Towards Attack-Agnostic Defense against Adversarial Inputs

Abstract

Deep neural networks (DNNs) are inherently vulnerable to adversarial inputs: such maliciously crafted samples trigger DNNs to misbehave, leading to detrimental consequences for DNN-powered systems. The fundamental challenges of mitigating adversarial inputs stem from their adaptive and variable nature. Existing solutions attempt to improve DNN resilience against specific attacks; yet, such static defenses can often be circumvented by adaptively engineered inputs or by new attack variants.

Here, we present EagleEye, an attack-agnostic adversarial tampering analysis engine for DNN-powered systems. Our design exploits the *minimality principle* underlying many attacks: to maximize the attack's evasiveness, the adversary often seeks the minimum possible distortion to convert genuine inputs to adversarial ones. We show that this practice entails the distinct distributional properties of adversarial inputs in the input space. By leveraging such properties in a principled manner, EagleEye effectively discriminates adversarial inputs and even uncovers their correct classification outputs. Through extensive empirical evaluation using a range of benchmark datasets and DNN models, we validate EagleEye's efficacy. We further investigate the adversary's possible countermeasures, which implies a difficult dilemma for her: to evade EagleEye's detection, excessive distortion is necessary, thereby significantly reducing the attack's evasiveness regarding other detection mechanisms.

1 Introduction

Recent years have witnessed the abrupt advances in deep learning (DL) techniques [24], which lead to breakthroughs in a number of long-standing artificial intelligence tasks (e.g., image classification, speech recognition, and even playing Go [38]). Internet giants, such as Google, Facebook and Amazon, all have heavily invested in offering DL-powered services and products.

However, designed to model highly nonlinear, nonconvex functions, deep neural networks (DNNs) are inherently vulnerable to adversarial inputs, which are malicious samples crafted by adversaries to trigger DNNs to misbehave [43]. Figure 1 shows an example: both original images are correctly recognized by a DNN; with a few pixels altered, the resulting adversarial images are misclassified by the same DNN, though the difference is barely discernible for human eyes. With the increasing use of DL-powered systems in security-critical domains, adversaries have strong incentive to manipulate

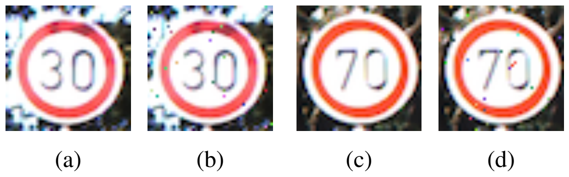


Figure 1: (a) (c) genuine inputs - both are correctly recognized; (b) (d) adversarial inputs - (b) is misclassified as "70 mph" and (d) is misclassified as "30 mph".

such systems via forcing misclassification of inputs: illegal content can bypass content filters that employ DL to discriminate inappropriate web content [17]; biometric authentications that apply DL to validate human faces can be manipulated to allow improper access [40]; in the near future, driverless vehicles that use DL to detect traffic signs may be misled to crashing.

The fundamental challenges of defending against adversarial input attacks stem from their adaptive and variable nature: they are created tailored to target DNNs, while crafting strategies vary greatly with concrete attacks. Existing solutions attempt to improve DNN resilience against specific attacks [18, 14, 21, 39, 35]; yet, such static defenses, once deployed, can often be circumvented by adaptively engineered inputs or by new attack variants. For instance, the training data augmentation mechanism [14, 33] suggests to train DNNs on adversarial inputs; as detailed in § 3, the resulting models often overfit to known attacks, thus being even more vulnerable to unseen variants. Further, most existing solutions require significant modifications to either DNN architectures or training procedures, which often negatively impact the classification accuracy of DNN models. Indeed, recent theoretical exploration [11] has confirmed the inherent trade-off between DNN robustness and expressivity, which significantly impedes the adoption of existing defense solutions in accuracy-sensitive domains.

In this paper, we take a completely new route: instead of striving to improve DNN robustness against specific attacks, we aim at defense mechanisms that make minimal assumptions regarding the attacks and adapt readily to their variable nature. To this end, we design, implement and evaluate EagleEye, an attack-agnostic adversarial tampering analysis engine for DL-powered systems.

At a high level, EagleEye leverages the *minimality principle* underlying many attacks: intuitively, to maximize the attack's evasiveness, the adversary often seeks the minimum possible distortion to convert a genuine input to an adversarial one. We show both empirically and analytically that this practice entails the distinct properties shared by adversarial inputs: compared with their genuine counterparts, adversarial inputs tend to distribute "closer" to the classification boundaries induced by DNNs in the input manifold space. By exploiting such properties in a principled manner, EagleEye effectively discriminates adversarial inputs and even uncovers their correct classification outputs. We also investigate the adversary's possible countermeasures by abandoning the minimality principle, which however implies a difficult dilemma for her: to evade EagleEye's detection, excessive distortion is necessary, thereby significantly reducing the attack's evasiveness with respect to other detection mechanisms (e.g., human vision).

Note that we are not arguing to replace existing defense solutions with EagleEye. Rather, their distinct designs entail their complementary nature. EagleEye exerts minimal interference to existing components of DL-powered systems and is thus compatible with existing defenses. Moreover, the synergistic integration of EagleEye with other mechanisms (e.g., defensive distillation [35]) delivers even stronger defenses for DNNs.

Our contributions can be summarized as follows.

- We expose the limitations of existing defenses against adversarial input attacks, which motivates the design of EagleEye. To our best knowledge, our empirical evaluation (§ 3) is the most comprehensive study to date on varied attack and defense models.
- We identify the minimality principle underlying most attacks, which entails the distinct properties shared by adversarial inputs. We design and implement EagleEye, which effectively exploits such properties in a principled manner (§ 4).
- We analytically and empirically validate EagleEye's efficacy (§ 5 and § 6), which achieves promising accuracy in discriminating adversarial inputs and even uncovering their correct classification outputs.
- We investigate the adversary's possible countermeasures and their implications. We also empirically explore the synergistic integration of EagleEye with existing defense mechanisms (§ 6 and § 7).

All the source code of this paper will be released on GitHub after the double-blind review is complete.

2 Attacks and Defenses

In this section, we introduce a set of fundamental concepts and assumptions, and survey representative attack and defense models in literature.

2.1 Deep Learning

DL represents a class of machine learning algorithms designed to learn high-level abstraction of complex data using multiple processing layers and nonlinear transformations. Figure 2 shows a typical DNN architecture.

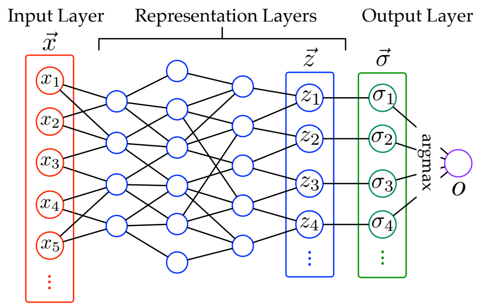


Figure 2: Illustration of DNN models.

In this paper, we primarily focus on image classification tasks, while our discussion generalizes to other settings (see § 7). In these tasks, the DNN encodes a mapping $f : \mathcal{X} \rightarrow \mathcal{O}$, which assigns a given image $\vec{x}$ (represented as a vector) in the input space $\mathcal{X}$ to one of a set of classes $\mathcal{O}$. For example, with the Mnist dataset as inputs, f classifies each image as one of ten digits '0'-'9'. As shown in Figure 2, the last layer of f often employs a softmax function. Specifically, let

$$\vec{z} \triangleq [z_1, z_2, \ldots] \qquad \vec{\sigma} = \frac{\exp(\vec{z})}{\sum_i \exp(z_i)} \triangleq [\sigma_1, \sigma_2, \ldots]$$

respectively be the input and output of this layer. Then σ_i is the probability that $\vec{x}$ belongs to the i^{th} class. The predicted class of $\vec{x}$ is given by $f(\vec{x}) = \arg\max_i \sigma_i$.

We consider DNNs obtained via supervised learning. Specifically, to train a DNN f, the algorithm takes a training set, of which each instance $(\vec{x}, o) \in \mathcal{X} \times \mathcal{O}$ constitutes an input and its ground-truth class, and determines the parameter setting of f via minimizing a loss function $\ell(f(\vec{x}), o)$ (e.g., the cross entropy of ground-truth classes and f's outputs).

2.2 Attack Methodologies

Targeting a DNN f deployed in use, the adversary attempts to trigger f to misbehave by feeding it with carefully crafted inputs. Given a genuine input $\vec{x}$ correctly classified by f, the adversary generates an adversarial one $\vec{x_\epsilon}$ by perturbing $\vec{x}$ with an insignificant amplitude (e.g., a few pixels). The difference of $\vec{x}$ and $\vec{x_\epsilon}$ ($\vec{r} = \vec{x_\epsilon} - \vec{x}$) is referred to as the *perturbation vector* (PV).

We differentiate two attack scenarios. In an untargeted attack, the adversary is interested in simply forcing f to misclassify, i.e., $f(\vec{x_\epsilon}) \neq f(\vec{x})$. In a targeted attack, she further desires for a particular target output o_ϵ, i.e., $f(\vec{x_\epsilon}) = o_\epsilon$. In the following we focus our discussion on targeted attacks, while the extension to untargeted attacks is straightforward.

A variety of attack models have been proposed in literature [14, 21, 34, 8]. Despite their variations in concrete crafting strategies, they all roughly follow a two-step procedure: (i) *saliency estimation:* the adversary assesses the impact of changing each input component on the classification output; (ii) *perturbation selection:* the adversary uses the input saliency information to select and perturb a subset of input components. Based on the concrete implementation of the two steps, we classify existing attacks in two major categories.

2.2.1 Linear Crafting Attacks

The class of linear attacks estimate the impact of distorting different input components on f's output via linear approximations and find the PV $\vec{r}$ that maximizes the probability of the target output o_ϵ. Next we detail two representative attack models.

Goodfellow's Attack. Goodfellow *et al.* [14] proposed the first linear attack model, which computes the gradient of the loss function ℓ with respect to the input $\vec{x}$ and determines $\vec{r}$ as a gradient sign step in the direction that increases the probability of the target output o_ϵ.

Specifically, let $\text{sign}(\nabla_{\vec{x}}\ell(f(\vec{x}), o_\epsilon))$ be the gradient sign of ℓ with respect $\vec{x}$ for given o_ϵ. Then the PV is defined as: $\vec{r} = -\delta \cdot \text{sign}(\nabla_{\vec{x}}\ell(f(\vec{x}), o_\epsilon))$, where δ is a parameter controlling the distortion amplitude (i.e., l_∞-norm of $\vec{r}$). Often, the adversary seeks the minimum δ to achieve misclassification: $\min_\delta f(\vec{x} + \vec{r}) = o_\epsilon$.

Huang's Attack. Huang *et al.* [21] introduced another linear attack model, which is constructed upon a linear approximation of the output of the softmax layer, i.e., the last layer of a DNN model (see Figure 2).

