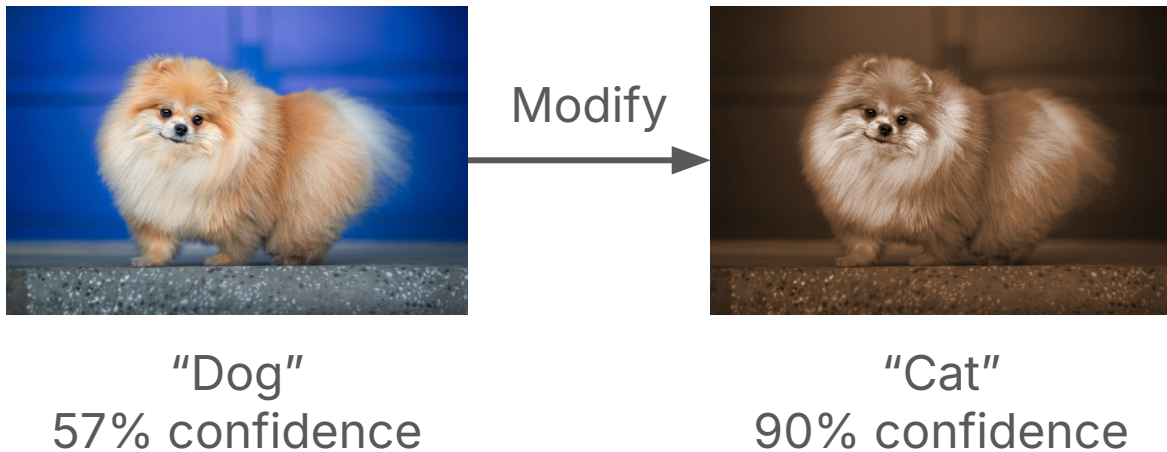


Safety - Adversarial Robustness

Group 3

What Are Adversarial Samples?

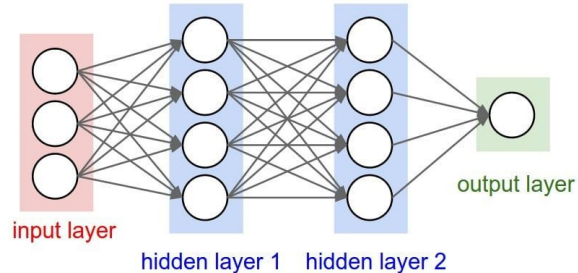
- **Definition:** Inputs modified to trick AI models into errors.
- **Example:** Slightly altered images misclassified by DNNs.
- **Impact:** Security risks (e.g., autonomous vehicles, malware detection).



Defensive Distillation: Protecting DNNs from Adversarial Attacks

Why Are DNNs Vulnerable?

- **High Sensitivity:** Small input changes cause large output shifts.
- **Attack Methods:**
 - Fast Gradient Sign Method (FGSM). [Perturbs inputs along gradient direction.]
 - Jacobian-based Saliency Map Attack. [Exploit model's input-output sensitivity.]
- **Goal:** Minimize perturbations to evade detection.



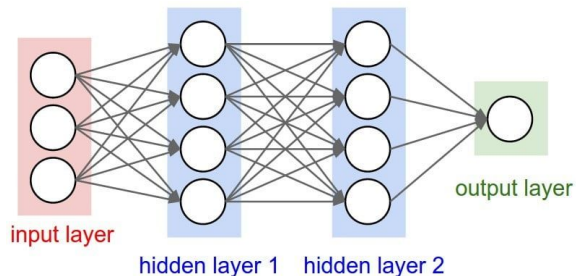
Solution: Defensive Distillation

- **Inspiration:** Knowledge transfer between DNNs.
- **Process:**
 - Train initial model with high-temperature softmax.
 - Use soft labels (probabilities) to retrain the same model.
- **Result:** Smoother decision boundaries, reduced gradients.

$$F(X)_i = \frac{e^{z_i(X)/T}}{\sum_j e^{z_j(X)/T}}$$

How Distillation Reduces Sensitivity

- **Key Idea:** High temperature (T) in softmax smooths outputs.
- **Effect:**
 - Gradients used in attacks reduced by 10^{30}
 - Perturbations require 8x more features to succeed.
- **Result:** Smoother decision boundaries, reduced gradients.



