers' efforts to evade the generated models.

The risk of evasion attacks against machine learning models under adversarial settings has been discussed in the machine learning community, mainly focused on simple models for spam detection (e.g., [29,64]). However, evasion attacks against malware classification can be much more complex in terms of the classification algorithm and the feature extraction as well as the mutability of highly-structured samples. Consequently, though evading malware classifiers has been partially explored by classifier authors as well as security researchers, previous studies significantly under-estimate the attackers' ability to manipulate samples. For example, previous studies may mistakenly assume the attackers can only insert new contents because removing existing contents would easily corrupt maliciousness [11,66,104]. In addition, previous works are ad hoc and limited to particular target classifiers or specific types of samples [66,105]. Other than suggesting point solutions, they do not provide methods to automatically evaluate the effectiveness of a classifier against adaptive adversaries.

We present a generic method to assess the robustness of a classifier by simulating attackers' efforts to evade the classifier. We do not assume the adversary has any detailed knowledge of the classifier or the features it uses, or can use targeted expert knowledge to manually direct the search for an evasive sample. Instead, drawing ideas from genetic programming (GP) [35,52], we perform stochastic manipulations and then evaluate the generated variants to select promising ones. By repeating this procedure iteratively, we aim to generate evasive variants. A sophisticated attacker, of course, can do manipulations that would not be found by a stochastic search, so we cannot claim that a classifier that resists such an attack is necessarily robust. On the other hand, if the automated approach finds evasive samples for a given classifier, it is a clear sign that the classifier is not robust against a motivated adversary.

We evaluated the proposed method on two PDF malware classifiers, and found that it could automatically find evasive variants for all the 500 sample seeds selected from the Contagio PDF malware archive [20]. The evasive variants exhibit the same malicious behaviors as the original samples, but are sufficiently different in the classifier's feature space to be classified as benign by the machine learning-based models.

Our analysis of the discovered evasive variants reveals that both classifiers are vulnerable because they employ non-robust features, which can be manipulated without disrupting the desired malicious behavior. Superficial features may work well on test datasets, but if the features used to classify malware are shallow artifacts of the training data rather than intrinsic properties of malicious content, it is possible to find ways to preserve the malicious behavior while disrupting the features.

Contributions. Our primary contributions involve developing and evaluating a general method for automatically finding variants that evade classifiers. In particular:

- We propose a general method to automatically find evasive variants for target classifiers. The method does not rely on any specific classification algorithms or assume detailed knowledge of feature extraction, but only needs the classification score feedback on generated variants and rough knowledge of the likely features used by the classifier (Section 3.2).

- We implement a prototype system that automatically finds variants that can evade structural feature-based PDF malware classifiers. This involves designing operators that perform stochastic manipulations on PDF files, an oracle that determines if a generated variant preserves maliciousness, a selection mechanism that promotes promising variants during the evolutionary process, and a fitness function for each target classifier (Section 3.4).

- We evaluate the effectiveness of our system in evading two recent PDF malware classifiers: PDFrate [101] and Hidost [104], a classifier designed with the explicit goal of resisting evasion attempts. Our system achieves 100% success rates in finding evasive variants against both classifiers in an experiment with 500 malware sample seeds. An analysis of the discovered evasive variants in the feature space of each classifier shows that many non-robust features employed in the classification facilitate evasion attacks (Sections 3.5 and 3.6).

We provide background on machine learning classifiers in Section 3.2 and on PDF malware in Section 3.3. Section 3.8 discusses related work on evasion attacks.

3.2 Overview

We propose an automated method to simulate an attacker attempting to find an evasive variant for a desired malware sample which is detected by a target classifier. The attacker's goal is to find a malware variant that preserves the malicious behavior of the original sample, but that is misclassified as benign by the target classifier. In addition to improving our understanding of how classifiers work in the presence of adaptive adversaries, we hope our results will lead to strategies for constructing classifiers that are more robust to adversaries, but in this work we focus on assessing evadability.

3.2.1 Threat Model

We assume an attacker starts with a desired malicious sample that is (correctly) classified by a target classifier as malicious, and wants to create a sample with the same malicious behavior, but that is misclassified as benign. The attacker is capable of manipulating the malicious sample in many ways, and is likely to have knowledge of samples that are (correctly) classified as benign.

We assume the attacker has black-box access to the target classifier, and can submit many variants to that classifier. For each submitted variant, the attacker learns its classification score. The classification score is a number (typically a real number between 0 and 1) that indicates the classifier's prediction of maliciousness, where values above some threshold (say 0.5) are considered malicious and samples with lower classification scores are considered benign. We do not assume the attacker has any internal information about the classifier, only that it can use it as a black-box that outputs the classification score for an input sample. We assume the classifier operator does not adapt the classifier to submitted variants (which must be the case if the attacker has offline access to the classifier).

3.2.2 Finding Evasive Samples

Our method uses genetic programming techniques to perform a directed search of the space of possible samples to find ones that evade the classifier while retaining the desired malicious behavior.

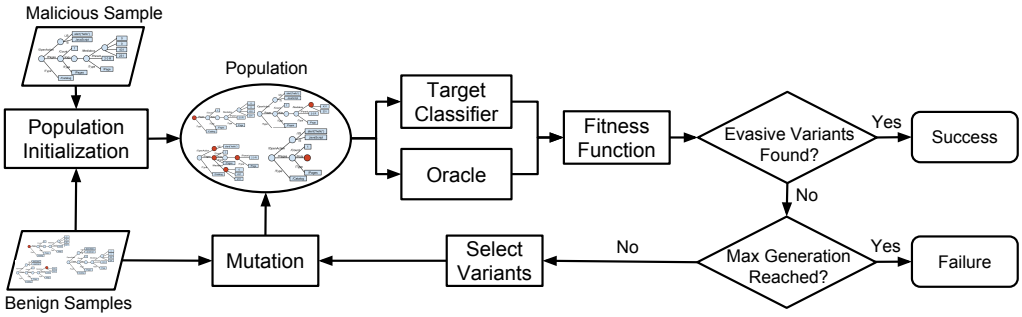


Figure 3.1: Generic classifier evasion method.

Genetic programming (GP) is a type of evolutionary algorithm, originally developed for automatically generating computer programs tailored to a particular task [35, 52]. It is essentially a stochastic search method using computational analogs of biological mutation and crossover to generate variants, and modeling Darwinian selection using a user-defined *fitness function*. Variants with higher fitness are selected for continued evolution, and the process continues over multiple generations until a variant with desired properties is found (or the search is terminated after exceeding a resource bound). Genetic programming has been shown to be effective in many tasks including fixing legacy software bugs [57], software reverse engineering [42], and software re-engineering [92].

Method. Our procedure is illustrated in Figure 3.1. It starts with a seed sample that exhibits malicious behavior, and is classified as malicious by the target classifier. Our method aims to find an evasive sample that preserves the malicious behavior but is misclassified as benign by the target classifier.

First, we initialize a population of variants by performing random manipulations on the malicious seed. Then, each variant is evaluated by a target classifier as well as an oracle. The *target classifier* is a black box that outputs a number that is a measure of predicted maliciousness of an input sample. There is a prescribed threshold used to decide if it is malicious or benign. The *oracle* is used to determine if a given sample exhibits particular malicious behavior. In most instantiations, the oracle will involve expensive dynamic tests.

A variant that is classified as benign by the target classifier, but found to be malicious by the oracle, is a successful evasive sample. If no evasive samples are found in the population, a subset of the generated variants are selected for the next generation based on a fitness measure designed to reflect progress towards finding an evasive sample. Since it is unlikely that the transformations

will re-introduce malicious behaviors into a variant, corrupted variants that have lost the malicious behavior are replaced with other variants or the original seed.

Next, the selected variants are randomly manipulated by mutation operators to produce next generation of the population. The process continues until an evasive sample is found or a threshold number of generations is reached.

To improve the efficiency of the search, we collect traces of the mutation operations used and reuse effective traces. If a search ends up finding any evasive variants, the mutation traces on the evasive variants will be stored as successful traces. Otherwise, the mutation trace of a variant with the highest fitness score is stored. These traces are then applied to other malware seeds to generate variants for their population initialization. Because of the structure of PDFs and the nature of the mutation operators, the same sequence of mutations can often be applied effectively to many initial seeds.

3.3 PDF Malware and Classifiers

This section provides background on PDF malware and the two target PDF malware classifiers.

3.3.1 PDF Malware

The *Portable Document Format* (PDF) is a popular document format designed to enable consistent content and layout in rendering and printing on different platforms. Although it was not openly standardized until 2008 [5], and there are various non-standard extensions supported by different PDF reader products, all PDF files roughly share the same basic structure depicted in Figure 3.2.

A PDF file consists of four parts: *header*, *body*, *cross-reference table* (CRT) and *trailer*. The *header* contains the PDF magic number and a format version indicator. The *body* is a set of PDF objects that comprise the content of the file, while the CRT indexes the objects in *body*. The *trailer* specifies how to find the CRT and other special objects such as the root object. Thus, PDF readers typically start reading a PDF from the end of the file for efficiency.

The *body* is the most important to a PDF since it holds almost all the visible document data. It contains eight basic types of objects, namely Booleans, numbers, strings, names, arrays, dictionaries,

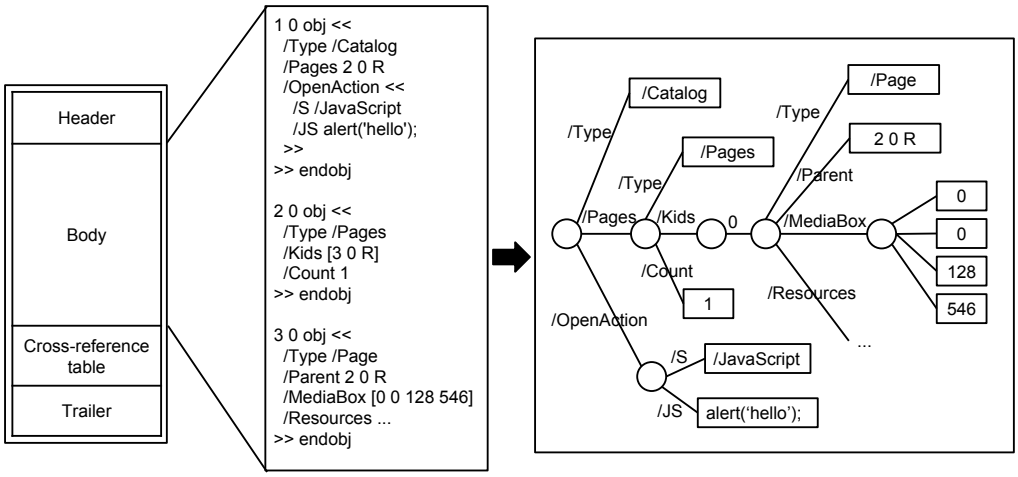


Figure 3.2: The physical and logical structure of a PDF file.

streams and the null objects. The objects can be labeled with a pair of integer identifiers as indirect objects so that they can be referenced by other objects. The inter-referencing objects form a tree-like logical structure, as is shown in the right of Figure 3.2. This tree-like structure is ideally suited to genetic programming techniques since it is easy to alter and move sub-trees to generate new variants.

PDF malware is becoming increasing prevalent because PDF is a widely accepted document format and victims are more willing to open PDFs than other files. According to a recent Internet security threat report [25], PDF is in top 7 attachment types in spear-phishing emails in 2014. We expect there will be continuing opportunities for PDF malware attacks because 128 new vulnerabilities in Acrobat readers have been reported in CVE so far in 2015 (through 8 December), which is almost three times the total number in 2014 [27].

PDF malware typically contains exploits in JavaScript objects or other objects that take advantage of vulnerabilities of particular PDF readers (most commonly, Adobe Acrobat). PDF malware may also carry other encoded payloads in stream objects which will be triggered after exploits [101].

3.3.2 Target Classifiers

Several projects have built PDF malware classifiers using machine learning techniques. Earlier works, such as Wepawet [26] and PJScan [56], focused on the embedded malicious JavaScript in

PDF malware. These tools consist of a JavaScript code extractor and a dynamic or static malicious JavaScript classifier.

Since not all PDF malware involves embedded JavaScript, and PDF malware authors have found many tricks for hiding JavaScript code [95], recent PDF malware classifiers have focused on structural features of PDF files. In this work, we target state-of-the-art structural feature-based classifiers.

Structural feature-based classifiers assume that the malicious PDFs have different patterns in their internal object structures than those found in benign PDFs. For example, the PDF Malware Slayer tool uses the object keywords as features, where each feature corresponds to the occurrences of a given keyword [67]. For our experiments, we selected PDFrate [100,101] and Hidost [104] as the target classifiers. They are representatives of recent PDF malware classifiers, and Hidost was developed with a particular goal of being resilient to evasion attacks. Both classifiers achieve extremely high accuracy in malware detection on their testing datasets. The other reason for choosing these classifiers as our targets is the availability of the open source implementations. Although our method only requires black-box access to the classifier, having access to the internal feature space is beneficial for understanding our results (Section 3.6).

PDFrate. PDFrate is a random forest classifier that uses an ensemble learning model consisting of a large number of decision trees designed to reduce variance in predictions. With a random subset of training data and a random subset of features, each decision tree is trained to minimize the prediction error on its training subset. After training, the output score of PDFrate is the fraction of trees that output "malicious", ranging from 0 to 1. The threshold value is typically 0.5, although the PDFrate authors claim that adjusting the threshold from 0.2 to 0.8 has little impact on accuracy because most samples have scores very close to either 0 or 1.

Besides object keywords, PDFrate also employs the PDF metadata and several properties of objects as the classification features. The PDF metadata includes the author, title, and creation date. The object properties includes positions, counts, and lengths.

PDFrate was trained with a random subset of the Contagio dataset [20] with 5,000 benign and 5,000 malicious PDFs. The two parameters are respectively the number of trees ($ntree = 1,000$) and the number of features in each tree ($mtry = 43$). The feature set is a total of 202 integer, floating point, and Boolean features, but only 135 of the features are described in the PDFrate documentation.

What we use in this work is an open-source re-implementation of PDFrate named Mimicus [103], implemented by Nedim Šrndic and Pavel Laskov to mimic PDFrate for malware evasion experiments [105]. Mimicus was trained with the 135 documented PDFrate features and the same training set as PDFrate.[2] Mimicus has been shown to have classification performance nearly equivalent to PDFrate [105].

Hidost. Hidost is a support vector machine (SVM) classification model. SVM is an optimal margin classifier that tries to find a small number of support vectors (data points) that separate all data points of two classes with a hyperplane of a high-dimensional space. With kernel tricks, it can be extended as a nonlinear classifier to fit more complex classification problems. Hidost uses a radial basis function (RBF) kernel to map data points into an infinite dimensional space. At testing time, the (positive or negative) distance of a data point to the hyper-plane is output as the prediction result. A positive distance is interpreted as malicious, and negative as benign.

Hidost uses the structural paths of objects as classification features. For example, the structural path of a typical Pages object is */Root/Pages*. If that object appears in the PDF file, its feature value is 1; if not, its feature value is 0. Since the number of possible structural paths of PDF objects is infinite, Hidost uses 6,087 selected paths as features. The selected paths are those which appeared in at least 1,000 of the files in a pool of 658,763 benign and malicious PDFs collected from VirusTotal [113] and a Google search. The resulting model provided by the authors of Hidost was trained using the randomly-sampled 5,000 malicious and 5,000 benign files. It is reported to be robust against adversaries, where the number of false negatives on another 5,000 random malicious files only increased from 28 to 30 under what the authors claim is the "strongest conceivable mimicry attack" [104].

3.4 Evading PDF Malware Classifiers

The proposed method could be applied to any security classifier, although its effectiveness depends on being able to find good genetic programming operators to search the feature space efficiently and an appropriate fitness function to direct the search. In this section, we show how to instantiate our design to find evasive PDF malware.

[2]The Mimicus authors were unable to locate one malicious file with the MD5 hash 35b621f1065b7c6ebebacb9a785b6d69 in Contagio.

3.4.1 PDF Parser and Repacker

The first step is to parse the PDF file as a tree-like representation. We will also need to regenerate a PDF file from the tree representation, after it has been manipulated to produce a new variant. For this, we use pdfrw [70], a python-based open source library for parsing PDF files into the tree-like structure and serializing that structure into an output PDF file.

It is important to note that pdfrw is not a perfect PDF parser and repacker, and a number of PDF malware samples have been malformed intentionally to bypass or confuse PDF parsers used in malware detectors (while still being processed by target PDF readers due to parser quirks). This means we cannot test our method on PDF seed samples that cannot be parsed by pdfrw, or that no longer exhibit malicious behavior when they are unpacked and packed using pdfrw.

To avoid losing too many samples because of PDF parsing issues, we modified pdfrw to loosen its grammar checking. This significantly increased the success rate of repacking PDF malware samples. The modified version of pdfrw is available at `https://github.com/mzweilin/pdfrw`.

In our modified pdfrw, we ignore several potentially corrupted, malformed, or misleading auxiliary elements. The *EOF* marks in PDF raw bytes are ignored; instead, the parser reads in all bytes of a file. The *cross-reference tables* are ignored; instead, it parses objects in the *body* directly without any index. Stream length indicators are ignored; instead, the parser detects the stream length with the *endstream* token. The unpaired keys or values are also ignored in parsing a dictionary. Ignoring these auxiliary elements significantly decreases parsing efficiency, thus, is only suitable for repacking seed malware samples. All seeds are repacked with correct auxiliary elements for efficient parsing later. In addition, we added support for parsing empty objects, which do exist in the malware samples. The dictionary data structure was modified to enable deep-copy in duplicating variants from seeds.

3.4.2 Genetic Operators

Since both of the classifiers we target employ the object structure of the PDF file as features, we need to generate variants by manipulating the PDF files at that level. (If we were targeting JavaScript-based classifiers instead, we would instead need to generate variants by manipulating the embedded JavaScript code.) Due to the limited number of possible static features, we believe it is reasonable to assume the attackers have the knowledge of the manipulation level.

We use computational analogs of *mutation* in biological evolution to generate evasive PDF malware variants. The mutation operator changes any object in a PDF file's tree-like structure with low probability. An object is mutated with probability given by the *mutation rate*, typically a number smaller than 0.5. The mutation is either a *deletion* (the object is removed), an *insertion* (another object is inserted after it), or a *replacement* (this object is replaced with some other object).

We choose among these options with uniform random probability. In the case of an insertion or replacement, a second object is also chosen uniformly at random from a large pool of objects segmented from benign PDFs. The external genome helps to generate a more diverse population.

The other well-known operator, *crossover*, commonly used in genetic algorithms, is not used in this work. We found it was possible to achieve an 100% evasion rate only using the simple mutation operations.

3.4.3 Oracle

We need an oracle to determine if a variant preserves the seed's malicious behavior. There is no perfectly accurate malware detection technique that works universally (indeed, if such a technique existed our work would not be necessary). In this case we have one advantage that enables a highly-accurate oracle for testing variants: we do not need an oracle that can test for arbitrary malicious behavior, but instead only need to verify that a particular known malicious action is performed by the variant.

To do this, we use the Cuckoo sandbox [40]. Cuckoo runs a submitted sample in a virtual machine installed with a PDF reader and reports the behavior of the sample including network APIs called and their parameters. Figure 3.3 shows an example of malware detection results from Cuckoo. The malware sample opened in a virtual machine exploited a disclosed buffer overflow vulnerability in Acrobat Readers (CVE-2007-5659). The injected shellcode downloads four additional pieces of malware from Internet and executes them. Since the execution of Cuckoo was isolated from the Internet to avoid spreading malware, the shellcode just received malformed executable files provided by INetSim, a network service simulator [48]. However, the downloading and execution behaviors detected by Cuckoo are enough to show that the shellcode has been executed. By comparing the behavioral signature of the original PDF malware and the manipulated variant, we determine if

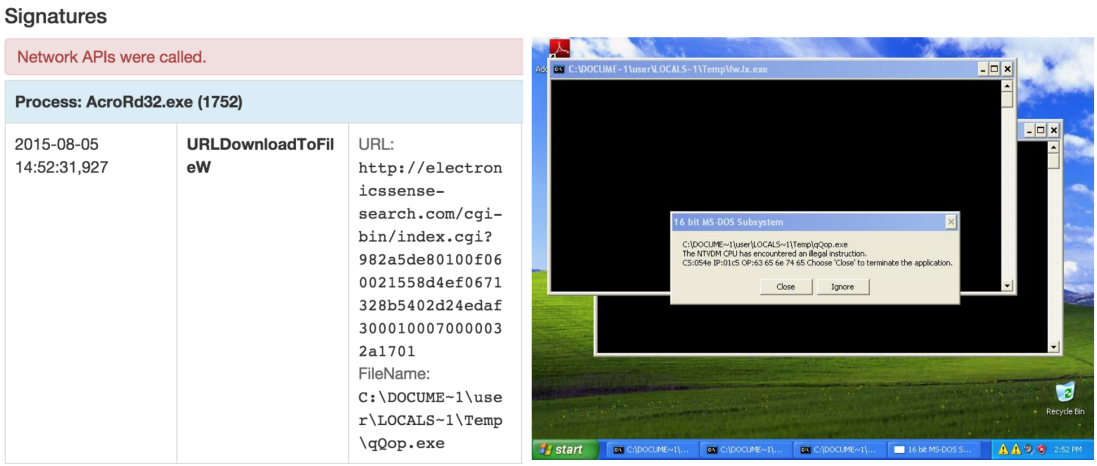


Figure 3.3: A PDF malware detection result given by the Cuckoo sandbox. The left side shows the key API execution trace, the right is a screenshot captured from the virtual machine.

the original malicious behavior is preserved. The details on how we select and compare behavioral signatures are deferred to Section 3.5.1.

We only focus on the network behaviors of malware samples in this work. Although this setting prevents our method from working on malware samples without network activity, we believe it is not a real constraint in practice since malware authors could always develop a way to verify the desired malicious behaviors.

Cuckoo sandbox works well as an oracle, but is computationally expensive. We experimented with other possible oracles, including using Wepawet. Wepawet and similar detection techniques only detect the malicious payloads, but do not verify that the payload is actually executed in a real PDF reader. Because many of the genetic mutations will disrupt that execution, oracles that do not actually dynamically observe the variant exhibiting the malicious behavior result in many false positives (apparently evasive variants that would not actually work as malware). Hence, it is important to use an oracle that confirms the malicious behavior is preserved through actual execution. This limits the samples we can use in our experiments to ones for which we can produce the malicious behavior in our oracle's test environment (Section 3.5.1).

3.4.4 Fitness Function

A fitness function gives the fitness score of each generated variant. Higher scores are better. Given 0 as a threshold value, a variant with a positive fitness score is evasive: it is classified as benign and

retains the malicious behavior.

In our case, the fitness function captures both the output of the oracle and the predicted result of the target classifier. The oracle is modeled as a binary function: $oracle(x) = 1$ if x exhibits the malicious signature; otherwise, $oracle(x) = 0$. In order to eliminate corrupted variants, we always assign the lowest possible fitness score to variants with $oracle(x) = 0$.

Based on the different scoring methods used by the target classifiers, the fitness functions are defined separately. PDFrate, as a random forest classifier, outputs a confidence value of maliciousness from 0 to 1, typically with a threshold of 0.5. Thus, we define its fitness function as

$$fitness_{pdfrate}(x) = \begin{cases} 0.5 - pdfrate(x) & oracle(x) = 1 \\ LOW\_SCORE & oracle(x) = 0 \end{cases}$$

with evasive range of $(0, 0.5]$.

The SVM model of Hidost outputs negative (positive) distance of a benign (malicious) sample to hyperplane. Therefore, for Hidost the fitness function is defined as

$$fitness_{hidost}(x) = \begin{cases} hidost(x) \times (-1) & oracle(x) = 1 \\ LOW\_SCORE & oracle(x) = 0 \end{cases}$$

with evasive range of $(0, +\infty)$.

3.4.5 Selection

A selection process in GP can be as simple as always selecting variants with higher fitness scores in a generation. However, it might happen that very few or even none of the variants in a generation preserve the malicious behavior during the evolutionary process. If the malicious behavior is lost from the population, it is very unlikely the GP will ever find an evasive sample that exhibits the original malicious behavior.

In order to avoid degeneration in the population, we designed a replacement mechanism in addition to the naïve selection process. The corrupted variants, which are judged by the oracle as non-malicious, are assigned the lowest fitness score (LOW_SCORE) and are replaced by either the original malicious

PDF, the best variant found so far, or the best variant found in the previous generation. We choose among these options with uniform random probability when corrupted variants occur, which ensures that a fixed number of variants are retained in each generation.