Let $\vec{\sigma_\epsilon}$ be the softmax output of $\vec{x_\epsilon} = \vec{x} + \vec{r}$. This attack approximates $\vec{\sigma_\epsilon}$ using a linear form: $\vec{\sigma_\epsilon} \approx \vec{\sigma} + J\vec{r}$, where $J = \frac{\mathrm{d}\vec{\sigma}}{\mathrm{d}\vec{x}}$ is the Jacobian matrix. Assume the original output o and target output o_ϵ respectively correspond to the j^{th} and j_ϵ^{th} row of J, denoted by J_j and J_{j_ϵ}. Let $\Delta_J \triangleq J_{j_\epsilon} - J_j$. To trigger $f(\vec{x_\epsilon}) = o_\epsilon$, the adversary seeks $\vec{r}$ that maximizes the difference of the j_ϵ^{th} and j^{th} component of $\vec{\sigma_\epsilon}$, i.e., $\max_{\vec{r}} \Delta_J \cdot \vec{r}$.

Similar to [14], this attack determines $\vec{r}$ as a step in the sign direction of Δ_J, i.e., $\vec{r} = \delta \cdot \text{sign}(\Delta_J)$, where δ controls the distortion amplitude[1].

2.2.2 Nonlinear Crafting Attacks

In linear attacks, the PVs are found in a single attempt. In comparison, nonlinear attacks construct the PVs iteratively. At each round, the adversary estimates the impact of each input component on the classification output, then selects several components to perturb, and checks whether the updated input causes the target misclassification. Next we detail two representative nonlinear attacks.

Papernot's Attack. Papernot *et al.* [34] proposed a saliency map to guide the crafting process. Intuitively, this map describes the impact of each input component on the output. Given the current input $\vec{x}$ (with previously selected components perturbed) and target output o_ϵ (corresponding to the j_ϵ^{th} component of $\vec{\sigma}$), the i^{th} component is associated with a pair of measures:

$$\alpha_i = \frac{\partial \sigma_{j_\epsilon}}{\partial x_i} \qquad \beta_i = \sum_{j \neq j_\epsilon} \frac{\partial \sigma_j}{\partial x_i}$$

where α_i is its impact on the probability of o_ϵ, while β_i is its impact on all the other classes.

The attack consists of multiple iterations of a greedy procedure. At each round, the component with the largest value of $(-\alpha \cdot \beta)$ is selected and flipped (to either '1' or '-1'), and the saliency map is updated accordingly. This process continues until the resulting input is misclassified as o_ϵ. The distortion amplitude is defined by the number of distorted components, i.e., l_1-norm of $\vec{r}$.

Carlini's Attack. In response to the defensive distillation method [35], Carlini and Wagner introduced a nonlinear attack [8], which differs from [34] in two aspects:

- To compensate for the gradient vanishing due to defensive distillation, the input to the softmax layer is artificially amplified by τ times, where τ is the "temperature" used by defensive distillation. The output of the softmax layer is thus: $\vec{\sigma} = \exp(\frac{\vec{z}}{\tau}) / \sum_j \exp(\frac{z_j}{\tau})$.
- The saliency values of input components are defined as $|\alpha - \beta|$ rather than $(-\alpha \cdot \beta)$. This modification reduces the complexity of perturbation selection from $O(n^2)$ to $O(n)$, where n is the number of input components. At each iteration, a pair of input components with the largest saliency values are selected and flipped.

2.2.3 Linear vs. Nonlinear Attacks

Linear attacks require computing gradient or Jacobian only once, while nonlinear attacks often involve multiple rounds of gradient or Jacobian computation. Given their efficiency advantage, linear attacks can be exploited to craft a large number of adversarial inputs.

Meanwhile, existing linear attacks often measure the distortion amplitude by l_∞-norm of the PV, while existing nonlinear attacks attempt to minimize l_1-norm of the PV.

The empirical comparison of the characteristics of different attacks is detailed in § 3.

2.3 Defense Methodologies

A DNN's resilience against adversarial inputs is inherently related to its stability [5]. Intuitively, a DNN f is stable, if for any "proximate" inputs $\vec{x}$ and $\vec{x_\epsilon}$ (i.e., $||\vec{x_\epsilon} - \vec{x}||$ is small), $f(\vec{x})$ and $f(\vec{x_\epsilon})$ are similar. Motivated by this rationale, a plethora of solutions [18, 14, 21, 39, 35] have been proposed to improve DNN stability, which can be roughly classified in three major categories.

Data Augmentation. This class of methods improve DNN stability by proactively generating a set of adversarial inputs and incorporating them in the training process.

[1]In [21] Huang *et al.* also give the definition of optimal $\vec{r}$ when the distortion amplitude is measured by l_1- or l_2-norm of $\vec{r}$.

Formally, given a DNN f and a known attack g, via applying g over f, one generates an adversarial input $\vec{x_\epsilon}$ for each genuine instance $(\vec{x}, o)$. A new DNN f' is trained using an augmented objective function:

$$\min_f \sum_{(\vec{x}, \vec{x_\epsilon}, o)} (\alpha \cdot \ell(f(\vec{x}), o) + (1-\alpha) \cdot \ell(f(\vec{x_\epsilon}), o))$$

where the parameter α $(0 \leq \alpha \leq 1)$ balances the relative weight of genuine and adversarial inputs. For instance, Goodfellow *et al.* [14] suggested equal importance of genuine and adversarial inputs ($\lambda = 0.5$),

Nevertheless, these methods are inherently heuristic, without theoretical guarantee on the robustness or accuracy of the trained DNN models.

Robust Optimization. Another line of work proposed to improve DNN stability via directly altering its objective function. To be specific, one prepares a DNN f for the worst possible inputs by training it with an minimax objective function:

$$\min_f \max_{||\vec{r}|| \leq \delta} \ell(f(\vec{x} + \vec{r}), o) \qquad (1)$$

The training algorithm first searches for the "worst" PV (constrained by δ) that maximizes the loss function ℓ under the current setting of f; it then optimizes f with respect to this PV. This objective function essentially captures the misclassification error under adversarial perturbations.

Due to the complexity of DNN models, it is intractable to search for exact worst adversarial inputs. Certain simplifications are often made. Szegedy *et al.* [43] and Gu and Rigazio [18] proposed to search for $\vec{r}$ along the gradient direction of loss function, while Shaham *et al.* [39] and Miyato *et al.* [28] reformulated this framework for Kullback-Leibler divergence like loss functions.

Model Transfer. In this method, one transfers the knowledge of a teacher DNN f to a student DNN f' such that the model stability is improved.

For instance, Papernot *et al.* [35] proposed to employ distillation [2, 19], a technique previously used to transfer the knowledge of an ensemble model into a single model, to improve DNN stability. Specifically,

- The teacher DNN f is trained on genuine inputs; in particular, the input to the softmax layer (see Figure 2) is modified as $\vec{z}/\tau$ for given "temperature" $\tau(\tau > 1)$.
- One evaluates f on the training set and produces a new training set $\{(\vec{x}, \vec{\sigma})\}$, where $\vec{\sigma}$ encodes the predicted probability distribution ("soft label") of $\vec{x}$.
- By training the student DNN f' on the new training set under temperature τ, f' is expected to generalize better to adversarial inputs than f.

Additionally, there is recent work attempting to design new DNN architectures [18] or learning procedures [9] to improve DNN stability. Yet, the resulting models fail to achieve satisfying accuracy on genuine inputs. Due to space limitations, we focus our discussion on the above three classes of defense mechanisms.

3 Empirical Study

Next we empirically evaluate the effectiveness of existing defense solutions against varied attacks. To our best knowledge, this evaluation represents the most comprehensive study to date on a range of attack and defense models, and is thus interesting in its own right.

In a nutshell, we show that it is fundamentally challenging to defend against adversarial inputs, which are tailored to target DNNs and crafted with varying strategies. Unfortunately, existing defenses are inherently static. Although they improve DNN resilience against specific attacks, the resulting models, once trained and deployed, are unable to adapt to *a priori* unknown attacks. The adversary can thus circumvent such defenses by creating inputs exploiting new vulnerability of target DNNs.

3.1 Setting of Study

Datasets and DNN Models. To show the prevalence of attack vulnerabilities across different tasks, in our study, we use three benchmark datasets, Mnist [25], Cifar10 [22], and Svhn [31], which have been widely used to evaluate image classification algorithms. The details of datasets can be found in Appendix A.

We also consider three distinct DNN architectures and apply each to one of the datasets above. To be specific, we apply the convolutional neural network (Cnn) [23], maxout network (Mxn) [26], and network-in-network (Nin) [26] models to classifying the Mnist, Cifar10, and Svhn datasets, respectively. The implementation details of these DNN models are referred to Appendix B.

Attacks and Defenses. We implement all the attack models in § 2.2, which we refer to as G-, H-, P-, and C-Attack for brevity. In particular, following [14, 21], we set the limit of distortion amplitude for G- and H-Attack as 0.25 (i.e., $\delta \leq 0.25$ in § 2.2); for P- and C-Attack, as in [35], we fix this limit to be 112, i.e., the adversary is allowed to perturb no more than 112 pixels.

We implement one representative solution from each defense category in § 2.3. In particular, as data augmentation is attack-specific, we refer to the resulting models as G-, H-, P-, and C-trained DNN; as robust optimization is norm-specific, we train robust DNNs under both l_∞- and l_1-norm criteria. The implementation details of defense methods are referred to Appendix C.

Data	Original Model	Defense-Enhanced Models						
		Data Augmentation ($\alpha = 0.5$)				Robust Optimization		Model Transfer
		G-trained	H-trained	P-trained	C-trained	l_1-norm	l_∞-norm	($\tau = 40$)
Mnist	99.5%	98.8%	99.0%	99.0%	98.8%	98.1%	98.5%	98.9%
Cifar10	85.2%	64.4%	57.6%	75.9%	76.7%	72.1%	71.3%	80.5%
Svhn	95.2%	91.3%	85.0%	91.2%	92.4%	90.2%	81.5%	86.0%

Table 1. Classification accuracy of original and defense-enhanced DNN models with respect to benchmark datasets.

Data	Attack	Original Model	Defense-Enhanced Models						
			Data Augmentation ($\alpha = 0.5$)				Robust Optimization		Model Transfer
			G-trained	H-trained	P-trained	C-trained	l_1-norm	l_∞-norm	($\tau = 40$)
Mnist	G-	15.3%	5.9%	82.4%	40.0%	70.6%	21.2%	0.0%	1.18%
	H-	22.4%	7.1%	87.1%	84.7%	91.8%	22.4%	0.0%	1.18%
	P-	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	1.0%
	C-	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	**100.0%**
Cifar10	G-	96.5%	**100.0%**	**100.0%**	93.3%	**100.0%**	**98.1%**	79.7%	39.8%
	H-	91.9%	100.0%	100.0%	**100.0%**	93.8%	98.1%	71.2%	38.6%
	P-	**100.0%**	100.0%	100.0%	100.0%	100.0%	94.3%	**100.0%**	21.7%
	C-	100.0%	100.0%	100.0%	100.0%	100.0%	64.2%	98.31%	**100.0%**
Svhn	G-	99.5%	36.9%	4.39%	99.5%	99.5%	100.0%	0.0%	7.5%
	H-	94.6%	35.4%	5.37%	93.4%	94.9%	96.7%	0.0%	7.5%
	P-	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**100.0%**	**99.7%**	3.0%
	C-	100.0%	98.0%	98.1%	100.0%	99.5%	95.6%	90.8%	**100.0%**

Table 2. Resilience of original and defense-enhanced DNN models against adversarial input attacks.

