



Boulder

Computer Vision; Image Classification; Robustness



[YouTube Playlist](#)

Maziar Raissi

Assistant Professor

Department of Applied Mathematics

University of Colorado Boulder

maziar.raissi@colorado.edu



Boulder



[YouTube Playlist](#)

Intriguing properties of neural networks

$x \in \mathbb{R}^m \rightarrow$ an input image

$\phi(x) \in \mathbb{R}^n \rightarrow$ activation values of some layer

Units of $\phi(x)$

$$x' = \arg \max_{x \in \mathcal{I}} \langle \phi(x), e_i \rangle$$

$e_i \rightarrow$ natural basis vector associated with the i -th hidden unit

$\mathcal{I} \rightarrow$ held-out set of images from the data distribution that the network was not trained on



(a) Unit sensitive to lower round stroke.



(b) Unit sensitive to upper round stroke, or lower straight stroke.



(c) Unit sensitive to left, upper round stroke.



(d) Unit sensitive to diagonal straight stroke.

$$x' = \arg \max_{x \in \mathcal{I}} \langle \phi(x), v \rangle$$

$v \in \mathbb{R}^n \rightarrow$ any random direction



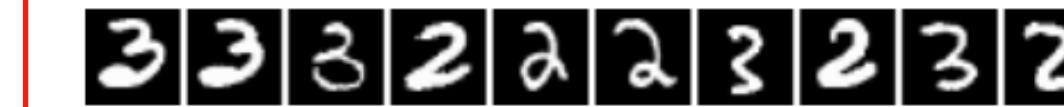
(a) Direction sensitive to upper straight stroke, or lower round stroke.



(b) Direction sensitive to lower left loop.



(c) Direction sensitive to round top stroke.



(d) Direction sensitive to right, upper round stroke.



Unit sensitive to round, spiky flowers.



Direction sensitive to spread shapes.

Blind Spots

$f : \mathbb{R}^m \rightarrow \{1, \dots, k\} \rightarrow$ a classifier

$$\min_r \|r\|_2 \quad \text{given } \ell \neq f(x)$$

$$\text{s.t. } f(x + r) = \ell$$

$$x + r \in [0, 1]^m$$

if $\ell = f(x)$ then $r = 0$ is a trivial solution

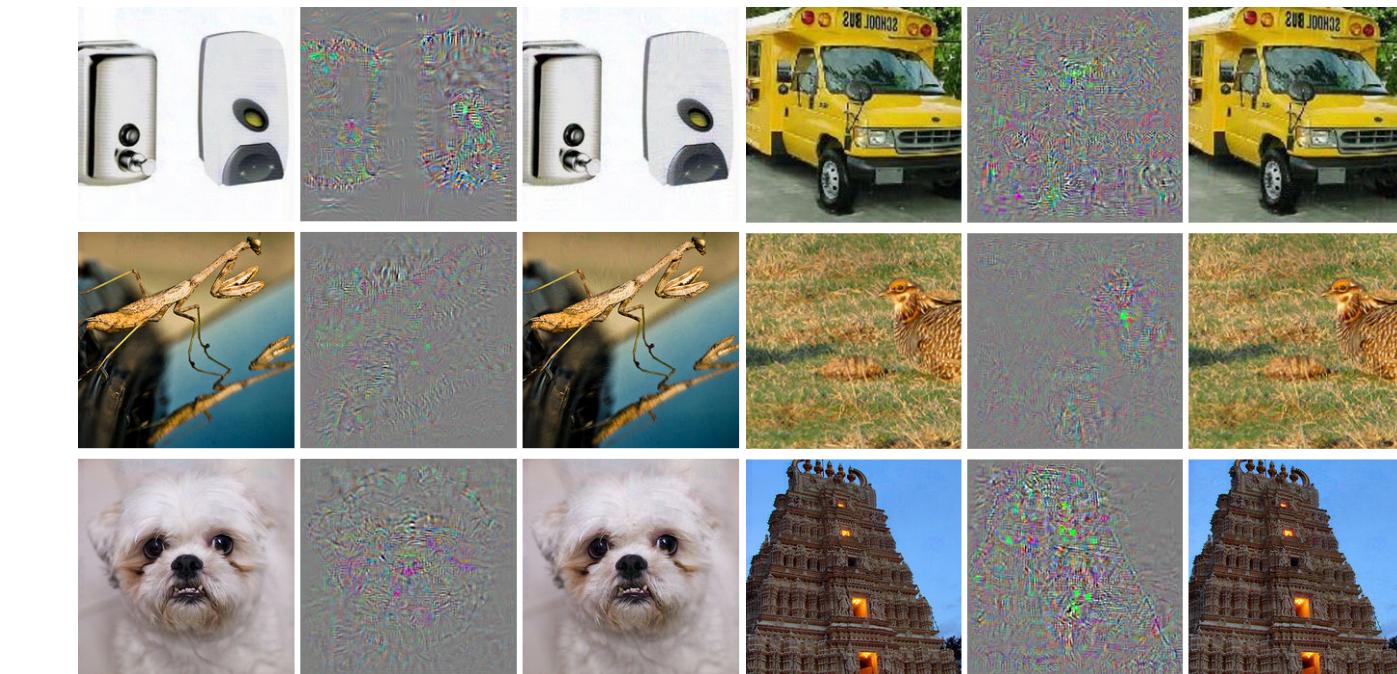
$\text{loss}_f : \mathbb{R}^m \times \{1, \dots, k\} \rightarrow \mathbb{R}^+$

$$\min_r c\|r\|_2 + \text{loss}_f(x + r, \ell)$$

$$\text{s.t. } x + r \in [0, 1]^m$$

Box-constrained L-BFGS

Cross-model generalization of adversarial examples



	FC10(10^{-4})	FC10(10^{-2})	FC10(1)	FC100-100-10	FC200-200-10	AE400-10	Av. distortion
FC10(10^{-4})	100%	11.7%	22.7%	2%	3.9%	2.7%	0.062
FC10(10^{-2})	87.1%	100%	35.2%	35.9%	27.3%	9.8%	0.1
FC10(1)	71.9%	76.2%	100%	48.1%	47%	34.4%	0.14
FC100-100-10	28.9%	13.7%	21.1%	100%	6.6%	2%	0.058
FC200-200-10	38.2%	14%	23.8%	20.3%	100%	2.7%	0.065
AE400-10	23.4%	16%	24.8%	9.4%	6.6%	100%	0.086
Gaussian noise, stddev=0.1	5.0%	10.1%	18.3%	0%	0%	0.8%	0.1
Gaussian noise, stddev=0.3	15.6%	11.3%	22.7%	5%	4.3%	3.1%	0.3

The general conclusion is that adversarial examples tend to stay hard even for models trained with different hyper-parameters.