3.4.6 Trace Collection and Replay

The most common way to initialize a population is duplicating the original seed and performing a random mutation operation on each copy. Considering the potentially common properties across evasive variants, we accelerate the search by reusing mutation traces that successfully led to evasive or promising variants.

A mutation trace consists of a series of mutations defined by 3-tuple (*mutation operator*, *target object path*, *file id: source object path*). For example,

(insert, */Root/Pages/Kids/1*, *1: /Root/Pages/Kids/4*)

inserts an external Page object from a benign file *1* to the targeted PDF file. The three possible mutation operators are defined in Section 3.4.2. Though the *target object path* has the same format as the *source object path*, they are paths in different PDF files. The *target object path* refers to an object in the variant, while the *source object path* points to an object in an external benign file with the specified file id.

Mutation traces are added to two pools at the end of each GP search. If a GP search successfully generates evasive variants, all of the corresponding mutation traces are added to the *success trace pool*. Otherwise, a mutation trace that generates the variant with the highest fitness score is added to the *promising trace pool*.

The traces in the two pools are replayed in the population initialization to produce some variants for the first generation. If the number of usable traces is smaller than the population size, additional variants are generated in the conventional way. If the number is larger than the population size, the selection process described in Section 3.4.5 shrinks the population to the specified size.

3.5 Experiment

We evaluate the effectiveness of the proposed method by conducting experiments on the two target PDF malware classifiers.

3.5.1 Dataset and Experiment Setup

We started with the 10,980 PDF malware samples in the Contagio archive [20], from which we selected 500 suitable samples for evaluation. These samples are verified by the oracle as exhibiting malicious behavior, are classified by both target classifiers as malicious, and can be correctly repacked by pdfrw.

Malicious PDF Dataset. Table 3.1 summarizes the sample selection procedure.

First, we filtered out the samples that don't have any network API calls by the shell code analysis of Wepawet, leaving 9,688 out of 10,980 samples. This is not necessary for our method, but useful since we use Wepawet to obtain additional information about the samples.

Second, the remaining samples were tested in the Cuckoo sandbox. According to the vulnerability information of each sample provided by Wepawet, Adobe Acrobat Reader 8.1.1 is the most common target PDF reader, except for CVE-2009-9837 which targets Foxit readers. Thus, these samples were loaded with Acrobat Reader 8.1.1. However, not all network behaviors indicated by the static analysis on shell code can be observed in Cuckoo even though we have selected a targeted PDF reader due to the imperfect network simulation in virtual machines as well as the potential sandbox detection features in malware. As a result, only 1,414 out of the 9,688 samples were observed to have malicious network activities running on Acrobat Reader 8.1.1 inside the Cuckoo sandbox.

Table 3.1: Seed selection.

Description	Number
PDF Malware samples in Contagio	10,980
Samples with network API calls detected by Wepawet	9,688
Samples with network activities observed by Cuckoo	1,414
Unique samples correctly repacked by pdfrw	1,384
True positives of PDFrate	1,378
True positives of Hidost	502
Intersection of TPs in PDFrate and Hidost	500

Table 3.2: Comparison of network-based malware signatures.

Source	Description	Example	Effective	Consistency Avg	Consistency Min
API traces	Combination of HTTP URL requests and host queries	[http://stortfordaircadets.org.uk/flash/exe.php?x=pdf, stortfordaircadets.org.uk]	500	0.95	0.50
API traces	Hosts queried through getaddrinfo()	[stortfordaircadets.org.uk]	497	0.95	0.50
Network traffic	Transport layer destination IP addresses	(udp: [192.168.57.2:53], tcp: [192.168.57.2:80])	476	0.85	0.10
API traces	URLs requested through raw socket, URLDownloadToFileW(), InternetOpenUrlA()	[http://stortfordaircadets.org.uk/flash/exe.php?x=pdf]	473	0.95	0.50
Network traffic	DNS queries	[stortfordaircadets.org.uk]	462	0.93	0.10
Network traffic	HTTP URL requests	[http://stortfordaircadets.org.uk/flash/exe.php?x=pdf]	460	0.93	0.10

Next, the 1,414 samples were repacked by the modified pdfrw with less strict grammar checking, then re-tested by Wepawet and Cuckoo. This resulted in 1,384 unique samples. Eleven of the samples were corrupted during repacking and no longer behaved maliciously in Wepawet or Cuckoo. The other 19 samples were found to be duplicates after being repacked. This is a clear sign that malware authors have attempted to evade detection through parsing obfuscation.

Since our goal is to evaluate the effectiveness of an evasion attack, we need to filter out the false negative samples of the target classifiers. PDFrate correctly classified 1,378 out of the 1,384 samples as malicious, while Hidost only correctly classified 502 of them. The intersection of the true positives from both classifiers left a suitable evaluation set of 500 unique PDF malware samples.

According to results from Wepawet, these 500 malware samples exploit two different vulnerabilities in Acrobat Readers: 333 of them exploit multiple buffer overflows reported in CVE-2007-5659, the other 167 exploit a stack-based buffer overflow reported by CVE-2009-0927. Both vulnerabilities can be exploited to execute arbitrary code. In summary, the payloads in the 500 samples access 255 different hosts to download additional malware from the Internet.

The selection process leaves us with 500 samples from the original 10,980 malware samples in the Contagio archive. Although this selects less than 5% of the original samples, it does not have implications for the success rate of a malware author attempting to find an evasive sample so long as

the selection criteria have no biases which would impact our results. Many of the down-selects are due to artifacts of the experiment, not reflective of what an actual malware producer would observe. For example, the most significant reduction is because of the particular dynamic environment we selected to verify the malicious behaviors. Malware authors can easily design an oracle that verifies the presence of the particular malicious behaviors they intend to inflict.

Reliable Malware Signatures. Since the dynamic behavior of malware samples may vary across executions, we need to select a reliable malware signature from a group of candidates. Even though the malware is executing in the same virtual environment, its behavior may be effected by the timing of events, service failures, and other sources of non-determinism.

Focusing on the network behaviors of malware samples, we may extract various network behaviors reported by Cuckoo as signatures, such as DNS queries, HTTP URL requests, and network destinations. Cuckoo generates these reports from the network-related API execution traces and the captured network traffic. Table 3.2 compares the effectiveness of six different types of signatures extracted from Cuckoo reports.

We tested the 500 malware seeds in Cuckoo virtual machines, running each seed ten times. Our goal is to determine which type of signature will have the best precision in capturing observed malicious behavior, while being consistent across multiple executions of the same sample.

If a signature extracts any relevant behavior for a seed in any of the ten tests, we count the signature effective on the seed. Obviously, an ideal signature would be effective on all 500 seeds. We also measure the consistency of a signature over the 10 repeated tests. We designate the extracted behavior observed most frequently over the ten tests as the reference signature for a seed. The consistency on a seed is calculated as $\frac{mode}{10}$ (that is, the fraction of times the reference signature occurred across the 10 trials).

The average and the minimum consistency of each type of signature over the ten executions for each of the 500 seeds are listed in Table 3.2. In general, the signatures extracted from API traces are more consistent than those extracted from network traffic. We choose the union of the HTTP URL requests and host queries extracted from API traces as the signature for our experiments. By combining those two behavioral signatures, we obtain a signature that is effective on all 500 malware

seeds and has the highest average and minimum consistency.

Benign PDF Dataset. We collected a set of 179 benign PDF documents using a Google search with filetype:pdf and no keywords. All files were confirmed to be benign by both VirusTotal [113] and Wepawet [26]. We only included files smaller than 1 MB to avoid introducing unnecessary computation costs manipulating extremely large PDF files. We picked the 3 benign samples with the lowest scores (that is, most benign) to the target classifiers as the source of external objects in the experiment. Our results show that just a few benign samples is sufficient for generating successful evasion attacks.

GP Parameters. Several GP parameters are arbitrarily chosen without any parameter fine-tuning other than one obvious constraint: we want the experiment to finish in a reasonable time. The population size is 48 and the maximum generation is 20. The mutation rate is 0.1. The fitness stop threshold is 0.0, which indicates that an evasive variant has been found.

Target Classifiers. Since we don't want to abuse the online deployed malware classification systems by submitting too many automatically generated malware variants, we always prefer locally executable code. We used the Mimicus re-implementation of PDFrate and the Hidost classifier, configured and trained as described in Section 3.3.2.

Machine. We used one typical desktop PC in the experiment (Intel Core i7-2600 CPU @ 3.40GHz and 32GB of physical memory running 64-bit Ubuntu 14.04 Server). The Cuckoo sandbox consists of 16 virtual machine instances running Windows XP SP3 32 bit and Adobe Acrobat Reader 8.1.1. The resources required to find evasive samples using our approach are readily available.

3.6 Results

The GP-based method achieves surprisingly good results in evading the two target classifiers. For both of the classifiers, it is able to generate a variant that preserves the malicious behavior but is classified as benign for all 500 seeds in our test set. Our code and data are available under an open source license from `http://www.evadeML.org`

3.6.1 PDFrate

After approximately one week of execution, the algorithm found 72 effective mutation traces that generated 16,985 total evasive variants for the 500 malware seeds (34.0 evasive variants per seed in average), achieving 100% evasion rate in attacking PDFrate.

Trace Analysis. All the mutation traces that generated evasive variants were re-executed on all of the 500 seeds afterwards to investigate the efficacy of each trace. Efficacy here measures for how many of the malware seeds applying the given trace produces an evasive variant.

The length of each mutation trace and its efficacy are illustrated in Figure 3.4. The traces are sorted by trace ID, which reflects the order in which traces are found. From the figure we observe that the method generally finds longer mutation traces as the evolution proceeds. Part of the reason for this is the initial population for later seeds is generated using the collected traces. If those initial variants are not evasive, subsequent mutations will be added to the original traces.

The efficacy of each seed is not strongly correlated with its length. One mutation trace consisting of a single operation that inserts a page object generated evasive variants for 155 malware seeds. There was also mutation trace with 189 operations that was effective for only two seeds.

The accumulated evasions sorted by the length of mutation traces are given by Figure 3.5 (for comparison, the figure show results for Hidost as well, which we discuss later). The difficulty of generating variants to evade PDFrate varies substantially over the seeds. It only took 15 short mutation traces (none longer than 45 operations) to generate evasive variants for 400 of the 500 seeds. Finding evasive variants for the other 100 seeds took 57 long mutation traces with lengths ranging from 48 to 354.

In order to understand why it takes much longer traces to generate evasive variants for those 100 seeds, we examined the original classification scores of each seed. Figure 3.6 groups the seeds by the minimum trace length required for generating evasive variants. The left side shows the original classification score distribution in PDFrate. We found that the original seeds with lower classification scores (<0.95) are mostly evadable by short traces. Thus, we believe some seeds require more mutations to evade because they are originally more clearly malicious to the classifier. (This is more obvious in Hidost as we discuss later.)

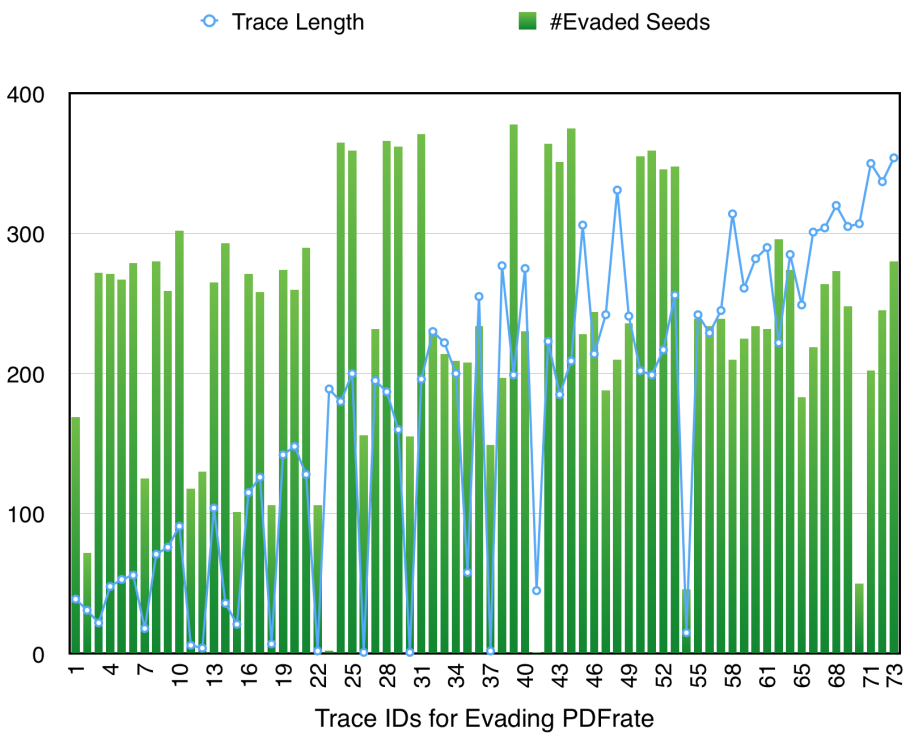


Figure 3.4: The length and efficacy of mutation traces for evading PDFrate.

Feature Analysis. To understand the evasion attacks, we examine the impact of the changes on the feature space used by PDFrate.

We first look at the two simplest mutation traces in length of 1 that are effective for 162 seeds:

(insert, */Root/Pages/Kids*,

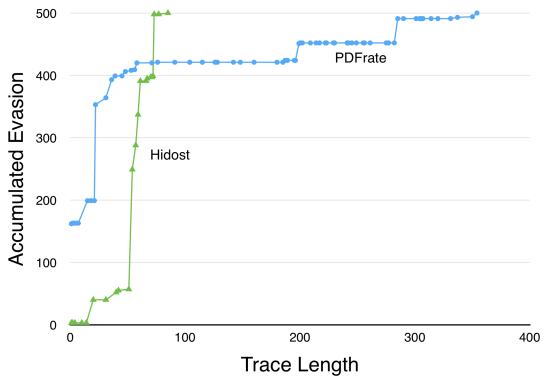


Figure 3.5: Accumulated evasions against PDFrate and Hidost, sorted by trace length.

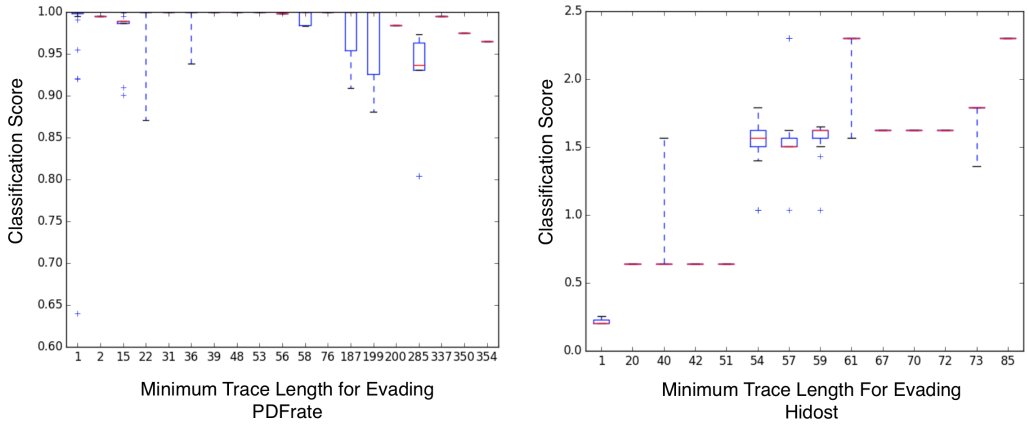


Figure 3.6: The distribution of the original classification score of seeds.

3:/Root/Pages/Kids/4/Kids/5/)

(replace, */Root/Type*, *3:/Root/Pages/Kids/1/Kids/3*)

Even though they are different operations, the common effect of the two mutations is that they both introduce new Page objects from external benign PDFs, resulting in significant changes in the feature space of PDFrate.

Table 3.3 lists one example of feature changes by simply inserting several Page objects. The classification score of the original seed is 0.998, approaching the maximum malicious score of 1.0. After inserting the new Page objects, the classification score decreases to 0.43, which is below the normal malware threshold of 0.5. The simple insert resulted in a large number of changes in the feature space. The counters of some objects like pages, fonts and streams as well as the file size directly increase due to the newly introduced objects. The object length statistics are decreased or increased due to the change of the object population. Some other features on object positions are also changed due to the relocation of objects at the raw byte level. All feature values are in the raw formats because feature normalization is not required with random forests. Even though the feature changes are so significant that PDFrate classifies the new variant as benign, the malicious behavior of the original seed does not change at all. The change just added some pages to the PDF file.

One simple manipulation introduces many feature changes, but the impact of changing each feature is not equivalent due to the varying importance of features in the classification. Though random forest is a complex non-linear model that is difficult to interpret, we estimate the impact of altering each feature independently. Intuitively, changing a high impact feature should significantly affect the classification scores.

Table 3.3: Impact of PDFrate Features.

Feature	Original	Evasive	Δscore1	Δscore2	Impact
count_font	0.0	70.0	0.114	0.392	0.506
count_obj	11.0	230.0	0.067	0.110	0.177
count_endobj	11.0	230.0	0.056	0.069	0.125
count_box_other	3.0	140.0	0.038	0.043	0.081
count_endstream	4.0	74.0	0.011	0.054	0.065
pos_box_max	0.0	0.8	0.052	0.013	0.065
count_stream	4.0	74.0	0.021	0.041	0.062
pos_box_avg	0.0	0.5	0.022	0.022	0.044
pos_eof_avg	1.0	1.0	0.000	0.032	0.032
pos_eof_min	1.0	1.0	-0.002	0.029	0.027
pos_page_max	0.0	0.8	0.003	0.018	0.021
pos_eof_max	1.0	1.0	0.002	0.016	0.018
pos_page_avg	0.0	0.5	0.002	0.010	0.012
size	36,028.0	503,739.0	-0.001	0.005	0.004
ratio_size_page	36,028.0	7,407.9	0.001	0.002	0.003
ratio_size_obj	3,275.3	2,190.2	0.000	0.002	0.002
ratio_size_stream	9,007.0	6,807.3	0.002	0.000	0.002
len_obj_avg	3,234.9	2,157.2	0.001	0.000	0.001
count_page	1.0	68.0	-0.004	0.004	0.000
len_obj_max	27,455.0	34,314.0	0.000	0.000	0.000
len_obj_min	44.0	49.0	0.000	0.000	0.000
len_stream_avg	8,700.3	6,390.2	0.000	0.000	0.000
len_stream_max	27,392.0	34,246.0	0.000	0.000	0.000
pos_page_min	0.0	0.0	-0.002	0.000	-0.002
pos_box_min	0.0	0.0	-0.003	0.000	-0.003

The difference in the feature space of PDFrate for a selected seed sample (with score 0.998) and corresponding evasive variant (with score 0.43). Original is the feature value of the original seed malware sample; Evasive is the feature value of the evasive variant. Features with the same value for both samples are not included. $\Delta score1$ is the original score subtracting the score after that feature is changed to Evasive value; $\Delta score2$ is the evasive score subtracted by the score after that feature is reset to the Original value. Impact is $\Delta score1 + \Delta score2$.

For an original malware sample, if the feature is changed to a different value in evasive variants, the classification score should decrease (indicating it appears more like benign samples to the classifier). On the other hand, for an evasive variant, if the feature value is reset to the original value, the classification score should increase (appears more like malicious samples to the classifier). Therefore, we model the impacts with two factors. The decrease of the classification score of a malware seed when a feature is changed to the evasive value is reported as $\Delta score1$. The increase of the classification score of an evasive variant when the feature is reset to the original value is reported as $\Delta score2$. The impact is the sum of the two scores. Table 3.3 lists the impact ranking of the affected features, which roughly matches with the feature importance ranking in PDFrate [100].

Table 3.4: Most Altered Features Evading PDFrate

Counts	Increased features	Counts	Decreased features
16,985	pos_eof_avg	14,234	pos_page_min
16,985	pos_eof_max	10,806	len_obj_min
16,985	pos_eof_min	10,728	count_javascript
16,985	size	8,834	len_stream_min
16,975	count_endstream	7,637	ratio_size_stream
16,975	count_stream	4,742	createdate_tz
16,941	count_endobj	4,742	delta_tz
16,941	count_obj	4,250	ratio_size_page
16,862	len_stream_max	3,448	len_stream_avg
16,812	pos_box_max	3,137	pos_page_avg

The most critical feature change for this example is *count_font*. The original malware sample does not have any font objects as fonts are not needed for the exploit. The classifier learns that this feature is important because most of the malware samples in the training set do not contain any font objects as the malware authors are too lazy to insert any text, but it is unlikely that any benign PDF file has no font objects. However, this is an artifact of the malware samples in the training set, not an inherent property for malicious PDFs. It is trivial to add font objects to an existing PDF malware sample to alter the value of this feature.

There are longer traces which contain at most 354 mutations and influence more features in PDFrate. Table 3.4 lists the features that were most frequently increased and decreased across all 16,985 evasive variants found. (The full list of all 68 mutable features of PDFrate found in evasion attacks is found in Appendix **??**.) The count is how many times the value of the feature is different for the evasive variant found compared to the original seed. High counts imply these features are not robust and should not be used in malware classification because they are easy to change without corrupting the malicious properties for many malware seeds.

Most non-robust features are unsurprising, because a PDF malware author can always change the visible contents (such as pages, text, images and metadata) in PDF malware samples without corrupting the malicious payloads. The only surprising feature is *count_javascript*. Since PDF malware heavily relies on JavaScript to carry exploits and shell code, it seems surprising that it is possible to decrease *count_javascript* without disrupting the malicious behavior. However, the *count_javascript* feature is not an accurate count of the number of embedded JavaScript code pieces in a PDF. It just extracts the number of JavaScript keywords, but these keywords are optional in script execution. The targeted PDF reader will execute the JavaScript even without the */Javascript* keyword.

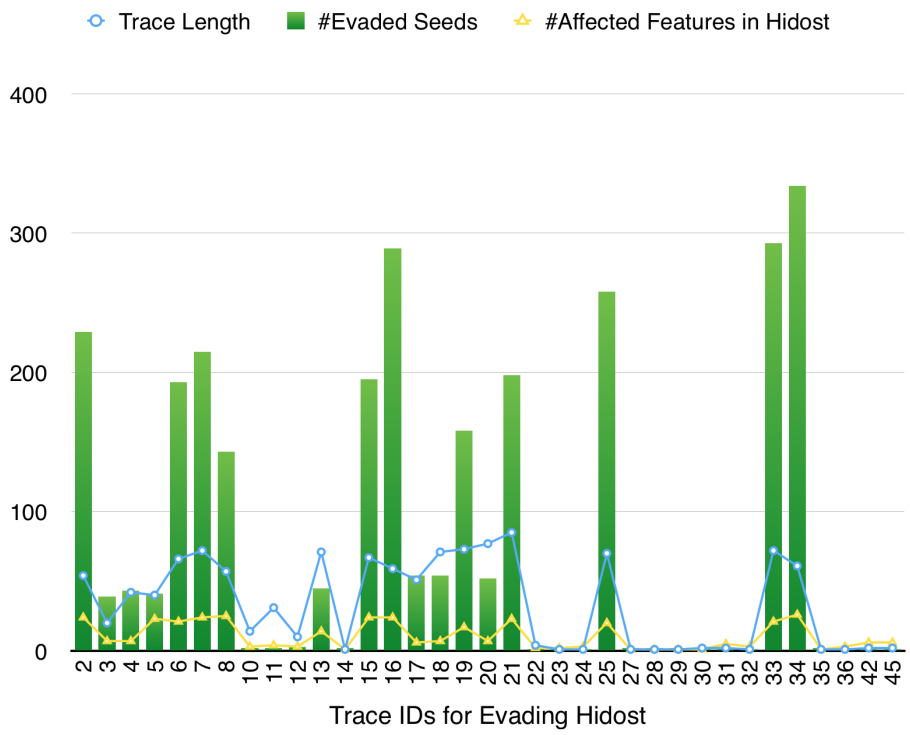


Figure 3.7: The length and efficacy of mutation traces for evading Hidost.

3.6.2 Hidost

The experiment of evading Hidost took around two days to execute. Although Hidost was designed specifically to resist evasion attempts,[3] our method achieves a 100% evasion rate, generating 2,859 evasive samples in total for 500 seeds (5.7 evasive samples per seed in average).