3.2 "No Free Lunch"

Table 1 summarizes the classification accuracy of the original DNN models (Cnn, Mxn, Nin) trained over legitimate inputs and their defense-enhanced variants on the benchmark datasets (Mnist, Cifar10, Svhn).

Observe that the original models achieve accuracy (i.e., 99.5%, 85.2%, 95.2%) close to the state of the art [6]. In comparison, most of their defense-enhanced variants observe non-trivial accuracy drop. For example, the accuracy decreases by 4.7% from the original Mxn model to its defensive distilled variant (model transfer), while this drop is as significant as 20.8% in the case of the G-trained variant (data augmentation).

We thus conclude that the improvement of attack resilience is not "free lunch", often at the expense of classification accuracy. This observation is consistent with the theoretical investigation on the trade-off between DNN expressivity and robustness [11].

3.3 "No Silver Bullet"

Next we evaluate different DNNs' attack resilience. Under the limit of distortion amplitude, we measure the percentage of legitimate inputs in each testing set which can be converted to adversarial inputs by varied attacks. Table 2 summarizes the results. The most successful attack under each setting is highlighted.

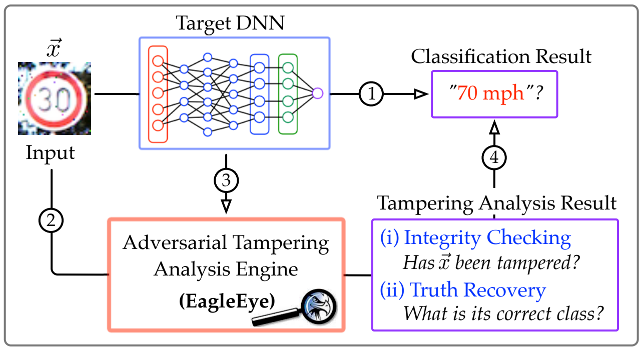
For the original models, most attacks, especially P- and C-Attack, achieve near-perfect success rates, implying the prevalence of vulnerabilities across DNN models.

The data augmentation method significantly improves DNN resilience against linear attacks. The success rate of G-Attack drops below 6% when facing G-trained Cnn. However, it is much less effective for more complicated DNNs or against nonlinear attacks. Both P- and C-Attack achieve near-perfect success rates against data augmented Mxn and Nin. This is because data augmentation is only capable of capturing simple, linear perturbations, while the space of PVs for nonlinear attacks and complex DNN models is much larger.

By considering worst-case inputs at every step of training, the robust optimization method leads to stronger resilience against linear attacks. For example, in the cases of Mnist and Svhn, the enhanced DNN models completely block G- and H-Attack. However, similar to data augmentation, robust optimization is ineffective against nonlinear attacks. This is partially explained by that the adversarial perturbations considered in training are essentially still linear (see Eq.(2)).

Model transfer is the only defense effective against nonlinear attacks. The success rate of P-Attack drops to 1% against defensive distilled Cnn, which is consistent with the results in [35]. However, this effectiveness is not universal. C-Attack, which is engineered to negate the gradient vanishing effects, is able to penetrate the protection of defensive distillation completely.

From the study above, we conclude that none of the existing defense solutions is a "silver bullet". While they improve DNN resilience against specific attacks, the resulting models are unable to adapt to new attack variants. There is thus an imperative need for attack-agnostic defense mechanisms.



Target DNN
$\vec{x}$
Input
Classification Result
"70 mph"?
Tampering Analysis Result
Adversarial Tampering Analysis Engine
(EagleEye)
(i) Integrity Checking
Has $\vec{x}$ been tampered?
(ii) Truth Recovery
What is its correct class?

Figure 3: Use case of EagleEye.

4 Defending DNN with EagleEye

Next we present EagleEye, an attack-agnostic adversarial tampering analysis engine. Its design is motivated by a set of desiderata, which we believe are expected for practical and effective defense mechanisms.

- Attack-agnostic defense. It should be universally effective against known and *a priori* unseen attacks.
- Intact DNN models. It should require no modification to DNN models, as such changes, especially to DNN architectures or training procedures, often result in unpredictable system behaviors.
- Light-weight execution. It should incur negligible performance overhead to the DL-powered system.

EagleEye satisfies all these desiderata. In its overview (§ 4.1), we show that EagleEye, following a modular design, requires no modification to DNN models or training methods; in its detailed description (§ 4.2), we demonstrate that EagleEye provides universal defense by making minimal assumptions regarding incoming attacks; in § 6, we further validate that EagleEye meets the requirement of light-weight execution by evaluating its empirical performance within DL-powered systems.

4.1 Overview

In contrast of existing solutions, EagleEye takes a completely new route: it attempts to discriminate adversarial inputs; moreover, for suspicious cases, it is instrumented to infer their correct classification outputs. Therefore, EagleEye provides much richer diagnosis information than existing defense solutions.

Specifically, as depicted in Figure 3, EagleEye is deployed as an auxiliary module within a DL-powered system. It exerts minimal interference with the classification task (①). Rather, for the given input $\vec{x}$ (②) and DNN (③), it offers on-demand adversarial tampering analysis: (i) it first runs *integrity checking* to assess the possibility that $\vec{x}$ has been maliciously tampered; (ii) if suspicious, it further performs *truth recovery* to infer $\vec{x}$'s correct classification. The analysis result is combined with the DNN's classification to form a comprehensive report for the operator's decision-making (④). Clearly the design of EagleEye satisfies the desiderata of intact DNN models. Next we focus on realizing attack-agnostic defense.

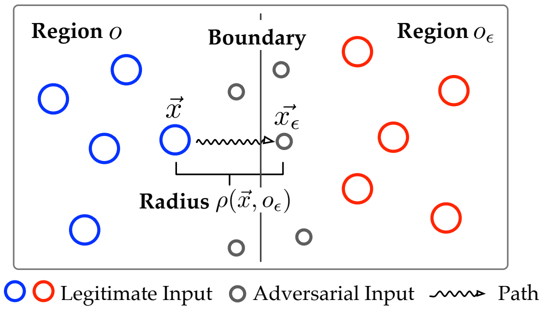


Figure 4: Concepts of boundary, path, and radius.

4.2 Minimality Principle

Despite their apparent variations, different attacks follow similar design principles, which entail invariant properties shared by adversarial inputs, independent of concrete attacks. In specific, we exploit the *minimality principle* underlying most attack models as the foundations for building attack-agnostic defenses.

Intuitively, to maximize the attack's evasiveness, the adversary often seeks the minimum possible distortion to convert genuine inputs to adversarial ones. Formally,

Definition 1 (Minimality Principle)**.** *Given the target DNN f, genuine input $\vec{x}$, and adversarial output o_ϵ, the attack seeks to solve an optimization problem as:*

$$\min_{\vec{r}} ||\vec{r}|| \quad s.t. \quad f(\vec{x}+\vec{r}) = o_\epsilon$$

For example, [14, 21] instantiate this problem with $||\cdot||$ defined as l_∞-norm, while [34, 8] consider l_1-norm.

To understand the entailments of this principle, we first introduce several key concepts (see Figure 4).

Definition 2 (Boundary)**.** *A DNN f partitions the input space (the topological space spanned by all the inputs) into non-overlapping regions. The inputs in each region are classified by f into the same class. Adjacent regions are separated by their* <u>*boundary*</u>*.*

Definition 3 (Path)**.** *For given inputs $\vec{x}$, $\vec{x}_\epsilon$ with $\vec{x}_\epsilon = \vec{x} + \vec{r}$, the PV $\vec{r}$ encodes a* <u>*path*</u> *from $\vec{x}$ to $\vec{x}_\epsilon$, of which the length is defined as the magnitude of $\vec{r}$, $||\vec{r}||$.*

Definition 4 (Radius)**.** *The path length of an input $\vec{x}$ to its nearest neighbor $\vec{x}_\epsilon$ in another class o_ϵ is referred to as $\vec{x}$'s* <u>*radius*</u> *to class o_ϵ, denoted by $\rho(\vec{x}, o_\epsilon)$.*

We now translate the minimality principle in the language of boundary, path, and radius: given a genuine input $\vec{x}$, among all the possible (adversarial) inputs in the target class o_ϵ, the adversary seeks $\vec{x}_\epsilon$ with the shortest path length from $\vec{x}$. Therefore, the minimality principle entails the following important properties:

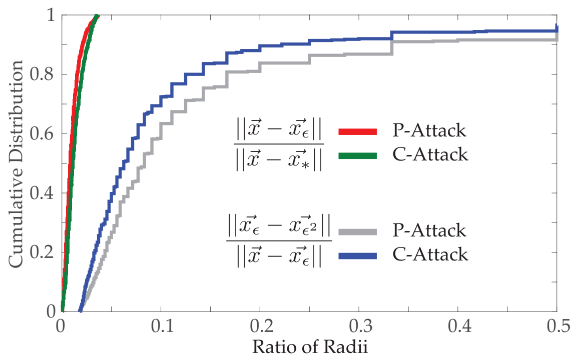


Figure 5: Cumulative distribution of the ratio of input $\vec{x}$'s shortest distance to adversarial and genuine inputs (on the Svhn dataset).

- Property 1: the path length of $\vec{x}$ to $\vec{x_\epsilon}$ approximates $\vec{x}$'s radius to o_ϵ, $\rho(\vec{x}, o_\epsilon)$.
- Property 2: $\vec{x_\epsilon}$ tends to distribute extremely close to the boundary of o and o_ϵ.

Next we empirically verify these properties, while their analytical treatment is referred to § 5.

Specifically, given a genuine input $\vec{x}$ (in class o) and an adversarial one $\vec{x_\epsilon}$ (in class o_ϵ) generated by an attack $\mathcal{A}$, in the given dataset, we find $\vec{x}$'s closest genuine counterpart $\vec{x_*}$ in class o_ϵ, i.e., $||\vec{x} - \vec{x_*}||$ is minimized. We then compute the ratio of $\vec{x}$'s distance to $\vec{x_\epsilon}$ and $\vec{x_\epsilon}$: $||\vec{x} - \vec{x_\epsilon}||/||\vec{x} - \vec{x_*}||$.

Figure 5 shows the cumulative distribution of such ratios with respect to P- and C-Attack on the Cifar10 dataset (similar results observed on other datasets and attacks). Observe that most ratios lie in the interval of $[0, 0.01]$, regardless of attacks, suggesting that $\vec{x}$ resides much closer to its nearest adversarial neighbor in o_ϵ than to its genuine counterpart. Thus $||\vec{x} - \vec{x_\epsilon}||$ approximates $\vec{x}$'s radius to o_ϵ.

Further, by applying $\mathcal{A}$ to the adversarial input $\vec{x_\epsilon}$, we generate another adversarial one[2] $\vec{x_{\epsilon^2}}$ in class o; similarly, $||\vec{x_\epsilon} - \vec{x_{\epsilon^2}}||$ approximates $\vec{x_\epsilon}$'s radius to o, $\rho(\vec{x_\epsilon}, o)$. We then compute the quantity of $||\vec{x_\epsilon} - \vec{x_{\epsilon^2}}||/||\vec{x} - \vec{x_\epsilon}||$, i.e., a proxy for $\rho(\vec{x_\epsilon}, o)/\rho(\vec{x}, o_\epsilon)$.

