A Comprehensive Survey of Data Poisoning Attacks and their Detection Techniques

Preprint · May 2025

DOI: 10.13140/RG.2.2.20084.67207

Deepon Halder
Indian Institute of Engineering Science and Technology, Shibpur
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Anshika Gupta
Indian Institute of Engineering Science and Technology, Shibpur
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Diya Ghosh
Indian Institute of Engineering Science and Technology, Shibpur
1 PUBLICATION **0** CITATIONS
SEE PROFILE

A Comprehensive Survey of Data Poisoning Attacks and their Detection Techniques

Deepon Halder[1], Anshika Gupta[1], Diya Ghosh[1], and Prof. Hafizur Rehman[2]

[1]IIEST Shibpur
{2023itb043.deepon, 2023itb096.anshika, 2023itb097.diya}@students.iiests.ac.in
[2]IIEST Shibpur
h.rehman@it.iiests.ac.in

Abstract

Data poisoning attacks present a significant threat to the integrity and reliability of machine learning models, particularly during the training phase. These attacks compromise model behavior by introducing maliciously crafted inputs into the training data, leading to degraded performance or targeted misclassification. In this survey, we provide a comprehensive overview of data poisoning attack strategies under both white-box and black-box threat models. We categorize and analyze a wide array of detection methods employed to mitigate the effects of adversarial examples (AEs). These include both supervised and unsupervised techniques, ranging from network-invariant approaches such as SafetyNet and RAID, to statistical and model uncertainty-based methods like Mahalanobis distance and Maximum Mean Discrepancy (MMD). We also explore auxiliary model-based defenses, feature squeezing, denoising methods like PixelDefend and MagNet, and gradient-based detectors. Special attention is given to recent advancements in adaptive and dynamic adversarial training techniques, as well as object-based detection strategies such as UnMask. By systematically comparing these methodologies, we highlight their strengths, limitations, and applicability across different attack scenarios.

1 Introduction

Machine learning (ML), a central branch of artificial intelligence (AI), has made impressive strides in recent years, especially in tasks that mimic human perception and reasoning. From recognizing objects in images and translating languages to diagnosing diseases and powering self-driving cars, ML is now at the heart of many everyday technologies (33; 31). This rapid progress is fueled by access to massive datasets, powerful computing hardware, and constant innovation in deep learning (DL) architectures.

Among the many challenges ML tackles, image classification stands out as a flagship application. It's everywhere-from helping doctors spot tumors in medical scans to enabling autonomous vehicles to safely navigate busy streets. Deep learning models, particularly Convolutional Neural Networks (CNNs) like VGG16, ResNet, InceptionV3, and MobileNet, have set new benchmarks for accuracy on popular datasets such as MNIST, CIFAR-10, SVHN, Tiny ImageNet, and ImageNet (57; 25; 64; 28). Specialized models like R-CNN, Fast R-CNN, and YOLO are now standard tools for object detection, while transformer-based models such as BERT and GPT-2 have transformed natural language processing (54; 53; 14; 52).

Despite these successes, deep learning systems have a critical weakness: they are surprisingly vulnerable to adversarial attacks. These attacks involve making tiny, carefully chosen changes to the input data-changes so subtle that humans can't notice them, but which can completely fool a model into making the wrong prediction with high confidence (63; 23). This vulnerability is especially concerning in high-stakes settings like healthcare and security, where mistakes can have serious consequences.

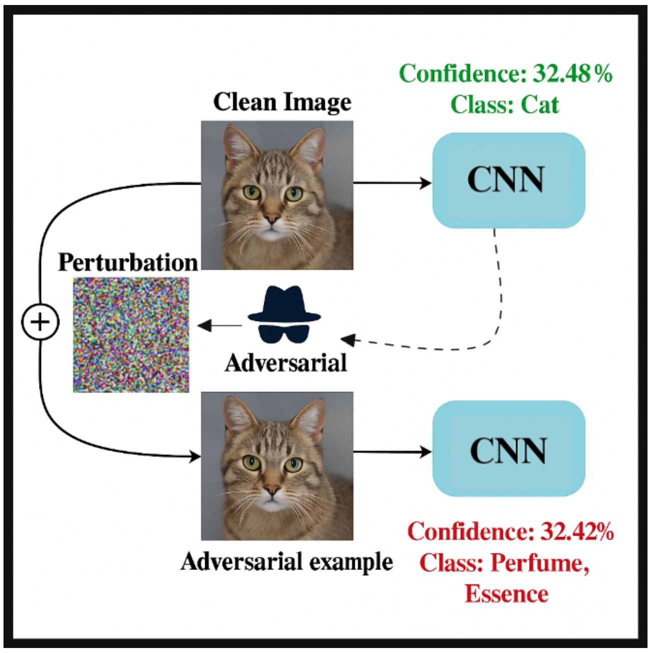

Figure 1: Addition of Noise/Pertubation to the image fools the model to think otherwise

Adversarial attacks generally fall into two main categories: poisoning attacks and evasion attacks. Poisoning attacks tamper with the training data to insert hidden backdoors into the model. Evasion attacks, on the other hand, happen at test time: attackers add small, calculated tweaks to inputs, causing misclassifications without changing the overall meaning or appearance of the image (5; 47). The attacker's knowledge of the model defines the attack scenario:

- **White-box attacks** assume the attacker knows everything about the model, including its structure and parameters, which makes it easier to craft effective adversarial examples.
- **Black-box attacks** assume the attacker has no internal knowledge and relies on the fact that adversarial examples often "transfer" between models.

These threats aren't limited to image classification. They also affect object detection, language processing, speech recognition, cybersecurity, and even real-world physical systems, making adversarial robustness a universal concern for the ML community (8; 32). In response, researchers have proposed various defenses, including adversarial training, input preprocessing, gradient masking, and feature denoising. Unfortunately, these methods often come with trade-offs, such as reduced model accuracy or vulnerability to more sophisticated attacks (65).

Recently, detection-based defenses have gained attention. Instead of trying to make models immune to adversarial examples, these approaches focus on spotting adversarial inputs before they reach the model. Detection methods range from statistical analyses and feature denoising to consistency checks, auxiliary classifiers, and strategies that look for invariance in the data (42; 17). While promising, there's still a lack of comprehensive evaluations comparing these techniques across different attack types and datasets.

In this review, we aim to fill that gap by offering a systematic, in-depth analysis of adversarial example detection methods, with a focus on test-time attacks in image classification. Our main contributions are:

- We provide a detailed taxonomy of state-of-the-art adversarial detection techniques, organized by both the attacker's knowledge (white-box, black-box) and the detection strategy.
- We present a large-scale experimental comparison of these methods across various attack types and benchmark datasets, including MNIST and CIFAR-10.

- We explore the characteristics of adversarial examples and how they affect detection performance, shedding light on why some methods work better than others.

2 Adversial Attacks

2.1 Introduction

In deep learning, **Deep Neural Networks (DNNs)** and **Convolutional Neural Networks (CNNs)** are commonly used as fitting functions f that learn to map raw input data to corresponding labels (33; 31; 57). These networks use interconnected neurons to extract and process features from labeled data.

Let X denote the input space (e.g., images), and Y the label space (e.g., classification labels). The data is assumed to be drawn from a joint probability distribution $P(X, Y)$ over $X \times Y$. A model $f : X \to Y$ defines a *prediction function*.

The training process involves a *loss function* $\ell : Y \times Y \to \mathbb{R}$, which quantifies the error between the predicted label $f(x)$ and the true label y. Let the training set be $S_m = \{(x_i, y_i)\}_{i=1}^m \subseteq (X \times Y)^m$, where each sample is independently and identically distributed (i.i.d.) according to $P(X, Y)$.