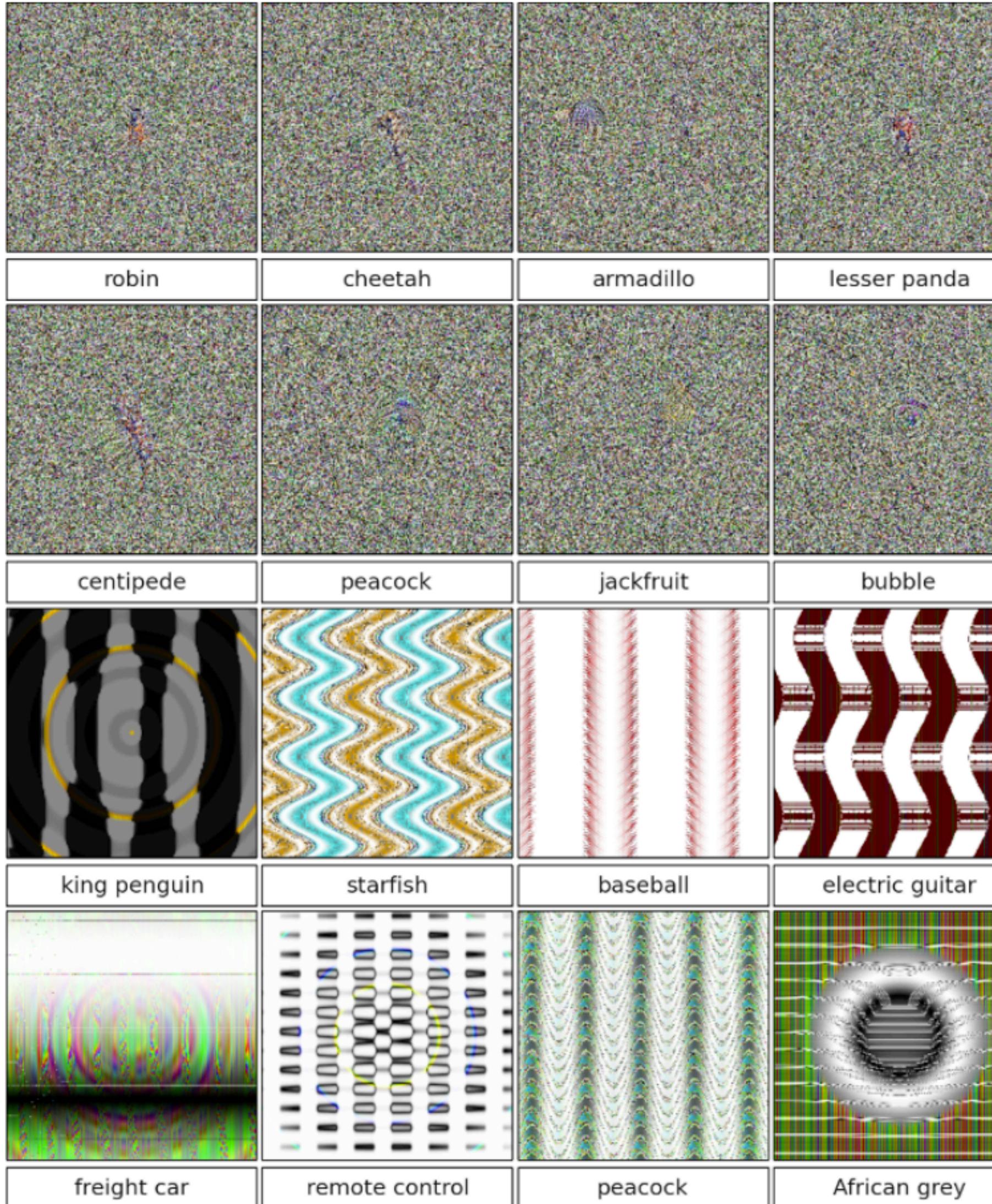
Cross-training-set generalization error rate for the set of adversarial examples generated for different models

	FC100-100-10	FC123-456-10	FC100-100-10'
Distorted for FC100-100-10 (av. stddev=0.062)	100%	26.2%	5.9%
Distorted for FC123-456-10 (av. stddev=0.059)	6.25%	100%	5.1%
Distorted for FC100-100-10' (av. stddev=0.058)	8.2%	8.2%	100%
Gaussian noise with stddev=0.06	2.2%	2.6%	2.4%
Distorted for FC100-100-10 amplified to stddev=0.1	100%	98%	43%
Distorted for FC123-456-10 amplified to stddev=0.1	96%	100%	22%
Distorted for FC100-100-10' amplified to stddev=0.1	27%	50%	100%
Gaussian noise with stddev=0.1	2.6%	2.8%	2.7%

Adversarial examples remain hard for models trained even on a disjoint training set.



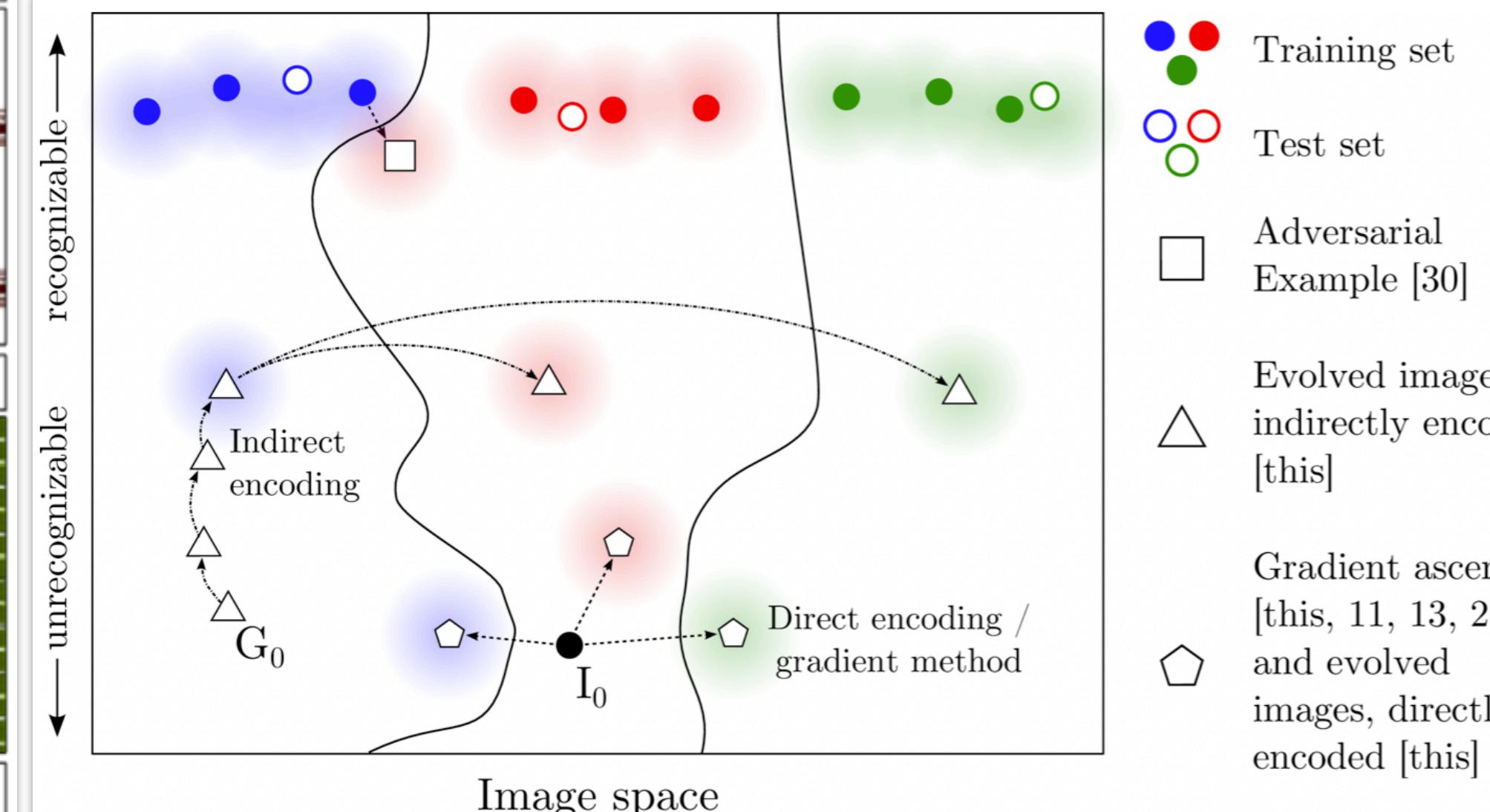
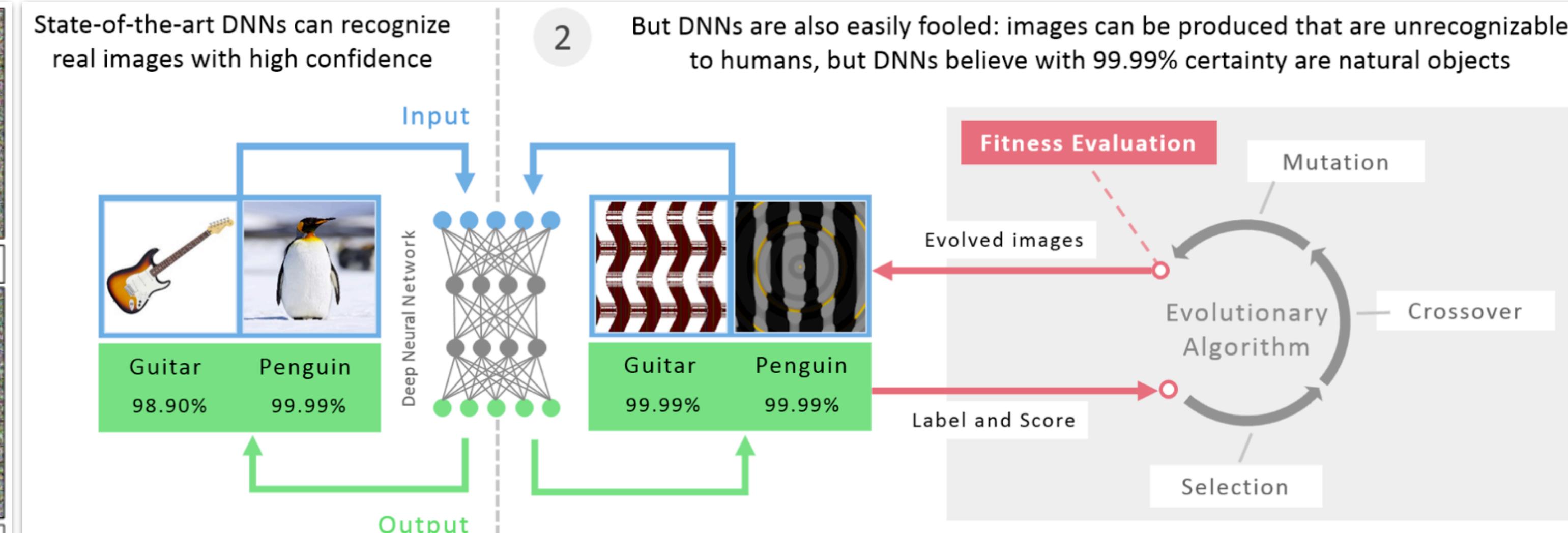
Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images