Figure 5 shows the cumulative distribution of such ratios. Across both attacks, over 80% of the ratios concentrate in the interval of $[0, 0.2]$, indicating $\vec{x_\epsilon}$ distributes closer to the boundary than its genuine counterpart $\vec{x}$.

4.3 Building EagleEye

These properties provide the premise for building effective differentiators to identify adversarial inputs: for a given input $\vec{x}$ (classified by f as o), we measure its radii to all other classes, among which we find the minimum one: $\min_{o_\epsilon \neq o} \rho(\vec{x}, o_\epsilon)$, referred to as its *adversarial radius* (AR).[3] With Property 1 and 2, we can differentiate genuine and adversarial inputs via examining their ARs.

However, to realize this idea, we face two major challenges. First, the radius metrics are attack-specific, e.g., it is measured differently by G- and P-Attack. Directly measuring radii is at most effective for specific attacks. Second, even for known attacks, finding an optimal threshold is difficult; even if it exists, it tends to vary with concrete datasets and DNN models.

To tackle the first challenge, we propose *adversarial radius probing* (ARP), an attack-neutral method to indirectly approximate AR. In specific, it employs semi-random perturbations and measures an input $\vec{x}$'s AR as the minimum distortion amplitude (referred to as its AR probe or probe) required to change its classification outputs.

To address the second challenge, we apply a bootstrapping method to remove the need for error-prone parameter tuning. In specific, for the given input $\vec{x}$, we generate a set of *shadow inputs* $\{\vec{x_\epsilon}\}$ via semi-random perturbations. By comparing the probes of $\vec{x}$ and $\{\vec{x_\epsilon}\}$ (differential analysis), we estimate the likelihood that $\vec{x}$ has been maliciously tampered. If suspicious, we further infer $\vec{x}$'s correct classification output by analyzing the consensus of $\{\vec{x_\epsilon}\}$ (consensus analysis).

The framework of EagleEye is illustrates in Figure 6. Below we elaborate its key components, ARP in § 4.3.1, bootstrapping in § 4.3.2, and probe analysis in § 4.3.3.

4.3.1 Adversarial Radius Probing

In this stage, by performing random perturbations on the given input $\vec{x}$, EagleEye estimates the minimum distortion amplitude (i.e., probe) necessary to change its classification by f. Intuitively, $\vec{x}$'s probe, denoted by $\rho(\vec{x})$, reflects its AR in an attack-neutral manner.

Yet, it is often infeasible to estimate $\rho(\vec{x})$ by running random perturbations on all of $\vec{x}$'s components given its high dimensionality. We use a semi-random perturbation method: (i) *magnification* - EagleEye first dynamically identifies a set of *saliency regions* in $\vec{x}$ that maximally impact its classification; (ii) *diversification* - it performs random perturbations over such regions to estimate $\rho(\vec{x})$. We detail these two operations below.

Magnification. The magnification operation is loosely based on attention mechanisms [16, 1], inspired by that human's vision automatically focuses on certain regions of an image with "high resolution" while perceiving surrounding regions in "low resolution". We use a simple attention mechanism with computational advantages.

For the given input $\vec{x}$, we define a set of $d \times d$ spatial regions ($d = 4$ in our implementation). We generate all possible regions of $\vec{x}$ by applying an identity kernel of size $d \times d$ over $\vec{x}$, similar to a convolution operation.

[2]Strictly speaking, $\vec{x_{\epsilon^2}}$ is adversarial, given that it is created by perturbing another adversarial input $\vec{x_\epsilon}$. Here we broadly refer to all the artificially generated inputs as adversarial inputs.

[3]Similar definitions have also been discussed in [11, 5, 12].

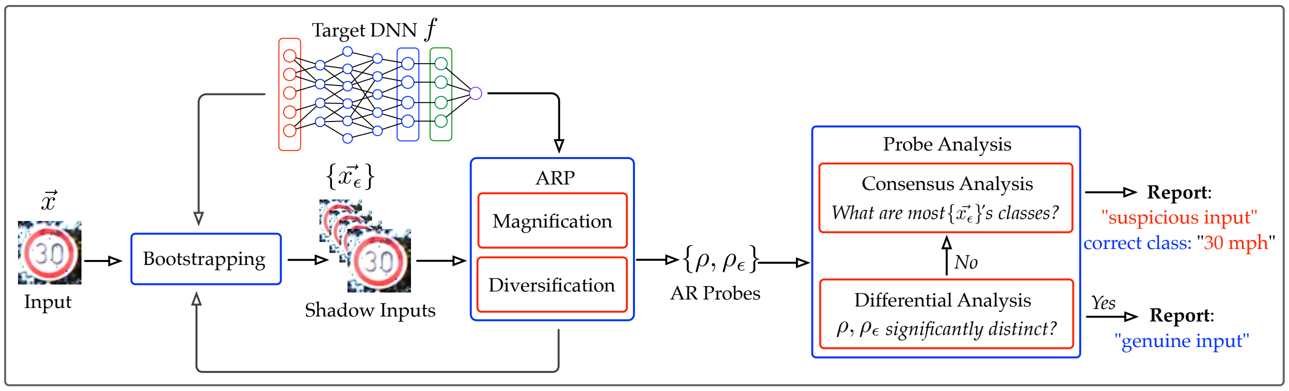


Figure 6: Illustration of EagleEye architecture. It discriminates adversarial inputs through the lens of adversarial radius analysis; for suspicious inputs, it further attempts to uncover their correct classification outputs.

To select the top n saliency regions that maximally impact $\vec{x}$'s classification, we apply a greedy approach. We sort the components of $\vec{x}$ according to their saliency (e.g., its gradient or Jacobian value) in descending order. Let $\text{rank}(\text{x})$ be the ranking of the component x. The saliency of a region π is the aggregated saliency contributed by all the components contained in π:

$$\text{saliency}(\pi) = \sum_{x \in \pi} c^{-\text{rank}(\text{x})}$$

where c ($c \geq 1$) is a constant.

This definition allows us to control the characteristics of selected regions. With large c, we focus on regions that cover the most influential components; while with small c (i.e., close to 1), we find regions that contain less influential components. The rationale behind balancing these two factors is as follows: the saliency landscape shifts as $\vec{x}$ is perturbed as $\vec{x_\epsilon}$; yet, due to the inherent continuity of DNNs, the variation tends to be local. By properly setting c, we are able to accommodate such shift and still capture the most influential component in $\vec{x_\epsilon}$.

We iteratively select the top n regions. Let $\mathcal{R}_i$ be the selected regions after the i^{th} iteration. We then update the saliency of each remaining region by removing the contributions by components contained in regions in $\mathcal{R}_i$. Formally, $\text{saliency}(\pi) = \sum_{x \in \pi \bigcap \nexists \pi' \in \mathcal{R}_i, x \in \pi'} c^{-\text{rank}(\text{x})}$. We then pick the region with the largest saliency among the remaining ones. We will discuss the optimal setting of k and c in § 6.

Diversification. At this step, given the saliency regions $\mathcal{R}$ of $\vec{x}$, we perform random perturbations on $\mathcal{R}$ to estimate $\vec{x}$'s probe $\rho(\vec{x})$.

With a little abuse of notations, let π be the set of components contained in the regions of $\mathcal{R}$. At each run, following a predefined distribution p (with parameter θ), we randomly select a subset of components in π to construct the perturbation vector $\vec{r}$, denoted by $\vec{r} \leftsquigarrow_\theta \pi$. A successful perturbation $\vec{r}$ results in $f(\vec{x} + \vec{r}) \neq o$, where o is $\vec{x}$'s current classification output by f.

In our current implementation, we instantiate θ as a uniform distribution and assume the flipping perturbation which sets an input component to either fully-on ('1') or fully-off ('1'). The distortion amplitude is therefore measurable by the sampling rate θ of p. While other instantiations are certainly possible (e.g., zipf distribution), we find that this instantiation is both (i) effective in discriminating adversarial inputs (§ 6) and (ii) simple to control by the system operator. Moreover, the large entropy of uniform distributions enhances the hardness for the adversary to evade the detection (§ 5 and § 6).

In the following, we consider the minimum sampling rate θ^* required to cause successful perturbations as $\vec{x}$'s probe, which indirectly reflects $\vec{x}$'s AR.

4.3.2 Bootstrap Operation

By randomly perturbing the given input $\vec{x}$, the bootstrap operation produces a set of adversarial inputs $\{\vec{x_\epsilon}\}$, which we refer to as $\vec{x}$'s *shadow inputs*. Intuitively, such shadow inputs represent $\vec{x}$'s near (if not the nearest) adversarial counterparts in other classes.

Specifically, to generate $\{\vec{x_\epsilon}\}$, we adopt the same semi-random perturbation strategy as in § 4.3.1, except for that the sampling rate is now fixed to be $\vec{x}$'s probe θ^*. This practice ensures that the generated shallow inputs are as close to $\vec{x}$ as possible. Further, as will be revealed in § 4.3.3, in consensus analysis, this practice also helps uncover the correct classification of $\vec{x}$ if it is adversarial. To provide stable estimation, we require the number of shadow inputs to be larger than a threshold k ($k \geq 4$ seems sufficient in practice § 6).

We then estimate the probes of all the shadow inputs, $\{\rho(\vec{x_\epsilon})\}$, which, together with $\rho(\vec{x})$, are fed as inputs to the phase of probe analysis. For simplicity of presentation, in the following, let ρ and ρ_ϵ denote the probes of $\vec{x}$ and $\vec{x_\epsilon} \in \{\vec{x_\epsilon}\}$ respectively.

4.3.3 Probe Analysis

In this phase, by analyzing the probes of the given input $\vec{x}$ and its shadow inputs $\{\vec{x_\epsilon}\}$, EagleEye determines the likelihood that $\vec{x}$ has been maliciously tampered (differential analysis); if so, it further attempts to recover its correct classification result (consensus analysis).

Differential Analysis. Recall that $\{\vec{x_\epsilon}\}$ represent $\vec{x}$'s near (if not the nearest) adversarial neighbors in other classes. Thus, if $\vec{x}$ itself is adversarial, $\vec{x}$ and $\vec{x_\epsilon}$ can be considered as adversarial versions of each other, thereby featuring similar ARs; otherwise, if $\vec{x}$ is a genuine input, the ARs of $\vec{x}$ and $\{\vec{x_\epsilon}\}$ tend to show significant difference (see Figure 5).

In differential analysis, we leverage this insight and examine the probes of $\vec{x}$ and each shadow input $\vec{x_\epsilon}$. Intuitively, a larger probe ratio of ρ/ρ_ϵ indicates that $\vec{x}$ is more likely to be genuine. Concretely, with a given shadow input $\vec{x_\epsilon}$, we estimates the likelihood that $\vec{x_\epsilon}$ is genuine as:

$$\text{genuine}_{\vec{x_\epsilon}}(\vec{x}) = \frac{1}{1+\exp(1-\rho/\rho_\epsilon)}$$

Here the sigmoid function converts ρ/ρ_ϵ to the interval of $(0,1)$, which we may roughly interpret as the "probability" that $\vec{x}$ is genuine. In specific, this probability is 0.5 if $\rho/\rho_\epsilon = 1$. The overall likelihood that $\vec{x}$ is genuine is computed by aggregating the results over all the shadow inputs: $\text{genuine}(\vec{x}) = \sum_{\vec{x_\epsilon}} \text{genuine}_{\vec{x_\epsilon}}(\vec{x})/|\{\vec{x_\epsilon}\}|$.