The objective during training is to minimize the **empirical risk**:

$$\hat{r}(f \mid S_m) = \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)$$

Training is typically carried out using **Stochastic Gradient Descent (SGD)**, which updates the model parameters θ by backpropagating the loss gradient (33).

A significant challenge in deep learning models is their vulnerability to *adversarial examples*. Given a trained model f, an input-label pair (x, y), and a maximum allowed perturbation $\varepsilon \in \mathbb{R}^n$, an adversary aims to find an adversarial input x' such that:

$$\|x' - x\| < \varepsilon \quad \text{and} \quad f(x') \neq f(x)$$

Such inputs deceive the model into misclassification despite being visually or semantically similar to the original input.

Threat Models

A threat model defines the conditions under which adversarial examples (AEs) are generated. Key factors used to classify threat models include adversary knowledge, adversarial goals, capabilities, and specificity (33). Here we focus on two principal axes:

1. Adversary Knowledge

- **White-box attacks**: The adversary has complete access to the model, including architecture, training data, outputs, and gradients. This enables precise AE generation.
- **Black-box attacks**: The adversary has no internal knowledge of the model and relies only on querying the model to observe outputs.
- **Gray-box attacks** (or semi-white-box): The adversary knows the training data but not the model architecture. A substitute model is used, leveraging the transferability of AEs.
- **Zero-knowledge adversary**: The adversary is unaware of the detection mechanisms and generates AEs without tailoring to defenses.

2. Adversarial Specificity

- **Targeted attacks**: The adversary generates an AE such that the model misclassifies the input into a specific incorrect label t. This typically involves optimizing to maximize the output probability of t. Targeted attacks are harder to generate and require higher perturbations.
- **Untargeted attacks**: The goal is to cause the model to misclassify the input into *any* label other than the correct one y. These attacks are easier and more likely to succeed with smaller perturbations.

2.2 Whitebox Attacks

2.2.1 LBFGS Attack

The **L-BFGS Attack** was one of the earliest adversarial attacks proposed(63). It is based on the *Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)* optimization algorithm. The idea is to generate an adversarial example x_0 by minimizing a specially designed loss function ℓ, defined to increase the model's confidence in a target class t(63).

To achieve this, the attack solves the following optimization problem:

$$\arg\min_{\delta} \; c\|\delta\| + \ell(x_0, t) \quad \text{subject to } x_0 \in [0,1]^n$$

Here, δ is the perturbation added to the original input x to produce $x_0 = x + \delta$, and c is a regularization parameter that balances the trade-off between minimizing the perturbation and maximizing the confidence in the target class(63). Since neural networks are non-convex, c is usually searched for iteratively to find the smallest possible perturbation.

To simplify the optimization, the authors reformulated the box-constrained problem as an unconstrained one by introducing a new variable w, such that:

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x$$

This transformation ensures that the resulting adversarial example stays within valid input bounds(8).

2.2.2 FGSM Attack

FGSM Attack (23), short for the *Fast Gradient Sign Method*, is the first adversarial attack designed specifically for the L_∞ norm. It leverages the gradients of the loss function with respect to the input to craft adversarial examples.

The main idea is to perturb the input x in the direction that increases the loss the most. This is done by taking the sign of the gradient of the loss with respect to the input and scaling it by a small parameter ϵ that controls the magnitude of the perturbation. The adversarial example x_0 is generated as:

$$x_0 = x + \epsilon \cdot \text{sign}(\nabla_x \ell(x, y))$$

Here, $\ell(x, y)$ is the loss function used by the model for input x and true label y, and $\nabla_x \ell(x, y)$ is the gradient of the loss with respect to x.

The constraint $\|x_0 - x\|_\infty < \epsilon$ ensures that the perturbation is small enough to be imperceptible to humans, yet large enough to fool the model. FGSM is simple, fast, and widely used as a baseline method in adversarial robustness research (33; 31; 57).

2.2.3 BIM Attack

BIM Attack (32), short for the *Basic Iterative Method*, is an extension of the FGSM attack that applies the perturbation multiple times in small steps, rather than in a single shot. This makes the attack more powerful and precise.

The attack starts from the original input x and repeatedly applies the FGSM update k times with a smaller step size α. The update rule at each iteration i is:

$$x_0^{i+1} = x_0^i + \alpha \cdot \mathrm{sign}(\nabla_x \ell(x_0^i, y))$$

where:

- $x_0^0 = x$ (i.e., the original input),
- x_0^i is the adversarial example after i steps,
- α is the step size for each iteration, and must satisfy $0 < \alpha < \epsilon$,
- $x_0^{i+1} \in [0,1]^n$ ensures that the perturbed input stays within valid bounds.

By taking multiple smaller steps in the direction of the gradient, BIM creates more effective adversarial examples than FGSM (23), while still keeping the overall perturbation within the allowed L_∞ constraint.

2.2.4 PGD Attack

PGD Attack (39), or *Projected Gradient Descent*, is an iterative adversarial attack method that builds upon the Basic Iterative Method (BIM) (32). However, unlike BIM which always starts from the original input, PGD begins from a randomly perturbed point within an L_p-norm ball around the input. This helps in finding stronger adversarial examples by escaping poor local minima.

The goal of PGD is to find an adversarial example that maximizes the model's loss (i.e., the most adversarial point) while keeping the perturbation within a bounded region. Formally, the perturbation must satisfy:

$$\|x_0 - x\|_p < \epsilon$$

where p can be 1, 2, or ∞, and ϵ defines the size of the perturbation budget.

PGD applies the following iterative update:

$$x_0^{i+1} = \Pi_{B_p(x,\epsilon)} \left(x_0^i + \alpha \cdot \mathrm{sign}(\nabla_x \ell(x_0^i, y)) \right)$$

Here, $\Pi_{B_p(x,\epsilon)}(\cdot)$ denotes the projection operator that ensures the updated point stays within the L_p ball of radius ϵ around the original input x.

PGD often uses multiple random restarts to increase the chances of finding a more adversarial example (39).

A more recent version, called **Auto-PGD** (11), improves on PGD by:

- Introducing a momentum term to the gradient updates,
- Automatically adjusting the step size across iterations based on the remaining attack budget,
- Restarting the attack from the best previously found adversarial point instead of a purely random one.

Auto-PGD is particularly effective in constrained settings where the number of iterations or the perturbation budget is limited (11).

2.2.5 CW Attack

CW Attack (8). Carlini and Wagner proposed an attack that improves upon the L-BFGS method with a few key modifications. First, they replaced the original loss function with a more targeted objective:

$$g(x') = \max\left(\max_{i \neq t} (Z(x')_i) - Z(x')_t, -k\right)$$

where Z is the softmax output of the model and k is a confidence parameter. This function aims to ensure that the model's output for the target class t is significantly higher than for any other class (8; 9).

In addition to this, they introduced six alternative objective functions (8), but the optimization problem generally became:

$$\min_{\delta} \|\delta\| + c \cdot g(x'), \quad \text{such that } x' \in [0,1]^n$$

Here, the goal is to find a small perturbation δ that changes the input x into an adversarial example $x' = x + \delta$, which the model confidently classifies as the target class t. The parameter c controls the balance between minimizing the perturbation and ensuring the target misclassification (9).

To simplify the optimization, they reformulated the box-constrained problem as an unconstrained one by introducing a new variable w and defining:

$$\delta = \frac{1}{2}(\tanh(w) + 1) - x$$

This ensures that the final perturbed input x' stays within valid bounds (9).

The CW attack was proposed in three main variants based on how similarity between the original and adversarial inputs is measured: using the L_0, L_2, or L_∞ norm (9; 55). It is widely considered one of the strongest attacks, outperforming earlier methods like FGSM and BIM (55).

Notably, the CW attack was one of the first to successfully break models trained with defensive distillation (8; 13). **DeepFool (DF) Attack** (44). Moosavi-Dezfooli *et al.* proposed an attack that produces smaller perturbations than FGSM, while maintaining the same fooling success rate (23).