State-of-the-art DNNs can recognize real images with high confidence

2

But DNNs are also easily fooled: images can be produced that are unrecognizable to humans, but DNNs believe with 99.99% certainty are natural objects



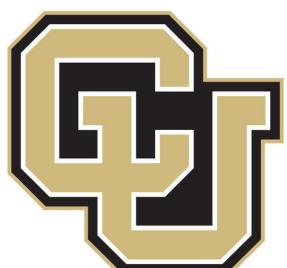
Directly Encoded

Each pixel value is initialized with uniform random noise within the $[0, 255]$ range. Those numbers are independently mutated.

Indirectly Encoded

Compositional Pattern-Producing Network (CPPN)

A CPPN takes in the (x, y) position of a pixel as input, and outputs pixel values using sine, sigmoid, Gaussian and linear functions.



Boulder

DeepFool: a simple and accurate method to fool deep neural networks

DeepFool: whale → turtle



$$\Delta(\mathbf{x}; \hat{k}) := \min_{\mathbf{r}} \|\mathbf{r}\|_2 \text{ subject to } \hat{k}(\mathbf{x} + \mathbf{r}) \neq \hat{k}(\mathbf{x})$$

robustness of classifier \hat{k} at point x

$$\rho_{\text{adv}}(\hat{k}) = \mathbb{E}_{\mathbf{x}} \frac{\Delta(\mathbf{x}; \hat{k})}{\|\mathbf{x}\|_2} \rightarrow \text{robustness of classifier } \hat{k}$$

DeepFool for binary classifiers

$$\hat{k}(\mathbf{x}) = \text{sign}(f(\mathbf{x}))$$

$f \rightarrow$ an arbitrary scalar-valued image classification function

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

$$\mathbf{r}_*(\mathbf{x}_0) := \arg \min \|\mathbf{r}\|_2$$

subject to $\text{sign}(f(\mathbf{x}_0 + \mathbf{r})) \neq \text{sign}(f(\mathbf{x}_0))$

$$= -\frac{f(\mathbf{x}_0)}{\|\mathbf{w}\|_2^2} \mathbf{w}.$$

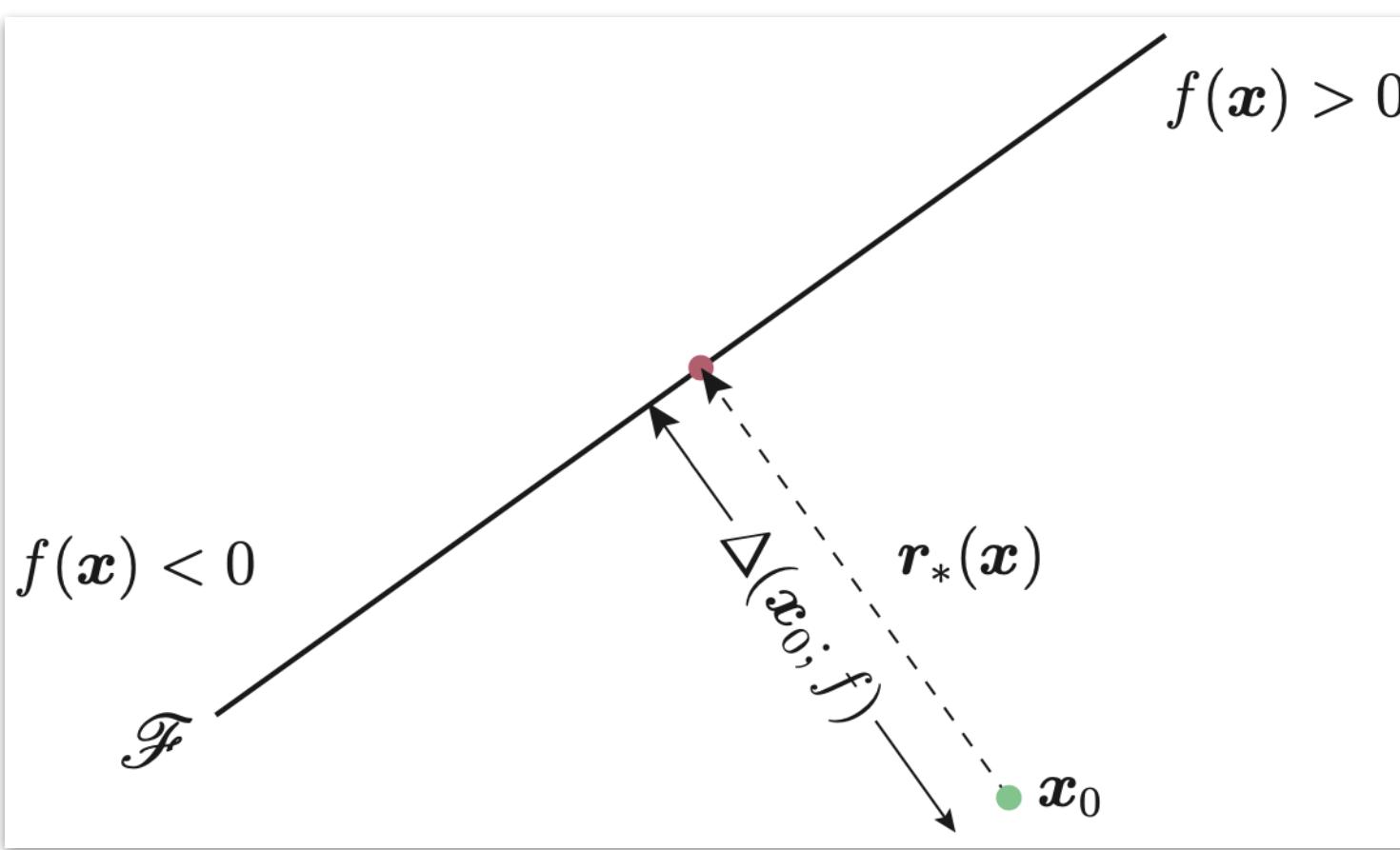
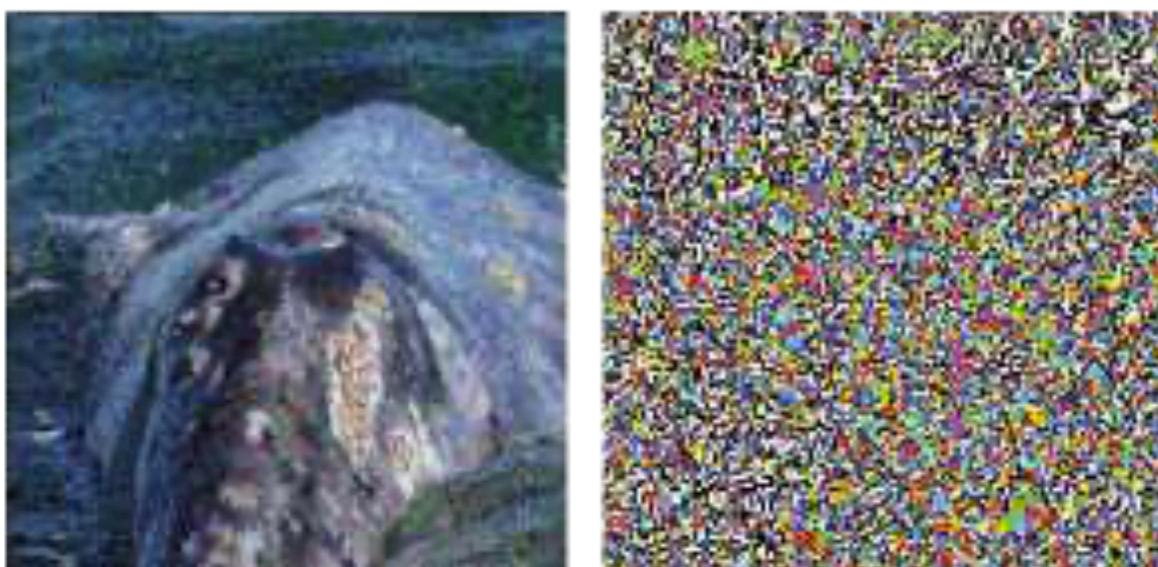
$$\mathbf{w}^T (\mathbf{x}_0 - \frac{f(\mathbf{x}_0)}{\|\mathbf{w}\|_2^2} \mathbf{w}) + b = f(\mathbf{x}_0) - f(\mathbf{x}_0) = 0$$