Datasets



MNIST

airplane

automobile

bird

cat

deer

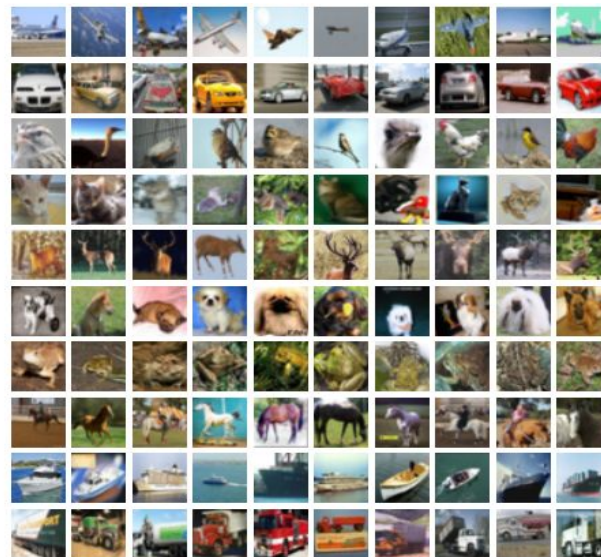
dog

frog

horse

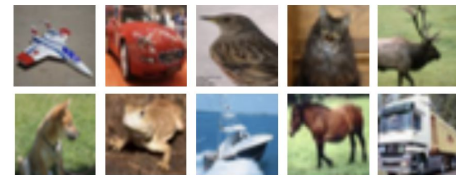
ship

truck

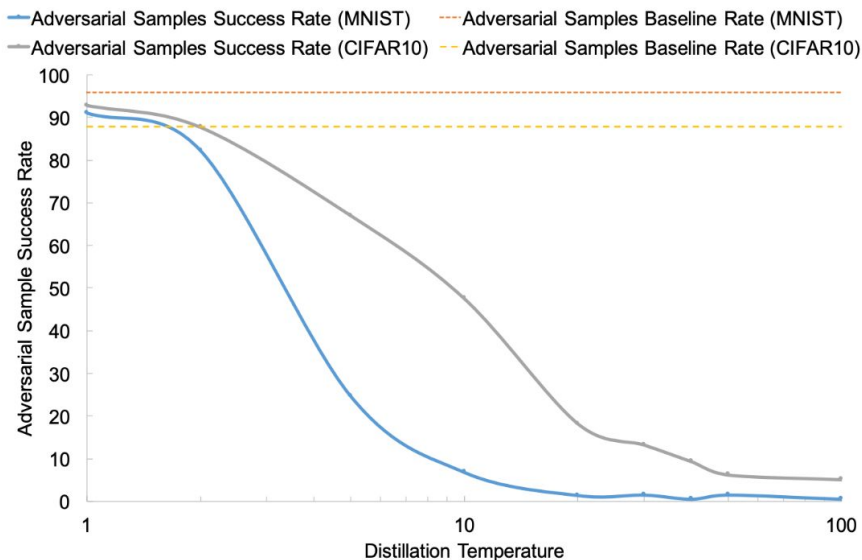


CIFAR10

Experimental Results



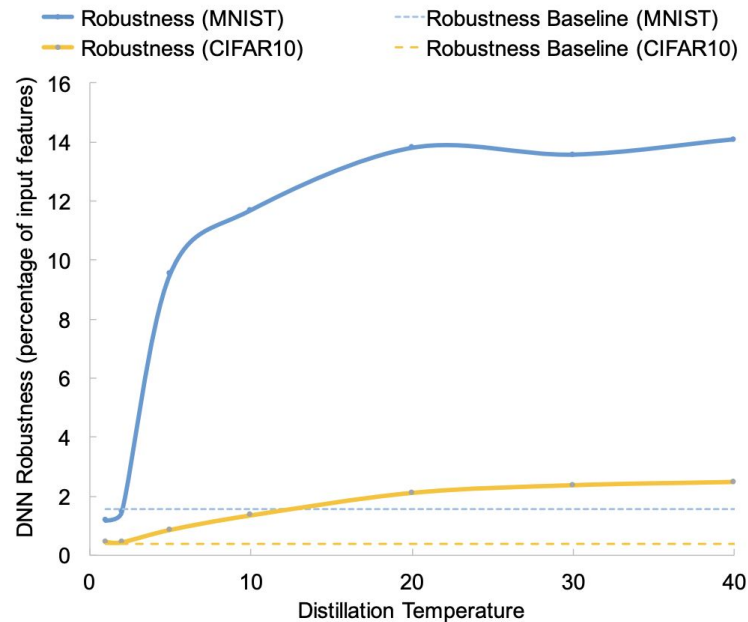
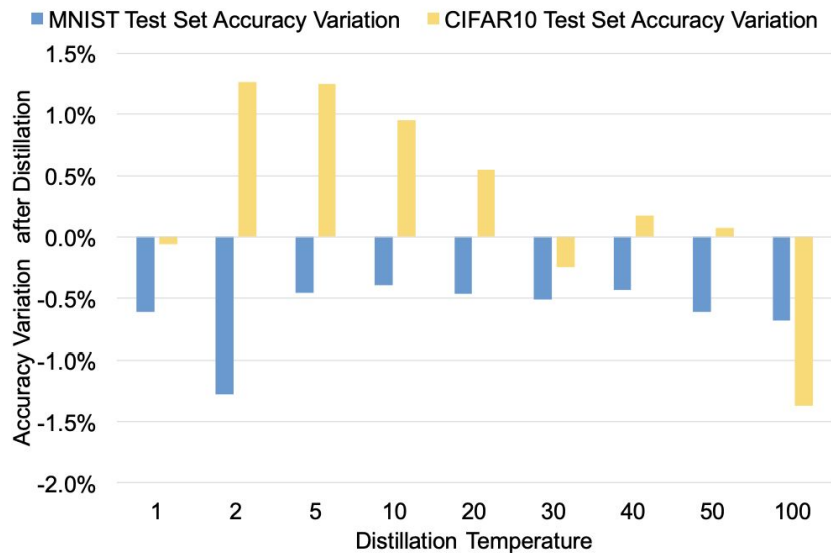
- Attack success rate drops down quickly with distillation temperature



Distillation Temperature	MNIST Adversarial Samples Success Rate (%)	CIFAR10 Adversarial Samples Success Rate (%)
1	91	92.78
2	82.23	87.67
5	24.67	67
10	6.78	47.56
20	1.34	18.23
30	1.44	13.23
40	0.45	9.34
50	1.45	6.23
100	0.45	5.11
No distillation	95.89	87.89

Experimental Results

- Influence of distillation on accuracy and robustness



Why Does Temperature Matter?

- **Higher T:**
 - Smoother probability distributions.
 - Low T (T=1): Spiky (e.g., [0.01, 0.93, 0.06]).
 - High T (T=20): Smooth (e.g., [0.3, 0.4, 0.3]).
 - Hiding sensitive gradients, thus lowering adversarial gradients.
- **Trade-off:** Very high T may slightly reduce accuracy ($\leq 1.5\%$).