2.2.6 DeepFool Attack

The idea is based on finding the smallest perturbation necessary to change the model's prediction. For a binary affine classifier $\mathcal{F} = \{x : f(x) = 0\}$, where $f(x) = w^T x + b$, the DF attack computes the orthogonal projection of the input x_0 onto the decision boundary of the classifier. This projection represents the minimal change required to alter the model's output (44).

The perturbation at each step is computed as:

$$\delta = -\frac{f(x)}{\|w\|^2} w$$

This ensures that the input moves just enough to cross the decision boundary.

At each iteration, DF solves the following optimization problem:

$$\arg\min_{\delta_i} \|\delta_i\|_2 \quad \text{subject to } f(x_i) + \nabla f(x_i)^T \delta_i = 0$$

This process is repeated, accumulating perturbations until the classifier's decision changes. The final perturbation is the sum of all intermediate steps (44).

DeepFool is especially effective at finding minimal adversarial examples and is widely used to evaluate model robustness against small but precise attacks (44).

2.3 Blackbox Attacks

2.3.1 ZOO Attack

ZOO Attack (10). When gradients of the model are not directly accessible-as is common in black-box settings-the *Zeroth Order Optimization (ZOO)* attack estimates them indirectly.

Instead of computing gradients explicitly, ZOO observes how the output of the model (typically the softmax confidence score $f(x)$) changes when small modifications are made to the input. These observations are then used to approximate gradients and second-order information (Hessian) (33).

ZOO uses symmetric difference formulas to estimate the gradient and second derivative with respect to the i-th input dimension (31):

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h}$$

$$\frac{\partial^2 f(x)}{\partial x_i^2} \approx \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h^2}$$

Here:

- h is a small constant step size,
- e_i is the standard basis vector with 1 at the i-th position and 0 elsewhere.

These approximations allow ZOO to craft adversarial examples without needing access to the internal gradients of the model, making it especially useful for attacking black-box models (57).

2.3.2 Pixel Attacks

Pixel Attacks (61). This family of attacks began with the *One-Pixel Attack*, where the idea is surprisingly simple yet effective: modify just a single pixel in the input image to fool a deep learning model.

To find the optimal pixel change, the attack uses *Differential Evolution (DE)*, which is a type of evolutionary algorithm (EA), to solve the following optimization problem (61; 6):

$$\max f_{\text{adv}}(x + \delta) \quad \text{such that} \quad \|\delta\|_0 \leq d$$

Here:

- f_{adv} is the adversarial objective function (e.g., model confidence for the target class),
- δ represents the perturbation,
- $\|\delta\|_0 \leq d$ ensures that only d pixels are changed, with $d = 1$ in the case of the original One-Pixel Attack (61).

Later, Kotyan *et al.* (30) extended this concept by proposing two new variants:

- **Pixel Attack (PA)**: This allows modification of more than one pixel by setting $d > 1$ in the same optimization formulation.
- **Threshold Attack (TA)**: Instead of constraining the number of changed pixels, TA uses the L_∞ norm to limit the magnitude of perturbations and solve the same adversarial objective.

These pixel-level attacks are particularly interesting because they show how even minimal and localized changes can deceive powerful deep learning models (61; 30).

2.3.3 ST Attack

Engström et al. (16) demonstrated that deep learning (DL) models can be sensitive to small spatial changes in input data, such as translations and rotations. To highlight this vulnerability, they introduced the Spatial Transformation (ST) attack. The goal of this attack is not just to fool the model, but also to improve its robustness by encouraging the use of data augmentation techniques during training (33; 31).

The ST attack is formulated as an optimization problem:

$$\max_{\delta_u,\delta_v,\theta} \ell(f(x'), y), \quad \text{where } x' = T(x; \delta_u, \delta_v, \theta)$$

Here:

- T is a spatial transformation function,
- δ_u is the translation in the x-direction,
- δ_v is the translation in the y-direction,
- θ is the rotation angle,
- x' is the transformed version of the original input x,
- $f(x')$ is the model's prediction on the transformed input,
- ℓ is the loss function comparing the prediction to the true label y.

In essence, the ST attack finds the transformation parameters that maximize the model's prediction error, exposing its sensitivity to spatial variations (57).

2.3.4 SA Attack

The Square Attack (SA) (3) is a black-box adversarial attack that uses a random search strategy. At each step of the algorithm, it selects localized, square-shaped regions within the input image and applies perturbations within an ϵ-bounded range. These perturbations are randomly positioned and colored, forming a patch-based attack pattern.

The goal of the attack is to find a perturbation δ that minimizes the model's confidence in the correct class while ensuring the perturbation remains within a certain norm constraint. This is expressed by the following optimization problem (3; 66):

$$\min_{\delta} \ \ell(f(x'), y) \quad \text{such that} \quad \|\delta\|_p \leq \epsilon, \quad x' = x + \delta, \quad x' \in [0,1]^n$$

The loss function is defined as:

$$\ell(f(x'), y) = f_y(x') - \max_{k \neq y} f_k(x')$$

Here:

- $f(x')$ represents the model's output probabilities for the perturbed input x',
- $f_y(x')$ is the probability assigned to the true class y,
- $f_k(x')$ is the probability assigned to any other class k,
- ϵ bounds the perturbation under the L_p norm,
- The attack supports two norm types: L_2 and L_∞.

By repeatedly applying these square-shaped perturbations, the algorithm tries to reduce the model's confidence in the correct label, ultimately causing misclassification (3; 66; 22).

3 Experiments on Attacks

3.1 Methodology

In this study, we designed and evaluated two convolutional neural network (CNN) models to analyze the effect of each attack. Each model was trained and evaluated on two widely used image datasets: MNIST and CIFAR-10. The architecture details for both CNNs are provided in Appendix A and Appendix B. Our experimental pipeline consisted of the following steps:

1. **Baseline Evaluation:** We trained both CNNs on their respective datasets and evaluated their performance on clean (unperturbed) test data.
2. **Adversarial Evaluation:** After obtaining baseline accuracies, we applied all the adversarial attacks
3. **Performance Comparison:** The results of the attacks were analyzed and compared against the baseline to assess each model's robustness to adversarial perturbations.

3.2 Results

- **Sample Selection**: A test set comprising 100 to 500 samples is randomly selected from the broader test dataset to ensure representativeness and mitigate selection bias.
- **Adversarial Generation**: For each selected sample, an adversarial variant is generated by applying targeted perturbations designed to challenge the model's robustness.
- **Model Evaluation**: Both the original and adversarial samples are processed through the model.
- **Performance Assessment**: The accuracy on adversarial samples serves as an indicator of the model's defences to adversarial attacks.