In our empirical evaluation in § 6, we find a threshold 0.625 works well across all the known attacks.

Consensus Analysis. As shown in Figure 6, if $\vec{x}$ passes the differential analysis, it is reported as "genuine"; otherwise, it is considered as a "suspicious" case and moved to the phase of consensus analysis, in which we attempt to infer its correct classification output.

To be specific, recall that an adversarial input $\vec{x}$ (in class o_ϵ) created from a genuine input (in class o) resides near to the boundary of o and o_ϵ. Thus, among its close adversarial neighbors in other classes, a majority of them should belong to class o. By leveraging this observation, we simply pick the most common class associated with the shadow inputs $\{\vec{x_\epsilon}\}$ as $\vec{x}$'s most likely correct classification output.

5 Analysis of EagleEye

In the preceding sections, we present EagleEye that applies adversarial radius analysis (ARA) to distinguish genuine and tampered inputs. Next, we analytically explore the effectiveness of ARA. Note that we do not intend to provide a definitive argument about using EagleEye (or ARA) to mitigate adversarial input attacks, but rather we view it as an initial step towards building universal, attack-agnostic defenses against adversarial inputs for DL and machine learning systems in general. In specific, our analysis attempts to draw the connection between ARA, DNN generalizability, and learning theory.

Furthermore, we discuss the adversary's possible countermeasures to evade EagleEye's detection. Recall that EagleEye is built on the premise that the attacks follow the minimality principle: the adversary attempts to minimize the distortion amplitude to maximize the attack's evasiveness. We explore attack variants that (partially) abandon this principle. This amounts to investigating the adversary's design spectrum: the tradeoff between the evasiveness with respect to EagleEye and the distortion amplitude (i.e., the evasiveness with respect to other detection mechanisms), which provides insights for the best practice of EagleEye.

5.1 Effectiveness of Radius Analysis

The fundamental premise of ARA is that genuine inputs are inclined to have larger adversarial radii than their adversarial counterparts. Below we provide the theoretical underpinnings of this property. For simplicity of exposition, we exemplify l_1-norm as the measure of distortion amplitude, while our discussion generalizes to other metrics as well.

From the view of an adversarial input $\vec{x_\epsilon}$, recall that attack $\mathcal{A}$ produces $\vec{x_\epsilon}$ by carefully perturbing a Ã§genuine input $\vec{x}$. Regardless of its concrete implementation, $\mathcal{A}$ is essentially designed to solve the optimization problem:

$$\min_{\vec{x_\epsilon}} ||\vec{x_\epsilon} - \vec{x}|| \quad \text{s.t.} \quad f(\vec{x_\epsilon}) = o_\epsilon$$

where $||\vec{x_\epsilon} - \vec{x}||$ reflects the perturbation amplitude. Assume that $\mathcal{A}$ operates on $\vec{x}$ with a sequence of t perturbations and $\vec{r}^{(i)}$ represent the perturbation vector at the end of the i^{th} iteration ($i = 1, 2, \ldots, t$), with $\vec{x_\epsilon} = \vec{x} + \vec{r}^{(t)}$. The minimality principle implies that $\mathcal{A}$ achieves the desired misclassification only after the t^{th} iteration, i.e.,

$$\begin{array}{c} f(\vec{x} + \vec{r}^{(t-1)}) = o \\ f(\vec{x} + \vec{r}^{(t)}) = o_\epsilon \end{array}$$

Therefore, $(\vec{x} + \vec{r}^{(t-1)})$ and $(\vec{x} + \vec{r}^{(t)})$ represent two inputs lying between the class boundary of o and o_ϵ. It is also noted that $\vec{r}^{(t-1)}$ and $\vec{r}^{(t-1)}$ differ only by a few components, depending on the concrete implementation of $\mathcal{A}$. For example, the attacks in [34, 8] perturb two input components at each round, and $\vec{r}^{(t-1)}$ and $\vec{r}^{(t-1)}$ differ by two components. We thus conclude that $\vec{x_\epsilon}$ resides extremely close to the class boundary of o and o_ϵ.

Next, we move on to explain the effectiveness of ARA from the perspective of a genuine input $\vec{x}$. Our intuition is that if $\vec{x}$ is classified as class o by DNN f with high confidence, its radii to the class boundaries induced by f must be reasonably large. To make this intuition more precise, we resort to statistical learning theory on the connection of classification confidence and ARA.

Definition 5 (Confidence). *The classification confidence of an input-output pair* $(\vec{x}, o)$ *(i.e., f classifies $\vec{x}$ as o) is measured by the difference of the largest and second largest probabilities in $f(\vec{x})$ (e.g., the softmax output). Formally,* $\phi(\vec{x}) = \min_{o_\epsilon \neq o} \sqrt{2}(\vec{\delta_o} - \vec{\delta_{o_\epsilon}}) \cdot f(\vec{x})$*, where* $\vec{\delta_i}$ *is the Kronecker delta vector with the* i^{th} *element being 1 and 0 otherwise and* $\cdot$ *denotes inner product.*

We now introduce an interesting result that connects the concept of classification confidence with ARA (adapted from [41] , pp. 14):

Theorem. *Given a DNN f and a genuine input $\vec{x}$, with $W^{(l)}$ being the weight matrix of the l^{th} layer of f, we have the following bound for the AR of $\vec{x}$:*

$$\rho(\vec{x}) \geq \frac{\phi(\vec{x})}{\prod_{W^{(l)}} ||W^{(l)}||_F}$$

where $|| \cdot ||_F$ represents Frobenius norm.

Intuitively, if a DNN f makes confident classification of $\vec{x}$, $\vec{x}$ tends to have a large AR. Astute readers may point to the possibility of using confidence instead of ARA to adversarial inputs. However, note that high confidence is only one sufficient condition for large ARs; as observed in our empirical evaluation in § 6 and previous work [14], there are adversarial inputs with high classification confidence but show small ARs.

Another implication of the above theorem is that increasing the classification confidence of DNNs is beneficial for discriminating adversarial inputs. Along this direction, defensive distillation [35] is designed exactly for this purpose: by increasing the temperature τ, it amplifies the probability difference in the outputs of DNNs. In § 6, we empirically show the synergistic effects of integrating defensive distillation and EagleEye.

5.2 Adversary's Dilemma

We now explore possible attack variants that attempt to evade EagleEye's detection or ARA in specific. Since EagleEye is built on top of the minimality principle underlying varied attack models, one possible way to evade its detection is to (partially) abandon this principle in preforming adversarial perturbations.

Specifically, following § 5.1, let $\vec{r}^{(i)}$ be the perturbation vector at the end of the i^{th} iteration and assume the adversary achieves the desired misclassification right after the t^{th} iteration: $f(\vec{x} + \vec{r}^{(t)}) = o_\epsilon$. Now, instead of stopping here, the adversary keeps on perturbing $\vec{x}$, attempting to increase its AR $\rho(\vec{x_\epsilon})$. Even in the ideal case, the adversary needs to perturb at least $\rho(\vec{x})$ input components, resulting in an approximate perturbation amplitude of $2 \cdot \rho(\vec{x})$. Intuitively, if $\vec{x}$ is selected at random, then the quantity of $2 \cdot \rho(\vec{x})$ represents the distance of two genuine inputs, while in real datasets, the difference of distinct inputs is fairly discernible even to human eyes. Furthermore, due to the high nonlinearity of DNNs, the extra perturbation amplitude is often much larger this lower bound. The empirical validation of this hypothesis is given in § 6.3, in which we also consider the adversary's another countermeasure of random perturbations.

This analysis above reveals a difficult dilemma for the adversary: she desires to preserve the AR of an adversarial input to evade EagleEye's detection; yet, to do so, she is forced to introduce extra perturbations sufficient to transform one genuine input to another, thereby significantly reducing the attack's evasiveness regarding other detection mechanisms (e.g., human vision).

6 Evaluation

In this section, we empirically evaluate the efficacy of EagleEye. Specifically, our experiments are designed to answer the following key questions.

- Q: *Does EagleEye effectively distinguish adversarial and genuine inputs in an attack-agnostic manner?*

  A: (§ 6.2) EagleEye achieves very high detection accuracy across benchmark datasets and attack models. For instance, its average recall and precision are 99.5% and 97.0% on the MNIST dataset. In particular, this performance is achieved under the same parameter setting without any tuning to datasets or attack models.

- Q: *Does EagleEye cause new attack-defense arm races?*

  A: (§ 6.3) To evade EagleEye's detection, the adversary has to abandon the minimality principle by significantly increasing the distortion amplitude, which weakens the attack's evasiveness with respect to other detection mechanisms and has fairly low success rate (less than 40% across benchmark datasets).

- Q: *Does EagleEye complement other defense solutions?*

  A: (§ 6.4) EagleEye exerts minimal interference to existing system components, and is compatible with any defense mechanisms. Further, defense-enhanced DNNs, with stronger generalization capabilities, provide even better foundations for EagleEye to operate, leading to above 4% increase in both precision and recall.

6.1 Experimental Settings

We use the same set of original DNN models and benchmark datasets as in the empirical study in § 3. The details of DNN models and datasets are referred to Appendix A and B. We also evaluate EagleEye against the set of representative attacks in § 2.2. The default setting of parameters is as follows: # patches $n = 8$, ranking coefficient $c = 1.25$, and # shadow inputs $k = 4$.

6.2 Discriminant Power of EagleEye

In this set of experiments, we show that EagleEye is capable of performing accurate detection of adversarial

Dataset	Metric	Attack Model			
		G-	H-	P-	C-
Mnist	Precision	95.0%	96.5%	98.4%	98.0%
	Recall	99.2%	100.0%	99.2%	99.6%
Cifar10	Precision	88.8%	90.8%	89.9%	91.2%
	Recall	98.4%	94.4%	96.4%	99.6%
Svhn	Precision	88.3%	88.6%	87.3	88.2%
	Recall	99.2%	96.4%	99.2%	98.8%

Table 3. EagleEye detection accuracy with respect to different benchmark datasets and attack models.

inputs in an attack-agnostic manner. We prepare the testing set as follows. We first randomly sample 5,000 inputs from each dataset, which form the pool of genuine inputs. We then apply all the attacks to each genuine input to generate its adversarial versions (with the adversarial target class randomly selected and the perturbation amplitude limited as 112 for P- and C-Attack and 0.25 for G- and H-Attack as in [14, 35]); those successfully crafted instances form the pool of adversarial inputs. Due to the high success rates of adversarial attacks (see Table 2), the genuine and adversarial pools are quite balanced.

We apply EagleEye to detect adversarial inputs and use the following metrics to measure its performance:

$$\text{Recall} = \frac{tp}{tp + fn} \qquad \text{Precision} = \frac{tp}{tp + fp}$$

where tp, fp, and fn represent the number of true positive, false positive, and false negative cases (adversarial: +, genuine: -). Intuitively, recall and precision measure the sensitivity and specificity of EagleEye.