Practical Implications

- **Security:** Harder for adversaries to craft stealthy attacks.
- **Usability:** Minimal impact on model accuracy.
- **Applications:** Critical systems (healthcare, finance, IoT).



Healthcare



Autonomous Vehicles



Finance

Conclusion

- **Key Takeaways:**
 - Defensive distillation drastically reduces attack success rates.
 - Increases robustness by requiring larger perturbations.
 - Easy to implement with existing DNN architectures.
- **Future Work:** Extend to non-DNN models, real-world testing.

Discussion

- Does distillation work for all attack types?
- How to choose the best temperature?

Provable Defenses via the Convex Outer Adversarial Polytope

Existing Adversarial Defenses & Limitations

Most existing defenses fail against stronger adversarial attacks due to computational inefficiency or lack of provable guarantees.

- SMT & Integer Programming: Exact optimization methods to verify robustness. Not scalable for large networks.
- Regularization-Based Defenses: Reduces adversarial vulnerability by controlling model sensitivity via weight constraints. Provides no provable robustness guarantee

Convex Outer Approximation

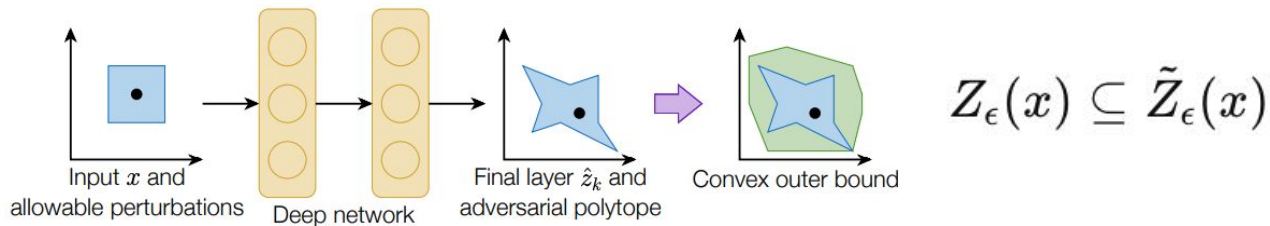


Figure 1. Conceptual illustration of the (non-convex) adversarial polytope, and an outer convex bound.

Why is the adversarial polytope non-convex?

- Neural networks, particularly with ReLU activations, introduce complex decision boundaries, making adversarial regions highly non-convex.
- This complexity makes robustness verification difficult.

How does convex approximation help?

- Instead of working with a complex, non-convex region, we over-approximate it with a convex shape.
- This allows us to solve adversarial robustness verification as a linear program (LP).

Efficient Optimization via Dual Networks

Instead of solving the original optimization problem directly, we transform it into a **dual problem** that can be efficiently solved using a neural network structure.

Primal Problem	$\max_{\ \Delta\ _\infty \leq \epsilon} L(f_\theta(x + \Delta), y)$
Dual Formulation	$J_\epsilon(x, g_\theta(\mathbf{e}_y \mathbf{1}^T - I))$

- Reformulate the problem in dual form, where it can be optimized efficiently using gradient-based methods.
- Reformulate the problem in dual form, where it can be optimized efficiently using gradient-based methods.

Computing Activation Bounds

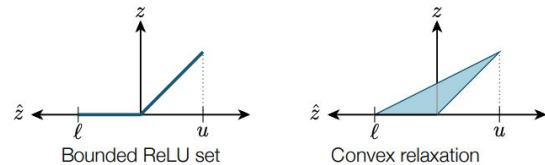


Figure 2. Illustration of the convex ReLU relaxation over the bounded set $[\ell, u]$.

ReLU function is non-convex

→ Relax it using convex approximation

Original ReLU Activation	$z = \max(0, \hat{z})$
Convex Relaxation Bounds	$z \geq 0, \quad z \geq \hat{z}, \quad -u\hat{z} + (u - \ell)z \leq -u\ell$

- Bounding the activation range $[\ell, u]$ enables efficient robustness verification.
- Convex relaxation transforms ReLU into a tractable linear constraint, ensuring provable adversarial robustness.

Why does tight bounding matter?

- Looser bounds allow greater adversarial freedom, making attacks easier.
- Tighter bounds ensure robust classification, reducing adversarial vulnerability.

Efficient Robust Optimization

- Traditional deep networks minimize the loss at given data points. Robust optimization instead minimizes the worst-case loss over an ϵ -ball of perturbations.
- The goal is to ensure the classifier remains robust under adversarial attacks.

$$\min_{\theta} \sum_{i=1}^N \max_{\|\Delta\|_{\infty} \leq \epsilon} L(f_{\theta}(x_i + \Delta), y_i)$$

- Outer maximization seeks the worst perturbation.
- Inner minimization trains the network to be robust against such perturbations.

Adversarial Guarantees

- Corollary 1: Robust Error Bound (Certification of Robustness) $J_\epsilon(x, g_\theta(e_{y^*}1^T - I)) \geq 0$
 - If this inequality holds, no perturbation $\|\tilde{x} - x\|_\infty \leq \epsilon$ can make the model misclassify x
 - The robust error measures how many samples do not satisfy this guarantee, meaning they could still be attacked
- Corollary 2: Detecting Adversarial Examples $J_\epsilon(x, g_\theta(e_{\hat{y}}1^T - I)) \geq 0$
 - This method allows us to detect adversarial examples at test time, preventing attacks before they reach deployment.
 - Unlike traditional defenses, this provides a formal guarantee—every real attack will be detected.

Experimental Results

Table 1. Error rates for various problems and attacks, and our robust bound for baseline and robust models.