Trace Analysis. We analyze the efficacy of each mutation trace which is examined in the same way as for PDFrate. The length and efficacy of each mutation trace are shown in Figure 3.7. In general, it required shorter mutation traces to achieve 100% evasion rate in attacking Hidost than it did for PDFrate.

We observed two major differences compared to PDFrate. First, there is no increasing trace length trend for newly found mutation traces, unlike for PDFrate where the trace length increases with the trace ID. Second, the trace length is more correlated with the efficacy: longer traces tend to

[3]Specifically, the Hidost authors claim, "The most aggressive evasion strategy we could conceive was successful for only 0.025% of malicious examples tested against an off-the-shelf nonlinear SVM classifier with the RBF kernel using the binary embedding. Currently, we do not have a rigorous mathematical explanation for such a surprising robustness. Our intuition suggests that the main difficulty on attacker's part lies in the fact that the input features under his control, i.e., the structural elements of a PDF document, are only loosely related to the true features used by a classifier. The space of true features is hidden behind a complex nonlinear transformation which is mathematically hard to invert." [104]

be more effective in generating evasive variants. Several short mutation traces with fewer than 5 mutations are only effective on 1 or 2 malware seeds. In contrast, a long mutation trace containing 61 mutations is effective on 334 malware seeds.

The accumulated number of evasions found sorted by the length of mutation traces is given in Figure 3.5. The plot is closer to linear, suggesting that, in contrast to PDFrate, there is little variation in the difficulty of finding evasive variants for different seeds. We believe the differences from PDFrate stem from the different feature set in Hidost. The mutation operations have more direct influence on the structural path features in Hidost. For example, an object deletion operation just deletes the corresponding path of a feature (along with those of its descendants). In contrast, feature changes in PDFrate resulting from the same operation are less tangible. Besides decreasing the counts of specific objects that we can expect, the other positional features may also change due to the relocation of objects in repacking the modified variant. As a result, there are more inter-influences among the mutation operations in evading PDFrate, and a larger number of mutations may be required to reach the evasion threshold. The box plot of the original classification score in Hidost of each seed shown in the right side of Figure 3.6 suggests that it usually requires more mutations to find an evasive variant for seeds that appear to be more clearly malicious to the classifier.

Feature Analysis. The binary features used in Hidost are much easier to interpret than the variety of features used by PDFrate.

We first look at the simplest mutation traces. There are 5 mutation traces in length 1, which are only effective on 1 or 2 malware seeds. They are:

(delete, */Root/OpenAction/JS/Length*)

(delete, */Root/Names*)

(delete, */Root/AcroForm/DR*)

(replace, */Root/AcroForm/DR*,
3: /Root/OpenAction/D/0/.../FontBBox/3)

(replace, */Root/AcroForm/DR*,
3: /Root/Pages/Kids/3/.../DescendantFonts/0/DW)

The first three mutations each delete a node from the original malware seeds, changing the value of the corresponding Hidost feature from 1 to 0. The first deleted object similar to the *count_javascript* feature in PDFrate. Both capture properties that frequently exist in malware samples but not in

benign files. However, they are optional in malicious code execution. The other deleted objects are artifacts in the training dataset that are not closely tied to malicious behavior. Although the last two traces use replace operations, the important effects of the replacements are to remove the features extracted from the children objects of the original */Root/AcroForm/DR* node.

Simply deleting some objects is not sufficient to evade Hidost (it is only effective on 1 or 2 malware seeds in our experiment), but additional mutations are enough to find evasive variants for all of the seeds. The longest mutation trace contains 85 operations, which is effective on 198 malware seeds for generating evasive variants to bypass Hidost. Table 3.5 lists the all of feature changes observed over the 198 malware seeds when executing that mutation trace. Unsurprisingly, several auxiliary objects are added or deleted to fool Hidost. For example, several metadata objects are inserted. Metadata widely exists in benign PDFs when users generate PDF documents with popular PDF writers. On the other hand, it is rare in PDF malware because malware authors did not add metadata in hand-crafting PDF exploits. However, this is just an artifact in the training dataset and not an essential difference between PDF documents and PDF malware. Inserting metadata into a PDF malware sample increases the likelihood of the sample being considered benign by Hidost.

As seen from this example, trace length itself is not a good measure of evasion complexity. Although the stochastic search process found an 85-operation trace to create these evasive variants, the trace only impacts the 23 features (each corresponding to a node in the PDF file) showing in Table 3.5. That is to say, there is a 23-operation trace that would be just as effective (and probably shorter traces since one mutation can impact many features), and the trace found by the search includes many useless or redundant mutations. For the purposes of creating evasive malware, it is not important to find the shortest effective trace, although it would be possible to develop techniques to automatically pare down a trace to its essential operations if desired. The yellow triangle plot in Figure 3.7 shows the number of affected features for each trace.

Although its authors claimed that Hidost was robust against evasion attacks involving just feature addition, we found many evasive variants that only added features. Among the 2,859 evasive variants, 761 are pure feature addition attacks, 21 of them are pure feature deletion attacks, and the other 2,077 involved both feature addition and deletion. It is already unrealistic to assume attackers can only insert features, and, as shown in the claims about non-evadability of Hidost, dangerous to assume a technique cannot be evaded because particular manual techniques fail.

A complete list of mutated features in evading Hidost is given in Appendix **??**. These non-robust

Table 3.5: Feature changes produced by longest Hidost mutation trace.

Added Features	Deleted Features
Threads	AcroForm
ViewerPreferences/Direction	Names/JavaScript/Names/S
Metadata	AcroForm/DR/Encoding/PDFDocEncoding
Metadata/Length	AcroForm/.../PDFDocEncoding/Differences
Metadata/Subtype	AcroForm/.../PDFDocEncoding/Type
Metadata/Type	Pages/Rotate
OpenAction/Contents	AcroForm/Fields
OpenAction/Contents/Filter	AcroForm/DA
OpenAction/Contents/Length	Outlines/Type
Pages/MediaBox	Outlines
	Outlines/Count
	Pages/Resources/ProcSet
	Pages/Resources

features should not be used in a malware classifier, as they can be easily changed while preserving the original malicious properties.

3.6.3 Cross-Evasion Effects

Even though the classifiers are designed very differently and trained with different training datasets, we suspected they must share some properties in the same classification task. Therefore, we conducted a cross-evasion experiment by feeding one classifier with the evasive variants found in evading the other classifier.

For 388 of the malware seeds, the evasive variants found by evading Hidost are also effective in evading PDFrate. That is to say, without any access to PDFrate, a malware author with access to Hidost could find evasive variants for 77.6% of the seeds. In contrast, the evasive variants found by evading PDFrate are only effective against Hidost for two of the malware seeds.

The significant difference in the cross evasion effects is due to the different feature sets in the two classifiers. Indeed, the primary design goal for Hidost was to be less easily evaded than other classifiers by using features based on structural properties. The evasive variants generated by the algorithm in evading PDFrate do change the measured features significantly, however, they have little effect on the structural features used in Hidost. In the reverse direction, the evasive variants targeting Hidost by directly altering structural features (necessary to evade Hidost), incidentally impact the features used by PDFrate.

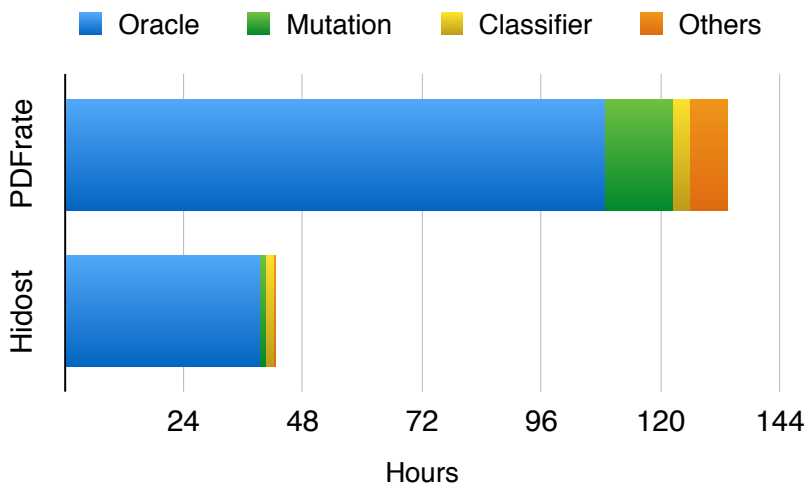


Figure 3.8: Time required to find evasive variants for 500 malware samples.

3.6.4 Execution Cost

One drawback of evolutionary algorithms is that they provide no guarantees about generating good results within a specific duration. For many problems, these methods can require a huge amount of computing resources before a desired result is found. Further, failing to find the desired result may be a sign that it doesn't exist, or just that more computing resources are required. Our experiments show that the resources required for this instantiation are very reasonable.

For each classifier target, the experiment was run in several rounds. The first round started with empty trace pools, so the search for evasive variants relies solely on the stochastic search directed by the fitness function. In later rounds, the successful and promising mutation traces found previously were used to accelerate the search. All the failed jobs in a round were re-run in the next round with the all mutation traces stored in previous rounds.

For PDFrate, it took four rounds to reach 100% evasion. The evasion rate on Hidost reached 100% in the second round. Figure 3.8 shows the total time needed to find evasive variants for all 500 malware seeds in evading the two classifiers. The duration for each job is not meaningful because it mainly depends on the job sequence. Later jobs are usually much faster because they may benefit from mutation traces found earlier, but the order of trying the seeds is arbitrary.

It took less than 6 days to achieve a 100% evasion rate on PDFrate. In other words, our method found an evasive variant for each seed in 16 minutes on average running on a commodity desktop. Evasive variants were found against Hidost three times faster, taking 5 minutes per seed in average.

The main computation time is running the generated variants in the Cuckoo sandbox, which we use as the oracle in our process. The machine with 16 virtual machines running in parallel is able to test 1,000 samples per hour. This could easily be accelerated by using more machines, since there are no dependencies between the executions.

We also observed that the time spent on other tasks (including mutation) in attacking PDFrate takes a larger proportion of the total duration than for Hidost (8.3% vs. 4.1%). This is because the benign files used as external object genome are larger than those in attacking Hidost. Hence, it produced larger variants, increasing the computational burden for parsing, manipulating, and repacking.

3.7 Discussion

In this section we discuss the potential defenses and future directions suggested by our results.

3.7.1 Defense

Beyond understanding the vulnerabilities of current classifiers, our ultimate goal is to improve the robustness of classifiers under attack. Based on the evasive samples we generated, and the non-robust features we found in Section 3.5, we consider several possible approaches.

Information Hiding and Randomization. One of the most direct solutions to protect classifiers is hiding the classification scores from the users or adding random noise to the scores [8]. Another proposed method is the multiple classifier system, in which the classification scores are somewhat randomly picked from different models trained with disjoint features [12]. As our method heavily relies on the classification scores of variants to calculate fitness scores that direct the evolution, the lack of accurate score feedback makes the search for evasive variants much harder and may make our approach infeasible.

However, the intrinsic non-robustness of superficial features should not be simply ignored. Considering the potential cross-evasion effects (Section 3.6.3), hiding or randomizing the information may not help much against an adversary who can infer something about the types of features used by the target classifier. Moreover, previous work has shown that accurately re-implementing a similar

classifier with a surrogate training set is possible (indeed, this is what the authors of Mimicus did to experiment with evadability of PDFrate [100, 105]).

Adapting to Evasive Variants. Our experiments assume that adversary can test samples without exposing them to the classifier operator. In an on-line scenario, the classifier may be able to adapt to attempted variants. Note, however, that retraining is expensive and opens up the classifier to alternate evasion strategies such as poisoning attacks.

Chinavle et al. proposed a method that would automatically retrain the classifier with pseudo labels once evasive variants were detected by a mutual agreement measure on the ensemble model, which had been shown effective on a spam detection task [21]. However, adapting to users' input without true labels introduces a new risk of poisoning attacks.

Defeating Overfitting. The evadability of classifiers we demonstrate could be just an issue of overfitting, in which case, well known machine learning practices should work to defeat overfitting. For example, collecting a much larger dataset for training the model, or using model averaging to lower the variance.

We don't expect these conventional methods will help, however. It is impossible to collect a complete dataset of future malware, and none of these techniques anticipate an adversary who is actively attempting to evade the classifier.

Selecting Robust Features. We found many non-robust features from the two classifiers in the evasion experiments. Obviously, they should be removed from the feature set as they can be easily manipulated by the attacker without corrupting the malicious properties. The problem with the features used by both Hidost and PDFrate, however, is that *all* of the features are likely non-robust. The superficial features used by these classifiers do not have any intrinsic distinguishability between benign and malicious PDFs, and it would be very surprising if superficial features were found that could be used for robust classification. Instead, it seems necessary to use deeper features to build classifiers that can resist evasion attempts by sophisticated adversaries. Such features will depend on higher-level semantic analysis of the input file, in ways that are difficult to change without disrupting the malicious behavior.

3.7.2 Improving Automatic Evasion

Our automatic evasion method provides a general method to evaluate the robustness of classifiers for security tasks. Its ability to find evasive variants against a target classifier demonstrates clear weaknesses, but if our method fails to find evasive variants against a particular classifier this is certainly not enough to be confident that other techniques (including manual effort) would not be able to find evasive variants. Hence, it is valuable to improve the method to enable more efficient searching to target more challenging classifiers.

Parameter Tuning. In this work, we just arbitrarily choose the search parameters. Tuning the parameters, or even trying dynamic mechanisms like parameter decay, could make the search algorithm more efficient.

Learnable GP. The current method we use to generate evasive variants is essentially a random search algorithm. Hence, it often generates corrupted variants that lose the malicious behavior. A probabilistic model would learn which mutations are more effective for generating evasive variants to direct the search more efficiently.

Other Applications. Our case study focused on PDF malware, but we believe similar approaches could be effective against other machine-learning based malware classifiers. The main challenges in applying our approach to a new domain are to develop suitable genetic mutation operations and find an appropriate oracle.

3.8 Related Work

There have been several papers on evasion attacks against classifiers in the machine learning community, mostly focused on spam detection with simple models (e.g., [21, 29, 64]. Chinavle et al. argued that the adversarial problem is essentially *concept drift*, which is a well studied field in machine learning that considers data distributions which change over time [21]. However, the concept drift solutions assume the data distribution changes are not due to the classifier itself, not resulting from an adversary intentionally adapting to it.

Evasion attacks against malware classifiers have been studied previously by Biggio et al. from the angle of classification models [11] and by Šrndic et al. [104]. However, these studies assumed that attackers can only insert new features and they conducted evasion experiments in the feature space without generating actual evasive PDF malware. In fact, the experiments in our work show attackers can also delete features while preserving maliciousness, and our experiments verified that the resulting evasive variants preserved maliciousness through dynamic execution in a test environment.

Šrndic et al. demonstrated how PDFrate could be evaded by exploiting an implementation flaw in the feature extraction [105]. Our method does not rely on any particular implementation flaw in a target classifier. Instead, it exploits the weak spots in a classifier model's feature space and employs a stochastic method to manipulate samples in diverse ways.

Maiorca et al. proposed reverse-mimicry attacks against PDF malware classifiers [66]. In reverse-mimicry, a benign sample is manipulated into a malicious one by inserting malicious payloads into the structure. The attack is generic to a class of classifiers based on structural features. However, the hand-crafted attack only works on malware with simple payloads. In contrast, our GP-based method is automatic and does not have this limitation.

Evolutionary algorithms have also recently been used to fool deep learning-based computer vision models [77]. In contrast, this work uses genetic programming, an important branch of evolutionary algorithms for generating highly-structured data like computer programs.

3.9 Conclusions

Our experiments show how the traditional approach of building machine learning classifiers can fail against determined adversaries. We argue that it is essential for designers of classifiers used in security applications to consider how adversaries will adapt to those classifiers, and important for the research community to develop better ways of predicting the actual effectiveness of a classifier in deployment.

3.10 Impact

Genetic Evasion, as the first practical and generic evasion attack against machine learning-based malware classifiers, is often discussed and compared in academic papers [31, 49, 85, 96, 125].

Our work has inspired more researchers to explore the security of machine learning models. For example, Dang et al. proposed a hill climbing algorithm-based method to strengthen the evasion attack [30]. Their improved method can evade a classifier that only gives binary feedback (malicious or benign), which makes it more practical to evade real-world black-box models that are deployed and hidden behind API calls. Wang et al. made a theoretical analysis on the robustness of machine learning models and concluded that one necessary condition of a strong-robust classifier is that it only uses relevant features [114].

Our work also draws more attention back to the malicious document problem. For example, Xu et al. proposed PlatPal, a non-machine-learning-based PDF malware detector that relies on simple heuristics about discrepant behaviors of malicious PDFs on diverse platforms [119]. Tong et al. further investigated the robustness of the Hidost classifier and found that only a handful of features are robust [111].

Chapter 4

Feature Squeezing[1]

Previous studies to defend against adversarial examples mostly focused on refining the DNN models, but have either shown limited success or required expensive computation. We propose a new strategy, *feature squeezing*, that can be used to harden DNN models by detecting adversarial examples. Feature squeezing reduces the search space available to an adversary by coalescing samples that correspond to many different feature vectors in the original space into a single sample. By comparing a DNN model's prediction on the original input with that on squeezed inputs, feature squeezing detects adversarial examples with high accuracy and few false positives. This chapter explores two feature squeezing methods: reducing the color bit depth of each pixel and spatial smoothing. These simple strategies are inexpensive and complementary to other defenses, and can be combined in a joint detection framework to achieve high detection rates against state-of-the-art attacks.

4.1 Introduction

The goal of this chapter is to harden DNN systems against adversarial examples by detecting adversarial inputs. Detecting an attempted attack may be as important as predicting correct outputs. When running locally, a classifier that can detect adversarial inputs may alert its users or take fail-safe actions (e.g., a fully autonomous drone returns to its base) when it spots adversarial inputs. For an

[1]This chapter is based on the paper: Weilin Xu, David Evans and Yanjun Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018 [123]

on-line classifier whose model is being used (and possibly updated) through API calls from external clients, the ability to detect adversarial examples may enable the operator to identify malicious clients and exclude their inputs. Another reason that detecting adversarial examples is important is because even with the strongest defenses, adversaries will occasionally be able to get lucky and find an adversarial input. For asymmetrical security applications like malware detection, the adversary may only need to find a single example that preserves the desired malicious behavior but is classified as benign to launch a successful attack. This seems like a hopeless situation for an on-line classifier operator, but the game changes if the operator can detect even unsuccessful attempts during an adversary's search process.

Most previous work on hardening DNN systems, including *adversarial training* and *gradient masking* (details in Section 2.2.2), focused on modifying the DNN models themselves. In contrast, our work focuses on finding simple and low-cost defensive strategies that alter the input samples but leave the model unchanged. Section 2.2.3 describes a few other recent proposals for detecting adversarial examples.

Our approach, which we call *feature squeezing*, is driven by the observation that the feature input spaces are often unnecessarily large, and this vast input space provides extensive opportunities for an adversary to construct adversarial examples. Our strategy is to reduce the degrees of freedom available to an adversary by "squeezing" out unnecessary input features. The key idea is to compare the model's prediction on the original sample with its prediction on the sample after squeezing, as depicted in Figure 4.1. If the original and squeezed inputs produce substantially different outputs from the model, the input is likely to be adversarial. By comparing the difference between predictions with a selected threshold value, our system outputs the correct prediction for legitimate examples and rejects adversarial inputs.

Although feature squeezing generalizes to other domains, here we focus on image classification. Because it is the domain where adversarial examples have been most extensively studied. We explore two simple methods for squeezing features of images: reducing the color depth of each pixel in an image and using spatial smoothing to reduce the differences among individual pixels. We demonstrate that feature squeezing significantly enhances the robustness of a model by predicting correct labels of non-adaptive adversarial examples, while preserving the accuracy on legitimate inputs (Section 4.3), thus enabling an accurate detector for static adversarial examples (Section 4.4). Feature squeezing appears to be both more accurate and general, and less expensive, than previous methods, though

the robustness against adaptive adversary needs further investigation in the future work.

Contributions. Our key contribution is introducing and evaluating feature squeezing as a technique for detecting adversarial examples. We show how the general detection framework (Figure 4.1) can be instantiated to accurately detect adversarial examples generated by several state-of-the-art methods. We study two instances of feature squeezing: reducing color bit depth (Section 4.2.1) and both local and non-local spatial smoothing (Section 4.2.2). We report on experiments that show feature squeezing helps DNN models predict correct classification on adversarial examples generated by eleven different and state-of-the-art attacks mounted without knowledge of the defense (Section 4.3). Feature squeezing is complementary to other adversarial defenses since it does not change the underlying model, and can readily be composed with other defenses such as adversarial training (Section 4.3.2).

Section 4.4 explains how we use feature squeezing for detecting static adversarial inputs, combining multiple squeezers in a joint detection framework. Our experiments show that joint-detection can successfully detect adversarial examples from eleven static attacks at the detection rates of 98% on MNIST and 85% on CIFAR-10 and ImageNet, with low (around 5%) false positive rates. Although we cannot guarantee an adaptive attacker cannot succeed against a particular feature squeezing configuration, our results show it is effective against state-of-the-art static methods, and it considerably complicates the task of an adaptive adversary even with full knowledge of the model and defense (Section 4.4.4). In Chapter 5, we show that bit depth reduction improves provable robustness of deep learning models.

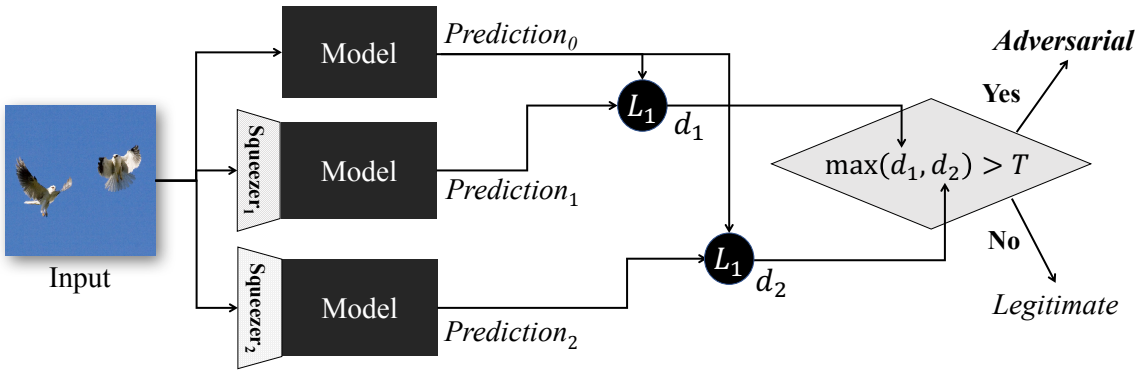


Figure 4.1: Feature-squeezing framework for detecting adversarial examples. The model is evaluated on both the original input and the input after being pre-processed by feature squeezers. If the difference between the model's prediction on a squeezed input and its prediction on the original input exceeds a threshold level, the input is identified to be adversarial.

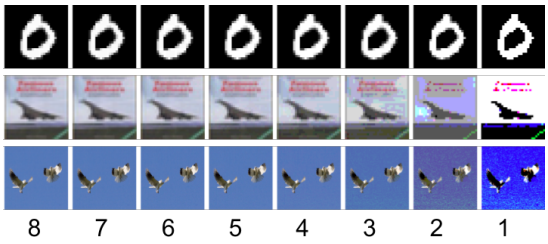


Figure 4.2: Image examples with bit depth reduction. The first column shows images from MNIST, CIFAR-10 and ImageNet, respectively. Other columns show squeezed versions at different color-bit depths, ranging from 8 (original) to 1.

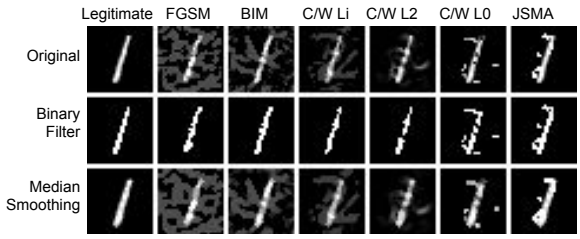


Figure 4.3: Examples of adversarial attacks and feature squeezing methods extracted from the MNIST dataset. The first column shows the original image and its squeezed versions, while the other columns present the adversarial variants. All targeted attacks are targeted-next.