Table 3 summarizes EagleEye's performance against varied attacks on the benchmark datasets. Observe that EagleEye provides universal, attack-agnostic protection for DNNs: across all the cases, EagleEye achieves high precision (above 87%) and recall (above 94%), indicating its strong discriminant power against adversarial inputs. Note that in all these cases, EagleEye has slightly better recall than precision. We hypothesize that those false positive cases are genuine inputs with low classification confidence (see § 5), while in the simplest Mnist dataset, most inputs have fairly high confidence scores, resulting in its lowest false positive rates.

We also examine EagleEye's impact on genuine cases misclassified by DNNs. For those cases, EagleEye detects them as genuine inputs with accuracy of 96.4%, 94.4%, and 95.6% on the Mnist, Cifar10, and Svhn dataset respectively, implying that EagleEye's performance is relatively independent of the DNN's accuracy.

We then evaluate EagleEye's effectiveness in uncovering the correct classification of adversarial inputs. Among the adversarial cases detected by EagleEye, we calculate the recovery rate as the proportion of inputs whose classes are correctly inferred. We find EagleEye's recovery is effective against both linear and nonlinear attacks.

Datasets	Attack Models			
	G-Attack	H-Attack	P-Attack	C-Attack
Mnist	62.2%	94.6%	61.2%	61.6%
Cifar10	70.0%	91.2%	42.8%	79.4%
Svhn	74.8%	91.6%	27.2%	47.6%

Table 4. Failure rates of adversarial inputs to reach desirable ARs with respect to benchmark datasets and attacks.

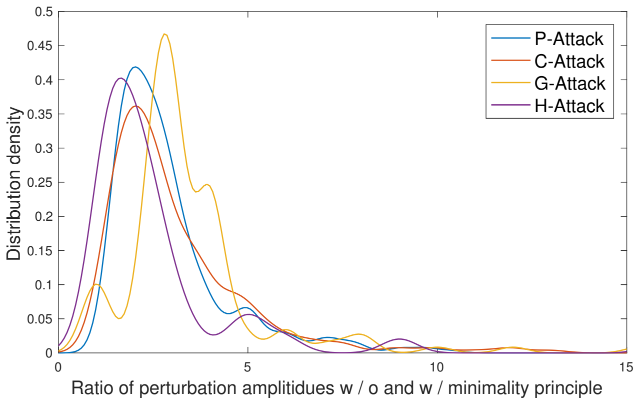


Figure 7: Distribution of ratio of distortion amplitudes (without vs. with minimality principle) on Svhn.

For example, against H-Attack, it achieves 85.6% recovery on the Mnist dataset; against P-Attack, it achieves 66.0% recovery on the Cifar10 dataset. Note that this performance is achieved under the same parameter setting without any adjustment towards datasets or attacks; thus, we believe EagleEye's performance can be further improved by fine parameter tuning. The experiments on parameter tuning are referred to Appendix E.

To summarize, in an attack-agnostic manner, EagleEye effectively discriminates maliciously tampered inputs and even uncovers their original classification outputs. It can be seamlessly deployed into any existing DL-powered systems. The analysis results of EagleEye can be combined with the DNN classification results to form comprehensive reports to enable more informative decision-making for the system operators.

6.3 Adversary's Countermeasures

Next we discuss adversary's possible countermeasures to evade EagleEye's detection and their implications. Remember that the fundamental cornerstone of EagleEye is the minimality principle underlying varied adversarial input attacks. Therefore, to evade EagleEye's detection, one natural option for the adversary is to abandon this principle. Rather than applying the minimum possible distortion, she attempts to find a suboptimal perturbation vector leading to larger ARs.

Specifically, after the adversarial input $\overrightarrow{x_\epsilon}$ achieves the misclassification o_ϵ, the adversary continues the perturbation process, in an attempt to increase $\overrightarrow{x_\epsilon}$'s AR. Following

Cases	**Dataset**		
	MNIST	CIFAR10	SVHN
defended by DD	100.0%	98.4%	90.8%
uncaptured by DD	100.0%	100.0%	100.0%

Table 5. Accuracy of EagleEye against adversarial inputs defended/uncaptured by defensive distillation (DD).

the analysis in § 5, here we empirically evaluate the implications of the adversary's countermeasure. Assume that for a given genuine input $\vec{x}$, the adversary desires to make $\vec{x}_\epsilon$ with AR comparable with that of $\vec{x}$.

We first investigate the cases that fail to achieve the desired AR under the given perturbation amplitude, which is increased to 448 for P- and C-Attack, and 1 for G- and H-Attack. Table 4 lists the failure rates of adversarial inputs on each dataset. For example, even by quadrupling the distortion amplitude, over 40% of the inputs cannot achieve ARs comparable with their genuine counterparts on CIFAR10. This is explained by that due to the highly nonlinear, nonconvex nature of DNNs, the AR of an adversarial input is a nonlinear function of the distortion amplitude as well. Thus, solely increasing the amplitude does not necessarily lead to the desired AR.

Moreover, we examine those cases that indeed reach the desired ARs. Let $\vec{x}_\epsilon$ and $\vec{x}_\epsilon'$ represent the adversarial input generated with and without the minimality principle. We measure the ratio of their distortion amplitudes. Figure 7 plots the distribution of such ratios on the SVHN dataset (experiments on other datasets in Appendix E). Note that regardless of the concrete attacks, for a majority of adversarial inputs, in order to make them evasive against EagleEye's detection, the adversary has to amplify the distortion amplitude by more than 2.5 times than that guided by the minimality principle. Such large distortion would be detectable by potential anomaly detection systems or even human vision [34]. This empirical evidence also validates our analysis in § 5.

Besides performing delicate perturbations to increase the ARs of adversarial inputs, the adversary may also try random perturbations in hope of finding adversarial inputs with satisfying ARs. We simulate this countermeasure by generating adversarial inputs with the minimum principle and then applying random perturbation to them. Figure 8 shows the ratio of distortion amplitudes after and before random perturbations (experiments on other datasets in Appendix E). Again, this countermeasure leads to significantly increased distortion amplitudes.

To summarize, EagleEye creates a difficult dilemma for the adversary: she has to balance the attack's evasiveness against EagleEye and other potential detection systems. In many cases, this balance is difficult to find, due to the highly nonlinear, nonconvex nature of DNNs.

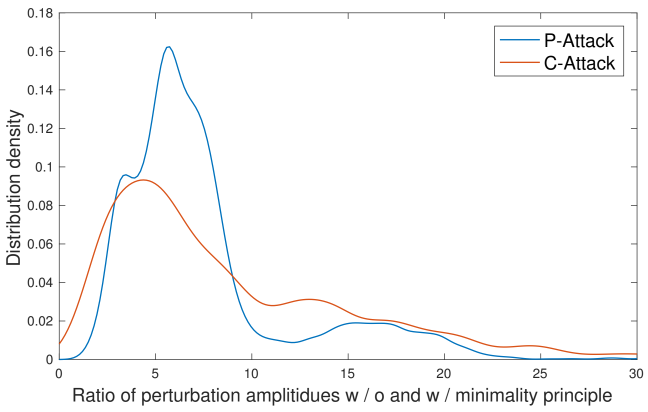


Figure 8: Distribution of ratio of distortion amplitudes (random perturbation vs. minimality principle) on SVHN.

6.4 EagleEye and Other Defenses

One advantage of EagleEye is that it exerts minimal interference with existing system components. This feature makes it complementary with other defense mechanisms, while their integration often leads to synergistic effects. In this set of experiments, we validate this hypothesis by investigating the effects of applying EagleEye on top of defensive distillation (DD) [35]. As shown in Table 2, while effective against G-, H-, and P-Attack, DD is vulnerable to adaptively designed attacks (e.g., C-Attack). For example, C-Attack achieves near-perfect success rates on benchmark datasets.

First we consider adversarial inputs that are successfully defended by DD. In particular, we apply P-Attack over original DNNs and collect all the adversarial inputs which pass original DNNs but are defended by DD. For each adversarial input $\vec{x}_\epsilon$ in this category, DD recognizes its correct class but is unaware that $\vec{x}_\epsilon$ is adversarial. The first row of Table 5 lists EagleEye's accuracy of detecting such cases as adversarial. It is clear that for cases successfully defended by DD, EagleEye can provide additional diagnosis information for the system operator.

Second we consider adversarial inputs that penetrate the protection of DD. In particular, we apply C-Attack over defensive distilled DNNs and collect all the successfully generated adversarial inputs. Table 5 lists EagleEye's accuracy of detecting such cases as adversarial. EagleEye achieves perfect detection rate in this category. It is clear for cases that penetrate the protection of DD, EagleEye provides another safe net.

We are not arguing to replace exiting defenses with EagleEye. Rather, we believe it is beneficial to integrate complementary defense mechanisms, which significantly sharpens the edge of vulnerability to adaptive attacks.

7 Discussion

The preceding analytical and empirical analysis shows that EagleEye, adversarial radius analysis (ARA) in specific, effectively discriminates adversarial inputs and even reveals their correct classification outputs.

One limitation of EagleEye is that its effectiveness, to some extent, depends on the generalization capabilities of DNNs, although practically useful DNNs need to have sufficient generalizability (DNN generalizability and robustness are two related but distinct properties [12]). We thus argue that the research on improving DNN generalizability and that on defense mechanisms against adversarial inputs complement each other. It is our ongoing research to improve the effectiveness of EagleEye against ambiguous inputs and weak DNN.

We measure the distortion amplitude using l_1- or l_∞-norm. There are other metrics for measuring input distance. For example, crafting adversarial malware samples to evade malware detection may require adopting other metrics [13, 17]. We plan to investigate how to extend our solution to other metrics and perturbations. Yet, we believe the minimality principle still holds. For example, the malware author still wishes to preserve malware's functional behaviors.

In § 6, we empirically show the synergistic effects of combining defensive distillation and EagleEye. It is expected because defense-enhanced DNNs, with stronger generalization capabilities than original models, provide better foundations for ARA to operate. Thus, we consider the integration of other defense mechanisms (e.g., data augmentation and robust optimization) and EagleEye as a promising future direction to explore.

Finally, it is worth emphasizing that EagleEye does not create new attack vectors. It can be deployed compatibly with existing defense solutions. Its premise, the minimality principle, is an underlying principle followed by many attack models [14, 21, 34, 8]. Even if the adversary knows that EagleEye is deployed, the only way to evade its detection is to amplify the adversarial distortion amplitude, which however reduces the attack's evasiveness with respect to other defense mechanisms. Therefore, EagleEye indeed creates a difficult dilemma for the adversary.

8 Additional Related Work

Next we review three categories of related work: adversarial machine learning, deep learning-specific attacks and defenses, and robustness of deep neural networks.

Lying at the core of many security-critical domains, machine learning systems are increasingly becoming the targets of malicious attacks [4, 20, 3]. Two primary threat models are considered in literature: (i) poisoning attacks, in which the attackers pollute the training data to eventually compromise the learning systems [7, 46, 36], and (ii) evasion attacks, in which the attackers modify the input data at test time to trigger the learning systems to misbehave [10, 27, 30]. Yet, for ease of analysis, most of the work assumes simple learning models (e.g., linear classifier, support vector machine, logistic regression) deployed in adversarial settings.