PROBLEM	ROBUST	ϵ	TEST ERROR	FGSM ERROR	PGD ERROR	ROBUST ERROR BOUND
MNIST	×	0.1	1.07%	50.01%	81.68%	100%
MNIST	✓	0.1	1.80%	3.93%	4.11%	5.82%
FASHION-MNIST	×	0.1	9.36%	77.98%	81.85%	100%
FASHION-MNIST	✓	0.1	21.73%	31.25%	31.63%	34.53%
HAR	×	0.05	4.95%	60.57%	63.82%	81.56%
HAR	✓	0.05	7.80%	21.49%	21.52%	21.90%
SVHN	×	0.01	16.01%	62.21%	83.43%	100%
SVHN	✓	0.01	20.38%	33.28%	33.74%	40.67%

- Robust models significantly reduce adversarial vulnerability.
- Certified robust error bounds accurately reflect adversarial robustness.
- There is a trade-off: slightly higher test error, but major gains in adversarial defense.

Scaling Provable Adversarial Defenses



Key Definitions and Current Methods

- In recent years, it has become possible to create deep learning models that come with formal guarantees of robustness against adversarial examples backed by proofs that ensure they will remain robust **under certain conditions**

Early Methods

- Provable guarantees have only been possible for reasonably small sized networks
- Only worked for *simple feedforward networks* with linear layers followed by activations like ReLU

Progress has been made toward tackling the challenge of scaling provable defenses to deeper, more complex architectures

Extending Provable Robustness for General Networks

A k-layer neural network is represented as:

$$z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \text{ for } i = 2, \dots, k$$

Adversarial Problem Formulation: goal is to make the network robust to perturbations

Perturbation Set: An adversary tries to slightly alter input (x). Perturbations are constrained within a small ϵ -bounded set:

$$B(x) = \{x + \Delta : \|\Delta\| \leq \epsilon\}$$

The adversary aims to minimize the model's confidence by finding the worst case perturbation Δ that maximizes the loss:

$$\underset{z_k}{\text{minimize}} \quad c^T z_k, \text{ subject to } z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \text{ for } i = 2, \dots, k, \quad z_1 \in B(x)$$

Leveraging Fenchel Duality

- Uses Fenchel duality to derive provably robust bounds as a dual optimization task

$$f^*(y) = \sup_x (x^T y - f(x))$$

- Each network operation has its own dual formation
- By applying Fenchel Duality to each operation, the dual of the entire network can be constructed by combining these modular components
- Reduces the problem of bounding adversarial loss to analyzing the dual form of individual layers

The adversarial problem is lower-bounded as:

$$J(x, \nu_{1:k}) = -\nu_1^T x - \epsilon \|\nu_1\|^* - \sum_{i=1}^{k-1} h_i(\nu_{i:k})$$

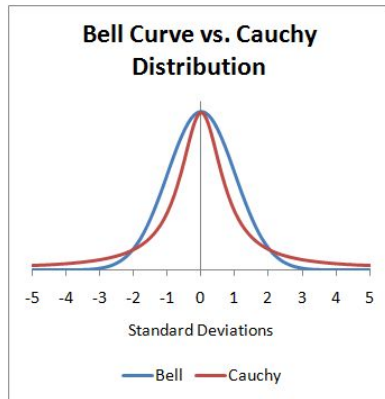
Efficient Bound Computation

Earlier methods computed the upper bound by calculating the contributions from every hidden unit, resulting in quadratic complexity. For ReLU networks with l_∞ bounded perturbations, the computation of the l_1 norm requires individual unit processing which is costly

Solution:

- Introducing nonlinear random projections to approximate these l_1 terms, reducing complexity from quadratic to linear
- The upper bound on robust loss is estimated efficiently, scaling linearly with the number of hidden units, making robust training feasible for larger models

Cauchy Random Projection



- Cauchy random projections are heavy-tailed making them ideal for approximating l_1 norms using random projections

Estimating the l_1 Norm:

- Let ν_1 represent the dual networks first layer, and R be a Cauchy random matrix
- L_1 norm can be approximated as:

$$\|\nu_1\|_1 \approx \text{median}(|\nu_1^T R|)$$

- For terms involving ReLU activations and perturbations:

$$\sum_{j \in I} \ell_{ij} [\nu_{ij}]^+ \approx \frac{1}{2} (-\text{median}(|\nu_i^T \text{diag}(d_i) R|) + \nu_i^T d_i)$$

- Random projections eliminate the need to explicitly pass each hidden unit through the dual network, saving computation time while maintaining the robustness bounds

Cascading Models

Problem: Robust training can over-regularize networks, reducing their accuracy on clean inputs

Robust Cascade: instead of training a single robust classifier, the cascade trains multiple classifiers sequentially

- Each classifier in the sequence focuses on examples that the previous classifiers failed to certify as robust
 1. Train the first model on the full dataset (f_1)
 2. Filter Certified Examples
 3. Train the next model (f_2 only on the uncertified examples of f_1)
 4. Continue this process training more models refining robust predictions

Why Cascades Work

- Earlier models in the cascade handle easy to certify examples, leaving more challenging examples for later models to specialize in
- Reduces the burden on each individual model
- Reduces verified robust error significantly compared to single robust classifier
- Allows the use of smaller more efficient models without sacrificing robustness



Experimentation Results

Dataset	Model	Epsilon	Single model error		Cascade error	
			Robust	Standard	Robust	Standard
MNIST	Small, Exact	0.1	4.48%	1.26%	-	-
MNIST	Small	0.1	4.99%	1.37%	3.13%	3.13%
MNIST	Large	0.1	3.67%	1.08%	3.42%	3.18%
MNIST	Small	0.3	43.10%	14.87%	33.64%	33.64%
MNIST	Large	0.3	45.66%	12.61%	41.62%	35.24%
CIFAR10	Small	2/255	52.75%	38.91%	39.35%	39.35%
CIFAR10	Large	2/255	46.59%	31.28%	38.84%	36.08%
CIFAR10	Resnet	2/255	46.11%	31.72%	36.41%	35.93%
CIFAR10	Small	8/255	79.25%	72.24%	71.71%	71.71%
CIFAR10	Large	8/255	83.43%	80.56	79.24%	79.14%
CIFAR10	Resnet	8/255	78.22%	71.33%	70.95%	70.77%

Standard - error rate on the original, unperturbed dataset (clean inputs)

- For MNIST: Model achieves error of 3.7% and on best cascade 3.13% (previous 5.8%)
- For CFAR: Model achieves best error of 46.1% and 36.4% (previous 80%)

Results Cont.