Table 1: Results By Attacks(by Accuracy)

ATTACK TYPE	ATTACK NAME	CIFAR 10 MODEL	MNIST MODEL
Baseline	No Attack	86.11%	99.00%
Whitebox Attacks	BIM	10.94%	7.81%
	CW	6.25%	14.06%
	DeepFool	14.06%	12.50%
	FGSM	26.56%	18.75%
	LBGFS	12.50%	15.62%
	PGD	13.71%	28.12%
Blackbox Attacks	Pixel	84.38%	76.56%
	Square	12.00%	10.94%
	ST	79.69%	65.62%
	Zoo	3.12%	46.88%

4 Adversial Detection Techniques

There's ongoing debate about whether AE detection methods should be considered proper defenses. While both detection and defense methods share the broader goal of combating adversarial attacks, they focus on different objectives. Defense methods try to ensure that both clean and adversarial

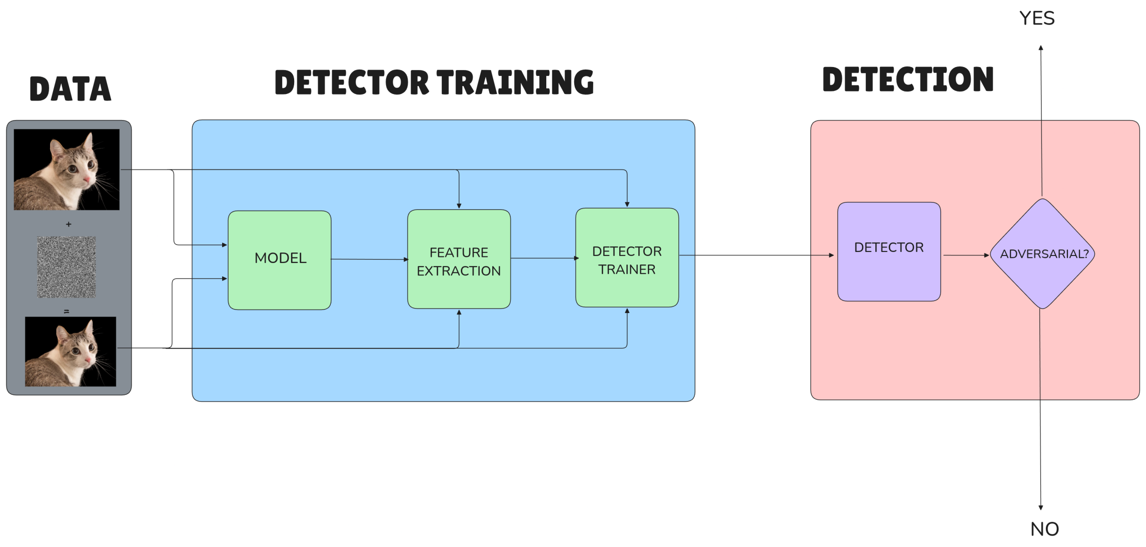

Figure 2: Diagram of how a detector works

versions of a sample are classified into the same category. On the other hand, detection methods focus solely on identifying whether an input is adversarial or not.

As highlighted in (33), most existing defenses still struggle to correctly classify adversarial examples. Because of this, some researchers have shifted focus towards building reliable detection techniques. Although these detection methods are not foolproof and can still be bypassed by carefully designed attacks (31), they can still serve as a valuable layer of protection. For instance, if there's a disagreement between the predictions of a baseline classifier and a robust classifier, detection mechanisms can help determine whether the input in question is actually adversarial.

In this section, we take a deep dive into adversarial example detection strategies. In general, these detectors are treated as third-party systems that act as filters-blocking adversarial inputs while allowing clean ones to reach the main deep learning model.

We classify detectors based on two main factors:

1. Whether or not they rely on prior knowledge of adversarial attacks.
2. The specific technique used to tell clean and adversarial inputs apart.

To evaluate how effective a detector is, we use the following criterion:

Detection Rate: This measures the accuracy of the detector. It's defined as the ratio of successfully detected adversarial examples to the total number of adversarial examples. A higher detection rate indicates a more effective detector.

4.1 Supervised Detection

In supervised detection, the defender uses adversarial examples generated by one or more attack algorithms to train a detector $\mathcal{D}$. The key assumption here is that adversarial examples have noticeable characteristics that differentiate them from clean inputs. Taking advantage of this, defenders aim to build a reliable detector $\mathcal{D}$ trained to recognize these differences. As a result, many strategies have been developed in the literature that fall under this supervised detection category (57).

4.1.1 Auxillary Model Approaches

The auxiliary model approach in supervised adversarial detection involves training an additional model (detector) alongside or separately from the main classifier. This auxiliary model learns to distinguish between clean and adversarial inputs using supervised labels (e.g., 0 for clean, 1 for adversarial), often based on extracted features like gradients, activations, or image statistics.

Model Uncertainty

One way defenders detect adversarial examples (AEs) is by measuring the model's uncertainty when classifying inputs. This is often done using *Dropout* during inference to introduce randomness (20; 33). The key idea is: for a clean input x, the model's prediction stays consistent even with dropout; for an AE $\tilde{x}$, predictions tend to fluctuate more (17).

This uncertainty is quantified and used as a feature for a binary classifier (detector) to distinguish between clean and adversarial inputs. Feinman et al. introduced *Bayesian Uncertainty (BU)*, leveraging *Monte Carlo Dropout* to estimate uncertainty near the data manifold (17). Formally, uncertainty is given by:

$$\mathbb{E}_{p(\theta|D)}[p(y|x,\theta)] \approx \frac{1}{T}\sum_{t=1}^{T} p(y|x,\theta_t)$$

where θ_t are sampled dropout masks. High variance in predictions suggests adversarial behavior.

Smith et al. alternatively proposed using *Mutual Information (MI)*:

$$\mathrm{MI}(y,\theta|x) = H\left(\mathbb{E}[p(y|x,\theta)]\right) - \mathbb{E}[H(p(y|x,\theta))]$$

which captures how uncertain the model is about its uncertainty, a useful signal for detecting AEs (58).

Raw AE-based Classifier

Several strategies have been proposed for detecting adversarial examples (AEs) (33). Gong et al. introduced a binary detector D, trained independently from the main classifier, to distinguish between clean and adversarial inputs (31). Another approach retrains the original classifier by adding a new output class specifically for adversarial inputs (57). Hosseini et al. applied adversarial training combined with *label smoothing*, where true labels are softened: for a K-class problem, the label vector becomes

$$y_i = \begin{cases} 1-\varepsilon & \text{for the true class} \\ \frac{\varepsilon}{K-1} & \text{for other classes} \end{cases}$$

A different method exploits parts of the input that the model typically ignores (33). By iteratively perturbing the input x (whether clean or adversarial), the model checks if the confidence of the predicted class $\max f(x)$ drops below a threshold τ. If so, x is flagged as adversarial:

$$\text{If } \max f(x) < \tau \Rightarrow x \text{ is adversarial}$$

NSS

Natural Scene Statistics (NSS) have long been useful in image processing, especially for assessing image quality, based on the insight that real (natural) images have different statistical patterns than altered or adversarial ones (33). Kherchouche et al. leveraged this idea to build a binary classifier. It uses features derived from the *Mean Subtracted Contrast Normalized* (MSCN) coefficients of images. These coefficients are modeled using two distributions: the *Generalized Gaussian Distribution* (GGD) and the *Asymmetric Generalized Gaussian Distribution* (AGGD) (31). For an image patch x, the MSCN coefficient is given by:

$$\hat{x}(i,j) = \frac{x(i,j)-\mu(i,j)}{\sigma(i,j)+\epsilon}$$

where $\mu(i,j)$ and $\sigma(i,j)$ are local mean and variance, and ϵ prevents division by zero. The model then learns to separate clean and PGD-generated adversarial examples based on the GGD/AGGD parameters extracted from $\hat{x}$ (57).

Gradient based Classifier

Lust et al. introduced a gradient-based adversarial detector called **GraN** (37). Their idea is to analyze how sensitive each layer of a neural network is to input changes. For a given input x (clean or adversarial), they apply smoothing and compute the gradient norm of the model's output (specifically, the predicted class score) with respect to the input at different layers:

$$g_l = \|\nabla_x f_l(x)\|$$

where $f_l(x)$ is the activation at layer l for input x. These gradient norms $\{g_l\}$ form the feature vector for training a binary classifier D, which distinguishes adversarial examples during inference.