Addressing the vulnerabilities of deep learning systems to adversarial inputs is more challenging for they are designed to model highly nonlinear, nonconvex functions [32, 37]. One line of work focuses on developing new attacks against DNNs [14, 21, 44, 34, 8], most of which attempt to find the minimum possible modifications to the input data to trigger the systems to misclassify. The detailed discussion of representative attack models is given in § 2.2. Another line of work attempts to improve DNNs resilience against such adversarial attacks [14, 18, 21, 35]. However, these defense mechanisms often require significant modifications to either DNN architectures or training processes, which may negatively impact the classification accuracy of DNNs. Moreover, as shown in § 3, the defense-enhanced models, once deployed, can often be fooled by adaptively engineered inputs or by new attack variants. To our best knowledge, this work represents an initial step to attack-agnostic defenses against adversarial attacks.

Finally, another active line of research explores the theoretical underpinnings of DNN robustness. For example, Fawzi *et al.* [11] explore the inherent trade-off between DNN capacity and robustness; Feng *et al.* [12] seek to explain why neural nets may generalize well despite poor robustness properties; and Tanay and Griffin [45] offer a theoretical explanation for the abundance of adversarial inputs in the input manifold space.

9 Conclusion

In this paper, we presented a new approach to defend deep learning (DL) systems against adversarial input attacks. Our work was motivated by the observations that the fundamental challenges to tackle adversarial inputs stem from their adaptive and variable nature, while static defenses can often be circumvented by adaptively engineered inputs or by new attack variants. We developed a principled approach that, by leveraging the underlying design principles shared by varied attacks, discriminates adversarial inputs in a universal, attack-agnostic manner. We designed and implemented EagleEye, a prototype defense engine which can be readily deployed into *any* DL systems, requiring no modification to existing components. Through comprehensive adversarial tampering analysis, EagleEye enables more informative decision-making for the operators of deep learning systems. Our empirical evaluations showed that EagleEye, when applied to three benchmark datasets, detected nearly 96% adversarial inputs generated by a range of attacks.

References

[1] Almahairi, A., Ballas, N., Cooijmans, T., Zheng, Y., Larochelle, H., and Courville, A. Dynamic capacity networks. In *ICML* (2016).

[2] Ba, J., and Caruana, R. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems 27*. 2014.

[3] Barreno, M., Nelson, B., Joseph, A. D., and Tygar, J. D. The security of machine learning. *Mach. Learn. 81*, 2 (2010), 121–148.

[4] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., and Tygar, J. D. Can machine learning be secure? In *ASIACCS* (2006).

[5] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., and Criminisi, A. Measuring Neural Net Robustness with Constraints. *ArXiv e-prints* (2016).

[6] Benenson, R. What is the class of this image? `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`, 2016.

[7] Biggio, B., Nelson, B., and Laskov, P. Poisoning attacks against support vector machines. In *ICML* (2012).

[8] Carlini, N., and Wagner, D. Defensive Distillation is Not Robust to Adversarial Examples. *ArXiv e-prints* (2016).

[9] Chalupka, K., Perona, P., and Eberhardt, F. Visual Causal Feature Learning. *ArXiv e-prints* (2014).

[10] Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D. Adversarial classification. In *KDD* (2004).

[11] Fawzi, A., Fawzi, O., and Frossard, P. Analysis of classifiers' robustness to adversarial perturbations. *ArXiv e-prints* (2015).

[12] Feng, J., Zahavy, T., Kang, B., Xu, H., and Mannor, S. Ensemble Robustness of Deep Learning Algorithms. *ArXiv e-prints* (2016).

[13] Fogla, P., and Lee, W. Evading network anomaly detection systems: Formal reasoning and practical techniques. In *CCS* (2006).

[14] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints* (2014).

[15] Goodfellow, I. J., Warde-farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout Networks. In *ICML* (2013).

[16] Gregor, K., Danihelka, I., Graves, A., Jimenez Rezende, D., and Wierstra, D. DRAW: A Recurrent Neural Network For Image Generation. *ArXiv e-prints* (2015).

[17] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *ArXiv e-prints* (2016).

[18] Gu, S., and Rigazio, L. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *ArXiv e-prints* (2014).

[19] Hinton, G., Vinyals, O., and Dean, J. Distilling the Knowledge in a Neural Network. *ArXiv e-prints* (2015).

[20] Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B. I., and Tygar, J. D. Adversarial machine learning. In *AISec* (2011).

[21] Huang, R., Xu, B., Schuurmans, D., and Szepesvari, C. Learning with a Strong Adversary. *ArXiv e-prints* (2015).

[22] Krizhevsky, A., and Hinton, G. Learning Multiple Layers of Features from Tiny Images. *Technical report, University of Toronto* (2009).

[23] LeCun, Y., and Bengio, Y. The handbook of brain theory and neural networks. MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258.

[24] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (2015), 436–444.

[25] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE* (1998), vol. 86, pp. 2278–2324.

[26] Lin, M., Chen, Q., and Yan, S. Network In Network. In *ICLR* (2014).

[27] Lowd, D., and Meek, C. Adversarial learning. In *KDD* (2005).

[28] Miyato, T., Maeda, S.-i., Koyama, M., Nakae, K., and Ishii, S. Distributional Smoothing with Virtual Adversarial Training. *ArXiv e-prints* (2015).

[29] Mnih, V., Heess, N., Graves, A., and kavukcuoglu, k. Recurrent models of visual attention. In *NIPs*. 2014.

[30] Nelson, B., Rubinstein, B. I. P., Huang, L., Joseph, A. D., Lee, S. J., Rao, S., and Tygar, J. D. Query strategies for evading convex-inducing classifiers. *J. Mach. Learn. Res. 13* (2012), 1293–1332.

[31] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2011).

[32] Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR* (2015).

[33] Økland, A. Improving Back-Propagation by Adding an Adversarial Gradient. *ArXiv e-prints* (2015).

[34] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swamil, A. The limitations of deep learning in adversarial settings. In *Euro S&P* (2016).

[35] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. Distillation as a defense to adversarial perturbations against deep neural networks. In *S&P* (2016).

[36] Rubinstein, B. I., Nelson, B., Huang, L., Joseph, A. D., Lau, S.-h., Rao, S., Taft, N., and Tygar, J. D. Antidote: Understanding and defending against poisoning of anomaly detectors. In *IMC* (2009).

[37] Sabour, S., Cao, Y., Faghri, F., and Fleet, D. J. Adversarial Manipulation of Deep Representations. In *ICLR* (2016).

[38] Sang-Hun, C. GoogleâĂŹs Computer Program Beats Lee Se-dol in Go Tournament. `http://www.nytimes.com/2016/03/16/world/asia/korea-alphago-vs-lee-sedol-go.html`, 2016.

[39] Shaham, U., Yamada, Y., and Negahban, S. Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization. *ArXiv e-prints* (2015).

[40] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS '16.

[41] Sokolic, J., Giryes, R., Sapiro, G., and Rodrigues, M. R. D. Robust Large Margin Deep Neural Networks. *ArXiv e-prints* (2016).

[42] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *ICML* (2013).

[43] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *ArXiv e-prints* (2013).

[44] Tabacof, P., and Valle, E. Exploring the Space of Adversarial Images. *ArXiv e-prints* (2015).

[45] Tanay, T., and Griffin, L. A Boundary Tilting Persepective on the Phenomenon of Adversarial Examples. *ArXiv e-prints* (2016).

[46] Xiao, H., Biggio, B., Nelson, B., Xiao, H., Eckert, C., and Roli, F. Support vector machines under adversarial label contamination. *Neurocomput. 160*, C (2015), 53–62.

Appendix

A. Datasets

The Mnist dataset [25] constitutes a set of 28×28 greyscale images of handwritten digits ('0'-'9'), with 60K training and 10K testing samples.

The Cifar10 dataset [22] consists of 32×32 color images from ten classes (e.g., 'airplane', 'automobile', 'bird'), split into 50K training and 10K testing samples.

The Svhn dataset [31] comprises color images of house numbers collected by Google Street View. We consider the format of 32×32 pixel images. The task is to classify the digit around the center of each image. There are 73K and 26K digits in the training and testing sets, respectively.

All the datasets are centered and normalized such that the value of each pixel lies in the interval of $[-1, 1]$.

B. Implementation of DNN Models

In the following we present the detailed architectures of DNN models used in our empirical evaluation.

Convolutional Neural Network. The DNN model applied to classifying the Mnist dataset is a convolutional neural network (Cnn). In particular, we adopt an architecture similar to [35], which is summarized in Table 6. In addition, we apply dropout (rate = 0.5) at both fully connected layers.

Layer	Definition
ReLU Convolutional	# filters: 32, kernel: 3×3
ReLU Convolutional	# filters: 32, kernel: 3×3
Max Pooling	pool: 2×2, stride: 1
ReLU Convolutional	# filters: 64, kernel: 3×3
ReLU Convolutional	# filters: 64, kernel: 3×3
Max Pooling	pool: 2×2, stride: 1
ReLU Fully Connected	# units: 256
ReLU Fully Connected	# units: 256
Softmax	# units: 10

Table 6. Convolutional network architecture (Mnist)

Maxout Network. The maxout network (Mxn) model generalizes conventional Cnn models by employing maxout activation functions, which pool over multiple affine feature maps in addition to pooling over adjacent spatial locations as in convolution operations. Therefore, an maxout convolutional layer is defined as the composition of one convolutional layer, one regular pooling layer, and one maxout pooling layer.

To classify the Cifar10 dataset, we adopt an architecture similar to that in [15], which consists of three maxout convolutional layers and one maxout fully connected layer, as detailed in Table 7. We apply dropout (rate = 0.5) at each maxout convolutional layer.

Layer	Definition
ReLU Convolutional	# filters: 128 kernel: 5×5, padding: 4
Max Pooling	pool: 3×3, stride: 2
Maxout	# units: 64 (2 pieces/unit)
ReLU Convolutional	# filters: 256 kernel: 5×5, padding: 3
Max Pooling	pool: 3×3, stride: 2
Maxout	# units: 128 (2 pieces/unit)
ReLU Convolutional	# filters: 256 kernel: 5×5, padding: 3
Max Pooling	pool: 3×3, stride: 2
Maxout	# units: 128 (2 pieces/unit)
ReLU Fully Connected	# units: 2,000
Maxout	# units: 400 (5 pieces/unit)
Softmax	# units: 10

Table 7. Maxout network architecture (Cifar10)

Network-in-Network. To classify the Svhn dataset, we apply an network-in-network (Nin) model, which features another distinct architecture. In conventional Cnn, the convolution filter is essentially a generalized linear model (Glm) for the underlying data patch. The Nin architecture replaces Glm with an "micro neural network" structure which is a nonlinear function approximator to enhance the abstraction capability of the local model.

Following [26], we use multilayer perceptron (Mlp) as the instantiation of micro network. In the resulting mlpconv layer, the Mlp is shared among all local receptive fields, while the feature maps are obtained by sliding the Mlp over the input in a similar manner as Cnn.

Specifically, our Nin model comprises three mlpconv layers, followed by one average pooling layer and one softmax layer, as summarized in Table 8. Dropout (rate = 0.5) is applied at each mlpconv layer.