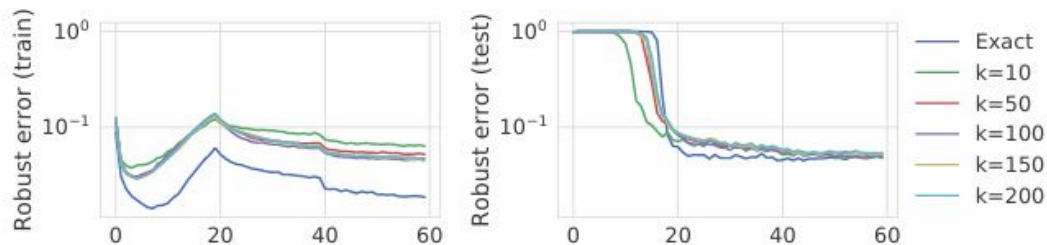


Figure 2: Training and testing robust error curves over epochs on the MNIST dataset using k projection dimensions. The ϵ value for training is scheduled from 0.01 to 0.1 over the first 20 epochs. The projections force the model to generalize over higher variance, reducing the generalization gap.

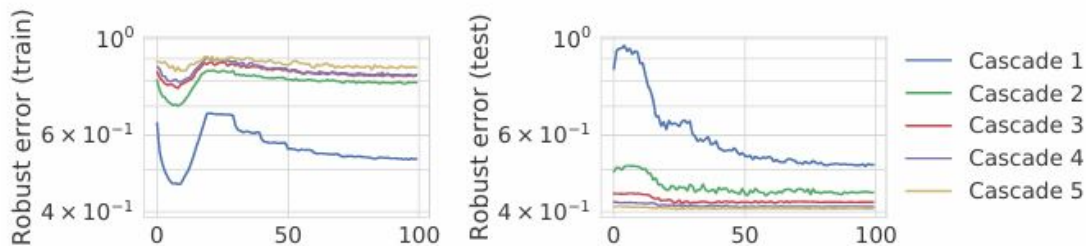


Figure 3: Robust error curves as we add models to the cascade for the CIFAR10 dataset on a small model. The ϵ value for training is scheduled to reach $2/255$ after 20 epochs. The training curves are for each individual model, and the testing curves are for the whole cascade up to the stage.

Discussion

- **How do cascade models help reduce bias in robust training? Can you think of any other technique that could complement cascade models to achieve better results?**
- **While cascade models improve robustness, they also increase non-robust error. How would you assess whether this trade-off is acceptable for different applications?**
- **What steps can be taken to ensure fairness and prevent bias when enhancing adversarial robustness in deep learning models as robustness improvements may not be uniformly effective across different data subgroups?**

Key Improvements

Extending Robust Training to Larger Networks:

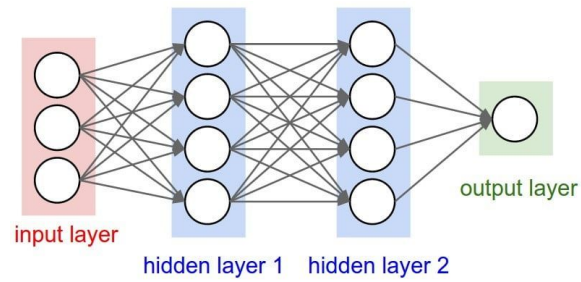
- Generalizes the approach to more complex architectures including those with skip connections and non linear movement
- Utilizes a modular technique to apply methods automatically to any network structure

Improved Computational Efficiency

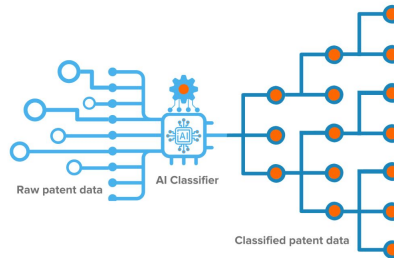
- Reduces the complexity to linear, making robust training much more scalable

Cascade Models

- Introduces cascade models to improve robustness which involves training multiple stages of classifiers where each stage handles the examples the previous stage could not classify robustly.



TRAINING DEEP LEARNING MODELS FOR ADVERSARIAL RESISTANCE



How Can We Train Deep Neural Networks that are Robust to Adversarial Inputs?