4.2 Feature Squeezing Methods

Although the notion of feature squeezing is quite general, we focus on two simple types of squeezing: reducing the color depth of images (Section 4.2.1), and using smoothing (both local and non-local) to reduce the variation among pixels (Section 4.2.2). Section 4.3 looks at the impact of each squeezing method on classifier accuracy and robustness against adversarial inputs. These results enable feature squeezing to be used for detecting adversarial examples in Section 4.4.

4.2.1 Color Depth

A neural network, as a differentiable model, assumes that the input space is continuous. However, digital computers only support discrete representations as approximations of continuous natural data. A standard digital image is represented by an array of pixels, each of which is usually represented as a number that represents a specific color.

Common image representations use color bit depths that lead to irrelevant features, so we hypothesize that reducing bit depth can reduce adversarial opportunity without harming classifier accuracy. Two common representations, which we focus on here because of their use in our test datasets, are 8-bit grayscale and 24-bit color. A grayscale image provides $2^8 = 256$ possible values for each pixel. An 8-bit value represents the intensity of a pixel where 0 is black, 255 is white, and intermediate numbers represent different shades of gray. The 8-bit scale can be extended to display color images with separate red, green and blue color channels. This provides 24 bits for each pixel, representing $2^{24} \approx 16$ million different colors.

Squeezing Color Bits

While people usually prefer larger bit depth as it makes the displayed image closer to the natural image, large color depths are often not necessary for interpreting images (for example, people have no problem recognizing most black-and-white images). We investigate the bit depth squeezing with three popular datasets for image classification: MNIST, CIFAR-10 and ImageNet.

Greyscale Images (MNIST). The MNIST dataset contains 70,000 images of hand-written digits (0 to 9). Of these, 60,000 images are used as training data and the remaining 10,000 images are used for testing. Each image is 28×28 pixels, and each pixel is encoded as 8-bit grayscale.

Figure 4.2 shows one example of class 0 in the MNIST dataset in the first row, with the original 8-bit grayscale images in the leftmost and the 1-bit monochrome images rightmost. The rightmost images, generated by applying a binary filter with 0.5 as the cutoff, appear nearly identical to the original images on the far left. The processed images are still recognizable to humans, even though the feature space is only $1/128^{\text{th}}$ the size of the original 8-bit grayscale space.

Figure 4.3 hints at why reducing color depth can mitigate adversarial examples generated by multiple attack techniques. The top row shows one original example of class 1 from the MNIST test set and six different adversarial examples. The middle row shows those examples after reducing the bit depth of each pixel into binary. To a human eye, the binary-filtered images look more like the correct class; in our experiments, we find this is true for DNN classifiers also (Table 4.3 in Section 4.3).

Color Images (CIFAR-10 and ImageNet). We use two datasets of color images in this chapter: the CIFAR-10 dataset with tiny images and the ImageNet dataset with high-resolution photographs. The CIFAR-10 dataset contains 60,000 images, each with 32×32 pixels encoded with 24-bit color and belonging to 10 different classes. The ImageNet dataset is provided by ImageNet Large Scale Visual Recognition Challenge 2012 for the classification task, which contains 1.2 million training images and the other 50,000 images for validation. The photographs in the ImageNet dataset are in different sizes and hand-labeled with 1,000 classes. However, they are pre-processed to 224×224 pixels encoded with 24-bit True Color for the target model MobileNet [46, 69] we use in this chapter.

The middle row and the bottom row of Figure 4.2 show that we can reduce the original 8-bit (per RGB channel) images to fewer bits without significantly decreasing the image recognizability to humans. It is difficult to tell the difference between the original images with 8-bit per channel color

and images using as few as 4 bits of color depth. Unlike what we observed in the MNIST datase, however, bit depths lower than 4 do introduce some human-observable loss. This is because we lose much more information in the color image even though we reduce to the same number of bits per channel. For example, if we reduce the bits-per-channel from 8 bits to 1 bit, the resulting grayscale space is 1/128 large as the original; the resulting RGB space is only $2^{-(24-3)} = 1/2,097,152$ of the original size. Nevertheless, in Section 4.3.1 we find that squeezing to 4 bits is strong enough to mitigate a lot of adversarial examples while preserving the accuracy on legitimate examples.

Implementation

We implement the bit depth reduction operation in Python with the NumPy library. The input and output are in the same numerical scale $[0, 1]$ so that we don't need to change anything of the target models. For reducing to i-bit depth ($1 \leq i \leq 7$), we first multiply the input value with $2^i - 1$ (minus 1 due to the zero value) then round to integers. Next we scale the integers back to $[0, 1]$, divided by $2^i - 1$. The information capacity of the representation is reduced from 8-bit to i-bit with the integer-rounding operation.

4.2.2 Spatial Smoothing

Spatial smoothing (also known as *blur*) is a group of techniques widely used in image processing for reducing image noise. Next, we describe the two types of spatial smoothing methods we used: *local smoothing* and *non-local smoothing*.

Local Smoothing

Local smoothing methods make use of the nearby pixels to smooth each pixel. By selecting different mechanisms in weighting the neighbouring pixels, a local smoothing method can be designed as Gaussian smoothing, mean smoothing or the median smoothing method [94] we use. As we report in Section 4.3.1, median smoothing (also known as *median blur* or *median filter*) is particularly effective in mitigating adversarial examples generated by ℓ_0 attacks.

The median filter runs a sliding window over each pixel of the image, where the center pixel is replaced by the median value of the neighboring pixels within the window. It does not actually reduce

the number of pixels in the image, but spreads pixel values across nearby pixels. The median filter is essentially squeezing features out of the sample by making adjacent pixels more similar.

The size of the window is a configurable parameter, ranging from 1 up to the image size. If it were set to the image size, it would (modulo edge effects) flatten the entire image to one color. A square shape window is often used in median filtering, though there are other design choices. Several padding methods can be employed for the pixels on the edge, since there are no real pixels to fill the window. We choose *reflect padding* , in which we mirror the image along with the edge for calculating the median value of a window when necessary.

Median smoothing is particularly effective at removing sparsely-occurring black and white pixels in an image (descriptively known as *salt-and-pepper noise*), whilst preserving edges of objects well.

Implementation. We use the median filter implemented in SciPy [94]. In a 2×2 sliding window, the center pixel is always located in the lower right. When there are two equal-median values due to the even number of pixels in a window, we (arbitrarily) use the greater value as the median.

Non-local Smoothing

Non-local smoothing is different from local smoothing because it smooths over similar pixels in a much larger area instead of just nearby pixels. For a given image patch, non-local smoothing finds several similar patches in a large area of the image and replaces the center patch with the average of those similar patches. Assuming that the mean of the noise is zero, averaging the similar patches will cancel out the noise while preserving the edges of an object. Similar with local smoothing, there are several possible ways to weigh the similar patches in the averaging operation, such as Gaussian, mean, and median. We use a variant of the Gaussian kernel because it is widely used and allows to control the deviation from the mean. The parameters of a non-local smoothing method typically include the search window size (a large area for searching similar patches), the patch size and the filter strength (bandwidth of the Gaussian kernel). We will denote a filter as "non-local means (a-b-c)" where "a" means the search window $a \times a$, "b" means the patch size $b \times b$ and "c" means the filter strength.

Implementation. We use the fast non-local means denoising method implemented in OpenCV [79]. It first converts a color image to the CIELAB colorspace, then separately denoises its L and AB

components, then converts back to the RGB space.

4.2.3 Other Squeezing Methods

Our results in this chapter are limited to these simple squeezing methods, which are surprisingly effective on our test datasets. However, we believe many other squeezing methods are possible, and continued experimentation will be worthwhile to find the most effective squeezing methods.

One possible area to explore includes lossy compression techniques. Kurakin et al. explored the effectiveness of the JPEG format in mitigating the adversarial examples [55]. Their experiment shows that a very low JPEG quality (e.g. 10 out of 100) is able to destruct the adversarial perturbations generated by FGSM with ϵ=16 (at scale of [0,255]) for at most 30% of the successful adversarial examples. However, they didn't evaluate the potential loss on the accuracy of legitimate inputs.

Another possible direction is dimension reduction. For example, Turk and Pentland's early work pointed out that many pixels are irrelevant features in the face recognition tasks, and the face images can be projected to a feature space named *eigenfaces* [112]. Even though image samples represented in the *eigenface*-space loose the spatial information a CNN model needs, the image restoration through *eigenfaces* may be a useful technique to mitigate adversarial perturbations in a face recognition task.

4.3 Robustness

The previous section suggested that images, as used in classification tasks, contain many irrelevant features that can be squeezed without reducing recognizability. For feature squeezing to be effective in detecting adversarial examples (Figure 4.1), it must satisfy two properties: (1) on adversarial examples, the squeezing reverses the effects of the adversarial perturbations; and (2) on legitimate examples, the squeezing does not significantly impact a classifier's predictions. This section evaluates the how well different feature squeezing methods achieve these properties.

Threat model. In evaluating robustness, we assume a powerful adversary who has full access to a trained target model, but no ability to influence that model. For now, we assume the adversary is

Table 4.1: Summary of the target DNN models.

Dataset	Model	Top-1 Accuracy	Top-1 Mean Confidence	Top-5 Accuracy
MNIST	7-Layer CNN [15]	99.43%	99.39%	-
CIFAR-10	DenseNet [47, 68]	94.84%	92.15%	-
ImageNet	MobileNet [46, 69]	68.36%	75.48%	88.25%

not aware of feature squeezing being performed on the operator's side. The adversary tries to find inputs that are misclassified by the model using white-box attack techniques.

Although we analyze the robustness of standalone feature squeezers here, we do not propose using a standalone squeezer as a defense because an adversary may take advantage of feature squeezing in attacking a DNN model. For example, when facing binary squeezing, an adversary can construct an image by setting all pixel intensity values to be near 0.5. This image is entirely gray to human eyes. By setting pixel values to either 0.49 or 0.51 it can result in an arbitrary 1-bit filtered image after squeezing, either entirely white or black. Such an attack can easily be detected by our detection framework (Section 4.4), however. Section 4.4.4 considers how adversaries can adapt to our detection framework.

Target Models. We use three popular datasets for the image classification task: MNIST, CIFAR-10, and ImageNet. For each dataset, we set up a pre-trained model with the state-of-the-art performance. Table 4.1 summarizes the prediction performance of each model and its DNN architecture. Our MNIST model (a seven-layer CNN [15]) and CIFAR-10 model (a DenseNet [47, 68] model) both achieve prediction performance competitive with state-of-the-art results [9]. For the ImageNet dataset, we use a MobileNet model [46, 69] because MobileNets are more widely used on mobile phones and their small and efficient design make it easier to conduct experiments. The accuracy achieved by this model (88.25% top-5 accuracy), is below what can be achieved with a larger model such as Inception v3 [23, 106] (93.03% top-5 accuracy).

Attacks. We evaluate feature squeezing on all of the attacks described in Section 2.2.2 and summarized in Table 4.2. For each targeted attack, we try two different targets: the *Next* class ($t = L + 1 \mod \#classes$), and the least-likely class (LL), $t = \min(\hat{\mathbf{y}})$. Here t is the target class, L is the index of the ground-truth class and $\hat{\mathbf{y}}$ is the prediction vector of an input image. This gives eleven total attacks: the three untargeted attacks (FGSM, BIM and DeepFool), and two versions each of the four targeted attacks (JSMA, CW_∞, CW_2, and CW_0). We use the implementations of

Table 4.2: Evaluation of attacks.

	Configration			Cost (s)	Success Rate	Prediction Confidence	Distortion		
		Attack	Mode				ℓ_∞	ℓ_2	ℓ_0
MNIST	ℓ_∞	FGSM		0.002	46%	93.89%	0.302	5.905	0.560
		BIM		0.01	91%	99.62%	0.302	4.758	0.513
		CW_∞	Next	51.2	100%	99.99%	0.251	4.091	0.491
			LL	50.0	100%	99.98%	0.278	4.620	0.506
	ℓ_2	CW_2	Next	0.3	99%	99.23%	0.656	2.866	0.440
			LL	0.4	100%	99.99%	0.734	3.218	0.436
	ℓ_0	CW_0	Next	68.8	100%	99.99%	0.996	4.538	0.047
			LL	74.5	100%	99.99%	0.996	5.106	0.060
		JSMA	Next	0.8	71%	74.52%	1.000	4.328	0.047
			LL	1.0	48%	74.80%	1.000	4.565	0.053
CIFAR-10	ℓ_∞	FGSM		0.02	85%	84.85%	0.016	0.863	0.997
		BIM		0.2	92%	95.29%	0.008	0.368	0.993
		CW_∞	Next	225	100%	98.22%	0.012	0.446	0.990
			LL	225	100%	97.79%	0.014	0.527	0.995
	ℓ_2	DeepFool		0.4	98%	73.45%	0.028	0.235	0.995
		CW_2	Next	10.4	100%	97.90%	0.034	0.288	0.768
			LL	12.0	100%	97.35%	0.042	0.358	0.855
	ℓ_0	CW_0	Next	367	100%	98.19%	0.650	2.103	0.019
			LL	426	100%	97.60%	0.712	2.530	0.024
		JSMA	Next	8.4	100%	43.29%	0.896	4.954	0.079
			LL	13.6	98%	39.75%	0.904	5.488	0.098
ImageNet	ℓ_∞	FGSM		0.02	99%	63.99%	0.008	3.009	0.994
		BIM		0.2	100%	99.71%	0.004	1.406	0.984
		CW_∞	Next	211	99%	90.33%	0.006	1.312	0.850
			LL	269	99%	81.42%	0.010	1.909	0.952
	ℓ_2	DeepFool		60.2	89%	79.59%	0.027	0.726	0.984
		CW_2	Next	20.6	90%	76.25%	0.019	0.666	0.323
			LL	29.1	97%	76.03%	0.031	1.027	0.543
	ℓ_0	CW_0	Next	608	100%	91.78%	0.898	6.825	0.003
			LL	979	100%	80.67%	0.920	9.082	0.005

Results are for 100 seed images for each the DNN models described in Table 4.1. The *cost* of an attack generating adversarial examples is measured in seconds per sample. The ℓ_0 distortion is normalized by the number of pixels (e.g., 0.560 means 56% of all pixels in the image are modified).

FGSM, BIM and JSMA provided by the Cleverhans library [81]. For DeepFool and the three CW attacks, we use the implementations from the original authors [15, 75]. Our models and code for our attacks, defenses, and tests are available at `https://evadeML.org/zoo`. We use a PC equipped with an i7-6850K 3.60GHz CPU, 64GiB system memory, and a GeForce GTX 1080 to conduct the experiments.

For the seed images, we select the first 100 correctly predicted examples in the test (or validation) set from each dataset for all the attack methods, since some attacks are too expensive to run on all the

seeds. We adjust the applicable parameters of each attack to generate high-confidence adversarial examples, otherwise they would be easily rejected. This is because the three DNN models we use achieve high confidence of the top-1 predictions on legitimate examples (see Table 4.1; mean confidence is over 99% for MNIST, 92% for CIFAR-10, and 75% for ImageNet). In addition, all the pixel values in the generated adversarial images are clipped and squeezed to 8-bit-per-channel pixels so that the resulting inputs are within the possible space of images.

Table 4.2 reports results from our evaluation of the eleven attacks on three datasets. The success rate captures the probability an adversary achieves their goal. For untargeted attacks, the reported success rate is $1 - \textit{accuracy}$; for targeted attacks, it is only considered a success if the model predicts the targeted class. Most of the attacks are very effective in generating high-confidence adversarial examples against three DNN models. The CW attacks often produce smaller distortions than other attacks using the same norm objective, but are much more expensive to generate. On the other hand, FGSM, DeepFool, and JSMA often produce low-confidence adversarial examples. We exclude the DeepFool attack from the MNIST dataset because it generates images that appear unrecognizable to humans.[2] We do not have JSMA results for ImageNet because our 64GiB test machine runs out of memory.

4.3.1 Results

Table 4.3 summarizes the effectiveness of different feature squeezers on classification accuracy in our experiments.

Color Depth Reduction. The resolution of a specific bit depth is defined as the number of possible values for each pixel. For example, the resolution of 8-bit color depth is 256. Reducing the bit depth lowers the resolution and diminishes the opportunity an adversary has to find effective perturbations.

MNIST. The last column of Table 4.3 shows the binary filter (1-bit depth reduction) barely reduces the accuracy on the legitimate examples of MNIST (from 99.43% to 99.33% on the test set). The binary filter is effective on all the ℓ_2 and ℓ_∞ attacks. For example, it improves the accuracy on CW_∞ adversarial examples from 0% to 100%. The binary filter works well even for large ℓ_∞ distortions.

[2] We believe this is because the linear boundary assumption doesn't hold for the particular MNIST model and DeepFool fails to approximate the minimal perturbation.

Table 4.3: Model accuracy with feature squeezing

Dataset	Squeezer		ℓ_∞ Attacks				ℓ_2 Attacks			ℓ_0 Attacks				All Attacks	Legitimate
	Name	Parameters	FGSM	BIM	CW_∞		Deep-Fool	CW_2		CW_0		JSMA			
					Next	LL		Next	LL	Next	LL	Next	LL		
MNIST	None		54%	9%	0%	0%	-	0%	0%	0%	0%	27%	40%	13.00%	99.43%
	Bit Depth	1-bit	**92%**	**87%**	**100%**	**100%**	-	**83%**	**66%**	0%	0%	50%	49%	**62.70%**	99.33%
	Median Smoothing	2x2	61%	16%	70%	55%	-	51%	35%	39%	36%	62%	56%	48.10%	99.28%
		3x3	59%	14%	43%	46%	-	51%	53%	**67%**	**59%**	**82%**	**79%**	55.30%	98.95%
CIFAR-10	None		15%	8%	0%	0%	2%	0%	0%	0%	0%	0%	0%	2.27%	94.84%
	Bit Depth	5-bit	17%	13%	12%	19%	40%	40%	47%	0%	0%	21%	17%	20.55%	94.55%
		4-bit	21%	29%	69%	74%	72%	84%	84%	7%	10%	23%	20%	44.82%	93.11%
	Median Smoothing	2x2	**38%**	**56%**	**84%**	**86%**	**83%**	**87%**	83%	**88%**	**85%**	**84%**	**76%**	**77.27%**	89.29%
	Non-local Means	11-3-4	27%	46%	80%	84%	76%	84%	**88%**	11%	11%	44%	32%	53.00%	91.18%
ImageNet	None		1%	0%	0%	0%	11%	10%	3%	0%	0%	-	-	2.78%	69.70%
	Bit Depth	4-bit	5%	4%	66%	79%	44%	84%	82%	38%	67%	-	-	52.11%	68.00%
		5-bit	2%	0%	33%	60%	21%	68%	66%	7%	18%	-	-	30.56%	69.40%
	Median Smoothing	2x2	22%	28%	75%	81%	**72%**	81%	84%	**85%**	**85%**	-	-	**68.11%**	65.40%
		3x3	**33%**	**41%**	73%	76%	66%	77%	79%	81%	79%	-	-	67.22%	62.10%
	Non-local Means	11-3-4	10%	25%	**77%**	**82%**	57%	**87%**	**86%**	43%	47%	-	-	57.11%	65.40%

No results are shown for DeepFool on MNIST because of the adversarial examples it generates appear unrecognizable to humans; no results are shown for JSMA on ImageNet because it requires more memory than available to run. We do not apply the non-local smoothing on MNIST images because it is difficult to find similar patches on such images for smoothing a center patch.

This is because the binary filter squeezes each pixel into 0 or 1 using a cutoff 0.5 in the $[0, 1)$ scale. This means maliciously perturbing a pixel's value by ± 0.30 does not change the squeezed value of pixels whose original values fall into $[0, .20)$ and $[.80, 1)$. In contrast, bit depth reduction is not effective against ℓ_0 attacks (JSMA and CW_0) since these attacks make large changes to a few pixels that are not reversed by the bit depth squeezer. However, as we will show later, the spatial smoothing squeezers are often effective against ℓ_0 attacks.

CIFAR-10 and ImageNet. Because the DNN models for CIFAR-10 and ImageNet are more sensitive to perturbations, adversarial examples at very low ℓ_2 and ℓ_∞ distortions can be found. Table 4.3 includes the results of 4-bit depth and 5-bit depth filters in mitigating the adversaries for CIFAR-10 and ImageNet. In testing, the 5-bit depth filter increases the accuracy on adversarial inputs for several of the attacks (for example, increasing accuracy from 0% to 40% for the CW_2 Next class targeted attack), while almost perfectly preserving the accuracy on legitimate data (94.55% compared with 94.84%). The more aggressive 4-bit depth filter is more robust against adversaries (e.g., accuracy on CW_2 increases to 84%), but reduces the accuracy on legitimate inputs from 94.84% to 93.11%. We do not believe these results are good enough for use as a stand-alone defense (even ignoring the risk of adversarial adaptation), but they provide some insight why the method is effective as used in our detection framework.

Median Smoothing. The adversarial perturbations produced by the ℓ_0 attacks (JSMA and CW_0) are similar to *salt-and-pepper noise*, though it is introduced intentionally instead of randomly. Note

that the adversarial strength of an ℓ_0 adversary limits the number of pixels that can be manipulated, so it is not surprising that maximizing the amount of change to each modified pixel is typically most useful to the adversary. This is why the smoothing squeezers are more effective against these attacks than the color depth squeezers.

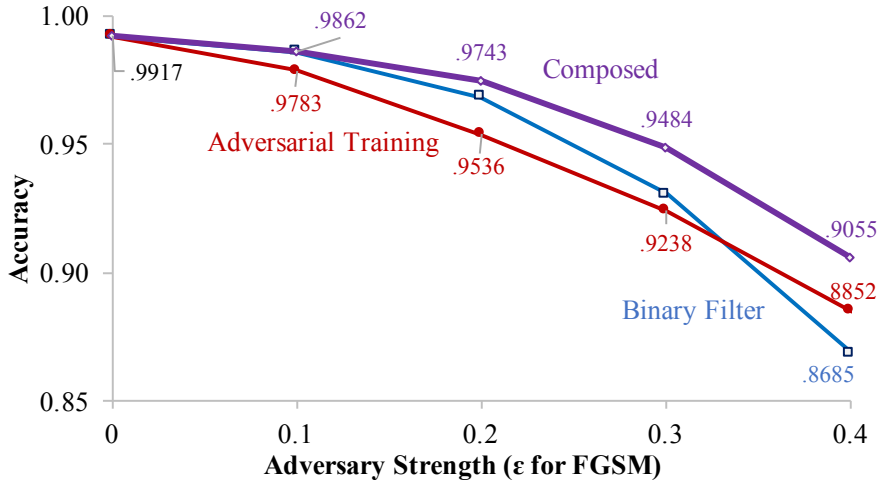
MNIST. We evaluate two window sizes on the MNIST dataset in Table 4.3. Median smoothing is the best squeezer for all of the ℓ_0 attacks (CW_0 and JSMA). The median filter with 2×2 window size performs slightly worse on adversarial examples than the one with 3×3 window, but it almost perfectly preserves the performance on the legitimate examples (decreasing accuracy from 99.43% to 99.28%).

CIFAR-10 and ImageNet. The experiment confirms the intuition that median smoothing can effectively eliminate the ℓ_0-limited perturbations. Without squeezing, the ℓ_0 attacks are effective on CIFAR-10, resulting in 0% accuracy for the original model ("None" row in Table 4.3). However, with a 2×2 median filter, the accuracy increases to over 75% for all the four ℓ_0 type attacks. We observe similar results on ImageNet, where the accuracy increases from 0% to 85% for the CW_0 attacks after median smoothing.

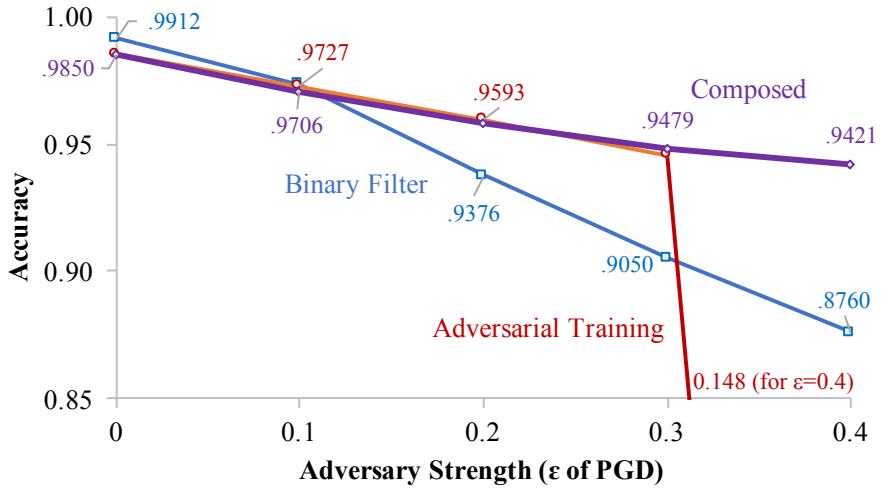
Non-local Smoothing. Non-local smoothing has comparable performance in increasing the accuracy on adversarial examples other than the ℓ_0 type. On the other hand, it has little impact on the accuracy on legitimate examples. For example, the 2×2 median filter decreases the accuracy on the CIFAR-10 model from 94.84% to 89.29% while the model with non-local smoothing still achieves 91.18%.