Algorithm 1 DeepFool for binary classifiers

```

1: input: Image  $\mathbf{x}$ , classifier  $f$ .
2: output: Perturbation  $\hat{\mathbf{r}}$ .
3: Initialize  $\mathbf{x}_0 \leftarrow \mathbf{x}, i \leftarrow 0$ .
4: while  $\text{sign}(f(\mathbf{x}_i)) = \text{sign}(f(\mathbf{x}_0))$  do
5:    $\mathbf{r}_i \leftarrow -\frac{f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|_2^2} \nabla f(\mathbf{x}_i)$ ,
6:    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \mathbf{r}_i$ ,
7:    $i \leftarrow i + 1$ .
8: end while
9: return  $\hat{\mathbf{r}} = \sum_i \mathbf{r}_i$ .
```

Fast Gradient Sign: whale → turtle

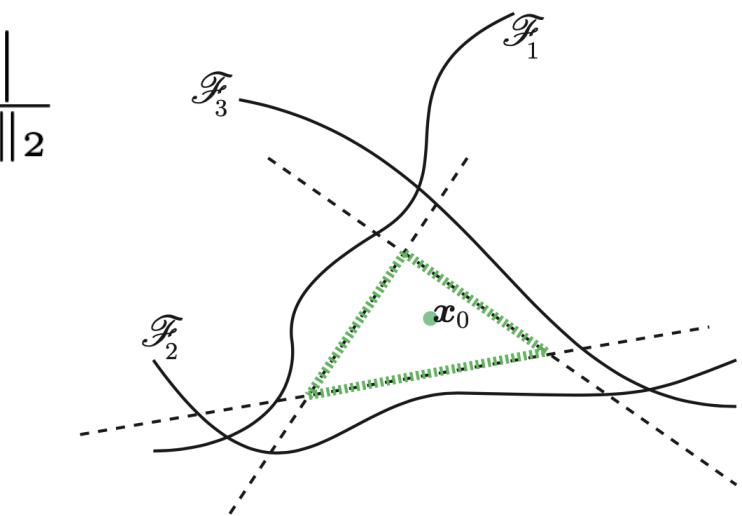


DeepFool for multiclass classifiers

Algorithm 2 DeepFool: multi-class case

```

1: input: Image  $\mathbf{x}$ , classifier  $f$ .
2: output: Perturbation  $\hat{\mathbf{r}}$ .
3:
4: Initialize  $\mathbf{x}_0 \leftarrow \mathbf{x}, i \leftarrow 0$ .
5: while  $\hat{k}(\mathbf{x}_i) = \hat{k}(\mathbf{x}_0)$  do
6:   for  $k \neq \hat{k}(\mathbf{x}_0)$  do
7:      $\mathbf{w}'_k \leftarrow \nabla f_k(\mathbf{x}_i) - \nabla f_{\hat{k}(\mathbf{x}_0)}(\mathbf{x}_i)$ 
8:      $f'_k \leftarrow f_k(\mathbf{x}_i) - f_{\hat{k}(\mathbf{x}_0)}(\mathbf{x}_i)$ 
9:   end for
10:   $\hat{l} \leftarrow \arg \min_{k \neq \hat{k}(\mathbf{x}_0)} \frac{|f'_k|}{\|\mathbf{w}'_k\|_2}$ 
11:   $\mathbf{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\mathbf{w}'_{\hat{l}}\|_2^2} \mathbf{w}'_{\hat{l}}$ 
12:   $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \mathbf{r}_i$ 
13:   $i \leftarrow i + 1$ 
14: end while
15: return  $\hat{\mathbf{r}} = \sum_i \mathbf{r}_i$ 
```



$$\mathcal{F}_k = \{x : f_k(x) - f_4(x) = 0\}$$

Classifier	Test error	$\hat{\rho}_{\text{adv}}$ [DeepFool]	time	$\hat{\rho}_{\text{adv}}$ [4]	time	$\hat{\rho}_{\text{adv}}$ [18]	time
LeNet (MNIST)	1%	2.0×10^{-1}	110 ms	1.0	20 ms	2.5×10^{-1}	> 4 s
FC500-150-10 (MNIST)	1.7%	1.1×10^{-1}	50 ms	3.9×10^{-1}	10 ms	1.2×10^{-1}	> 2 s
NIN (CIFAR-10)	11.5%	2.3×10^{-2}	1100 ms	1.2×10^{-1}	180 ms	2.4×10^{-2}	> 50 s
LeNet (CIFAR-10)	22.6%	3.0×10^{-2}	220 ms	1.3×10^{-1}	50 ms	3.9×10^{-2}	> 7 s
CaffeNet (ILSVRC2012)	42.6%	2.7×10^{-3}	510 ms*	3.5×10^{-2}	50 ms*	-	-
GoogLeNet (ILSVRC2012)	31.3%	1.9×10^{-3}	800 ms*	4.7×10^{-2}	80 ms*	-	-