- Natural saddle point (min-max) formulation

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- **Inner Maximization Problem**

- Aims to find an adversarial version of a given data point x that achieves a high loss

- **Outer Minimization Problem**

- Aims to find model parameters so the “adversarial loss” given by the inner attack problem is minimized

- Allows for resistance against a broad class of attacks rather than defending against only specifically known attacks

Goals

- How can we produce strong adversarial examples, i.e., adversarial examples that fool a model with high confidence while requiring only a small perturbation?
- How can we train a model so that there are no adversarial examples, or at least so that an adversary cannot find them easily?
- Goal: Attaining small adversarial loss gives a guarantee that no allowed attack will fool the network

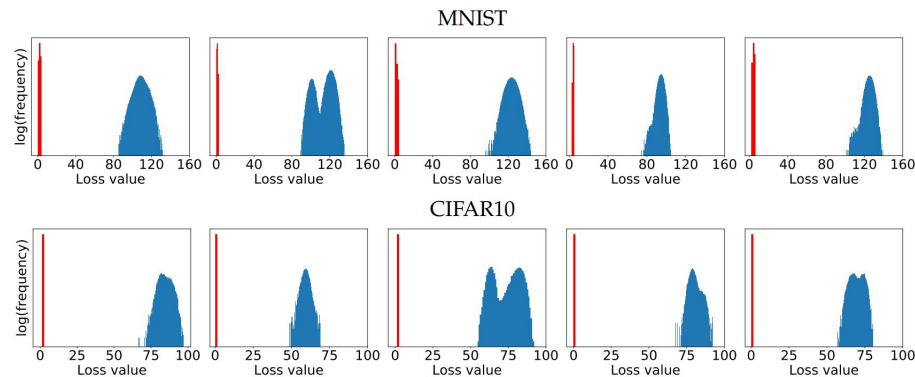
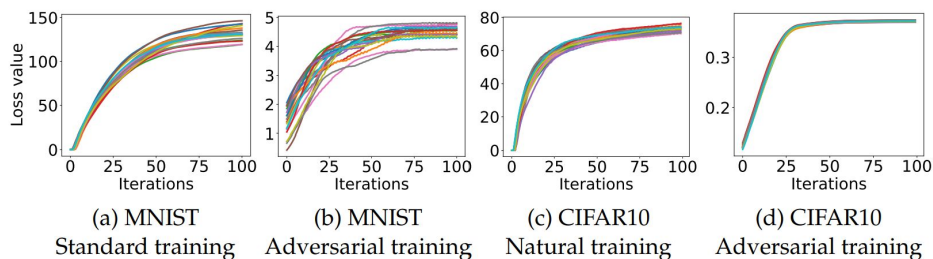
Inner Maximization Problem

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

- Corresponds to finding an adversarial example for a given network and data point
- Goal: To produce adversarial examples that fool a model with high confidence while only requiring a small perturbation
- Requires us to maximize a highly non-concave function
 - Found using Projected Gradient Descent (PGD) which produces adversarial examples

Finding Adversarial Examples

- Final loss values on adversarially trained networks are significantly smaller than their standard counterparts
- Local maxima found by PGD all have similar loss values



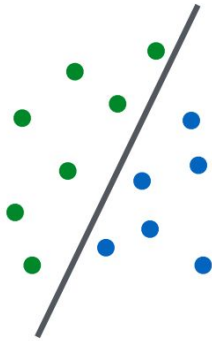
Outer Minimization Problem

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$

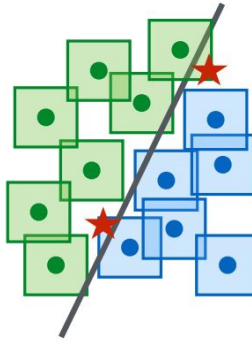
- Goal: To find model parameters that minimize the “adversarial loss”, the value of the inner maximization problem
- Method for minimizing the loss function: Stochastic Gradient Descent (SGD)
 - Replace the input points by their corresponding adversarial perturbations and train the network on the perturbed input
 - Apply SGD to reduce the loss of the saddle point problem during training

Crucial Component for Adversarial Robustness: Capacity

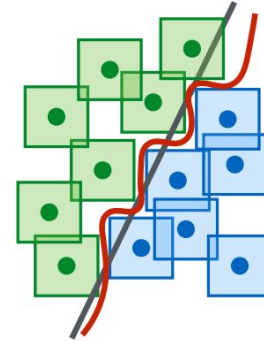
- Increasing the capacity of the network when training increases robustness
- Small capacity networks sacrifice performance on natural examples to provide any kind of robustness against adversarial inputs
 - Network converges to always predicting a fixed class



Linear Decision Boundary



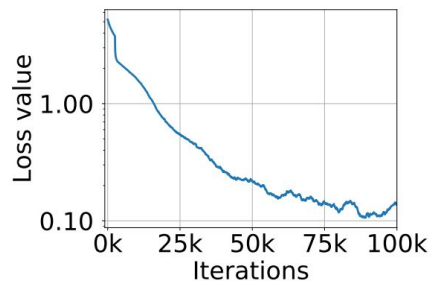
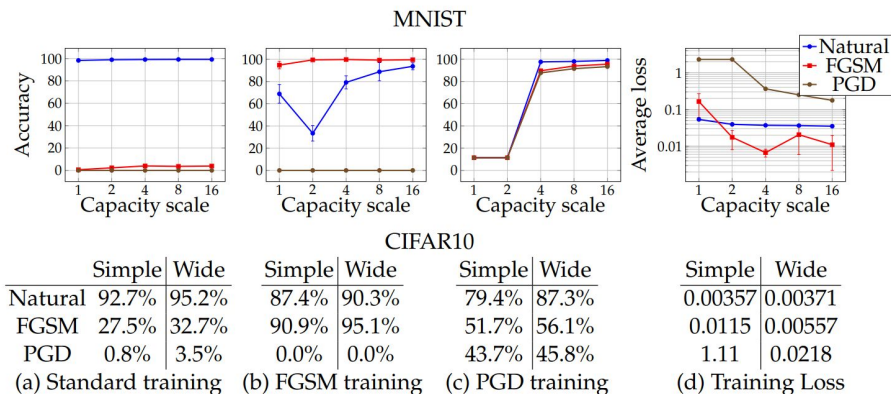
Red stars represent misclassified adversarial examples