4.3.2 Combining with Adversarial Training

Since our approach modifies inputs rather than the model, it can easily be composed with any defense technique that operates on the model. The most successful previous defense against adversarial examples is adversarial training (Section 2.2.2). To evaluate the effectiveness of composing feature squeezing with adversarial training, we combined it with two different adversarial training methods: the FGSM-based version implemented in Cleverhans [81] and the PGD-based version implemented by Madry et al. [65]. We evaluated the accuracy on all 10,000 MNIST testing images and compared



(a) FGSM attacks.



(b) PGD attacks.

Figure 4.4: Composing adversarial training with feature squeezing. The horizontal axis is the adversary's strength (ϵ), increasing to the right. The adversarial training uses $\epsilon = 0.3$ for both FGSM are PGD. Composing the 1-bit filter with the adversarial-trained model often performs the best.

the three different configurations: 1. the base model with a binary filter; 2. the adversarial-trained model; 3. the adversarial-trained model with a binary filter.

Figure 4.4a shows that bit-depth reduction by itself often outperforms the adversarial training methods, and the composition is nearly always the most effective. For FGSM, binary filter feature squeezing outperforms adversarial training for ϵ values ranging from 0.1 to 0.3. This is the best case for adversarial training since the adversarially-trained model is learning from the same exact adversarial method (training is done with FGSM examples generated at $\epsilon = 0.3$) as the one used to produce adversarial examples in the test. Nevertheless, feature squeezing outperforms it, even at

$\epsilon = 0.3$ (93.03% accuracy on adversarial examples compared to 92.38%).

PGD-based adversarial training [65] does better, outperforming the simple binary filter feature squeezing for $\epsilon = 0.2$ and 0.3 but slightly reducing accuracy on legitimate ($\epsilon = 0$) examples, as shown in Figure 4.4b. Composing both methods typically leads to the highest accuracy or the one comparable to the best single approach. For example, both methods encounter significant drop on accuracy when $\epsilon = 0.4$: 87.60% with the binary filter and 14.84% with adversarial training. However, the composed method still achieves 94.21%.

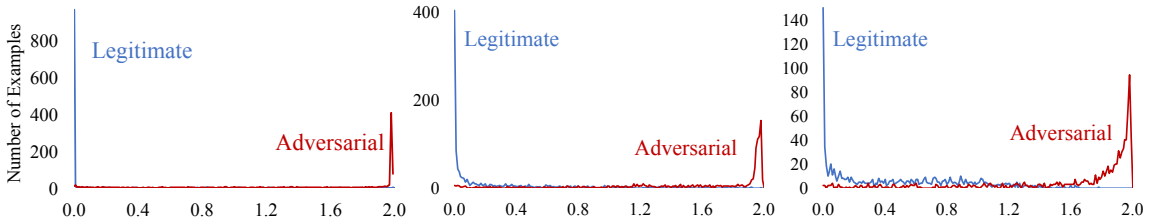
Feature squeezing is far less expensive than adversarial training. It is almost cost-free, as we simply insert a binary filter before the pre-trained MNIST model. On the other hand, adversarial training is very expensive as it requires both generating adversarial examples and retraining the classifier for many epochs. When its cost is not prohibitive, though, adversarial training is still beneficial since it can be combined with feature squeezing.

4.4 Detecting Adversarial Inputs

The results from Section 4.3 show that feature squeezing is capable of obtaining accurate model predictions for many adversarial examples with little reduction in accuracy for legitimate examples. This enables detection of adversarial inputs using the framework introduced in Figure 4.1. The basic idea is to compare the model's prediction on the original sample with the same model's prediction on the sample after squeezing. The model's predictions for a legitimate example and its squeezed version should be similar. On the contrary, if the original and squeezed examples result in dramatically different predictions, the input is likely to be adversarial. Figure 4.5 confirms this intuition visually by comparing the ℓ_1 distances between the predictions for squeezed and non-squeezed examples for legitimate and adversarial examples, and Table 4.4 shows the results of our experiments. The following subsections provide more details on our detection method, experimental setup, and discuss the results. Section 4.4.4 considers how adversaries may adapt to our defense.

4.4.1 Detection Method

A prediction vector generated by a DNN classifier normally represents the probability distribution how likely an input sample is to belong to each possible class. Hence, comparing the model's original



(a) MNIST examples. (b) CIFAR-10 examples. (c) ImageNet examples.

Figure 4.5: Differences in ℓ_1 distance between original and squeezed sample, for legitimate and adversarial examples across three datasets. The ℓ_1 score has a range from 0.0 to 2.0 . Each curve is fitted over 200 histogram bins each representing the ℓ_1 distance range of 0.01. Each sample is counted in the bin for the maximum ℓ_1 distance between the original prediction and the output of the best joint-detection squeezing configuration shown in Table 4.4. The curves for adversarial examples are for all adversarial examples, including unsuccessful ones (so the separation for successful ones is even larger than shown here).

prediction with the prediction on the squeezed sample involves comparing two probability distribution vectors. There are many possible ways to compare the probability distributions, such as the ℓ_1 norm, the ℓ_2 norm and K-L divergence [13]. For this work, we select the ℓ_1 norm[3] as a natural measure of the difference between the original prediction vector and the squeezed prediction:

$$score^{(\mathbf{x},\mathbf{x}_{squeezed})} = \|g(\mathbf{x}) - g(\mathbf{x}_{squeezed})\|_1 \tag{4.1}$$

Here $g(\mathbf{x})$ is the output vector of a DNN model produced by the softmax layer whose i^{th} entry describes the probability how likely input $\mathbf{x}$ is in the i^{th} class. The ℓ_1 score has a range from 0 to 2 for the prediction vectors. Higher scores mean there are greater differences between the original prediction and the squeezed prediction. The maximum value of 2 is reached when each prediction vector consists of a 1 and all zeros, but with different classes as the 1. Based on the accuracy results in Section 4.3, we expect the score to be small for legitimate inputs and large for adversarial examples. The effectiveness of detection depends on selecting a threshold value that accurately distinguishes between legitimate and adversarial inputs.

Even though we can select an effective feature squeezer for a specific type of adversarial method, an operator typically does not know the exact attack method that would be used in practice. Hence, we combine multiple feature squeezers for detection by outputting the maximum distance:

$$score^{joint} = \max\left(score^{(\mathbf{x},\mathbf{x}_{sq1})},\ score^{(\mathbf{x},\mathbf{x}_{sq2})}, \dots\right) \tag{4.2}$$

We chose the max operator based on the assumption that different squeezers will be effective for

[3] This turned out to work well, but it is certainly worth exploring in future work if other metrics can work better.

different types of perturbations, and that the effective squeezer can be identified through the highest ℓ_1 score. On the other hand, this may increase the false positive rate because the max operator also selects the most destructive squeezer on legitimate inputs. We found that using max is enough to allow a reasonable trade-off in the empirical results, but it may be worthwhile to investigate better ways to combine the squeezers in future work.

Figure 4.5 shows the histogram of $score^{joint}$ for both legitimate (blue) and adversarial examples (red) on the three datasets. The peak for legitimate examples is always near 0, and the peak for adversarial examples is always near 2. Picking a threshold value between the two peaks is a balance between high detection rates and acceptable false positive rates.

4.4.2 Experimental Setup

We report on experiments using all attacks from Section 4.3 with the three types of squeezers in different configurations.

Datasets. To get a balanced dataset for detection, we select the same number of legitimate examples from the test (or validation) set of each dataset. For each of the attacks in Section 4.3, we use the 100 adversarial examples generated for each attack in the robustness experiments. This results in 2,000 total examples for MNIST (of which 1,000 are legitimate examples, and 1,000 are adversarial), 2,200 examples for CIFAR-10 and 1,800 examples for ImageNet. We randomly split each detection dataset into two groups: one-half for training the detector and the remainder for validation. Note that some of the adversarial examples are failed adversarial examples which do not confuse the original model, so the number of successful adversarial examples varies slightly across the attacks.

Squeezers. We first evaluate how well each squeezing configuration does against adversarial examples generated by each attack method. Then, we consider the realistic scenario where the defender does not know that attack method used by the adversary and needs to select a configuration that works well against a distribution of possible attacks.

Training. The training phase of our detector is simply selecting an optimal threshold of $score^{joint}$. One typical practice is to find the one that maximizes the training accuracy. Since the actual expected distribution of samples is not balanced and mostly benign, however, a detector with high accuracy

but a high false positve rate would be useless for many security-sensitive tasks. Therefore, we instead select a threshold that targets a false positive rate below 5% by choosing a threshold that is exceeded by no more than 5% of legitimate samples. Note that the training threshold is set using only the legitimate examples, so does not depend on the adversarial examples. Consequently, our method is inexpensive in the training phase compared with other methods like *sample statistics* or *training a detector* as we introduce in Section 2.2.3, but could be more expensive than *training a detector* during inference due to the squeezing operations and the multiple inputs.

Validation. Next, we use the chosen threshold value to measure the detection rate on three groups: successful adversarial examples (SAEs), failed adversarial examples (FAEs), and legitimate examples (for false positive rate). Except when noted explicitly, "detection rate" means the detection rate on successful adversarial examples. It is important to distinguish failed adversarial examples from legitimate examples here since detecting failed adversarial examples is useful for detecting attacks early, whereas an alarm on a legitimate example is always undesirable and is counted as a false positive.

4.4.3 Results

Table 4.4 shows the detection rates for successful adversarial examples for each attack method with a variety of configurations. For each dataset, we first list the detection rate for several detectors built upon single squeezers. For each squeezing method, we tried several parameters and compare the performance for each dataset. The "Best Attack-Specific Single Squeezer" row gives the detection rate for the best single squeezer against a particular attack. This represents the (unrealistically optimistic) case where the model operator knows the attack type and selects a single squeezer for detection that may be different for each attack. Below this, we show the best result of joint detection (to be discussed later) with multiple squeezers where the same configuration must be used for all attacks.

The best bit depth reduction for MNIST is squeezing the color bits to one, which achieves at least 97.87% detection for all the ℓ_∞ and ℓ_2 attacks and 100% detection rate for seven of the attacks. It is not as effective on CW_0 attacks, however, since these attacks are making large changes to a small number of pixels. On the contrary, the 3×3 median smoothing is the most effective on detecting the ℓ_0 attacks with detection rates above 88%. This matches the observation from Table 4.3 that they

Table 4.4: Detection rate for squeezing configurations on successful adversarial examples.

	Configuration			ℓ_∞ Attacks				ℓ_2 Attacks			ℓ_0 Attacks				Overall Detection Rate
	Squeezer	Parameters	Threshold	FGSM	BIM	CW_∞		Deep Fool	CW_2		CW_0		JSMA		
						Next	LL		Next	LL	Next	LL	Next	LL	
MNIST	Bit Depth	1-bit	0.0005	**1.000**	**0.979**	**1.000**	**1.000**	-	**1.000**	**1.000**	0.556	0.563	**1.000**	**1.000**	0.903
		2-bit	0.0002	0.615	0.064	0.615	0.755	-	0.963	0.958	0.378	0.396	0.969	**1.000**	0.656
	Median Smoothing	2x2	0.0029	0.731	0.277	**1.000**	**1.000**	-	0.944	**1.000**	0.822	0.938	0.938	**1.000**	0.868
		3x3	0.0390	0.385	0.106	0.808	0.830	-	0.815	0.958	0.889	**1.000**	0.969	**1.000**	0.781
	Best Attack-Specific Single Squeezer		-	**1.000**	**0.979**	**1.000**	**1.000**	-	**1.000**	**1.000**	0.889	**1.000**	**1.000**	**1.000**	-
	Best Joint Detection (1-bit, 2x2)		0.0029	**1.000**	**0.979**	**1.000**	**1.000**	-	**1.000**	**1.000**	**0.911**	0.938	**1.000**	**1.000**	**0.982**
CIFAR-10	Bit Depth	1-bit	1.9997	0.063	0.075	0.000	0.000	0.019	0.000	0.000	0.000	0.000	0.000	0.000	0.013
		2-bit	1.9967	0.083	0.175	0.000	0.000	0.000	0.000	0.000	0.000	0.018	0.000	0.000	0.022
		3-bit	1.7822	0.125	0.250	0.755	0.977	0.170	0.787	0.939	0.365	0.214	0.000	0.000	0.409
		4-bit	0.7930	0.125	0.150	0.811	0.886	0.642	0.936	0.980	0.192	0.179	0.041	0.000	0.446
		5-bit	0.3301	0.000	0.050	0.377	0.636	0.509	0.809	0.878	0.096	0.018	0.041	0.038	0.309
	Median Smoothing	2x2	1.1296	0.188	**0.550**	**0.981**	**1.000**	0.717	0.979	**1.000**	**0.981**	**1.000**	**0.837**	**0.885**	0.836
		3x3	1.9431	0.042	0.250	0.660	0.932	0.038	0.681	0.918	0.750	0.929	0.041	0.077	0.486
	Non-local Mean	11-3-2	0.2770	0.125	0.400	0.830	0.955	0.717	0.915	0.939	0.077	0.054	0.265	0.154	0.484
		11-3-4	0.7537	0.167	0.525	0.868	0.977	0.679	0.936	**1.000**	0.250	0.232	0.245	0.269	0.551
		13-3-2	0.2910	0.125	0.375	0.849	0.977	0.717	0.915	0.939	0.077	0.054	0.286	0.173	0.490
		13-3-4	0.8290	0.167	0.525	0.887	0.977	0.642	0.936	**1.000**	0.269	0.232	0.224	0.250	0.547
	Best Attack-Specific Single Squeezer		-	0.188	**0.550**	**0.981**	**1.000**	0.717	0.979	**1.000**	**0.981**	**1.000**	**0.837**	**0.885**	-
	Best Joint Detection (5-bit, 2x2, 13-3-2)		1.1402	**0.208**	**0.550**	**0.981**	**1.000**	**0.774**	**1.000**	**1.000**	**0.981**	**1.000**	**0.837**	**0.885**	**0.845**
ImageNet	Bit Depth	1-bit	1.9942	0.151	0.444	0.042	0.021	0.048	0.064	0.000	0.000	0.000	-	-	0.083
		2-bit	1.9512	0.132	0.511	0.500	0.354	0.286	0.170	0.306	0.218	0.191	-	-	0.293
		3-bit	1.4417	0.132	0.556	**0.979**	**1.000**	0.476	0.787	**1.000**	0.836	**1.000**	-	-	0.751
		4-bit	0.7996	0.038	0.089	0.813	**1.000**	0.381	0.915	**1.000**	0.727	**1.000**	-	-	0.664
		5-bit	0.3528	0.057	0.022	0.688	0.958	0.310	**0.957**	**1.000**	0.473	**1.000**	-	-	0.606
	Median Smoothing	2x2	1.1472	0.358	0.422	0.958	**1.000**	0.714	0.894	**1.000**	**0.982**	**1.000**	-	-	0.816
		3x3	1.6615	0.264	0.444	0.917	0.979	0.500	0.723	0.980	0.909	**1.000**	-	-	0.749
	Non-local Mean	11-3-2	0.7107	0.113	0.156	0.813	0.979	0.357	0.936	0.980	0.418	0.830	-	-	0.618
		11-3-4	1.0387	0.208	0.467	0.958	**1.000**	0.548	0.936	**1.000**	0.673	0.957	-	-	0.747
		13-3-2	0.7535	0.113	0.156	0.813	0.979	0.357	0.936	0.980	0.418	0.851	-	-	0.620
		13-3-4	1.0504	0.226	0.444	0.958	**1.000**	0.548	0.936	**1.000**	0.709	0.957	-	-	0.751
	Best Attack-Specific Single Squeezer		-	0.358	0.556	**0.979**	**1.000**	0.714	**0.957**	**1.000**	**0.982**	**1.000**	-	-	-
	Best Joint Detection (5-bit, 2x2, 11-3-4)		1.2128	**0.434**	**0.644**	**0.979**	**1.000**	**0.786**	0.915	**1.000**	**0.982**	**1.000**	-	-	**0.859**

have different strengths for improving the model accuracy. For MNIST, there is at least one squeezer that provides good ($> 91\%$) detection results for all of the attacks.

For CIFAR-10, 2×2 median smoothing is the best single squeezer for detecting every attack except DeepFool, which is best detected by non-local means. This is consistent with the robustness results in Table 4.3. For the ImageNet dataset, we find several different squeezers are similarly effective on each attack type. For example, the CW_2-LL attack can be detected 100% of the time with several bit depth filters, 2×2 median smoothing, and some of the non-local mean filters.

The third column in the table gives the distance threshold setting needed to satisfy the maximum false positive rate of 5% in the training dataset. These threshold values provide some insight into how well a particular squeezer distinguishes between adversarial and legitimate examples. For the binary filter on MNIST, a tiny threshold value of 0.0005 was sufficient to produce a false positive rate below 5%, which means the squeezing has negligible impact on the legitimate examples: 95% of the legitimate examples have the ℓ_1-based distance score below 0.0005. On the other hand, the best median smoothing filter (2×2) on MNIST needs a larger threshold value 0.0029 to achieve a similar false positive rate, which means it is slightly more destructive than the binary filter on

Table 4.5: Summary results for the best joint detectors.

Dataset	Detector	Threshold	Detection (SAEs)	Detection (FAEs)	FPR	ROC-AUC (Excluding FAEs)
MNIST	Bit Depth (1-bit), Median (2×2)	0.0029	98.15%	20.00%	3.98%	99.44%
CIFAR-10	Bit Depth (5-bit), Median (2×2), Non-local Means (13-3-2)	1.1402	84.53%	22.22%	4.93%	95.74%
ImageNet	Bit Depth (5-bit), Median (2×2), Non-local Means (11-3-4)	1.2128	85.94%	25.00%	8.33%	94.24%

SAE: successful adversarial example. FAE: failed adversarial example.

the legitimate examples. The more aggressive median smoothing with 3×3 window results in an even higher threshold 0.039, because the legitimate examples could get over-squeezed to the target classifier. A lower threshold is always preferred for detection since it means the detector is more sensitive to adversarial examples.

For some of the attacks, none of the feature squeezing methods work well enough for the color datasets. The worst cases, surprisingly, are for FGSM and BIM, two of the earlier adversarial methods. The best single-squeezer-detection only recognizes 18.75% of the successful FGSM examples and 55% of BIM examples on the CIFAR-10 dataset, while the detection rates are 35.85% and 55.56% on ImageNet. We suspect the reason the tested squeezers are less effective against these attacks is because they make larger perturbations than the more advanced attacks (especially the CW attacks), and the feature squeezers we use are well suited to mitigating small perturbations. Understanding why these detection rates are so much lower than the others, and developing feature squeezing methods that work well against these attacks is an important avenue for future research.

Joint-Detection with Multiple Squeezers.

By comparing the last two rows of each dataset in Table 4.4, we see that joint-detection often outperforms the best detector with a single squeezer. For example, the best single-squeezer-detection detects 97.87% of the CW_2-Next examples for CIFAR-10, while joint detection detects 100%.

The main reason to use multiple squeezers, however, is because this is necessary to detect unknown attacks. Since the model operator is unlikely to know what attack adversaries may use, it is important to be able to set up the detection system to work well against any attack. For each data set, we try several combinations of the three squeezers with different parameters and find out the configuration that has the best detection results across all the adversarial methods (shown as the "Best Joint

Detection" in Table 4.4, and summarized in Table 4.5). For MNIST, the best combination was the 1-bit depth squeezer with 2×2 median smoothing (98.15% detection), combining the best parameters for each type of squeezer. For the color image datasets, different combinations were found to outperform combining the best squeezers of each type. The best joint detection configuration for ImageNet (85.94% detection) includes the 5-bit depth squeezer, even though the 3-bit depth squeezer was better as a single squeezer.

Since the joint detector needs to maintain the 5% false positive rate requirement, it has a higher threshold than the individual squeezers. This means in some cases its detection rate for a particular attack will be worse than the best single squeezer achieves. However, comparing the "Best Attack-Specific Single Squeezer" and "Best Joint Detection" rows in Table 4.4 reveals that the joint detection is usually competitive with the best single squeezers over all the attacks. For MNIST, the biggest drop is for detection rate for CW_0 (LL) attacks drops from 100% to 93%; for CIFAR-10, the joint squeezer always outperforms the best single squeezer; for ImageNet, the detection rate drops for CW_2 (Next) (95% to 91%). For simplicity, we use a single threshold across all of the squeezers in a joint detector; we expect there are better ways to combine multiple squeezers that would use different thresholds for each of the squeezers to avoid this detection reduction, and plan to study this in future work.

We report ROC-AUC scores in Table 4.5 excluding the failed adversarial examples from consideration, since it is not clear what the correct output should be for a failed adversarial example. Our joint-detector achieves around 95% ROC-AUC score on CIFAR-10 and ImageNet. The ROC-AUC of the detector is as high as 99.44% for MNIST. The false positive rates on legitimate examples are all near 5%, which is expected considering how we select a threshold value in the training phase. The detection rate for the best configuration on successful adversarial examples exceeds 98% for MNIST using a 1-bit filter and a 2×2 median filter and near 85% for the other two datasets using a combination of three types feature squeezing methods with different parameters. The detection rates for failed adversarial examples are much lower than those for successful adversarial examples, but much higher than the false positive rate for legitimate examples. This is unsurprising since FAEs are attempted adversarial examples, but since they are not successful the prediction outputs for the unsqueezed and squeezed inputs are more similar.

We compare our results with MagNet [72] in Table 4.6. We configured the MagNet detectors on two datasets following the description in their paper and reported the detection performance with

Table 4.6: Comparison with MagNet.

Dataset	Method	AEs	SAEs
MNIST	Feature Squeezing	69.08%	78.75%
	MagNet	**91.77%**	**95.61%**
CIFAR-10	Feature Squeezing	**60.87%**	**61.88%**
	MagNet	50.36%	50.46%

False positive rate for both is 0.40% on MNIST, 1.28% on CIFAR-10.

our target models and the detection dataset. In order to fairly compare the detection rates, we adjusted the threshold values of our detectors accordingly on the two datasets to produce the false positive rates matching the MagNet results: 0.40% for MNIST and 1.28% for CIFAR-10. MagNet achieves higher detection rates on MNIST (91.77% over 69.08%), while our method outperformed on CIFAR-10 (60.87% over 50.36%). The detection rates excluding failed adversarial examples were similar. MagNet's detection rates on our adversarial examples for MNIST are impressive, and superior to what the best feature squeezing configuration achieves. However, this advantage does not apply to CIFAR-10. Further, it is more expensive to use MagNet because it requires training an autoencoder on a whole dataset. In addition, the MagNet pipeline is end-to-end differentiable, making it vulnerable to trivial white-box adversary.

4.4.4 Adversarial Adaptation

So far, we have only considered static adversaries who do not adapt to attack our feature squeezing method directly. Now, we consider adaptive adversaries who have full knowledge of the defense. To be successful against our detection framework, an adversary needs to find an input where the original classifier produces the wrong output and the ℓ_1 score between the model's predictions on squeezed and original inputs is below the detection threshold. This is a much harder problem than just finding an adversarial example, as is supported by our experimental results.

He et al. [43] recently proposed an adaptive attack which can successfully find adversarial examples that defeat one configuration of a feature squeezing defense.[4] The approach finds adversarial examples that both confuse the original model and have a $score^{joint}$ lower than a pre-selected threshold for squeezed inputs. Their approach adapts the CW_2 attack by adding a penalty term for the ℓ_1 prediction distance. It requires that all the feature squeezing operators are differentiable so that it is possible

[4] This work was done following initial public reports on the work in our papers [120, 121]; we shared details of our approach and code with the authors of [43], and much appreciate their sharing their implementation with us to enable the experiments reported here.

to compute the gradient of the loss function in the optimization process. For the non-differentiable feature squeezers such as the bit depth reduction, their approach requires restarting the algorithm several times with random initialization and hoping it finds an example that is resilient against the non-differentiable squeezers. This means the attack is non-deterministic and more time-consuming in face of non-diffrentiable components [43]. The attack takes roughly 20 seconds on each MNIST sample, which is around 60 times slower than the original CW_2 attack.