Boulder

Adversarial Examples in the Physical World

$M \rightarrow$ machine learning system

$C \rightarrow$ clean example

$M(C) = y_{true} \rightarrow$ correctly classified

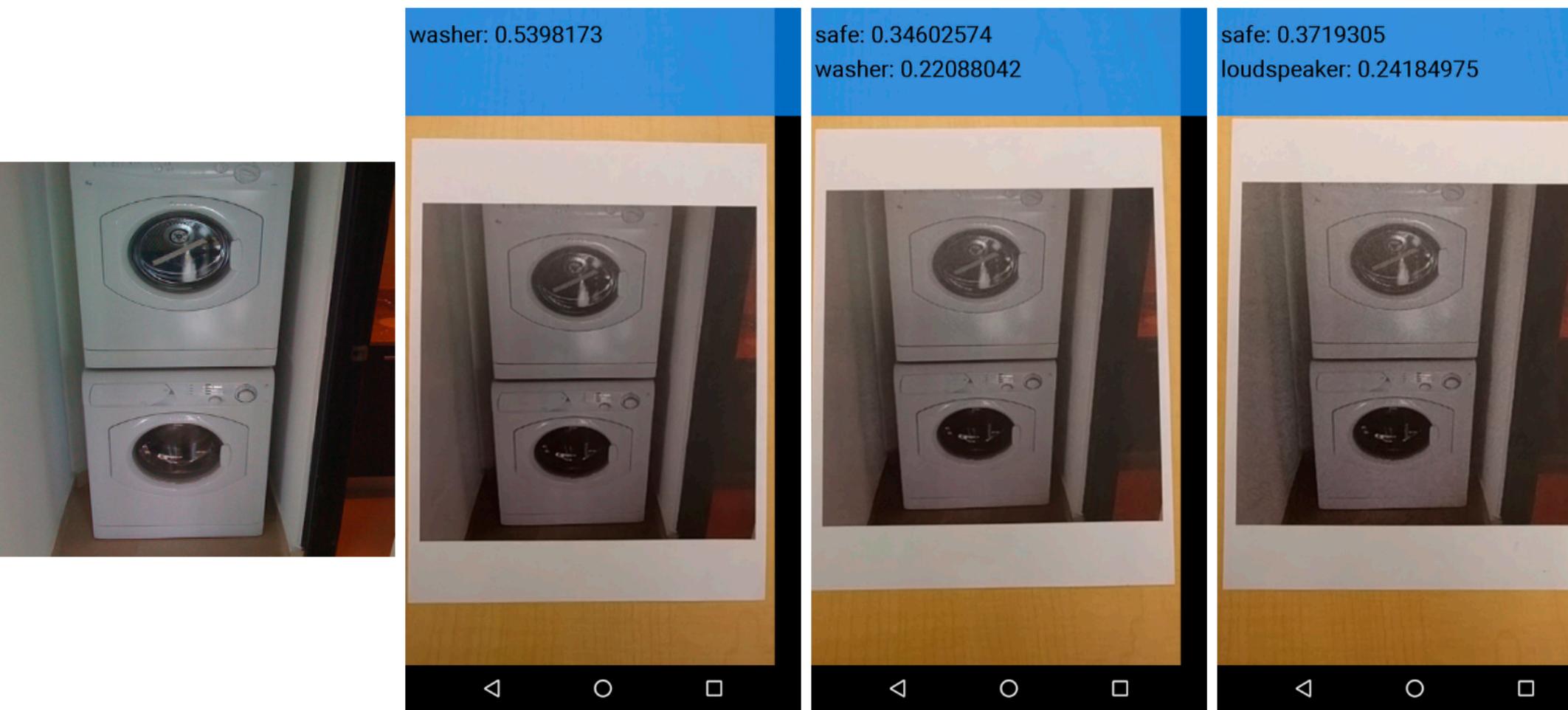
$A \rightarrow$ adversarial example (perceptually indistinguishable from C)

$M(A) \neq y_{true}$

An adversarial example that was designed to be misclassified by a model M_1 is often also misclassified by a model M_2 .

- robots perceiving world through cameras and other sensors
- video surveillance systems
- mobile applications for image or sound classification

Do adversarial examples survive transformations
(e.g., printing the image and taking a photo of the result)?



(a) Image from dataset

(b) Clean image

(c) Adv. image, $\epsilon = 4$

(d) Adv. image, $\epsilon = 8$

Methods for generating adversarial examples

X - an image y_{true} - true class for the image X

Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." (2016).

$J(\mathbf{X}, y)$ - cross-entropy cost function

$J(\mathbf{X}, y) = -\log p(y|\mathbf{X}) \rightarrow$ log probability of the class given the image

$$Clip_{X,\epsilon} \{\mathbf{X}'\}(x, y, z) = \min \left\{ 255, \mathbf{X}(x, y, z) + \epsilon, \max \left\{ 0, \mathbf{X}(x, y, z) - \epsilon, \mathbf{X}'(x, y, z) \right\} \right\}$$

\curvearrowleft will be in L_∞ ϵ -neighbourhood of the source image \mathbf{X}

Fast Method

$$\mathbf{X}^{adv} = \mathbf{X} + \epsilon \operatorname{sign}(\nabla_{\mathbf{X}} J(\mathbf{X}, y_{true})) \rightarrow$$
 decrease the probability of the true class

Basic Iterative Method

$$\mathbf{X}_0^{adv} = \mathbf{X}, \quad \mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon} \left\{ \mathbf{X}_N^{adv} + \alpha \operatorname{sign}(\nabla_{\mathbf{X}} J(\mathbf{X}_N^{adv}, y_{true})) \right\}$$

Iterative Least-Likely Class Method

$$y_{LL} = \arg \min_y \{p(y|\mathbf{X})\}$$

\curvearrowleft least-likely class according to the prediction of the trained network

$$\mathbf{X}_0^{adv} = \mathbf{X}, \quad \mathbf{X}_{N+1}^{adv} = Clip_{X,\epsilon} \left\{ \mathbf{X}_N^{adv} - \alpha \operatorname{sign}(\nabla_{\mathbf{X}} J(\mathbf{X}_N^{adv}, y_{LL})) \right\} \curvearrowleft$$

Destruction Rate

increase the probability of the least likely class

$$d = \frac{\sum_{k=1}^n C(\mathbf{X}^k, y_{true}^k) \overline{C(\mathbf{X}_{adv}^k, y_{true}^k)} C(T(\mathbf{X}_{adv}^k), y_{true}^k)}{\sum_{k=1}^n C(\mathbf{X}^k, y_{true}^k) \overline{C(\mathbf{X}_{adv}^k, y_{true}^k)}}$$

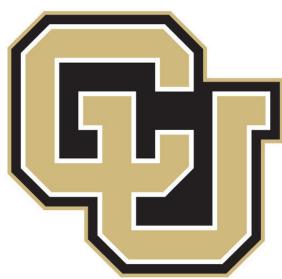
Fraction of adversarial images which are no longer misclassified after the transformations!

$$C(\mathbf{X}, y) = \begin{cases} 1, & \text{if image } \mathbf{X} \text{ is classified as } y; \\ 0, & \text{otherwise.} \end{cases}$$

$$\overline{C(\mathbf{X}, y)} = 1 - C(\mathbf{X}, y) \rightarrow \text{binary negation}$$

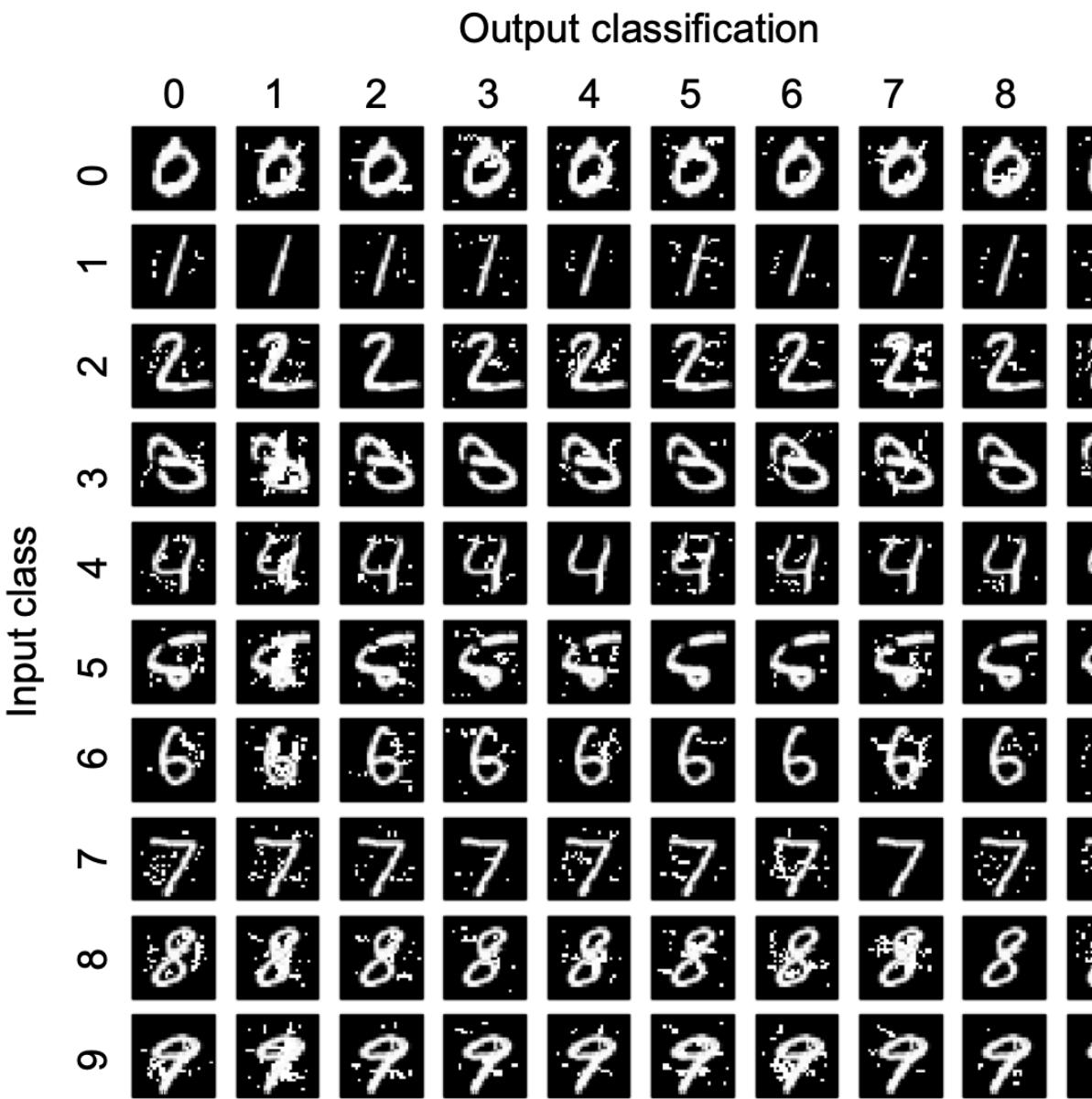
$n \rightarrow$ number of images used to compute the destruction rate