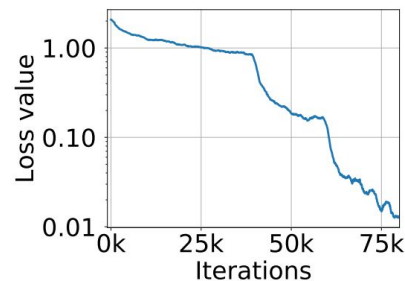
Robust to Adversarial Examples

Experiments

- Two Key Elements
 - Train a sufficiently high capacity network
 - Use the strongest possible adversary
 - Adversary of Choice: Projected Gradient Descendant (PGD)
- Performance increased when given a higher-capacity network



(a) MNIST



(b) CIFAR10

Model Performance Against Different Adversaries

- The MNIST dataset had higher accuracy compared to the CIFAR10 dataset

Method	Steps	Restarts	Source	Accuracy
Natural	-	-	-	98.8%
FGSM	-	-	A	95.6%
PGD	40	1	A	93.2%
PGD	100	1	A	91.8%
PGD	40	20	A	90.4%
PGD	100	20	A	89.3%
Targeted	40	1	A	92.7%
CW	40	1	A	94.0%
CW+	40	1	A	93.9%
FGSM	-	-	A'	96.8%
PGD	40	1	A'	96.0%
PGD	100	20	A'	95.7%
CW	40	1	A'	97.0%
CW+	40	1	A'	96.4%
FGSM	-	-	B	95.4%
PGD	40	1	B	96.4%
CW+	-	-	B	95.7%

MNIST

Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	45.8%
CW	30	A	46.8%
FGSM	-	A'	67.0%
PGD	7	A'	64.2%
CW	30	A'	78.7%
FGSM	-	A _{nat}	85.6%
PGD	7	A _{nat}	86.0%

CIFAR10

Limitations

- Having a large capacity network is computationally expensive
- Accuracy of networks on different datasets yields differing results
 - High accuracy on MNIST dataset, yet same level of performance not reached on the CIFAR10 dataset
 - Need to evaluate on more datasets



Discussion Questions

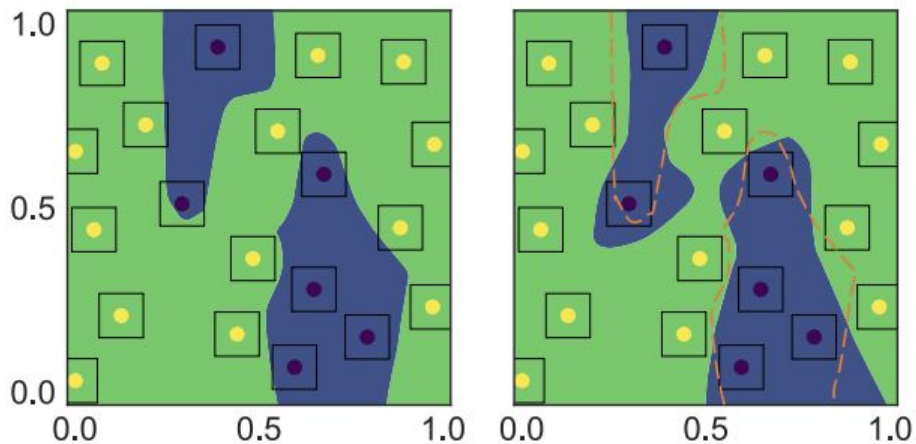
- Can adversarial training fully eliminate adversarial vulnerabilities, or will attackers always find stronger methods?
- How does adversarial robustness relate to real-world security applications (e.g., autonomous vehicles, biometric authentication)?
- What are the ethical considerations of adversarial attacks and defenses? Could there be unintended consequences of deploying robust models?



Theoretically Principled Trade-Off Between Robustness and Accuracy

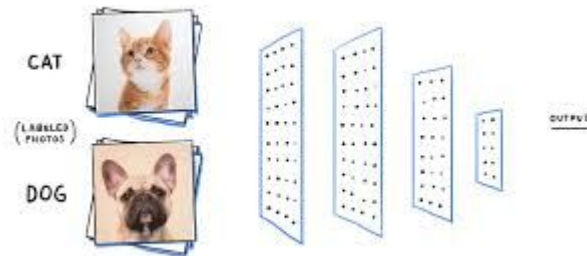
Motivation

- Vulnerable to adversarial attacks
- Can we design models that are both robust and accurate?



Key Idea - Robustness vs. Accuracy

- **Natural Error:** Misclassification on clean data
- **Robust Error:** Misclassification on adversarially perturbed data
- **Key Theorem:** Robust Error = Natural Error + Boundary Error
- Robust models need wider decision boundaries



TRADES

- Empirical risk minimization → improves accuracy
- Regularization loss → pushes boundary away from data
- Hyperparameter (lambda) controls balance

Algorithm 1 Adversarial training by TRADES

```
1: Input: Step sizes  $\eta_1$  and  $\eta_2$ , batch size  $m$ , number of iterations  $K$  in inner optimization, network architecture parametrized by  $\theta$ 
2: Output: Robust network  $f_\theta$ 
3: Randomly initialize network  $f_\theta$ , or initialize network with pre-trained configuration
4: repeat
5:   Read mini-batch  $B = \{x_1, \dots, x_m\}$  from training set
6:   for  $i = 1, \dots, m$  (in parallel) do
7:      $x'_i \leftarrow x_i + 0.001 \cdot \mathcal{N}(\mathbf{0}, \mathbf{I})$ , where  $\mathcal{N}(\mathbf{0}, \mathbf{I})$  is the Gaussian distribution with zero mean and identity variance
8:   for  $k = 1, \dots, K$  do
9:      $x'_i \leftarrow \Pi_{\mathbb{B}(x_i, \epsilon)}(\eta_1 \text{sign}(\nabla_{x'_i} \mathcal{L}(f_\theta(x_i), f_\theta(x'_i))) + x'_i)$ , where  $\Pi$  is the projection operator
10:   end for
11: end for
12:  $\theta \leftarrow \theta - \eta_2 \sum_{i=1}^m \nabla_\theta [\mathcal{L}(f_\theta(x_i), y_i) + \mathcal{L}(f_\theta(x_i), f_\theta(x'_i))]/\lambda / m$ 
13: until training converged
```

$$\min_f E \left[\phi(f(X)Y) + \max_{X' \in B(X, \epsilon)} \phi(f(X)f(X')/\lambda) \right]$$