We only evaluate their adaptive attack on the MNIST dataset, because we currently don't have a Tensorflow implementation of the non-local means filter used on CIFAR-10 and ImageNet.[5] We adapted their attacking code to our MNIST model and the joint-detector we report in Table 4.5.

In addition to the untargeted attack He et al. tested, we also tested the two targeted attacks mentioned earlier in Section 4.3. We used the same parameters as He et al. [43] in generating the adaptive adversarial examples except that we change the target threshold value to 0.0029 that is used by the detector in this chapter.[6] The seed examples are the same 100 images used in Section 4.3. With the exact threshold we use in this chapter as the target value, the success rate of the untargeted attack is 68% (compared with 100% success rate in He et al.'s report [43]). The success rates of the targeted attacks are even lower: 44% for the targeted-next attack and 24% for the targeted-ll attack. We believe these lower adversarial success rates are due to using a different MNIST model (representative of a state-of-the-art model) and our detector has a much smaller threshold value (0.0029) compared to the one they used (over 0.3076).

However, the success rates reported in the adaptive attack do not provide a useful measure of robustness because the discussed adaptive adversary is allowed to modify any pixel arbitrarily. Given unlimited perturbation bounds, one could always force a model to misclassify an example into any class, but often by producing images that are unrecognizable or obviously suspicious to humans. Figure 4.6 shows typical examples produced by the adaptive attack. In order to understand the robustness of our detector in a more realistic setting, we clipped the perturbations with different ϵ values ranging from 0 to 1. Figure 4.7 shows the adversary success rates as the adversary's strength ϵ increases to the right. At 0.3, the ϵ value considered a reasonable limit on what is a permissible adversarial example by many papers [65], the adversary's success rate is 0% for both of the target attacks and 1% for the untargeted attack.

[5] He et al. reported results for CIFAR-10 with only bit depth reduction and median smoothing [43]. These results were similar to the results they reported on MNIST.

[6] He et al.'s target detector was based on an earlier version of this work that used 0.3076 as threshold and uses a slightly different max function in combining multiple squeezers.

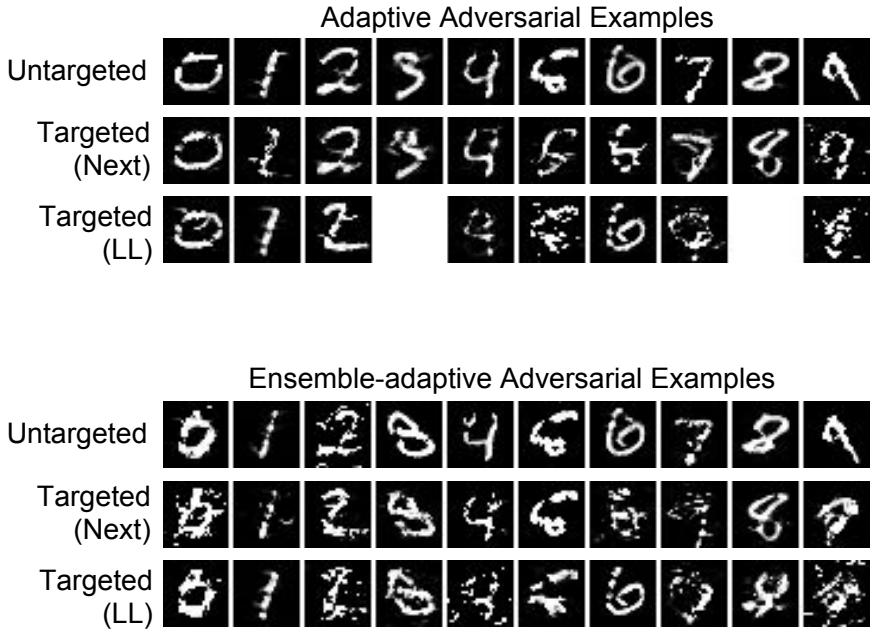


Figure 4.6: Adversarial examples generated by the adaptive adversary. The images are randomly sampled from the successful adversarial examples generated by the adaptive adversarial methods. No successful adversarial examples were found for Targeted (LL) 3 or 8. The average ℓ_2 norms of the successful adversarial examples are respectively 2.80, 4.14, 4.67 for the untargeted, targeted (next) and targeted (ll) examples; while the corresponding values are 3.63, 5.48, 5.76 for the ensemble-adaptive adversarial examples. The average ℓ_∞ norm are 0.79, 0.89, 0.88 for the adaptive adversarial examples; while the corresponding values are 0.89, 0.95 and 0.96 for the ensemble-adaptive adversarial examples.

Countermeasures. One obvious strategy to combat adaptive adversaries is to introduce randomness in the squeezing method. This is very different from attempts to obfuscate models, which have been shown vulnerable to transfer attacks. Instead, we can use cryptographic randomness to make the deployed framework unpredictable, since the adversary's search requires knowledge of the exact squeezing operation. The defender has many opportunities to use randomness in selecting squeezing parameters. For example, instead of using a fixed 0.5 threshold for the 1-bit filter, using $0.5 \pm rand(0.1)$ (which could be done with a different random value for each pixel), or selecting random regions for the median smoothing instead of a fixed 2×2 region).

We conducted an experiment in which the cutoff value of the binary filter follows a normal distribution with 0.5 mean and standard deviation 0.0625. The success rates (with no limit on perturbation magnitude) decrease from $(68\%, 44\%, 24\%)$ to $(17.0 \pm 2.4\%, 16.2 \pm 2.6\%, 7.6 \pm 2.2\%)$ (respectively for untargeted, targeted to next class, and targeted to least likely; measured over 10 executions).

An adversary may attempt to adapt to the randomness by attacking an ensemble of random squeezers.

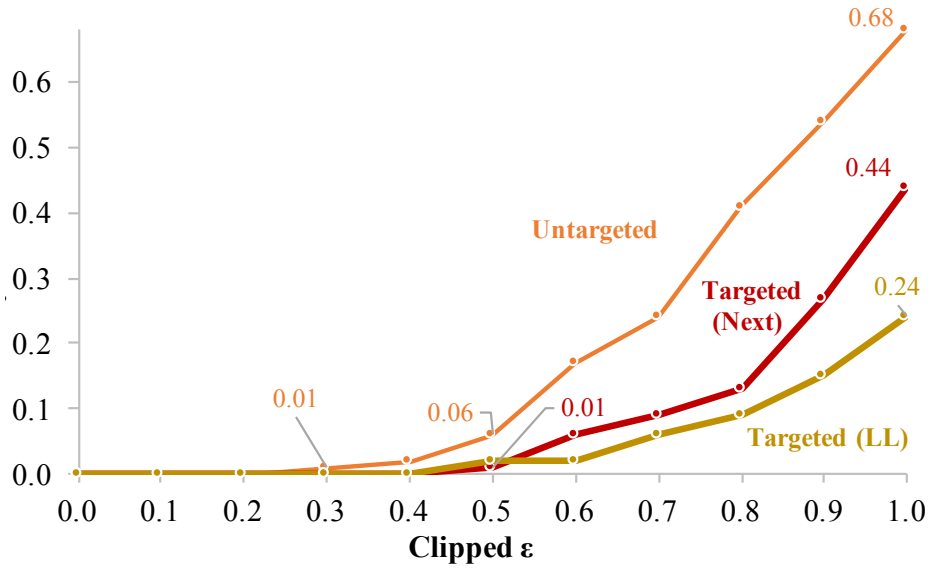


Figure 4.7: Adaptive adversary success rates.

We consider an ensemble-adaptive adversary that considers three thresholds of the binary filter together: 0.4, 0.5 and 0.6. The success rates increase to $(46.5 \pm 2.1\%, 34.5 \pm 3.5\%, 28.5 \pm 2.1\%)$ measured over 10 executions. However, the perturbations became even larger, resulting in many unrecognizable and suspicious-looking images (shown in the right part of Figure 4.6).

4.5 Conclusion

The effectiveness of feature squeezing seems surprising since it is so simple and inexpensive compared to other proposed defenses. Developing a theory of adversarial examples remains an illusive goal, but our intuition is that the effectiveness of squeezing stems from how it reduces the search space of possible perturbations available to an adversary.

Although we have so far only experimented with image classification models, the feature-squeezing approach could be used in many domains where deep learning is used. For example, Carlini et al. demonstrated that lowering the sampling rate helps to defend against the adversarial voice commands [16]. Hosseini et al. proposed that correcting the spelling on inputs before they are provided to a character-based toxic text detection system can defend against adversarial examples [45].

As discussed in Section 4.4.4, feature squeezing is not immune to adversarial adaptation, but it substantially changes the challenge an adversary faces. Our general detection framework opens a

new research direction in defending against adversarial examples and understanding the limits of deep neural networks in adversarial contexts.

Impact. Our *Feature Squeezing* framework, as one of the earliest adversarial detection work, is often discussed and compared in other research papers [6, 41, 43, 61, 63, 71, 74, 89, 97, 99, 102, 109, 127]. Several survey papers cited *Feature Squeezing* as an effective and representative work [22, 62, 91]. In addition, *Feature Squeezing* has been implemented by others in several open-source adversarial machine learning toolboxes, making it more accessible to general users [2, 78].

Chapter 5

Provable Robustness

The robustness of neural network models against adversarial examples is a challenging problem. Simple and inexpensive pre-processing methods have been found to be an effective defense against adversarial examples empirically in Chapter 4 and other studies, but provide no guarantee of robustness against more sophisticated adversaries.

5.1 Introduction

Simple pre-processing methods have drawn a lot of attention in the research community as a promising method to defend deep neural networks against adversaries. Several studies have used simple pre-processing to thwart adversarial examples, such as bit depth reduction [123], thermometer encoding [14], defensive quantization [61], random resizing and croping [118], smoothing [123] and quilting [41].

However, subsequent studies have pointed out that pre-processing-based methods are often vulnerable to adaptive attacks [7, 18, 43]. Hence, it is unclear if simple pre-processing can be used to provide a strong defense, or merely can thwart some fixed attacks.

The goal of this chapter is to develop a theoretical basis for the impact of simple pre-processing on model robustness.

Provable robustness is a research direction to disrupt the arms race. Researchers have extended formal methods to verify the robustness properties of neural networks [50,110,115], so that we can precede an adversary in finding the weaknesses of a model. Although formal methods can verify robustness against restricted adversaries, they do not suggest any approach to improve it. Therefore, researchers have used robust optimization to improve the certifiable robustness by minimizing some over-approximation of prediction errors [37,51,88].

In this chapter, we combine the idea of simple pre-processing with provable robustness methods and develop novel techniques to train provably robust models incorporating simple pre-processing. We focus on the simple pre-processing of *bit depth reduction* on image classification tasks in this work and find that it not only helps train accurate and robust models but also improves the state-of-the-art certifiable defense.

Our work is distinct from other provable robustness work on randomized simple pre-processing [24, 126]. Since randomized simple pre-processing behaves differently every time, those works generate probabilistic robustness proofs. In contrast, we use deterministic simple pre-processing and provide sound robustness guarantees.

The contributions of this chapter include:

- We propose a framework to train robust machine learning models by incorporating simple pre-processing with provable robust methods. We feed many inputs to a simple pre-processing function and measure the adversarial capability in several ℓ_p-norms and ℓ_0-"norm" (Section 5.3). If we observe any reduced adversarial capability after pre-processing, we propagate the interval bound of inputs through the pre-processing function as the starting point to train robust certificates (Section 5.4.1). If we observe reduced adversarial capability in the ℓ_p-norm, we train a robust model simply by regularizing the dual ℓ_q-norm of weights (Section 5.4.2).

- We validate our framework with experimental results on MNIST and CIFAR-10 datasets. The trained models are examined by an exact verifier to prove the robustness on many test examples. (Section 5.2.1). We assume an adversary bounded by ℓ_∞ distance and choose *bit depth reduction* as a simple pre-processing functions. We show that the weight regularization method alone produces accurate MNIST models with state-of-the-art verified robustness (Section 5.5.2). We also show that the transformed adversarial bound improves the results of the state-of-the-art certification method on MNIST and CIFAR-10 (Section 5.5.1).

5.2 Provable Robustness Methods

Researchers have proposed several techniques to gain provable robustness since the empirical results suffer from endless arms race with adversaries. We categorize existing works into two groups: (1) *verification*, which employs formal methods to exactly model the robustness properties of neural networks and uses off-the-shelf solvers to verify the properties; (2) and *certification*, which relies on a differentiable over-approximation of the model output.

5.2.1 Formal Verification

The formal verification of a system contains two steps. First, we use a modeling language to create the desired specifications of a system. Second, we use a solver to either prove the specifications are satisfied or to find a counterexample.

Regarding the robustness of neural network, we can use various modeling languages, such as Satisfiability Modulo Theories (SMT) [50] or Mixed Integer Linear Programming (MILP) [110] to describe the desired properties. Typically, we assume there is an adversarial example within the adversarial norm ball of a normal example that would make the neural network predict a different class. We then use a solver to search such an adversarial example. If the solver reports that such an adversarial example does not exist, it proves that the model is robust with respect to the input. If it finds a counter example, it proves the model is not robust. Next, we elaborate on the MILP-based method which is used in this work.

MILP. Tjeng et al. has shown that MILP is a powerful modeling language for neural network robustness [110]. MILP is a useful extension to Linear Programming (LP) that also allows integer variables. We can encode a neural network and the robustness property as MILP formulas. The formulation of linear layers such as the fully connected layer and the convolutional layer is straightforward in LP. To incorporate non-linear operations, such as the ReLU activation and the max function, we need to introduce integer variables in the formulation.

MILP Solver. The MILP problem is known to be NP-complete. However, there are many established heuristics that help to solve most MILP instances that are found in typical problems efficiently [4]. Given a very hard MILP problem, we can first perform some presolve methods to simplify

the formulas by removing unnecessary constraints and variables. Second, we can use LP relaxation and *branch-and-bound* to convert the problem into many solvable LP sub-problems organized in a search tree. Third, we can use *cutting planes* to recognize the infeasible zones and reduce the search effort on each sub-problem.

We note that MILP solving in this context is different from the traditional optimization scenarios. Our goal is to prove that there is no feasible solution as an adversarial example that satisfies all constraints. We stop the search as soon as we find one incumbent solution, regardless of whether it is optimal.

MILP solving is sound and complete in theory. However, it occasionally returns incorrect results in practice due to the inexact floating point representation in modern computers. There are some heuristics to avoid numerical errors, but the soundness of MILP solvers is in general not guaranteed [1]. Verifying an adversarial example reported by the solver is easy. However, it will be difficult (NP-complete) to check the correctness if the solver says that the model is robust and it can not find any adversarial example. We have encountered concrete examples where the verification was unsound, and perform weight pruning in Section 5.5 as the remedy.

5.2.2 Robust Certification

Certification is the other method to prove the robustness of neural networks. It is sound but incomplete. The basic idea of certification is to give an over-approximation of a neural network's output assuming some bounded adversary. We can safely conclude that one model is robust against a bounded adversary with respect to an input example if the over-approximation is still a correct prediction. Unlike verification methods, it is easy to verify an input is invulnerable given a robust certificate, typically with some efficient matrix multiplication operations.

Researchers have proposed several certification methods using semi-definite relaxation [88], convex outer approximation [51] or interval bound propagation (IBP) [37]. Since these certificates are differentiable, it is straightforward to develop a robust training method by adding the certification term in the loss function. As a result, we can train a robust model that will be able to issue robust certificates to many inputs, though we have to sacrifice some model capacity due to the over-approximation in generating certificates.

Table 5.1: Comparison between verification and certification.

	Verification	Certification
Core Technique	Formal methods	Robust optimization
Preserve Model Capacity	✓	✗
Computational Cost	Expensive	Cheaper
Verifiable Result	✗	✓
Enable Robust Training	✗	✓
Soundness	✓	✓
Practical Soundness	Not guaranteed	✓
Completeness	✓	✗
Practical Completeness	Not guaranteed	✗
Examples	Reluplex [50], MIPVerify [110]	Raghunathan et al. [88], Wong and Kolter [51], IBP [37]

5.2.3 Comparison

Even though verification and certification have the same goal of proving the robustness of neural network models, they are distinct from each other in many ways. Table 5.1 summarizes the differences of the two techniques.

The verification-based methods have two advantages over certification in theory. First, they are sound and complete because of the exact modeling languages. Second, they do not have to sacrifice model capacity and can work with pre-trained models.

However, the theoretical advantages of verification are often not easily achievable in practice. First, the soundness is not guaranteed if floating point numbers are used in the model. We may use fixed precision numbers to make up the soundness, but the solving could be slower in several orders of magnitude. Second, completeness is not guaranteed due to the nature of NP-complete problems. We should expect that verification could take unreasonably long time occasionally, which makes it not complete in practice. As a result, we typically need to limit the model capacity to make verification affordable.

In contrast, certification-based methods are preferred for most practical use. First, they are sound, and the result is easily verifiable, which enables the issuance of robust certificates. Second, even though they are incomplete in theory, we can adapt them into robust training methods to improve the completeness as well as the model's actual robustness. The practical completeness of certification could be even better than that of verification-based methods on large scale models.

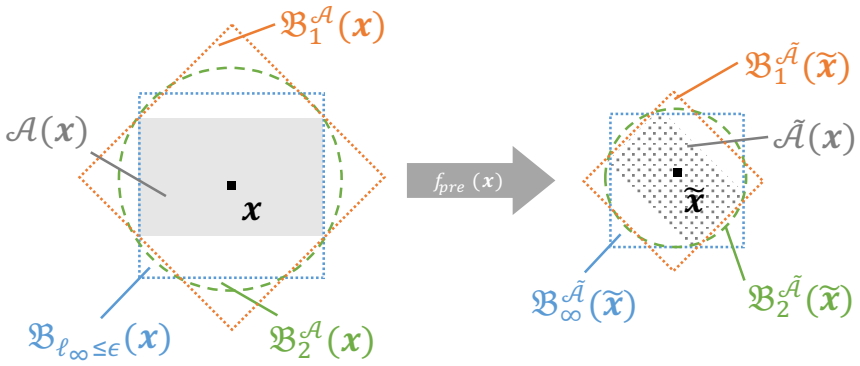


Figure 5.1: Simple pre-processing influences the adversary capability on input $\boldsymbol{x}$.

In our experiments, we combine the two techniques to train and verify robust neural network models. We use certification methods to train a robust model first. While it may efficiently certify the robustness of many input examples, we also employ verification methods to verify the uncertified cases to tighten the robustness error bounds further.

5.3 Adversarial Capability Measurement

We consider an adversary whose strength is bounded by the ℓ_∞-norm and investigate how pre-processing influences *adversarial capability*, which we define next.

5.3.1 Definitions

We illustrate how we define and measure *adversarial capability* in this section.

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be the considered d-dimensional input space. For any $\boldsymbol{x} \in \mathcal{X}$ and $\epsilon > 0$, we use $\mathcal{B}_{\ell_p \leq \epsilon}(\boldsymbol{x}) = \{\boldsymbol{x}' \in \mathcal{X} : \|\boldsymbol{x}' - \boldsymbol{x}\|_p \leq \epsilon\}$ to denote the ℓ_p-norm ball around $\boldsymbol{x}$ with radius ϵ in $\mathcal{X}$.

Let's assume an adversary bounded by $\ell_\infty \leq \epsilon$, *i.e.* the *adversarial strength* is ϵ in ℓ_∞-norm. As illustrated in the left part of Figure 5.1, given a seed example $\boldsymbol{x}$, we can get an adversarial norm ball $\mathcal{B}_{\ell_\infty \leq \epsilon}(\boldsymbol{x})$ centered at $\boldsymbol{x}$. Since data points usually have interval bounds, such as the common interval $[0, 1]$ constraint for image pixels, we denote the feasible set of the adversarial norm ball as $\mathcal{A}(\boldsymbol{x})$, which we define as *adversarial capability*.

Note that *adversarial strength* can be arbitrarily large, but the *adversarial capability* is bounded by a domain. For example, an ℓ_∞ adversary with adversarial strength $\epsilon = 3$ should have the same *adversarial capability* as one with $\epsilon = 1$ because the adversarial perturbations will be truncated to fit $[0, 1]$.

Even though $\mathcal{A}(\boldsymbol{x})$ is a subset of an ℓ_∞-norm ball, we have other ℓ_p-norm balls that cover $\mathcal{A}(\boldsymbol{x})$. We measure *adversarial capability* in ℓ_p-norm as the minimal radius $r_p^{\mathcal{A}}(\boldsymbol{x})$ of an ℓ_p-norm ball centered at $\boldsymbol{x}$ that covers $\mathcal{A}(\boldsymbol{x})$, such that $r_p^{\mathcal{A}}(\boldsymbol{x}) = \min\{s > 0 : \mathcal{A}(\boldsymbol{x}) \subseteq \mathcal{B}_{\ell_p \leq s}(\boldsymbol{x})\}$. We measure the adversarial capability in different ℓ_p-norms for p in $\{0, 1, 2, \infty\}$[1].

As in the right part of Figure 5.1, given a pre-processing function $f_{pre} : \mathcal{X} \to \mathcal{X}$, we transform $\boldsymbol{x}$ into $\tilde{\boldsymbol{x}}$ and the corresponding feasible set $\mathcal{A}(\boldsymbol{x})$ into $\tilde{\mathcal{A}}(\boldsymbol{x})$. We measure the *adversarial capability* after simple pre-processing as the minimal radius $r_p^{\tilde{\mathcal{A}}}(\tilde{\boldsymbol{x}})$ of an ℓ_p-norm ball centered at $\tilde{\boldsymbol{x}}$ that covers $\tilde{\mathcal{A}}(\boldsymbol{x})$.

We further extend the concept of *adversarial capability* on a single example to a whole dataset. We first compute the minimal radius $r_p^{\mathcal{A}}(\boldsymbol{x})$ for each example, then average the minimal radiuses to obtain the adversarial capability expectation over the dataset. The adversarial capability expectation after a pre-processing function is calculated in the same way. This provides an intuitive measure of the size of the effective search space available to a strength-bounded adversary.

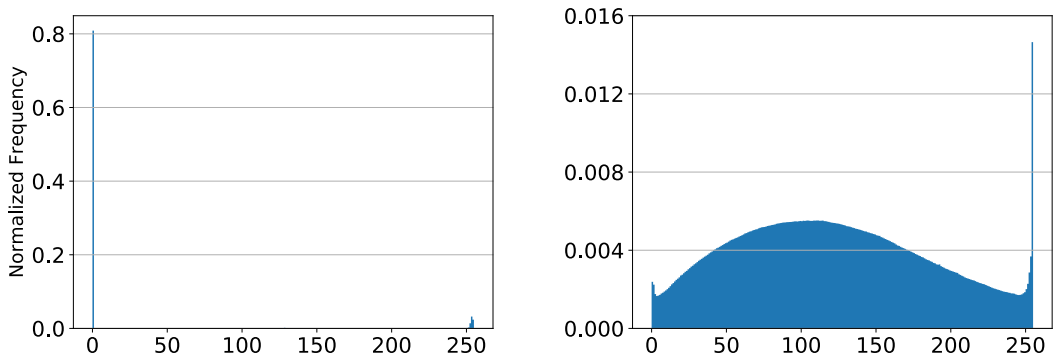
5.3.2 Bit Depth Reduction

Bit depth reduction is a simple pre-processing method we introduce in Chapter 4. Instead of using *bit depth reduction* in an ensemble framework for detecting adversarial examples, we aim to improve the provable robustness of a model in this chapter.

Bit depth reduction borrows the idea of *uniform quantization* from the signal processing field to further quantize digital signals with fewer bits. The intuition is that when a signal is encoded with fewer bits, a tiny perturbation is less likely to affect the signal value; therefore the neural network output is not changed either.

An image pixel is typically encoded with 8 bits, which implies it is susceptible to tiny perturbations on input. Using bit depth reduction, we can encode the pixel values with fewer bits to make a

[1] We abuse the ℓ_p-norm definition here because ℓ_0 is not a norm.



(a) MNIST (b) CIFAR-10

Figure 5.2: Histogram of pixel values, respectively measured on 60,000 MNIST training images and 50,000 CIFAR-10 training images.

model less sensitive to tiny input perturbations. Intuitively, it should be useful to defend against ℓ_∞ bounded adversaries.