Erase and Restore

Zuo et al. proposed an **Erase & Restore (E&R)** method for detecting adversarial examples using a binary classifier D (33). The idea is simple: for each input (clean or L_2-based adversarial), certain pixels are intentionally erased, and then restored using a deep generative model (31). This processed image is passed to D, which decides whether it's adversarial (57). If $D(x) = 1$, the input is flagged as an adversarial example. The baseline classifier remains unchanged and is used as the reference for prediction. Formally, let $\tilde{x}$ be the restored version of x:

$$\tilde{x} = \text{Restore}(\text{Erase}(x))$$

$$D(\tilde{x}) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

Softmax Based

Hendrycks et al. showed that softmax probabilities can help detect adversarial examples (26). They proposed adding a decoder to reconstruct the clean input from the softmax output and training it with the main classifier. A separate binary classifier, D, is then trained using the reconstructed input, logits, and confidence scores:

$$\hat{x} = \text{Decoder}(\text{Softmax}(x))$$

$$D(\hat{x}, \text{logits}) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

In a similar approach, Pertigkiozoglou et al. used confidence outputs from the classifier to calculate regularized vector features and retrained the classifier by adding these features to the last layer (51). The detector D flags an input as adversarial if there's no match between the baseline and retrained classifier. Another method by Ajgiran et al. trained a simple neural network detector D using logits from the baseline model to classify clean and adversarial inputs (1):

$$D(\text{logits}) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

Finally, Monteiro et al. proposed a bimodal mismatch detection approach, assuming different models make distinct errors on adversarial inputs. Their detector D is a binary SVM with radial basis function (RBF) kernels, which compares the outputs of two baseline classifiers (one for clean, one for adversarial) (43).

4.1.2 Network Invariant Approaches

The network invariant approach in supervised adversarial detection relies on features or behaviors of a neural network that remain stable for clean inputs but change noticeably for adversarial ones. These invariants, such as activation patterns, internal representations, or neuron relationships, are used to train a detector that flags deviations as adversarial.

SafetyNet

SafetyNet is based on the idea that adversarial attacks create distinct activation patterns in the late-stage ReLU layers compared to natural examples (36). To detect these differences, SafetyNet quantizes the activations from the final ReLU layer and trains a binary Support Vector Machine (SVM) with a radial basis function (RBF) kernel (36). Let a_L be the activation vector at the last ReLU layer, and $\tilde{a_L}$ its quantized version:

$$\tilde{a_L} = \text{Quantize}(a_L)$$

The SVM classifier D then uses these quantized activations to classify the input as either clean or adversarial:

$$D(\tilde{a_L}) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

Dynamic Adversary Training

Metzen et al. introduced dynamic adversary training to improve the robustness of detectors (42). In their method, the classifier is trained on adversarial examples (AEs), and the detector D is integrated into a pre-trained model at a specific layer. The detector takes the output from this layer for both clean and dynamically generated AEs as input to train a binary classifier. Let $h_l(x)$ be the output of layer l for input x:

$$h_l(x) = \text{Layer}_l(x)$$

The detector D is then trained to classify the inputs based on these outputs:

$$D(h_l(x)) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

Histogram Based

Pertigkiozoglou et al. (50) noticed that adversarial examples (AEs) cause certain peaks in the output of a classifier's first convolutional layer to increase, while the values at other points decrease. Based on this observation, they proposed a binary SVM classifier that uses the histogram of the first convolutional layer's output as input. Let $h_1(x)$ represent the output of the first convolutional layer for input x, and $H(h_1(x))$ its histogram:

$$H(h_1(x)) = \text{Histogram}(h_1(x))$$

The binary SVM classifier D is trained to distinguish between clean and adversarial inputs:

$$D(H(h_1(x))) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

RAID

Eniser et al. proposed **RAID**, a binary classifier that detects adversarial examples (AEs) by analyzing changes in neuron activations between clean and adversarial inputs (15). Let $a(x)$ and $a(x')$ be activation vectors for clean input x and its adversarial version x', then the detector uses their difference:

$$\Delta a = a(x) - a(x')$$

To resist adaptive attacks, they extended this idea to **Pooled-RAID**, which trains a pool of detectors $\{D_1, D_2, \ldots, D_k\}$, each using activations from randomly selected neurons (15). During inference, one detector is randomly chosen from the pool to make the decision:

$$D_{\text{rand}}(\Delta a) = \begin{cases} 1 & \text{if adversarial} \\ 0 & \text{if clean} \end{cases}$$

4.1.3 Statistical Approaches

The statistical approach in supervised adversarial detection uses statistical tests or distribution-based metrics (like MMD or histograms) to identify differences between clean and adversarial inputs. A detector is trained on these statistical features to classify inputs based on whether they align with the distribution of normal training data.

MMD

Grosse et al. (24) proposed using **Maximum Mean Discrepancy (MMD)**-a kernel-based, model-agnostic statistical test-to detect adversarial examples (AEs). The idea is to compare the distribution of clean inputs x and adversarial ones x'. First, compute the MMD score between the two:

$$a = \text{MMD}(x, x')$$

Then, randomly shuffle elements of x and x' to form two new sets y_1 and y_2, and compute:

$$b = \text{MMD}(y_1, y_2)$$

If $a < b$, we conclude that x and x' likely come from different distributions and the input is flagged as adversarial. However, subsequent work showed limitations in this approach due to kernel choice and test power (21).

LID

Ma et al. proposed using **Local Intrinsic Dimensionality (LID)** as an alternative to kernel density (KD) to detect adversarial examples (38). LID estimates how densely a point is surrounded by its neighbors, helping measure how much the input "fills" its local space. For an input x and distances r_i to its k nearest neighbors, LID is computed as:

$$\text{LID}(x) = -\left(\frac{1}{k}\sum_{i=1}^{k}\log\frac{r_i}{r_k}\right)^{-1}$$

Higher LID values often indicate adversarial inputs due to irregular local geometry (38).

Mahalnobis Approach

Lee et al. introduced a **Mahalanobis distance-based** confidence score to detect out-of-distribution (OOD) and adversarial examples as an alternative to KD and LID (34). Instead of relying on softmax, they use a generative classifier based on Gaussian Discriminant Analysis (GDA). For a sample x from class c, the Mahalanobis score is:

$$M(x) = (f(x) - \mu_c)^\top \Sigma^{-1} (f(x) - \mu_c)$$

where $f(x)$ is the feature vector from the network, μ_c is the class mean, and Σ is the shared covariance matrix. Larger $M(x)$ suggests the input is far from the class distribution and likely adversarial.

k-Nearest Neighbours

Early k-NN-based work measured how much each training sample influenced a validation input, identifying the most supportive samples for classification (29). At each layer, deep features $f_l(x)$ were extracted and a k-NN model was fitted to rank similar training points. These rankings from clean and adversarial samples were used to train a detector D:

$$\text{Support}(x) = \text{k-NN}(f_l(x))$$

Later, Mao et al. proposed **Neighbor Context Encoder (NCE)**, which uses a transformer to model the local feature space around an input using its k nearest neighbors (40). The transformer learns to classify whether the subspace indicates a clean or adversarial input.

Kernel Density

It was observed that adversarial examples (AEs) tend to lie in low-density regions of the feature space, especially when far from the true class manifold (**?**). To capture this, Kernel Density (KD) estimation was used to model the feature distribution of each class (**?**). For a test sample x, its density under class c is:

$$\text{KD}_c(x) = \frac{1}{n}\sum_{i=1}^{n}\exp\left(-\frac{\|f(x) - f(x_i^c)\|^2}{2\sigma^2}\right)$$

where $f(x)$ is the feature embedding, x_i^c are training samples from class c, and σ is a bandwidth parameter. A detector D is then trained on these densities along with uncertainty measures for clean, noisy, and adversarial inputs (**?**).

4.2 Unsupervised Detection

A major drawback of supervised detection is its dependency on prior knowledge of specific attack types, which limits its effectiveness against new or unknown attacks. Unsupervised detection addresses this by using only clean (non-adversarial) training data to design and train the detector $\mathcal{D}$.