Experimental Setup

- Datasets: MNIST, CIFAR-10
- Models: CNN, Wide ResNet
- Threat models:
 - White-box attacks: (PGD, FGSM, C&W)
 - Black-box attacks: Transferability tests
- Baseline defenses: Adversarial Training, Logit Pairing

TRADES vs. Baselines






- Under White-box attacks (CIFAR-10)
 - Trades robust accuracy: 56.6%
 - Madry et al: 47.0%
 - Other defenses < 50%
- Under Black-box attacks:
 - TRADES outperforms all other defenses
 - Maintains high accuracy against unseen attacks.

Table 5: Comparisons of TRADES with prior defense models under white-box attacks.





Defense	Defense type	Under which attack	Dataset	Distance	$\mathcal{A}_{\text{nat}}(f)$	$\mathcal{A}_{\text{rob}}(f)$
[BRRG18]	gradient mask	[ACW18]	CIFAR10	0.031 (ℓ_∞)	-	0%
[MLW ⁺ 18]	gradient mask	[ACW18]	CIFAR10	0.031 (ℓ_∞)	-	5%
[DAL ⁺ 18]	gradient mask	[ACW18]	CIFAR10	0.031 (ℓ_∞)	-	0%
[SKN ⁺ 18]	gradient mask	[ACW18]	CIFAR10	0.031 (ℓ_∞)	-	9%
[NKM17]	gradient mask	[ACW18]	CIFAR10	0.015 (ℓ_∞)	-	15%
[WSMK18]	robust opt.	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	27.07%	23.54%
[MMS ⁺ 18]	robust opt.	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	87.30%	47.04%
[ZSLG16]	regularization	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	94.64%	0.15%
[KGB17]	regularization	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	85.25%	45.89%
[RDV17]	regularization	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	95.34%	0%
TRADES (1/ λ = 1)	regularization	FGSM ^{1,000} (PGD)	CIFAR10	0.031 (ℓ_∞)	88.64%	48.90%
TRADES (1/ λ = 6)	regularization	FGSM ^{1,000} (PGD)	CIFAR10	0.031 (ℓ_∞)	84.92%	56.43%
TRADES (1/ λ = 1)	regularization	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	88.64%	49.14%
TRADES (1/ λ = 6)	regularization	FGSM ²⁰ (PGD)	CIFAR10	0.031 (ℓ_∞)	84.92%	56.61%
TRADES (1/ λ = 1)	regularization	DeepFool (ℓ_∞)	CIFAR10	0.031 (ℓ_∞)	88.64%	59.10%
TRADES (1/ λ = 6)	regularization	DeepFool (ℓ_∞)	CIFAR10	0.031 (ℓ_∞)	84.92%	61.38%
TRADES (1/ λ = 1)	regularization	LBFGSAttack	CIFAR10	0.031 (ℓ_∞)	88.64%	84.41%
TRADES (1/ λ = 6)	regularization	LBFGSAttack	CIFAR10	0.031 (ℓ_∞)	84.92%	81.58%
TRADES (1/ λ = 1)	regularization	MI-FGSM	CIFAR10	0.031 (ℓ_∞)	88.64%	51.26%
TRADES (1/ λ = 6)	regularization	MI-FGSM	CIFAR10	0.031 (ℓ_∞)	84.92%	57.95%
TRADES (1/ λ = 1)	regularization	C&W	CIFAR10	0.031 (ℓ_∞)	88.64%	84.03%
TRADES (1/ λ = 6)	regularization	C&W	CIFAR10	0.031 (ℓ_∞)	84.92%	81.24%
[SKC18]	gradient mask	[ACW18]	MNIST	0.005 (ℓ_2)	-	55%
[MMS ⁺ 18]	robust opt.	FGSM ⁴⁰ (PGD)	MNIST	0.3 (ℓ_∞)	99.36%	96.01%
TRADES (1/ λ = 6)	regularization	FGSM ^{1,000} (PGD)	MNIST	0.3 (ℓ_∞)	99.48%	95.60%
TRADES (1/ λ = 6)	regularization	FGSM ⁴⁰ (PGD)	MNIST	0.3 (ℓ_∞)	99.48%	96.07%
TRADES (1/ λ = 6)	regularization	C&W	MNIST	0.005 (ℓ_2)	99.48%	99.46%

Strengths & Weaknesses of TRADES

Strengths

- Mathematically grounded → built on 
- formal theorem 
- Improves robust accuracy 
- Flexible and Adjustable 
- Scalable 

Weaknesses

- Trade-offs still exist 
- Hyperparameter sensitivity 
- Limited Attacks 
- 

Discussion

- **Scenario:** ML engineer at financial institution tasked with developing a model that predicts if a customer will repay or default on a loan
- **Challenge:** Hackers use adversarial attacks to slightly modify records, tricking the AI into approving risky loans

Discussion Questions:

- Should we prioritize robustness or accuracy? Why?
- What happens if we set λ too high? How would that impact legitimate customers?
- How do we ensure fairness while making the model robust? Could robust training affect different demographics unfairly?