Adversarial method	Average case		Prefiltered case	
	top-1	top-5	top-1	top-5
fast $\epsilon = 16$	12.5%	40.0%	5.1%	39.4%
fast $\epsilon = 8$	33.3%	40.0%	14.6%	70.8%
fast $\epsilon = 4$	46.7%	65.9%	32.4%	91.2%
fast $\epsilon = 2$	61.1%	63.2%	49.5%	91.9%
iter. basic $\epsilon = 16$	40.4%	69.4%	60.0%	87.8%
iter. basic $\epsilon = 8$	52.1%	90.5%	64.7%	91.2%
iter. basic $\epsilon = 4$	52.4%	82.6%	77.5%	94.1%
iter. basic $\epsilon = 2$	71.7%	81.5%	80.8%	96.9%
1.1. class $\epsilon = 16$	72.2%	85.1%	87.5%	97.9%
1.1. class $\epsilon = 8$	86.3%	94.6%	88.9%	97.0%
1.1. class $\epsilon = 4$	90.3%	93.9%	91.9%	98.0%
1.1. class $\epsilon = 2$	82.1%	93.9%	93.1%	98.0%



Boulder

The Limitations of Deep Learning in Adversarial Settings



$$f : x \mapsto y$$

Deep Neural Network

$x \rightarrow$ raw feature vector

$y \rightarrow$ output vector

$x^* = x + \delta_x \rightarrow$ adversarial sample

$\delta_x \rightarrow$ perturbation vector

$$\arg \min_{\delta_x} \|\delta_x\| \text{ s.t. } f(x^*) = y^* \neq y$$

Adversarial Saliency Maps

$$\text{label}(x) = \arg \max_j f_j(x)$$

score of class j

Modify x s.t. it is misclassified as $t \neq \text{label}(x)$

Increase $f_t(x)$ and decrease $f_j(x), \forall j \neq t$

Finding features that the adversary should increase to achieve misclassification!

$$J_{ij}(x) := \frac{\partial f_j(x)}{\partial x_i} \quad i \rightarrow \text{an input feature}$$

$$s(i; x, t) := 0 \text{ if } J_{it}(x) < 0 \text{ or } \sum_{j \neq t} J_{ij}(x) > 0 \\ := |J_{it}(x)| \sum_{j \neq t} |J_{ij}(x)| \text{ otherwise}$$

Algorithm 1 Crafting adversarial samples

X is the benign sample, \mathbf{Y}^* is the target network output, \mathbf{F} is the function learned by the network during training, Υ is the maximum distortion, and θ is the change made to features. This algorithm is applied to a specific DNN architecture and dataset in Algorithm 2.

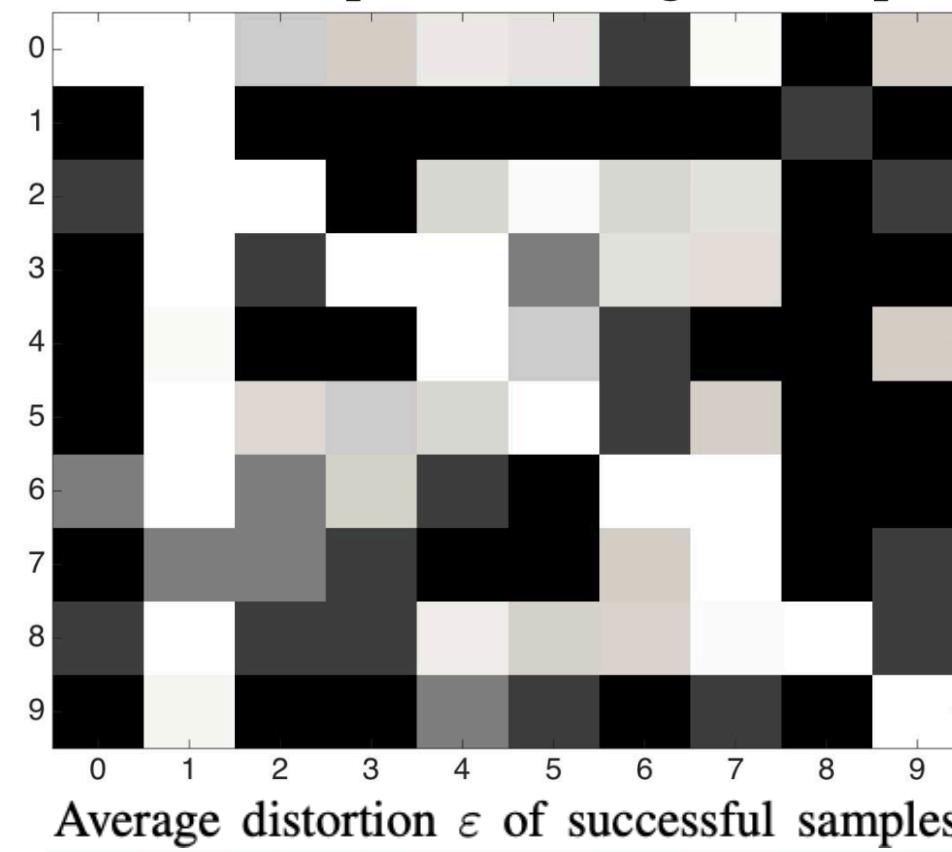
Input: $\mathbf{X}, \mathbf{Y}^*, \mathbf{F}, \Upsilon, \theta$

```

1:  $\mathbf{X}^* \leftarrow \mathbf{X}$ 
2:  $\delta_{\mathbf{X}} \leftarrow \vec{0}$ 
3:  $\Gamma = \{1 \dots |\mathbf{X}|\}$ 
4: while  $\mathbf{F}(\mathbf{X}^*) \neq \mathbf{Y}^*$  and  $\|\delta_{\mathbf{X}}\| < \Upsilon$  do
5:   Compute forward derivative  $J_{\mathbf{F}}(\mathbf{X}^*)$ 
6:    $S(\mathbf{X}, \mathbf{Y}^*) = \text{saliency\_map}(J_{\mathbf{F}}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$ 
7:    $i_{\max} \leftarrow \arg \max_i S(\mathbf{X}, \mathbf{Y}^*)[i]$ 
8:   Modify  $\mathbf{X}^*_{i_{\max}}$  by  $\theta$ 
9:    $\delta_{\mathbf{X}} \leftarrow \mathbf{X}^* - \mathbf{X}$ 
10: end while
11: return  $\mathbf{X}^*$ 
```