This approach is also known as inconsistency-based detection, because it hinges on the idea that adversarial examples may not fool every neural network model the same way. The core goal of unsupervised methods is to reduce the exploitable input feature space available to adversaries. Despite the limitations, a wide range of unsupervised approaches have also been explored in the literature.

4.2.1 Object Based Detection

Object-based unsupervised detection compares object-level features (like shapes or textures) in the input to those of known classes. If the predicted class lacks sufficient feature overlap with expected patterns, the input is flagged as adversarial. It focuses on the semantics of objects, not just pixel-level cues.

UnMask

UnMask (18) is an object-based adversarial detector that compares low-level features of the input with features from training data of the predicted class. Suppose an adversarial image of a *bicycle* is misclassified as a *bird*. UnMask extracts object-level features F_{adv} from the input and compares them with features F_{cls} from true training examples of the predicted label:

$$\text{Overlap} = \frac{|F_{\text{adv}} \cap F_{\text{cls}}|}{|F_{\text{cls}}|}$$

If the overlap is too small, the detector D flags the input as adversarial. As a defense, UnMask also searches across all class features to find the class with the highest overlap and recovers the correct prediction.

4.2.2 Feature Squeezing

Feature squeezing in unsupervised detection reduces input complexity (e.g., by lowering bit-depth or applying smoothing) to eliminate adversarial noise. If the model's prediction changes significantly between the original and "squeezed" input, the sample is likely adversarial. It doesn't require labeled adversarial data to function.

Bit depth and Squeezing

Xu et al. proposed a defense based on **feature squeezing**, where the input is transformed to reduce adversarial noise (67). They applied: (1) color bit-depth reduction, (2) local smoothing (e.g., median filter), and (3) non-local smoothing (non-local means). The idea is that adversarial perturbations are sensitive to such transformations. A detector D flags an input x as adversarial if the prediction changes significantly after squeezing:

$$\|f(x) - f(x')\| > \tau$$

Here, x' is the squeezed version of x, $f(\cdot)$ is the model prediction, and τ is a threshold. Large prediction shift implies x is likely adversarial.

Adaptive Noise Reduction

Liang et al. proposed an **adaptive noise reduction** technique to detect adversarial examples(35**?**). The input image x is processed using scalar quantization and a spatial smoothing filter. The strength of noise reduction is guided by the image's entropy $H(x)$-higher entropy leads to stronger smoothing. After squeezing, if the model prediction changes, the detector D flags it as adversarial:

$$\text{if } \operatorname{argmax}(f(x)) \neq \operatorname{argmax}(f(x')) \Rightarrow x \text{ is adversarial}$$

Here, x' is the denoised version of x, and $f(\cdot)$ is the model output. Entropy-driven filtering helps remove perturbations without harming clean inputs(**?**).

4.2.3 Auxillary Approaches

The auxiliary approach in unsupervised detection uses additional models or networks to assist in identifying adversarial examples. It typically involves training an auxiliary model that is separate from the main classifier to detect discrepancies between the clean and potentially adversarial inputs, helping to flag adversarial examples based on features learned from the data.

k-NN Classifier

Carrara et al. (7) proposed a method where features from an intermediate layer of a deep model are used to build a k-nearest neighbor (k-NN) classifier. This k-NN model does **not** predict the final label, but instead scores how well the baseline classifier's prediction aligns with its neighborhood in feature space. If the score is low:

$$\text{if score}(x) < \tau \Rightarrow x \text{ is adversarial}$$

They also applied Principal Component Analysis (PCA) (49) to reduce the dimensionality of intermediate features before applying the k-NN scoring, making the method more efficient.

Reverse Cross Entropy

Pang et al. introduced a detection method using a modified training loss and density estimation (46). First, the classifier is retrained using a reverse cross-entropy loss to make its internal features more separable for clean vs adversarial inputs. Then, a kernel density (KD) is estimated per class (46). An input x is declared adversarial if its density falls below a threshold τ:

$$\text{if KD}(x) < \tau \Rightarrow x \text{ is adversarial}$$

They also experimented with non-maximal entropy (Non-ME) for density scoring, but KD-based detection generally performed better (46).

DNR

Sotgiu et al. proposed the Deep Neural Rejection (DNR) method, where the outputs of the last N layers of a baseline classifier are used to train N separate SVM classifiers with RBF kernels (60). The confidence probabilities from these classifiers are combined to form a final SVM-RBF classifier (60; 19). The detector $\mathcal{D}$ declares an input as adversarial if the maximum confidence score, $p_{\max}$, is below a predefined threshold τ (60):

$$\text{if } p_{\max} < \tau \Rightarrow x \text{ is adversarial}$$

Selective Detection

Aldahdooh et al. introduced the Selective and Feature-Based Adversarial Detection (SFAD) technique(2). It integrates three detection modules:

1. The selective detection module uses SelectiveNet's uncertainty to create a threshold-based detector for clean data(2).
2. The confidence detection module uses threshold-based softmax probabilities from SFAD's classifiers for clean data(2).
3. The feature extraction module analyzes the last N layers' outputs, applying autoencoding, up/down sampling, bottleneck, and noise blocks to capture robust features(2).
4. The ensemble prediction module compares predictions between the detector and the baseline classifier for mismatch detection(2).

$$\text{if mismatch score} > \tau \Rightarrow x \text{ is adversarial}$$

4.2.4 Statistical Approaches

The statistical approach in unsupervised detection involves analyzing the statistical properties of the input data, such as its distribution or density. By comparing these properties to those of clean, in-distribution data, the method identifies adversarial examples as outliers or anomalies based on their statistical differences.

PCA

Hendrycks et al. observed that the variance of later PCA components is higher for adversarial examples (AEs) than for clean inputs(27; 4). Based on this, they proposed a detector $\mathcal{D}$ that flags an input as adversarial if the variance of the later PCA components exceeds a predefined threshold τ:

$$\text{if variance}_{\text{PCA}} > \tau \Rightarrow x \text{ is adversarial}$$

Gaussian Mixing Models

Zheng et al. proposed a detection method called I-defender, which uses a Gaussian Mixture Model (GMM) to approximate the hidden state distribution of each class in the model(62; 45). I-defender focuses on the fully connected hidden layers and calculates a threshold for each class. The detector $\mathcal{D}$ flags an input as adversarial if its hidden state probability is below the threshold for the predicted class:

$$P(x \mid \text{predicted class}) < \tau$$

Additionally, I-defender assumes that adversarial inputs are either:

1. **Too Atypical**: Having low likelihood for the predicted class, and
2. **Too Typical**: Having high likelihood for a different class.

For both cases, the Kullback-Leibler (KL) divergence is used to compute the density score(12):

$$D_{\text{KL}}(P_{\text{input}}, P_{\text{model}}) > \tau$$

where τ is the threshold, and the detector flags the input as adversarial if the score exceeds this threshold(12).

4.2.5 Denoiser Approach

The denoiser approach in unsupervised detection involves using a model, like a denoiser or autoencoder, to reconstruct input samples. Adversarial examples are detected by evaluating the reconstruction error, with a higher error indicating a likely adversarial sample, as they typically fail to reconstruct well compared to clean samples.

PixelDefend

PixelDefend utilizes a generative model, PixelCNN, to detect adversarial examples(59). First, PixelDefend reconstructs clean data using PixelCNN and computes the prediction probabilities with the baseline classifier. It is observed that reconstructed images have higher prediction probabilities under the in-distribution of the training data(59). The detector $\mathcal{D}$ works as follows:

1. Compute the probability density of the test input P_{test}(59).
2. Rank this density against the densities of the training data, P_{train}(59).
3. Calculate the p-value for the rank as a test statistic to determine if the input belongs to the in-distribution or is adversarial(**?**).

$$p = \frac{\#(\text{samples ranked below test input})}{\text{total number of samples}}$$

The input is flagged as adversarial if the p-value is below a certain threshold(59).