Layer	Definition
ReLU Convolutional	# filters: 96, kernel: 5×5, padding: 2
ReLU Convolutional	# filters: 96, kernel: 1×1
Max Pooling	pool: 3×3, stride: 2
ReLU Convolutional	# filters: 192, kernel: 5×5, padding: 2
ReLU Convolutional	# filters: 192, kernel: 1×1
Max Pooling	pool: 3×3, stride: 2
ReLU Convolutional	# filters: 192, kernel: 3×3, padding: 1
ReLU Convolutional	# filters: 192, kernel: 1×1
ReLU Convolutional	# filters: 10, kernel: 1×1
Average Pooling	pool: 8×8
Softmax	# units: 10

Table 8. Network-in-network architecture (Svhn)

C. Implementation of Defense Methods

We implement one representative defense mechanism from each category of defense strategies in § 2.3.

Data Augmentation. Recall that with data augmentation, adversarial inputs are incorporated in training a more robust DNN f'. Our implementation of this defense mechanism proceeds as follows.

- We begin with an initialized DNN f and an attack of interest $\mathcal{A}$;
- At each iteration, regarding the current f, we apply

$\mathcal{A}$ to a minibatch $\{(\vec{x}, o)\}$ randomly sampled from the training set, and generate an augmented minibatch $\{(\vec{x}, \vec{x_\epsilon}, o)\}$; We update f using the objective function:

$$\min_f \sum_{(\vec{x}, \vec{x_\epsilon}, o)} \left(\ell(f(\vec{x}), o) + \ell(f(\vec{x_\epsilon}), o)\right)$$

We instantiate the attack $\mathcal{A}$ as each of G-, H-, P-, C-Attack and refer to the resulting models as G-, H-, P-, and C-trained DNN respectively.

Robust Optimization. Recall that with robust optimization, one improves DNN stability by preparing it for the worst-case inputs, which is often formulated as an minimax optimization framework.

We implement the framework in [39], wherein the loss function is minimized over adversarial inputs generated at each parameter update. To be specific, it instantiates Eqn. (1) with finding the perturbation vector $\vec{r}$ (with respect to an input instance $(\vec{x}, o)$) that maximizes its inner product with the gradient of objective function:

$$\vec{r} = \arg\max_{\vec{r}} \langle \nabla \ell(f(\vec{x}), o)), \vec{r} \rangle \tag{2}$$

For l_∞-norm, we limit $||\vec{r}||_\infty \leq 0.2$; for l_1-norm, we limit $||\vec{r}||_1 \leq 2 * 2$ (no more than 2 pixels).

Model Transfer. Recall that with model transfer, the knowledge in a teacher DNN f (trained on legitimate inputs) is extracted and transferred to a student DNN f', such that f' generalizes better to adversarial inputs. Specifically, we implement the defensive distillation mechanism in [35] as a representative method of model transfer. We set the temperature $\tau = 40$ in the experiments as suggested by [35].

D: Training of DNN Models

The training process identifies the optimal setting for a DNN's parameters w. Due to complex structures of DNN models and massive amount of training data, we apply Stochastic Gradient Descent with Nesterov momentum [42] as the optimization algorithm. More specifically, let $\ell(w)$ represent the objective function (e.g., the cross entropy of ground-truth class labels and DNN models' outputs). At each epoch, the gradient of ℓ with respect to w is computed over a "mini-batch" (e.g., each of 128 samples) sampled from the training data via a back-propagation procedure. The update rule of w is given by:

$$\begin{cases} v = \mu v - \lambda \cdot \nabla \ell(w + \mu v) \\ w := w + v \end{cases}$$

where v represents the "velocity", μ denotes the "momentum", λ is the learning rate, and ∇ is the gradient operator. The training process repeats until the objective function converges.

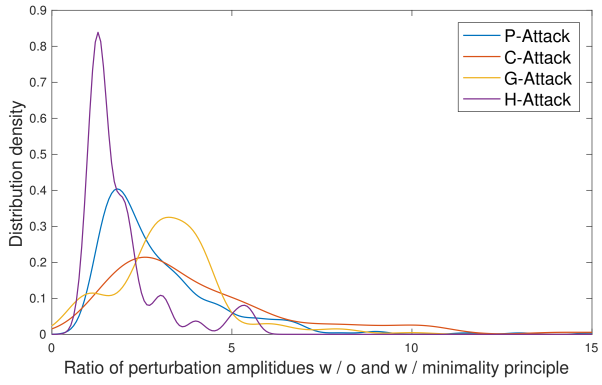


Figure 9: Distribution of ratio of distortion amplitudes (without vs. with minimality principle) on Cifar10.

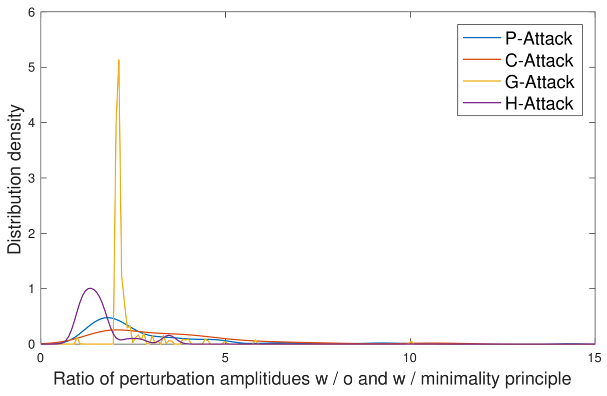


Figure 10: Distribution of ratio of distortion amplitudes (without vs. with minimality principle) on Mnist.

In implementation, the default learning rates for Mnist, Cifar10, and Svhn datasets are respectively set as 0.1, 0.01, and 0.01; the default momentum μ is fixed as 0.9; the optimization algorithm is run for up to 240 epochs. In addition, we apply an adaptive learning rate scheme: at each epoch, let ℓ and ℓ^* respectively be the loss of current epoch and the best loss thus far; the learning rate is adjusted as: $\lambda = \lambda \cdot \exp(\frac{\ell^* - \ell}{s})$, where $s = 2.5$ if $\ell^* \geq \ell$ and $s = 0.75$ if $\ell_* < \ell$. The DNN models are trained using the training set of each dataset. All the algorithms are implemented on top of Theano[4], a Python-based DL library. All the experiments are performed using an array of 4 Nvidia GTX 1080 GPUs.

E: Additional Experiments

Here we list the experiment results in addition to those in § 6, including the impact of parameter tuning on EagleEye's performance, and the tradeoff between the attack's evasiveness regarding EagleEye and other defenses.

[4]Theano: http://deeplearning.net/software/theano/

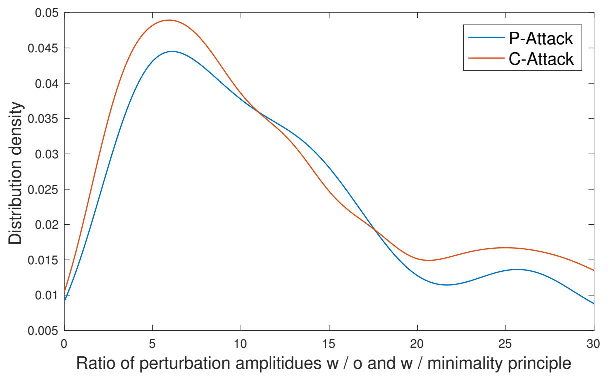


Figure 11: Distribution of ratio of distortion amplitudes (random perturbation vs. minimality principle) on Cifar10.

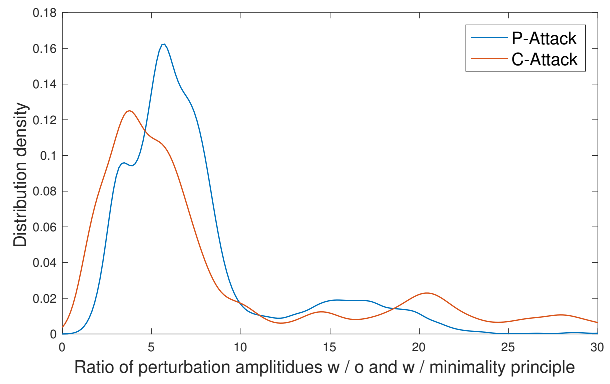


Figure 12: Distribution of ratio of distortion amplitudes (random perturbation vs. minimality principle) on Mnist.

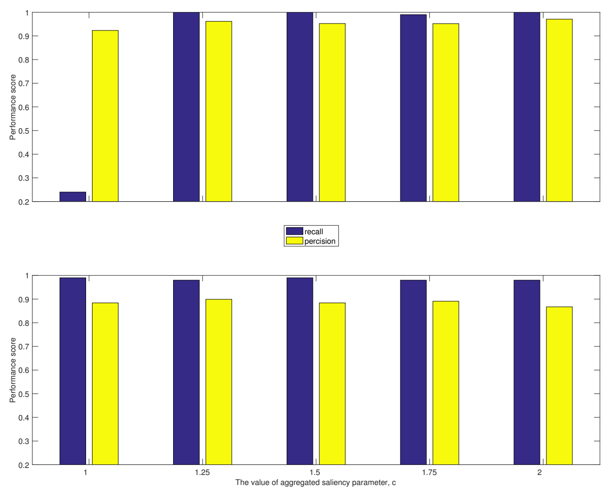


Figure 13: Impact of number of patches c on EagleEye's performance.

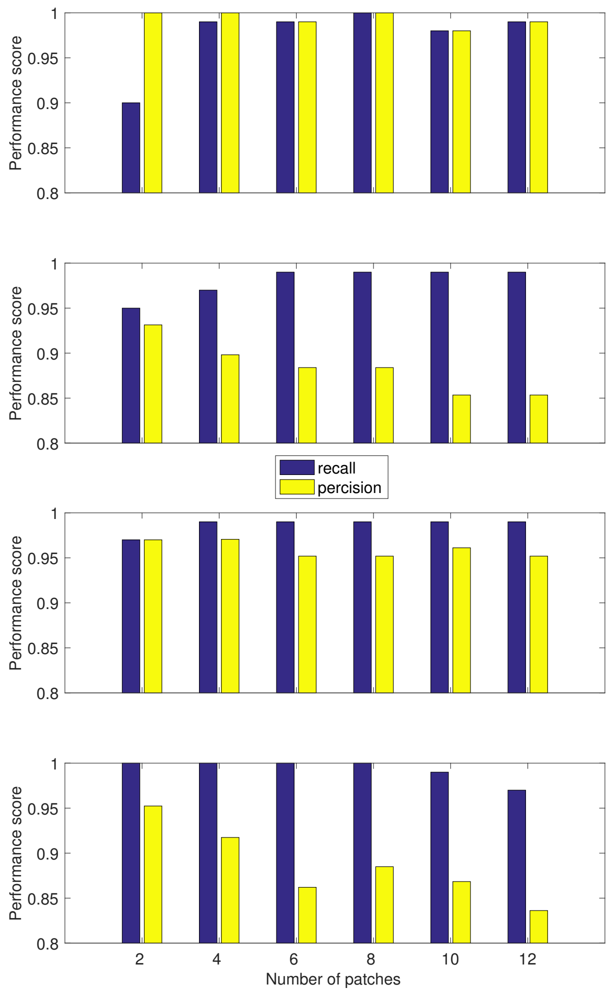


Figure 14: Impact of number of patches n on EagleEye's performance.