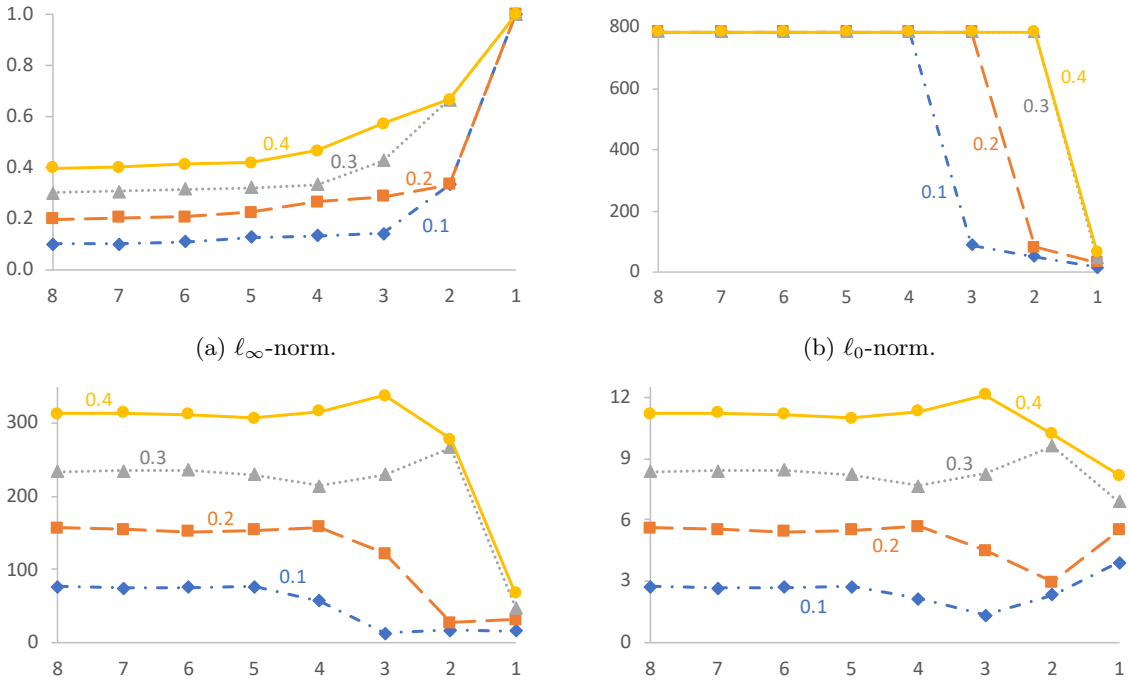
Formally, a pixel value x in the range of $[0, 1]$ can be quantized into b bits while maintaining the same scale using the formula in Equation 5.1.

$$f_{bdr}(x, b) = \frac{min(\lfloor x \times 2^b \rfloor, 2^b - 1)}{2^b - 1} \tag{5.1}$$

We measure the expected adversarial capability after *Bit Depth Reduction* on all training examples of two datasets: MNIST and CIFAR-10. The metrics we use are ℓ_∞-norm, ℓ_1-norm, ℓ_2-norm and ℓ_0-"norm".

MNIST. MNIST is a handwritten digit dataset consisting of 70,000 8-bit-encoded gray-scale images in size of 28×28 [60]. Each pixel is encoded as an 8-bit unsigned integer number ranging from 0 to 255: 0 as black, 255 as white, and the shades of gray between them. We present the pixel value distribution of the 60,000 training examples in Figure 5.2a. We find that the pixel value distribution of MNIST images is highly skewed: most pixels are 0 representing the black background while a few are close to 255 representing white strokes, and hardly any are in the range of [1,255].

We consider ℓ_∞ adversary bounded by 0.1, 0.2, 0.3 and 0.4 while scaling the value of image pixels into a range of $[0, 1]$ We want to understand the influence of bit depth reduction on adversarial capability.



(a) ℓ_∞-norm. (b) ℓ_0-norm.

(c) ℓ_1-norm. (d) ℓ_2-norm.

Figure 5.3: *Bit Depth Reduction* affects ℓ_∞-norm bounded adversarial capability on MNIST. The x-axis represents the number of bits and the y-axis represents the averaged ℓ_p-norm.

Figure 5.3 summarizes the average adversarial capability measured in various ℓ_p-norms on 60,000 training images after *Bit Depth Reduction*.

The ℓ_∞-norm in Figure 5.3a with 8 bits represents the original adversarial strength, respectively 0.1, 0.2, 0.3 and 0.4. As we reduce to fewer bits, the average ℓ_∞-norm increases. If we make an extreme case by reducing to one bit (binary filter), the average ℓ_∞-norm increases significantly to 1 since an adversary is now able to flip the bits for 0 to 1 or vice versa. This means *Bit Depth Reduction* potentially increases the *adversarial capability*, which contradicts the previous empirical study that it is effective on defending against ℓ_∞ adversary [123].

The other ℓ_p measurements in Figure 5.3 help to explain the counterintuitive result. We found a clear trend that reducing to fewer bits decreases the adversarial capability in terms of ℓ_0 and ℓ_1.

The ℓ_0 case is quite intuitive. Take the binary filter and the $\ell_\infty \leq 0.1$ adversary as an example. A binary filter converts pixel values that are not smaller than 0.5 to 0, and all others to 1. For the original pixel values lower than 0.4, an $\ell_\infty \leq 0.1$ adversary would not be able to alter the pixels to reach the binary filter threshold 0.5. Thus, all pixel values smaller than 0.4 will stick to 0 after the binary filter regardless of adversarial perturbations. The pixel values larger than 0.6 are similarly

frozen to 1. As a result, an adversary can only perturb pixels in the range of $(0.4, 0.6]$. Only 2.05% of pixels in the MNIST training dataset are in the $(0.4, 0.6]$ range. This accounts for the reason why the average ℓ_0 adversarial capability drops from 784 to 15.8. A similar result holds even if we increase the ℓ_∞ adversary strength to 0.4, because the MNIST pixel value distribution is so strongly polarized, as in Figure 5.2a.

The ℓ_1 and ℓ_2 are related to the ℓ_0 case. The adversary-immutable pixels in the ℓ_0 case would not contribute to either ℓ_1 or ℓ_2. However, the remaining adversary-mutable pixels are not negligible, especially for ℓ_2-norm. An adversary can perturb those pixels from 0 to 1 or vice versa after the binary filter, which means the magnitude of adversarial perturbation is magnified from {0.1, 0.2, 0.3, 0.4} to 1. As a result, the average ℓ_2-norm after the binary filter is not always smaller than the original one, but still has an advantage in some cases.

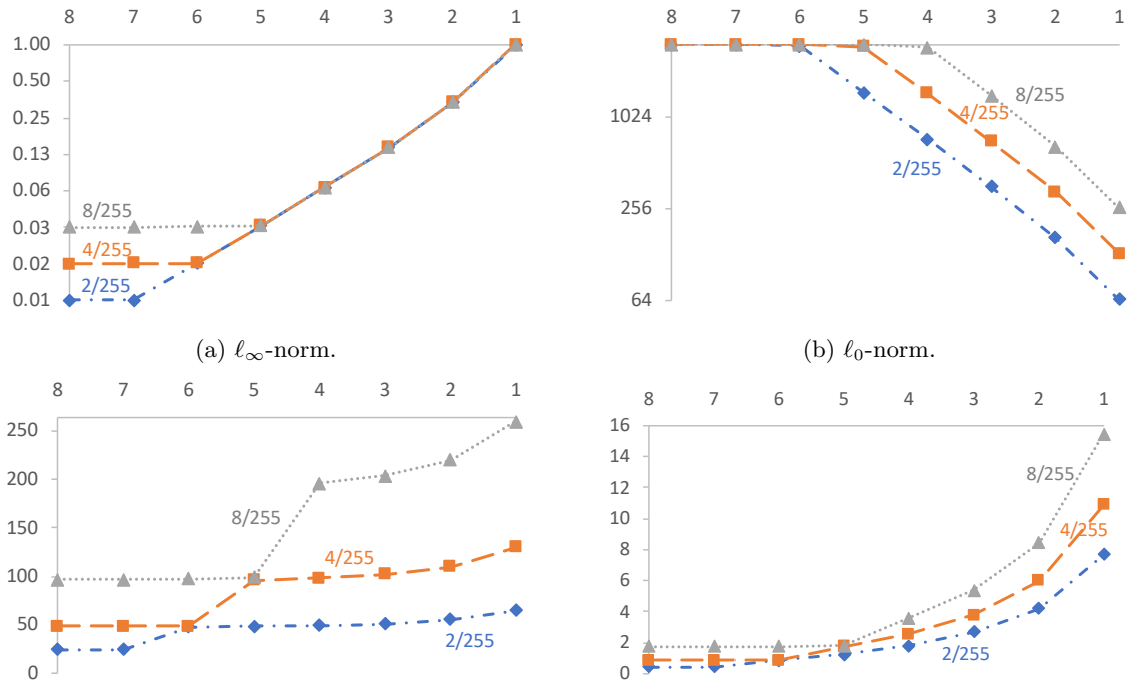
The reduced adversarial capability in terms of ℓ_0, ℓ_1 and ℓ_2 after *Bit Depth Reduction* indicates that *Bit Depth Reduction* should help to defend against ℓ_∞ adversaries on the MNIST dataset, which we explore further in Section 5.4.

CIFAR-10. CIFAR-10 is a dataset consisting of 60,000 colored images in size of 32×32 [53]. Each pixel is encoded with three unsigned 8-bit integers, respectively representing the red, green and blue (RGB) channels. The distribution of the `uint8` numbers is drastically different from that of MNIST, as is depicted in Figure 5.2b. The natural image signal generally follows a normal distribution, except that it has two peaks on both edges due to the limited dynamic range of sensing devices: extremely bright signals are truncated to 255, while nearly dark signals are truncated to 0.

We assume ℓ_∞ adversaries respectively bounded by 2/255, 4/255 and 8/255 and report the average adversarial capability after *Bit Depth Reduction* on 50,000 training images in Figure 5.4.

We found that *Bit Depth Reduction* magnifies the adversarial capability in terms of ℓ_∞, ℓ_1 and ℓ_2 for CIFAR-10. This is very different from what we observe on MNIST, because the CIFAR-10 pixel values in all channels follow the normal distribution. The adversarial norm ball of each pixel often (though not always) crosses the step function thresholds of bit depth reduction, resulting in amplified magnitudes.

However, the adversarial capability in terms of ℓ_0 is significantly reduced. This is because there are a significant number of pixels whose adversarial norm ball stays on one level of the step function



(a) ℓ_∞-norm.

(b) ℓ_0-norm.

(c) ℓ_1-norm.

(d) ℓ_2-norm.

Figure 5.4: *Bit Depth Reduction* affects ℓ_∞-norm bounded adversarial capability on CIFAR-10.

of bit depth reduction. These adversary-immutable pixels give us a chance to improve the model robustness using bit depth reduction, which we explore in Section 5.5.1.

5.4 Robustness

We explore how to train a robust classifier by incorporating simple pre-processing with provable robustness methods. As we have demonstrated in Section 5.3, *Bit Depth Reduction* can significantly decrease the adversarial capability measured in some ℓ_p-norm while a different norm bounds the original adversarial strength. This suggests that *Bit Depth Reduction* may convert a bounded adversary problem to a hopefully much easier problem bounded by other ℓ_p-norms.

5.4.1 Adversarial Bound Transformation

We propose a general approach to make use of simple pre-processing in training robust models by propagating the transformed interval bounds of input.

As we have demonstrated in Section 5.3, simple pre-processing often suppresses the ℓ_0-norm of adversarial capability, which means many pixels are immutable to an adversary after pre-processing. For example, for an ℓ_∞-norm adversary bounded by 0.1, the original adversarial interval of a pixel value 0.2 is [0.1, 0.3]. After pre-processing with a binary filter, the adversarial interval becomes [0.0,0.0].

The shorter length of the adversarial interval is a clear sign that pre-processing reduces adversarial capability. We can easily exploit such an advantage with many bound propagation-based techniques. Some robust certification methods propagate the adversarial intervals of input layer by layer to over-approximate the output bounds. With shorter intervals on the first layer, the output bounds tighten accordingly. As a result, the over-approximation takes less model capacity for robustness certification.

Assume one input $\boldsymbol{x}$ is bounded by $[\boldsymbol{l}, \boldsymbol{u}]$, the output $\tilde{\mathcal{B}}(\boldsymbol{x})$ of a monotonic pre-processing function $f_{pre}(\boldsymbol{x})$ is strictly bounded by:

$$[\inf\{\tilde{\mathcal{B}}(\boldsymbol{x})\}, \sup\{\tilde{\mathcal{B}}(\boldsymbol{x})\}] = [f_{pre}(\boldsymbol{l}), f_{pre}(\boldsymbol{u})] \tag{5.2}$$

In this way, we can combine pre-processing with other bound propagation-based robust training methods, such as *Interval Bound Propagation (IBP)* [37].

5.4.2 Robustness Regularization

We introduce an alternate method to make use of pre-processing in training robust models by regularizing the model weights. The method works if one pre-processing function significantly reduces the adversarial capability measured in ℓ_1, ℓ_2 or ℓ_∞.

Assume we have a linear model $g(x)$ parameterized with weights W and biases b:

$$g(x) = W^T x + b \tag{5.3}$$

Given one adversarial example $x' = x + \delta$ in which δ is bounded by some ℓ_p norm, the prediction of such an adversarial example is denoted as

$$\begin{aligned} g(x') &= W^T(x+\delta) + b \\ &= (W^T x + b) + W^T\delta \\ &= g(x) + W^T\delta \end{aligned} \tag{5.4}$$

The adversarial power comes from $W^T\delta$, which is the difference between the prediction of a normal input and the prediction of a bounded adversarial example. According to Hölder's inequality [3], the adversarial power is upper bounded by the product of a pair of dual norms:

$$\begin{aligned} W^T\delta &=< W, \delta > \\ &\leq ||W||_q ||\delta||_p, \end{aligned} \tag{5.5}$$

where ℓ_p-norm and ℓ_q-norm are dual such that $\frac{1}{p} + \frac{1}{q} = 1$.

Since $||\delta||_p$ is explicit for a given adversary, the adversarial capability is directly upper bounded by $||W||_q$. When p is in $\{1, 2, \infty\}$, we can construct a dual $||W||_q$ and suppress the adversary impact using proper weight regularization.

This analysis is based on linear assumptions about the target model. However, we found empirically that it also applies to ReLU activated neural network models. We believe this is because a deep neural network model often has near-linear behavior in the local region, which is the case when we consider tiny adversarial perturbations for a given input [36].

ℓ_1-Robust Training. We instantiate the ℓ_p-robust training framework on $p = 1$ and derive a robust training approach.

As we concluded in Section 5.3.2, *Bit Depth Reduction* effectively suppresses ℓ_1-norm for a given ℓ_∞ bounded adversary on the MNIST dataset. If we can train a model that is robust against ℓ_1-norm bounded adversaries, the model with *Bit Depth Reduction* should also be robust against ℓ_∞-norm adversaries. To reduce the capability of such a converted ℓ_1-norm adversary, we need to minimize $||W||_\infty$ when training the model.

The derived approach is consistent with our intuition. We know that ℓ_1-norm usually implies sparse

perturbations, which is also the case of the converted ℓ_1-norm bounded adversary that we depict in Figure 5.3c. The best opportunity of an ℓ_1-norm adversary arises if the most substantial weight magnifies the perturbation of large magnitudes. Therefore, depressing $||W||_\infty$ adversely affects the power of an ℓ_1-norm bounded adversary.

ℓ_∞ Regularization. Even though reducing $||W||_\infty$ is a straightforward approach to enhance the ℓ_1 robustness, performing ℓ_∞ regularization directly is not effective, because the gradient of ℓ_∞ oscillates among different components in the training phase.

We design an ℓ_∞ regularization term in Equation 5.6:

$$loss_\infty(W) = \alpha||W||_2 + \beta||min(0, |W| - \gamma||\hat{W}||_\infty)||_2 \tag{5.6}$$

Our ℓ_∞ regularization term has three hyper-parameters: α, β and γ. α is the coefficient of the base ℓ_2 regularization, while β is the coefficient of the extra penalization for large components. γ is a real number selected from an interval of $[0, 1]$. The smaller γ, the more components of W are penalized. We choose to build upon ℓ_2 instead of ℓ_1 because it depresses the ℓ_∞-norm more effectively. The large components in W also have large gradient values in ℓ_2-norm, so receive heavier penalization than small components in the backward pass.

5.4.3 Verification

We use formal verification to confirm the robustness of our models. We incorporate pre-processing into the MILP verification framework we introduce in Section 5.2.1.

Formulate Binary Filter in MILP. We illustrate how we formulate the binary filter, a special case of *bit depth reduction* when $b = 1$ into MILP. The formulation of *bit depth reduction* with more bits is similar.

We derive the definition of the binary filter from Equation 5.1:

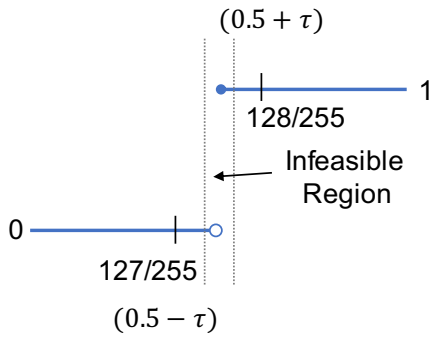


Figure 5.5: We specify an infeasible region in MILP formulation of the binary filter.

$$
\begin{aligned}
f_{bin}(x) = f_{bdr}(x,1) &= \frac{min(\lfloor x \times 2^1 \rfloor, 2^1 - 1)}{2^1 - 1} \\
&= \begin{cases} 0 & \text{if } 0 \leq x < 0.5 \\ 1 & \text{if } 0.5 \leq x \leq 1 \end{cases}
\end{aligned} \tag{5.7}
$$

The big-M method is often used to encode piece-wise linear functions as MILP formulation. However, $f_{bin}(x)$ in Equation 5.7 is an upper semi-continuous function with a breakpoint at 0.5. Thus, it cannot be directly encoded. We introduce an extra variable τ in the MILP formulation of $y = f_{bin}(x)$ to resolve the issue as shown in Figure 5.5.

$$
\begin{aligned}
l \cdot \omega_1 \leq & x_1 \leq (0.5 - \tau) \cdot \omega_1 \\
(0.5 + \tau) \cdot \omega_2 \leq & x_2 \leq u \cdot \omega_2 \\
\omega_1, \omega_2 \in & \{0, 1\} \\
\omega_1 + \omega_2 = & 1 \\
x_1 + x_2 = & x \\
y = & \omega_2
\end{aligned} \tag{5.8}
$$

Our formulation contains two mutually exclusive binary variables ω_1 and ω_2, respectively representing the zero output and the one output of $f_{bin}(x)$. The lower bound l and upper bound u of input help to determine if we can encode the function as constant so that the resulting MILP is easier to solve. For example, if l is larger than $(0.5 - \tau)$, ω_1 must be 0, then ω_2 must be 1 and the output would be constant 1. Similarly, $u < (0.5 + \tau)$ would cause a constant output 0.

The formula has to split if $l < (0.5 - \tau)$ and $u > (0.5 + \tau)$. We introduce τ, a tiny real number

Table 5.2: Architecture of two ReLU-activated models used in the experiments. "Conv k $w \times h + s$" means 2D convolutional layer with k filters of size $w \times h$ using a stride of s in both dimensions. "FC n" means a fully connected layer with n outputs.

		Adversarial Bound Transformation	Robustness Regularization
Layers		Conv 16 4×4+2 Conv 32 4×4+1 FC 100 FC 10	Conv 16 4×4+2 Conv 32 4×4+2 FC 100 FC 10
Padding		VALID	SAME
Trainable Params	MNIST	330K	166K
	CIFAR-10	471K	-

that creates an infeasible open interval $(0.5 - \tau, 0.5 + \tau)$, because $f_{bin}(x)$ is not continuous at input 0.5. Considering the *uint8* input data type of pixel values, there's a gap $(\frac{127}{255}, \frac{128}{255})$ near 0.5 that we can safely assume infeasible. Any real number in $(0, \frac{128}{255} - 0.5)$ is appropriate to handle this non-continuous point. In practice, we let $\tau = 0.001$ as it is large enough that it would not be rounded off as zero.

5.5 Experiments

We use two methods from Section 5.4 to train robust models and verify their robustness. We evaluate the adversarial bound transformation method that combines *bit depth reduction* with certifiable defense in Section 5.5.1 and evaluate the robustness regularization method in Section 5.5.2. We report test accuracy and verified robust accuracy for each model. The verified robust accuracy is the percentage of testing examples that are verified as robust against an assumed adversary.

5.5.1 Adversarial Bound Transformation

We incorporate *Bit Depth Reduction* with Interval Bound Propagation [37] by propagating the adversarial interval bounds through the *Bit Depth Reduction* layer to train robust models.

Model Architecture. We use the "small" model architecture proposed in IBP [37], described in the first column of Table 5.2.

Training Procedure. We use the training procedures proposed in IBP [37] respectively for MNIST and CIFAR-10. However, since we could not replicate the results reported in their paper, we create a baseline using their open source code instead.[2] To avoid the numeric issue in MILP solving that we introduce in Section 5.2.1, we prune the trained weights of MNIST models whose absolute value is lower than 0.01.

Verification Procedure on MNIST. We use MIPVerify to examine the uncertified inputs in the MNIST experiment. The binary filter is encoded in a MILP manner as in Section 5.4.3. As a result, we can encode everything, including the model, the restricted adversary and the binary filter as a MILP problem. We set an objective to minimize the ℓ_1-norm of the adversarial perturbation and use the off-the-shelf Gurobi solver. We use Interval Arithmetic (IA), the most efficient way, to propagate the input bounds layer by layer and the solver times out in 100 seconds. We test all the 10,000 MNIST test images and report the verified robust accuracy, which is the percentage of examples for which we verify the infeasibility of the assumed adversary.

MNIST Result. We assume an ℓ_∞ adversary bounded by 0.1, 0.2, 0.3 and 0.4. For each setup, we add a binary filter to the model architecture and retrain the model from scratch using the identical procedure.

From Table 5.3 we see that adding the binary filter always improves both the accuracy and verified robustness. Take the $\epsilon = 0.1$ case as an example. The verified robust accuracy increases from 96.51% to 96.91% and the test accuracy increases from 98.26% to 98.32%. The mean verification cost is also reduced from 0.56 seconds to 0.43 seconds.

CIFAR-10 Result. We assume ℓ_∞ adversary bounded by 2/255, 4/255 and 8/255. We successively

[2] The authors of IBP stated that they were not allowed to open source all the code they used in their paper at https://github.com/deepmind/interval-bound-propagation/issues/1

Table 5.3: *Bit Depth Reduction* combined with *Interval Bound Propagation* improves provable robustness against ℓ_∞ bounded adversary on 10,000 MNIST test images.

ϵ	$bin(x)$	Test Accuracy	Verified Robust Accuracy	Mean Verification Time (sec)
0.1		98.26%	96.51%	0.56
	✓	**98.32%**	**96.91%**	0.43
0.2		96.43%	91.75%	0.52
	✓	**96.74%**	**92.48%**	0.48
0.3		96.43%	86.80%	0.64
	✓	**96.74%**	**88.46%**	0.65
0.4		96.43%	77.14%	0.98
	✓	**96.74%**	**80.06%**	1.12

Table 5.4: *Bit Depth Reduction* improves model robustness against ℓ_∞ adversary on 10,000 CIFAR-10 test images, when combined with *Interval Bound Propagation*. We repeated each experiment three times and report the result with the highest robustness accuracy. The bold results are the best.

ϵ		8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit	1-bit
2/255	TestAcc	47.74%	49.41%	**50.15%**	48.98%	49.82%	47.55%	47.23%	42.65%
	RobAcc	21.27%	21.75%	**25.89%**	21.52%	23.04%	20.37%	21.53%	19.74%
4/255	TestAcc	47.26%	**49.60%**	46.64%	48.92%	47.46%	48.02%	45.84%	43.68%
	RobAcc	23.09%	**23.46%**	20.56%	20.60%	19.87%	19.50%	19.94%	19.65%
8/255	TestAcc	42.25%	44.00%	44.50%	43.40%	**45.20%**	44.50%	43.55%	41.67%
	RobAcc	17.75%	19.19%	18.71%	17.11%	**20.76%**	17.35%	17.79%	19.12%

TesAcc: test accuracy. RobAcc: verified robust accuracy.

reduce the number of encoding bits of input and train a new model from scratch for each ϵ. We repeat each experiment three times and report the result with the best robustness. We don't use MIPVerify to examine the uncertified inputs, because it is substantially more expensive than verifying the MNIST models. We report the test accuracy and verified robust accuracy in Table 5.4.