Magnet

Magnet trains denoisers using clean training data to reconstruct input samples (41). It detects adversarial examples (AEs) in two ways:

1. **Reconstruction Error**: For clean images, the reconstruction error is small, while for AEs, it is large. The reconstruction error E is computed as:

$$E = \|x - \hat{x}\|_2$$

where x is the original input, and $\hat{x}$ is the denoised version. If E exceeds a predefined threshold, the input is flagged as adversarial (41).

2. **Prediction Distance**: This method calculates the distance between the predictions of the original input and its denoised version. The detector $\mathcal{D}$ announces the input as adversarial if this distance is greater than a threshold (41).

$$D(x, \hat{x}) = |f(x) - f(\hat{x})|$$

where $f(x)$ is the model's prediction for x, and $f(\hat{x})$ is the prediction for the denoised input.

5 Experiments on Detection Methods

5.1 Model Overview

The models employed in this study are CNN architectures trained on CIFAR-10 and MNIST datasets. The detailed architecture for each of these CNN models is provided in Appendix B for CIFAR-10 and Appendix A for MNIST.

5.2 Methodology

For each attack applied, a detection mechanism was implemented to assess the extent of the attack's effectiveness. We designed the detection method to focus on identifying adversarial perturbations introduced to the input data. The detection process involved evaluating the model's performance after applying each attack and comparing it with baseline (unperturbed) results.

Detection Evaluation

For each type of adversarial attack, we calculated the detection rate, defined as the percentage of attacks that were successfully detected by the model. The detection rate was computed as follows:

$$\text{Detection Rate} = \frac{\text{Number of detected attacks}}{\text{Total number of attacks}} \times 100 \tag{1}$$

Each attack type was evaluated separately, and the detection rate was recorded for every instance of detection. The results were then compared across different attack types and models to analyze the efficacy of the detection mechanism.

Comparison with Baseline

To further understand the effect of adversarial attacks on the models, we compared the detection performance against the baseline results. The baseline refers to the model's performance without any adversarial perturbations applied. This comparison allowed us to gauge the impact of the attacks and the effectiveness of the detection mechanism.

The results for each model, dataset, and attack type were compiled, and the detection rates were analyzed in detail to draw meaningful conclusions about the robustness of each CNN model against the selected adversarial attacks.

5.3 Results

- **Sample Selection**: A subset of 100 to 500 samples is systematically and randomly selected from the designated test set for evaluation.
- **Adversarial Generation**: For each selected sample, an adversarial counterpart is created by applying targeted perturbations.
- **Model Evaluation**: Both the original and adversarial samples are processed through the model to assess its performance, with a focus on measuring accuracy on adversarial samples.
- **Detector Training**: A dedicated adversarial detector is trained using a combination of adversarial and original samples to learn distinguishing features.

6 Conclusion

This survey systematically analyzed modern data poisoning detection strategies, classifying them into **supervised** and **unsupervised** frameworks with specialized subtypes like *statistical analysis*, *network-invariant defenses*, and *feature refinement*. Evaluations on **CIFAR-10** and **MNIST** under adversarial attacks (FGSM, PGD, DeepFool, etc.) revealed critical trade-offs between detection accuracy and adaptability.

6.1 Key Findings

Supervised Methods dominated simpler tasks, with notable trends:

Table 2: Results By Detection (by Detection Rate)

TYPE	SUBTYPE	DETECTION	ATTACK	CIFAR 10	MNIST
Supervised	Auxillary	Model Uncertainty	FGSM	87.50%	75.44%
		E&R	CW	93.75%	85.94%
		GRaN	FGSM	80.5%	71.0%
		NSS	CW	93.75%	85.94%
		Raw AE	BIM	35.00%	84.00%
		Softmax	BIM	88%	99.22%
Supervised	Network Invariant	SafetyNet	FGSM	93.75%	85.94%
		Histogram	BIM	92.0%	97.82%
		DAT	BIM	96.42%	100%
		RAID	DeepFool	94.06%	90.09%
Supervised	Statistical	Kernel Density	BIM	100%	93.22%
		k-NN	FGSM	100%	100%
		LID	FGSM	52.23%	97.55%
		MMD	FGSM	73.00%	74.00%
		Mahalnobis	FGSM	100%	100%
Unsupervised	Object Based	UnMask	PGD	69.76%	92.33%
Unsupervised	Auxillary	DNR	PGD	97%	99%
		SFAD	CW	100%	100%
		Reverse Entropy	BIM	88%	91%
		k-NN	FGSM	61.54%	35.24%
Unsupervised	Denoiser	Magnet	FGSM	100%	89.06%
		PixelDefend	FGSM	13.92%	54.55%
Unsupervised	Feature Squeezing	Adaptive Noise Reduction	FGSM	28.92%	90.09%
		Bit Depth	FGSM	93.75%	58.33%
Unsupervised	Statistical	PCA	BIM	94.32%	100%
		GMM	FGSM	62.5%	50%

- **Statistical detectors** (e.g., Mahalanobis distance) achieved flawless 100% accuracy against FGSM perturbations on both datasets, proving ideal for well-characterized data distributions.
- **Network-invariant approaches** like DAT maintained >90% efficacy on MNIST against complex attacks (BIM, DeepFool) but struggled on CIFAR-10, exposing limitations in high-dimensional spaces.

Unsupervised Techniques showed potential for label-free environments but faced inconsistencies:

- *SFAD* and *DNR* excelled (97–100%) against adaptive attacks (PGD, CW), highlighting their robustness.
- Methods like *PixelDefend* and *GMM* faltered (13.92%), emphasizing the need for attack-specific tuning.
- **Feature squeezing** exhibited dataset dependence: strong on MNIST (e.g., Adaptive Noise Reduction) but ineffective on CIFAR-10 due to color-channel complexity.

6.2 Dataset Dynamics

- **MNIST** facilitated higher detection rates, as its low variability simplified perturbation identification.
- **CIFAR-10**'s intricate features and chromatic diversity reduced method reliability, stressing the need for scalable, multimodal defenses.

6.3 Future Directions

No universal solution emerged; instead, **hybrid systems** integrating statistical rigor, invariance learning, and real-time adaptability are critical. Challenges persist in mitigating adaptive attacks and ensuring cross-dataset generalization. Priorities for future work include:

- Combining two or three detection methods together
- Test out all the methods with every attacks [was not possible due to time-constraint]

This study underscores the urgency of advancing detection paradigms alongside attack sophistication, ensuring AI systems remain secure in decentralized and resource-constrained environments.

References

[1] Ali Ajgiran, Arash Rahnama, and Fatemeh Afghah. 2021. Adversarial example detection using neural network-based detector. *IEEE Access*, 9:14255–14265.

[2] A. Aldahdooh, W. Hamidouche, and O. Deforges. Revisiting model's uncertainty and confidences for adversarial example detection. *Applied Intelligence*, 2022. https://doi.org/10.1007/s10489-022-03373-y

[3] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. 2020. Square Attack: a query-efficient black-box adversarial attack via random search. *European Conference on Computer Vision (ECCV)*, pages 484–501.

[4] Nicholas Carlini and David Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 2017.

[5] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 387–402.

[6] Ying Cao and Jun Luan. 2024. A novel differential evolution algorithm with multi-population and elites regeneration. *PLOS ONE*, 19(4):e0302207.

[7] F. Carrara, F. Falchi, R. Caldelli, R. Becarelli, G. Amato Adversarial Image Detection in Deep Neural Networks In *Proceedings of the ACM Workshop on Information Hiding and Multimedia Security*, 2017, pp. 1–8.

[8] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57.

[9] Edward S. Chan. 2020. Chapter 3 Adversarial Attack. *Purdue Engineering.*

[10] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26.