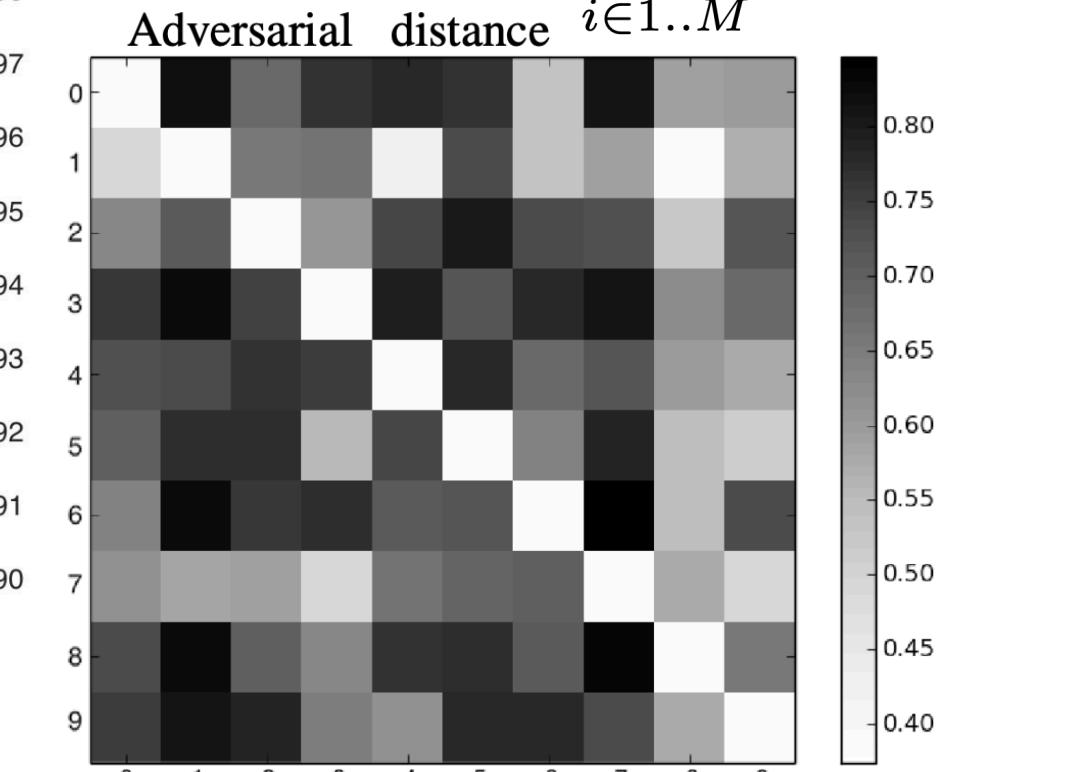
$$H(s, t) \approx \sum_{k=1}^{K-1} (\tau_{k+1} - \tau_k) \frac{\varepsilon(s, t, \tau_{k+1}) + \varepsilon(s, t, \tau_k)}{2}$$

Success rate per source-target class pair.



Adversarial Distance

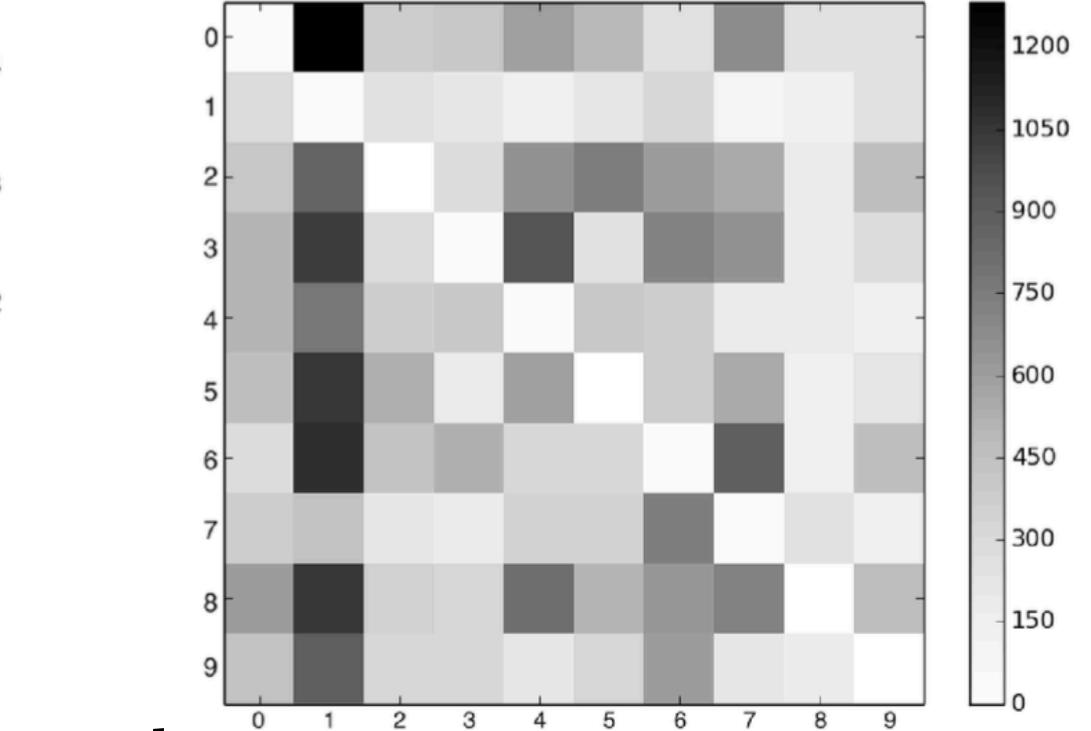
$$A(\mathbf{X}, t) = 1 - \frac{1}{M} \sum_{i \in 1..M} 1_{S(\mathbf{X}, t)[i] > 0}$$



Robustness

$$R(\mathbf{F}) = \min_{(\mathbf{X}, t)} A(\mathbf{X}, t)$$

Hardness matrix

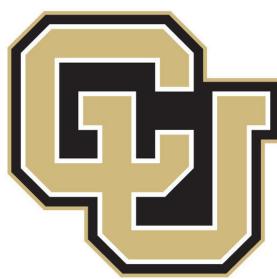


Hardness measure

$$H(s, t) = \int_{\tau} \varepsilon(s, t, \tau) d\tau$$

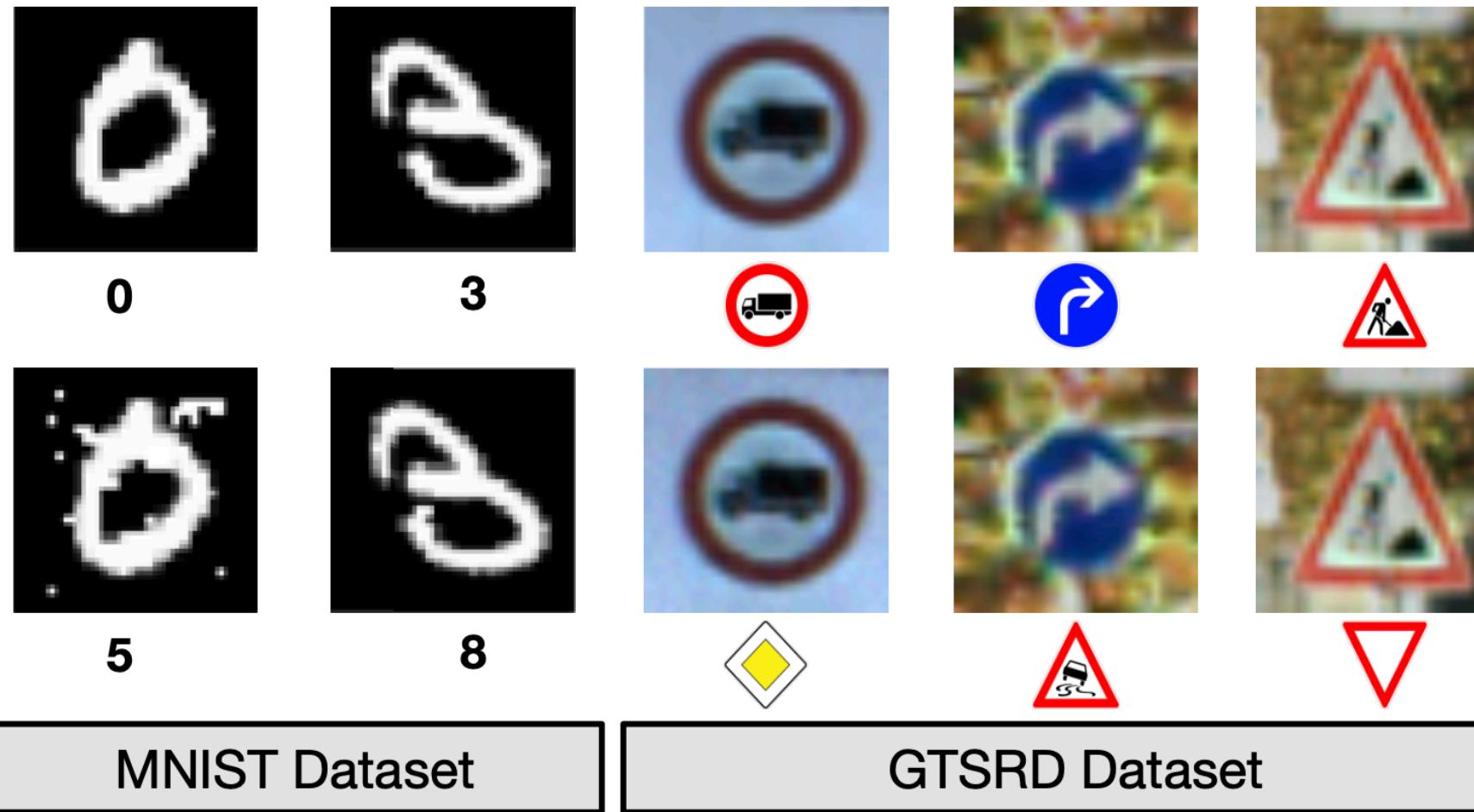
average distortion of a set of samples for the corresponding success rate τ