We see that adding bit depth reduction achieves better test accuracy and verified robust accuracy. Take $\epsilon = 2/255$ as an instance, if we squeeze the input into 6 bits, the test accuracy increases from 47.74% to 50.15% and the verified robust accuracy increases from 21.27% to 25.89%.

5.5.2 Robustess Regularization

As we have observed in Section 5.3.2, *Bit Depth Reduction* converts the ℓ_∞-norm bounded adversary problem into an easier ℓ_1-norm problem on MNIST.[3] We use the method we derive in Section 5.4.2 to

[3]We don't consider the CIFAR-10 dataset in this experiment, because the ℓ_1 adversarial capability is not decreased with *Bit Depth Reduction* as shown in Figure 5.4c.

train a robust model using *Bit Depth Reduction*, assuming an adversary bounded by $\ell_\infty \leq 0.1$.

Model Architecture. We use the same CNN architecture as in Wong et al. [51]. The model has two convolutional layers and two fully connected layers, as is illustrated in the second column of Table 5.2. Though the major difference from the model in the first column is the 2×2 stride in the second convolutional layer instead of 1×1, the model we use in this experiment only has half the number of the trainable parameters, resulting in smaller model capacity.

Training Procedure. We use an Adam optimizer with a learning rate of 0.0001 to minimize the cross-entropy loss in 200 epochs, with a batch size of 32. We configure the hyper-parameters of the ℓ_∞ regularizer in Equation 5.6 as $\alpha = 0.0004$, $\beta = 0.004$ and $\gamma = 0.2$. Besides, we perform weight pruning and ReLU pruning on all models to make the verification more efficient [117]. First, all the trained weights whose absolute values are not larger than 0.001 are set to 0. Second, if one ReLU unit has consistent behavior on more than 90% input examples, it is frozen to that behavior, either rectified or zero.

Verification Procedure. We use the open source MIPVerify [110] framework to verify the robustness of models against the assumed $\ell_\infty \leq 0.1$ adversary. The verification procedure is identical to that in Section 5.5.1, except that we perform up to three rounds with different tightening methods. The unverified inputs in previous rounds will be revisited in later rounds using less efficient methods until they are verified or the third round times out. The first round uses Interval Arithmetic (IA), the most efficient method, to propagate the input bounds layer by layer and the solver times out in 100 seconds. The second round uses Linear Programming (LP), a relaxation of MILP to propagate tighter bounds at each layer and the solver times out in 300 seconds. The third round uses the most time-consuming MILP at each layer to get the (hopefully) tightest bounds and the solver times out in one hour. We interrupt the experiment after one round or two if the average time consumption exceeds 100 seconds for each example.

Result. First, we compare three models in the experiment: a baseline model without any adversarial training or robust training techniques (Vanilla Training); the baseline model with a binary filter (VT+f_{bin}); and the baseline model with a binary filter and ℓ_∞ regularization (VT+f_{bin}+ℓ_∞ Reg.).

[4]The exact verification result of Wong et al. [51] is from Tjeng et al. [110].

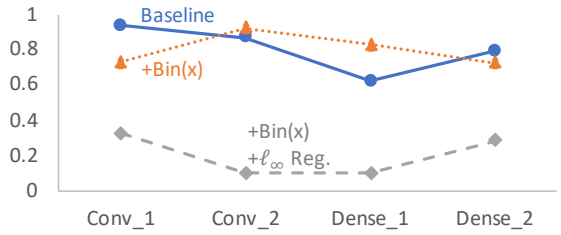


Figure 5.6: ℓ_∞ regularization suppresses $||W||_\infty$ at each layer.

Table 5.5: *Bit Depth Reduction* improves provable robustness against $\ell_\infty \leq 0.1$ adversary on 10,000 MNIST test examples.

Method	Test Accuracy	Verified Robust Accuracy	Mean Verification Time (sec)
Vanilla Training (VT)	**99.37%**	0.00%	100.00
VT+f_{bin}	99.16%	70.43%	31.60
VT+f_{bin}+ℓ_∞ Reg.	99.22%	**96.94%**	**16.27**
Wong et al. [51, 110][4]	98.11%	95.62%	3.52*
ReLU-Stable [117]	98.68%	94.33%	0.49*

* The verification cost was measured on different machines and reported by Xiao et al. [117].

We show the effectiveness of the ℓ_∞ regularization in Figure 5.6. It is clear that the ℓ_∞ norm is significantly decreased with our ℓ_∞ regularization compared with the counterparts.

We report the verified robustness results in Table 5.5. First, we find that the baseline model has an extremely high accuracy of 99.37% on normal inputs, but we couldn't verify if any input example is robust against the assumed adversary given the time constraint.

If we add a binary filter to the model architecture and retrain a new model using the same procedure, we can verify that 70.43% of test images are robust against the assumed adversary, while the accuracy of the new model $+f_{bin}(x)$ is similar to the baseline.

We further test the ℓ_1-robust training we derive in Section 5.4.2. In addition to the binary filter, we add the ℓ_∞ regularizer in the training procedure. The result in the third row of Table 5.5 shows that this method produces a model that is both accurate and robust. The accuracy of 99.22% on normal test inputs is comparable to the baseline of 99.37%. In addition, we can verify that 96.94% of the test inputs are robust against the assumed adversary. The average verification time also decreases substantially from 31.6 seconds to 16.3 seconds.

We also compare with two models reported in other papers [110, 117] in Table 5.5. Our model outperforms theirs in both test accuracy and verified robust accuracy, while we share the same model

architecture. Their reported verification cost is lower than ours though, which is not a surprise because their methods explicitly use expensive training methods to reduce the verification cost while our method only uses simple pre-processing and the inexpensive ℓ_∞ regularization.

In summary, we confirm that a simple binary filter combined with the ℓ_∞ regularization results in an accurate MNIST model with impressive provable robustness, even without expensive adversarial training.

5.6 Conclusion

We formally analyze the implication of simple pre-processing on adversarial capability and incorporate *bit depth reduction* with formal verification and robust certification methods in a novel way to train provably robust models against restricted adversaries. The experimental results confirm the intuition that simple pre-processing reduces the search space available to an adversary, so improving the model robustness, even against adaptive adversaries.

Although we have only experimented on image classification datasets with *bit depth reduction*, we believe other pre-processing should have similar properties in the domains where deep learning is used. Our work opens a new research direction in defending against adversary by combining inexpensive simple pre-processing with formal methods and robust certification.

Chapter 6

Conclusion

This chapter begins with a summary of the thesis work and closes with final remarks.

6.1 Summary

Our work shows that using domain knowledge is an important step towards robust machine learning systems.

First, we developed *Genetic Evasion*, a general framework to automatically find weaknesses of machine learning models. We embedded domain knowledge of PDF documents into the framework and attacked two state-of-the-art machine learning-based PDF malware classifiers with 100% success. Our practical attacking method broke the trust of black-box machine learning models in the malware detection field and highlighted the importance of using domain knowledge in examining the robustness of machine learning models.

Second, we proposed *Feature Squeezing*, a generic framework to detect adversarial examples against deep learning models for perception tasks. Based on the domain knowledge that natural signals are often redundant, we designed an ensemble framework which pre-processes inputs with different squeezers and measures the distance between the model's predictions on input with different pre-processing to detect adversarial examples. We showed that *Feature Squeezing* is empirically effective

and inexpensive in detecting adversarial examples generated by various attacking algorithms, compared with previous methods that do not exploit domain knowledge.

Third, we incorporated simple pre-processing with provable robustness methods to improve the robustness of machine learning models. We investigated the implications of *Bit Depth Reduction* on adversarial capability and accordingly developed novel methods to train more robust models by either transforming the input bounds or regularizing the model weights. Our approach successfully trained accurate and robust models without expensive adversarial training and advanced the state-of-the-art of robust certification methods.

6.2 Paths Forward

Our work presents solid evidence about the weaknesses of machine learning models and proposes practical solutions to defend non-robust models and improve provable robustness. The results improves understanding of the robustness properties of machine learning models and enhance the confidence in deploying machine learning for security-sensitive tasks.

There remain many problems to solve before we can attain robust machine learning models. Even if we utilize formal methods in our robustness studies, the result comes with several limitations for important tasks. First, it is difficult to scale the provable robustness techniques to large datasets or large models that are widely used in the real world. Second, strict proofs of model robustness need to consider the whole input space rather than specific examples from a dataset, or at least be efficient enough to use at inference time. Third, the ℓ_p-norms are insufficient to model the power of adversaries accurately [32, 116].

The root cause of machine learning models lack of robustness is that *correlation does not imply causation*—machine learning models often do not learn causal factors in their decision rules. The gap between the ground-truth decision rules and the approximate ones learned by a machine learning model inevitably creates opportunities for adversaries. Domain knowledge should play a vital role in training robust models that learn causal factors in the future.

Bibliography

[1] Gurobi Guidelines for Numerical Issues. http://files.gurobi.com/Numerics.pdf, 2017.

[2] AdvBox by Baidu X-Lab. https://github.com/baidu/AdvBox, 2018.

[3] Hölder's Inequality. https://en.wikipedia.org/wiki/Hlder%27s_inequality, 2018.

[4] MIP - A Primer on the Basics. http://www.gurobi.com/resources/getting-started/mip-basics, 2019.

[5] Adobe, Inc. PDF Reference and Adobe Extensions to the PDF Specification. http://www.adobe.com/devnet/pdf/pdf_reference.html.

[6] Hassan Ali, Hammad Tariq, Muhammad Abdullah Hanif, Faiq Khalid, Semeen Rehman, Rehan Ahmed, and Muhammad Shafique. QuSecNets: Quantization-based Defense Mechanism for Securing Deep Neural Network against Adversarial Attacks. *arXiv preprint arXiv:1811.01437*, 2018.

[7] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *35th International Conference on Machine Learning (ICML)*, 2018.

[8] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can Machine Learning Be Secure? In *1st ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, 2006.

[9] Rodrigo Benenson. Classification Datasets Results. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

[10] Arjun Nitin Bhagoji, Daniel Cullina, and Prateek Mittal. Enhancing Robustness of Machine Learning Systems via Data Transformations. *arXiv preprint 1704.02654*, 2017.

[11] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion Attacks against Machine Learning at Test Time. In *6th European Machine Learning and Data Mining Conference (ECML/PKDD)*, 2013.

[12] Battista Biggio, Giorgio Fumera, and Fabio Roli. Multiple Classifier Systems for Adversarial Classification Tasks. In *Multiple Classifier Systems*. Springer, 2009.

[13] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer Encoding: One Hot Way to Resist Adversarial Examples. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[15] Nicholas Carlini. Robust Evasion Attacks against Neural Network to Find Adversarial Examples. https://github.com/carlini/nn_robust_attacks/.

[16] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden Voice Commands. In *25th USENIX Security Symposium (USENIX)*, 2016.

[17] Nicholas Carlini and David Wagner. Defensive Distillation is not Robust to Adversarial Examples. *arXiv preprint 1607.04311*, 2016.

[18] Nicholas Carlini and David Wagner. Adversarial Examples Are not Easily Detected: Bypassing Ten Detection Methods. In *10th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2017.

[19] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *38th IEEE Symposium on Security and Privacy (Oakland)*, 2017.

[20] Stephan Chenette. Malicious Documents Archive for Signature Testing and Research - Contagio Malware Dump. http://contagiodump.blogspot.de/2010/08/malicious-documents-archive-for.html.

[21] Deepak Chinavle, Pranam Kolari, Tim Oates, and Tim Finin. Ensembles in Adversarial Classification for Spam. In *18th ACM Conference on Information and Knowledge Management (CIKM)*, 2009.

[22] Travers Ching, Daniel S Himmelstein, Brett K Beaulieu-Jones, Alexandr A Kalinin, Brian T Do, Gregory P Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M Hoffman, et al. Opportunities and Obstacles for Deep Learning in Biology and Medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[23] Francois Chollet. Keras Implementation of Inception v3. https://github.com/fchollet/deep-learning-models.

[24] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified Adversarial Robustness via Randomized Smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[25] Symantec Corporation. Symantec Internet Security Threat Report. Technical report, Symantec Corporation, 2015.

[26] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and Analysis of Drive-By-Download Attacks and Malicious JavaScript Code. In *19th International World Wide Web Conference (WWW)*, 2010.

[27] CVE Details. Adobe Acrobat Reader — CVE Security Vulnerabilities, Versions and Detailed Reports. http://www.cvedetails.com/product/497.

[28] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-Scale Malware Classification Using Random Projections and Neural Networks. In *38th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[29] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial Classification. In *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[30] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading Classifiers by Morphing in the Dark. In *24th ACM Conference on Computer and Communications Security (CCS)*, 2017.

[31] Saeed Ehteshamifar, Antonio Barresi, Thomas R Gross, and Michael Pradel. Easy to Fool? Testing the Anti-evasion Capabilities of PDF Malware Scanners. *arXiv preprint arXiv:1901.05674*, 2019.

[32] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A Rotation and A Translation Suffice: Fooling CNNs with Simple Transformations. *arXiv preprint arXiv:1712.02779*, 2017.

[33] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting Adversarial Samples from Artifacts. *arXiv preprint 1703.00410*, 2017.

[34] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial Attacks on Medical Machine Learning. *Science*, 363(6433):1287–1289, 2019.

[35] Stephanie Forrest. Genetic Algorithms: Principles of Natural Selection Applied to Computation. *Science*, 261(5123), 1993.

[36] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[37] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the Effectiveness of Interval Bound Propagation for Training Verifiably Robust Models. *arXiv preprint arXiv:1810.12715*, 2018.

[38] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (Statistical) Detection of Adversarial Examples. *arXiv preprint 1702.06280*, 2017.

[39] Shixiang Gu and Luca Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *arXiv preprint 1412.5068*, 2014.

[40] Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, and Mark Schloesser. Cuckoo Sandbox: A Malware Analysis System. `http://www.cuckoosandbox.org/`.

[41] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering Adversarial Images using Input Transformations. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[42] Mark Harman, William B Langdon, and Westley Weimer. Genetic Programming for Reverse Engineering. In *20th IEEE Working Conference on Reverse Engineering (WCRE)*, 2013.

[43] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial Example Defenses: Ensembles of Weak Defenses are not Strong. In *11th USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[44] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. *arXiv preprint 1207.0580*, 2012.

[45] Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Deceiving Google's Perspective API Built for Detecting Toxic Comments. *arXiv preprint 1702.08138*, 2017.

[46] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint 1704.04861*, 2017.

[47] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely Connected Convolutional Networks. In *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[48] Thomas Hungenberg and Matthias Eckert. INetSim: Internet Services Simulation Suite. http://www.inetsim.org/.

[49] Alex Kantchelian, JD Tygar, and Anthony Joseph. Evasion and Hardening of Tree Ensemble Classifiers. In *33rd International Conference on Machine Learning (ICML)*, 2016.

[50] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *International Conference on Computer Aided Verification*, 2017.

[51] J Zico Kolter and Eric Wong. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *35th International Conference on Machine Learning (ICML)*, 2018.

[52] John R Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, volume 1. MIT press, 1992.

[53] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto, 4 2009.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *26th Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[55] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial Examples in the Physical World. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

[56] Pavel Laskov and Nedim Šrndić. Static Detection of Malicious JavaScript-Bearing PDF Documents. In *27th ACM Annual Computer Security Applications Conference (ACSAC)*, 2011.

[57] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering*, 2012.

[58] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation Applied to Handwritten ZIP Code Recognition. *Neural computation*, 1989.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[60] Yann Lecun, Corinna Cortes, and Christopher JC Burges. The MNIST Database of Handwritten Digits. http://yann.lecun.com/exdb/mnist, 2009.

[61] Ji Lin, Chuang Gan, and Song Han. Defensive Quantization: When Efficiency Meets Robustness. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[62] Tao Liu, Zihao Liu, Qi Liu, and Wujie Wen. Enhancing the Robustness of Deep Neural Networks from "Smart" Compression. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2018.

[63] Zihao Liu, Qi Liu, Tao Liu, Yanzhi Wang, and Wujie Wen. Feature Distillation: DNN-Oriented JPEG Compression Against Adversarial Examples. *arXiv preprint arXiv:1803.05787*, 2018.

[64] Daniel Lowd and Christopher Meek. Adversarial learning. In *11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2005.

[65] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[66] Davide Maiorca, Igino Corona, and Giorgio Giacinto. Looking at the Bag Is Not Enough to Find the Bomb: An Evasion of Structural Methods for Malicious PDF Files Detection. In *8th ACM Symposium on Information, Computer and Communications Security (AsiaCCS)*, 2013.

[67] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A Pattern Recognition System for Malicious PDF Files Detection. In *8th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*. 2012.

[68] Somshubra Majumdar. DenseNet Implementation in Keras. `https://github.com/titu1994/DenseNet/`.

[69] Somshubra Majumdar. Keras Implementation of Mobile Networks. `https://github.com/titu1994/MobileNetworks/`.

[70] Patrick Maupin. PDFRW: A Pure Python Library That Reads and Writes PDFs. `https://github.com/pmaupin/pdfrw`.

[71] François Menet, Paul Berthier, José M Fernandez, and Michel Gagnon. Spartan Networks: Self-Feature-Squeezing Neural Networks for increased robustness in adversarial settings. *arXiv preprint arXiv:1812.06815*, 2018.

[72] Dongyu Meng and Hao Chen. MagNet: a Two-Pronged Defense against Adversarial Examples. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[73] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On Detecting Adversarial Perturbations. *arXiv preprint 1702.04267*, 2017.

[74] João Monteiro, Zahid Akhtar, and Tiago H Falk. Generalizable Adversarial Examples Detection Based on Bi-model Decision Mismatch. *arXiv preprint arXiv:1802.07770*, 2018.

[75] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal Adversarial Perturbations. `https://github.com/LTS4/universal/`.

[76] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: a Simple and Accurate Method to Fool Deep Neural Networks. In *29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[77] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images. In *28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[78] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial Robustness Toolbox v0.6.0. *CoRR*, 1807.01069, 2018.

[79] OpenCV-Python Tutorials. Image Denoising. `https://docs.opencv.org/3.2.0/d5/d69/tutorial_py_non_local_means.html`, 2017.

[80] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Pérez-Cabo. No Bot Expects the DeepCAPTCHA! *IEEE Transactions on Information Forensics and Security*, 2017.

[81] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. Cleverhans v1.0.0: an Adversarial Machine Learning Library. *arXiv preprint 1610.00768*, 2016.

[82] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. In *12th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2017.

[83] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The Limitations of Deep Learning in Adversarial Settings. In *1st IEEE European Symposium on Security and Privacy(EuroS&P)*, 2016.

[84] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. SoK: Towards the Science of Security and Privacy in Machine Learning. In *3rd IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[85] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. SoK: Security and Privacy in Machine Learning. In *3rd IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[86] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks. In *37th IEEE Symposium on Security and Privacy (Oakland)*, 2016.

[87] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep Face Recognition. In *British Machine Vision Conference*, 2015.

[88] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified Defenses against Adversarial Examples. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[89] Adnan Siraj Rakin, Zhezhi He, Boqing Gong, and Deliang Fan. Blind Pre-Processing: A Robust Defense Method Against Adversarial Examples. *arXiv preprint arXiv:1802.01549*, 2018.

[90] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft Malware Classification Challenge. *arXiv preprint arXiv:1802.10135*, 2018.

[91] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[92] Conor Ryan. *Automatic Re-Engineering of Software Using Genetic Programming*, volume 2. Springer Science & Business Media, 2012.

[93] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A Unified Embedding for Face Recognition and Clustering. In *28th IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[94] SciPy. Median Filter (scipy.ndimage.median_filter). https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html#scipy.ndimage.median_filter, 2017.

[95] Karthik Selvaraj and Nino Fred Gutierrez. The Rise of PDF Malware. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_rise_of_pdf_malware.pdf, March 2010.

[96] Tegjyot Singh Sethi and Mehmed Kantardzic. Data Driven Exploratory Attacks on Black Box Classifiers in Adversarial Domains. *Neurocomputing*, 289:129–143, 2018.

[97] Kumar Sharad, Giorgia Azzurra Marson, Hien Thi Thu Truong, and Ghassan Karame. Mix'n'Squeeze: Thwarting Adaptive Adversarial Samples Using Randomized Squeezing. *arXiv preprint arXiv:1812.04293*, 2018.

[98] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition. In *23rd ACM Conference on Computer and Communications Security (CCS)*, 2016.

[99] Yash Sharma and Pin-Yu Chen. Bypassing Feature Squeezing by Increasing Adversary Strength. *arXiv preprint arXiv:1803.09868*, 2018.

[100] Charles Smutz and Angelos Stavrou. Malicious PDF Detection Using Metadata and Structural Features. In *28th ACM Annual Computer Security Applications Conference (ACSAC)*, 2012.

[101] Charles Smutz and Angelos Stavrou. Malicious PDF Detection Using Metadata and Structural Features. Technical report, 2012.

[102] Sibo Song, Yueru Chen, Ngai-Man Cheung, and C-C Jay Kuo. Defense Against Adversarial Attacks with Saak Transform. *arXiv preprint arXiv:1808.01785*, 2018.

[103] Nedim Šrndic and Pavel Laskov. Mimicus: A Library for Adversarial Classifier Evasion. https://github.com/srndic/mimicus.

[104] Nedim Šrndic and Pavel Laskov. Detection of Malicious Pdf Files Based on Hierarchical Document Structure. In *20th Network and Distributed System Security Symposium (NDSS)*, 2013.

[105] Nedim Šrndic and Pavel Laskov. Practical Evasion of a Learning-Based Classifier: A Case Study. In *35th IEEE Symposium on Security and Privacy (Oakland)*, 2014.

[106] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[107] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[108] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the Gap to Human-level Performance in Face Verification. In *27th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[109] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks Meet Interpretability: Attribute-steered Detection of Adversarial Samples. In *32nd Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[110] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[111] Liang Tong, Bo Li, Chen Hajaj, Chaowei Xiao, and Yevgeniy Vorobeychik. A Framework for Validating Models of Evasion Attacks on Machine Learning, with Application to PDF Malware Detection. *arXiv preprint arXiv:1708.08327*, 2017.

[112] Matthew A Turk and Alex P Pentland. Face Recognition using Eigenfaces. In *4th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1991.

[113] VirusTotal. Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/.

[114] Beilun Wang, Ji Gao, and Yanjun Qi. A Theoretical Framework for Robustness of (Deep) Classifiers Under Adversarial Noise. In *International Conference on Learning Representations (ICLR) Workshop*, 2017.

[115] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal Security Analysis of Neural Networks using Symbolic Intervals. In *27th USENIX Security Symposium (USENIX)*, 2018.

[116] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially Transformed Adversarial Examples. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[117] Kai Y Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing relu stability. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[118] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating Adversarial Effects through Randomization. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[119] Meng Xu and Taesoo Kim. PlatPal: Detecting Malicious Documents with Platform Diversity. In *26th USENIX Security Symposium (USENIX)*, 2017.

[120] Weilin Xu, David Evans, and Yanjun Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. *arXiv preprint 1704.01155*, 2017.

[121] Weilin Xu, David Evans, and Yanjun Qi. Feature Squeezing Mitigates and Detects Carlini/Wagner Adversarial Examples. *arXiv preprint 1705.10686*, 2017.

[122] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *23rd Network and Distributed System Security Symposium (NDSS)*, 2016.

[123] Weilin Xu, Yanjun Qi, and David Evans. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *25th Network and Distributed System Security Symposium (NDSS)*, 2018.

[124] Weilin Xu, Zhenyu Zhong, and Yunhan Jia. Defcon CAAD 2018: Magic Tricks for Self-driving Cars. https://speakerdeck.com/mzweilin/magic-tricks-for-self-driving-cars, 2018.

[125] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[126] Yuchen Zhang and Percy Liang. Defending against Whitebox Adversarial Attacks via Randomized Discretization. In *22nd International Conference on Artificial Intelligence and Statistics(AISTATS)*, 2019.

[127] Fei Zuo, Lannan Luo, and Qiang Zeng. Countermeasures Against L0 Adversarial Examples Using Image Processing and Siamese Networks. *arXiv preprint arXiv:1812.09638*, 2018.