[11] Francesco Croce and Matthias Hein. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 2206–2216.

[12] Cui, J., et al. Generalized Kullback-Leibler Divergence Loss. arXiv preprint arXiv:2503.08038, 2022.

[13] DeepAI. 2019. Defensive Distillation Definition. *DeepAI Glossary.*

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

[15] Hasan Ferit Eniser, Maria Christakis, Valentin Wüstholz. *RAID: Randomized Adversarial-Input Detection for Neural Networks.* arXiv preprint arXiv:2002.02776, 2020. URL: https://arxiv.org/abs/2002.02776

[16] Logan Engström, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. 2019. Exploring the Landscape of Spatial Robustness. *International Conference on Machine Learning*, pp. 1802–1811.

[17] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410.*

[18] Scott Freitas, Shang-Tse Chen, Zijie J. Wang, and Duen Horng Chau. *UnMask: Adversarial Detection and Defense Through Robust Feature Alignment.* IEEE Access, 8:125344–125353, 2020. DOI: https://doi.org/10.1109/ACCESS.2020.3007253.

[19] G. Fumera, A. Sotgiu, M. Melis, A. Demontis, and F. Roli. Detection of Face Recognition Adversarial Attacks. *Computer Vision and Image Understanding*, 2020. https://iris.cnr.it/bitstream/20.500.14243/383446/3/CVIU20_SI___Detection_of_Face_Recognition_Adversarial_Attacks.pdf

[20] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning*, pp. 1050–1059.

[21] Ruize Gao, Feng Liu, Jingfeng Zhang, Bo Han, Tongliang Liu, Gang Niu, Masashi Sugiyama. *Maximum Mean Discrepancy Test is Aware of Adversarial Attacks.* Proceedings of the 38th International Conference on Machine Learning (ICML), 2021.

[22] Maksym Andriushchenko et al. 2019. Square Attack: a query-efficient black-box adversarial attack via random search. *GitHub repository*, https://github.com/max-andr/square-attack.

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572.*

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

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

[26] Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *International Conference on Learning Representations.*

[27] Dan Hendrycks and Kevin Gimpel. Early Methods for Detecting Adversarial Images. arXiv preprint arXiv:1608.00530, 2017.

[28] Andrew G. Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861.*

[29] Kipf, T. N., Welling, M. Semi-Supervised Classification with Graph Convolutional Networks *International Conference on Learning Representations (ICLR)*, 2017.

[30] Shashank Kotyan and Danilo Vasconcellos Vargas. 2022. Adversarial robustness assessment: Why in evaluation both L0 and L1 attacks are necessary. *PLOS ONE*, 17(3):e0265723.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105.

[32] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533.*

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature*, 521(7553):436–444.

[34] Kimin Lee, Kibok Lee, Honglak Lee, Jinwoo Shin. *A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks.* Advances in Neural Information Processing Systems (NeurIPS), 2018.

[35] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting Adversarial Image Examples in Deep Networks with Adaptive Noise Reduction. arXiv preprint arXiv:1705.08378, 2017.

[36] J. Lu, T. Issaranon, D. Forsyth, SafetyNet: Detecting Adversarial Examples that Silence Neural Networks, *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[37] Kilian Lust, Jonas Rauber, Matthias Bethge, Wieland Brendel. 2020. GraN: Gradient Norm as a Defense Against Adversarial Attacks. *arXiv preprint arXiv:2002.10439.*

[38] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, et al. *Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality.* In International Conference on Learning Representations (ICLR), 2018.

[39] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083.*

[40] Mao, C., Chen, L., Zhong, Z., et al. Adversarial Detection via Local Neighborhood Analysis *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[41] Meng, D., Chen, H. (2017). Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 135-147).

[42] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *International Conference on Learning Representations (ICLR).*

[43] João Monteiro, Pedro Morgado, and Nuno Vasconcelos. 2020. Bimodal mismatch detection for robust deep learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13174–13183.

[44] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. DeepFool: a simple and accurate method to fool deep neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582.

[45] Zhang, Pengfei; Ju, Xiaoming. Research on Adversarial Sample Detection Method Based on Image Hidden Feature Distribution. Journal of Internet Technology, 2020.

[46] Pang, T., Xu, K., Du, C., Chen, N., Zhu, J. (2018). Improving Adversarial Robustness via Promoting Ensemble Diversity. In Proceedings of the 36th International Conference on Machine Learning (ICML).

[47] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. 2016. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814.*

[48] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2017. Practical black-box attacks against machine learning. *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 506–519.

[49] K. Pearson LIII. On Lines and Planes of Closest Fit to Systems of Points in Space *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901, 2(11):559–572.

[50] Stefanos Pertigkiozoglou and Petros Maragos. Detecting adversarial examples in convolutional neural networks. *arXiv preprint arXiv:1812.03303*, 2018.

[51] Ioannis Pertigkiozoglou, Christos Tjortjis, and Panagiotis Sarigiannidis. 2022. Detecting adversarial examples using classifier confidence and regularized feature vectors. *Expert Systems with Applications*, 187:115892.

[52] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

[53] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 779–788.

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28:91–99.

[55] Sergio Luján Mora. 2024. An Analysis with FGSM, PGD and CW. *Big Data Cogn. Comput.*, 8, 8.

[56] F. Sheikholeslami, S. Jain, and G. B. Giannakis. Minimum Uncertainty Based Detection of Adversaries in Deep Neural Networks. *arXiv preprint arXiv:1904.02841*, 2019.

[57] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

[58] Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533.*

[59] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, Nate Kushman. PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples. arXiv preprint arXiv:1710.10766, 2017.

[60] A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security*, 2020. https://arxiv.org/abs/1910.00470

[61] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841.

[62] Sun, Y., Zheng, Y., et al. Robust Detection of Adversarial Attacks by Modeling the Intrinsic Properties of Deep Neural Networks. NeurIPS 2018.

[63] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199.*

[64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

[65] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. 2018. Ensemble adversarial training: Attacks and defenses. *International Conference on Learning Representations (ICLR).*

[66] Wikipedia. 2025. Adversarial machine learning: Square Attack. *Wikipedia*,

[67] Xu, W., Evans, D., & Qi, Y. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[68] Zhilin Yang et al. 2019. XLNet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32.s

Appendix

A MNIST Classifier Architecture

Table 3: Layer-wise architecture of the MNIST CNN model.

Layer	Operation	Output Shape
Input	1×28×28	-
Conv1	32 filters, 3×3, ReLU	32×28×28
Conv2	32 filters, 3×3, ReLU, MaxPool(2×2)	32×14×14
Conv3	64 filters, 3×3, ReLU	64×14×14
Conv4	64 filters, 3×3, ReLU, MaxPool(2×2)	64×7×7
FC1	Linear (3136→256), ReLU, Dropout(0.3)	256
FC2	Linear (256→256), ReLU	256
FC3	Linear (256→10)	10

B CIFAR-10 Classifier Architecture

Table 4: Layer-wise architecture of the CIFAR-10 CNN model.

Layer	Operation	Output Shape
Input	3×32×32	-
Conv1	64 filters, 3×3, BN, ReLU	64×32×32
Conv2	64 filters, 3×3, BN, ReLU, MaxPool, Dropout(0.1)	64×16×16
Conv3	128 filters, 3×3, BN, ReLU	128×16×16
Conv4	128 filters, 3×3, BN, ReLU, MaxPool, Dropout(0.2)	128×8×8
Conv5	256 filters, 3×3, BN, ReLU	256×8×8
Conv6	256 filters, 3×3, BN, ReLU, MaxPool, Dropout(0.3)	256×4×4
Conv7	512 filters, 3×3, BN, ReLU, MaxPool, Dropout(0.4)	512×2×2
FC1	Linear (2048→512), ReLU	512
FC2	Linear (512→10)	10