$(\varepsilon_k, \tau_k), k = 1, \dots, K \rightarrow$ correspond to different Υ_k



Boulder

Practical Black-Box Attacks against Machine Learning



Black-Box Attacks: The adversary (a) has no information about the structure or parameters of the DNN, and (b) does not have access to any large training dataset.

The adversary is only capable of observing labels assigned by the DNN for chosen inputs.

Attack Strategy: Train a local substitute DNN with a “synthetic” dataset.

$O \rightarrow$ Oracle (targeted DNN)

$x \rightarrow$ input

$\tilde{O}(x) \rightarrow$ label (the class assigned the largest probability by O)

$$\tilde{O}(x) = \arg \max_j O_j(x)$$

$O(x) \rightarrow$ probability vector

$x^* = x + \delta_x \rightarrow$ adversarial example

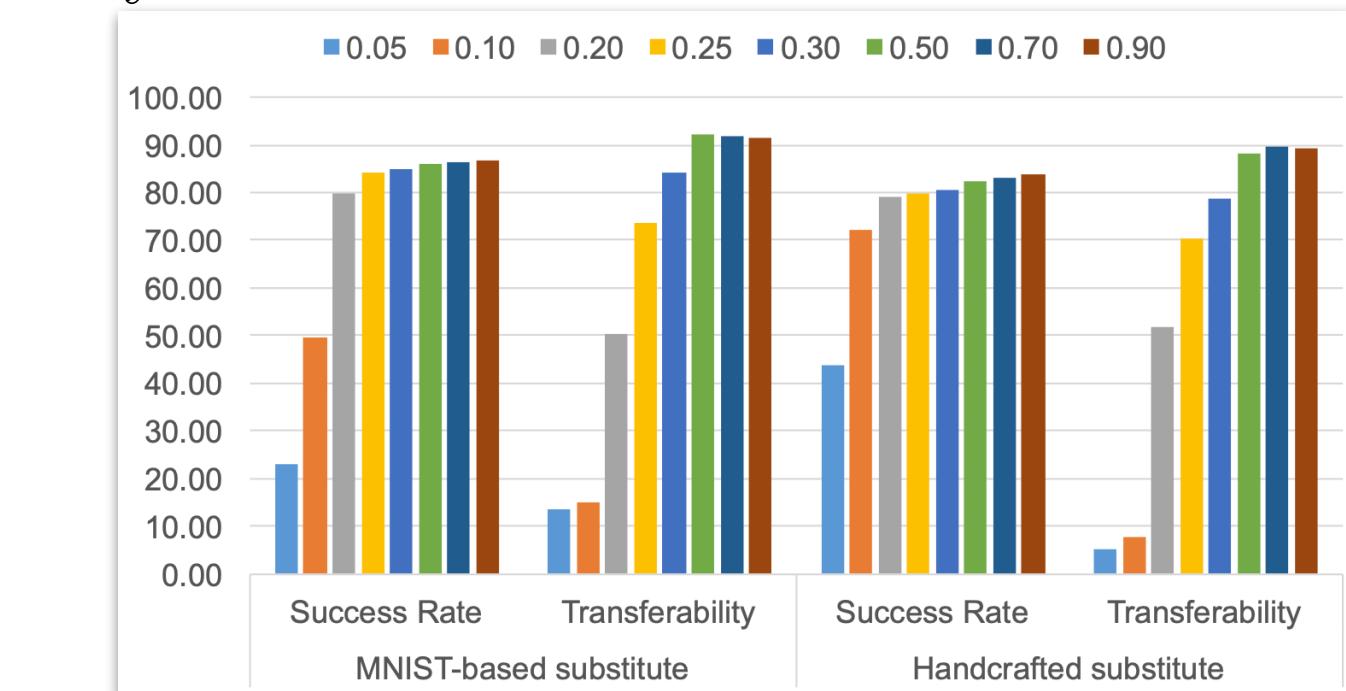
$$\delta_x = \arg \min\{z : \tilde{O}(x + z) \neq \tilde{O}(x)\}$$

$F \rightarrow$ substitute network (learn similar decision boundary to O)

Jacobian-based Dataset Augmentation

$$J_F(x) = \left[\frac{\partial F_j(x)}{\partial x_i} \right]_{i,j} \rightarrow \text{model's Jacobian matrix}$$

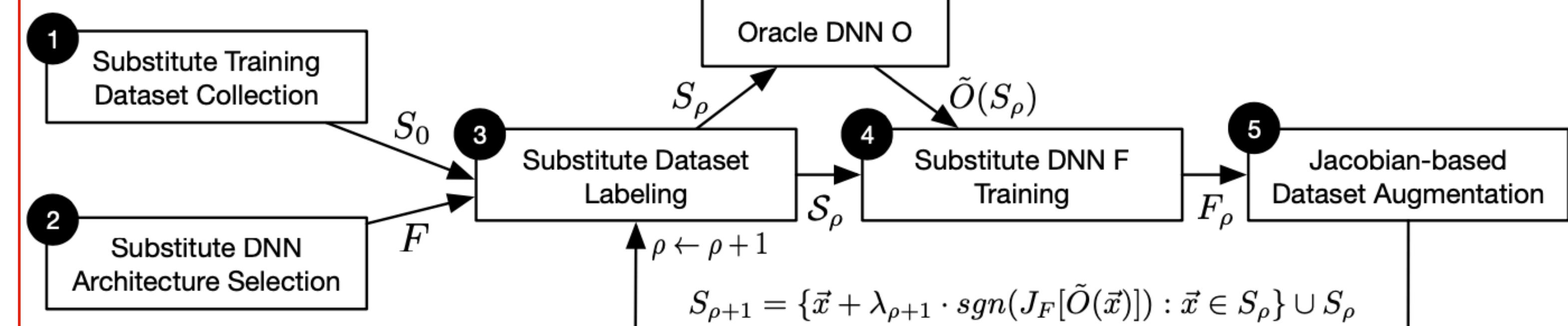
$$x + \lambda \operatorname{sgn}(J_F(x)[\tilde{O}(x)])$$



Algorithm 1 - Substitute DNN Training: for oracle \tilde{O} , a maximum number \max_ρ of substitute training epochs, a substitute architecture F , and an initial training set S_0 .

Input: \tilde{O} , \max_ρ , S_0 , λ

- 1: Define architecture F
- 2: **for** $\rho \in 0 .. \max_\rho - 1$ **do**
- 3: // Label the substitute training set
- 4: $D \leftarrow \{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\}$
- 5: // Train F on D to evaluate parameters θ_F
- 6: $\theta_F \leftarrow \text{train}(F, D)$
- 7: // Perform Jacobian-based dataset augmentation
- 8: $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \operatorname{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
- 9: **end for**
- 10: **return** θ_F



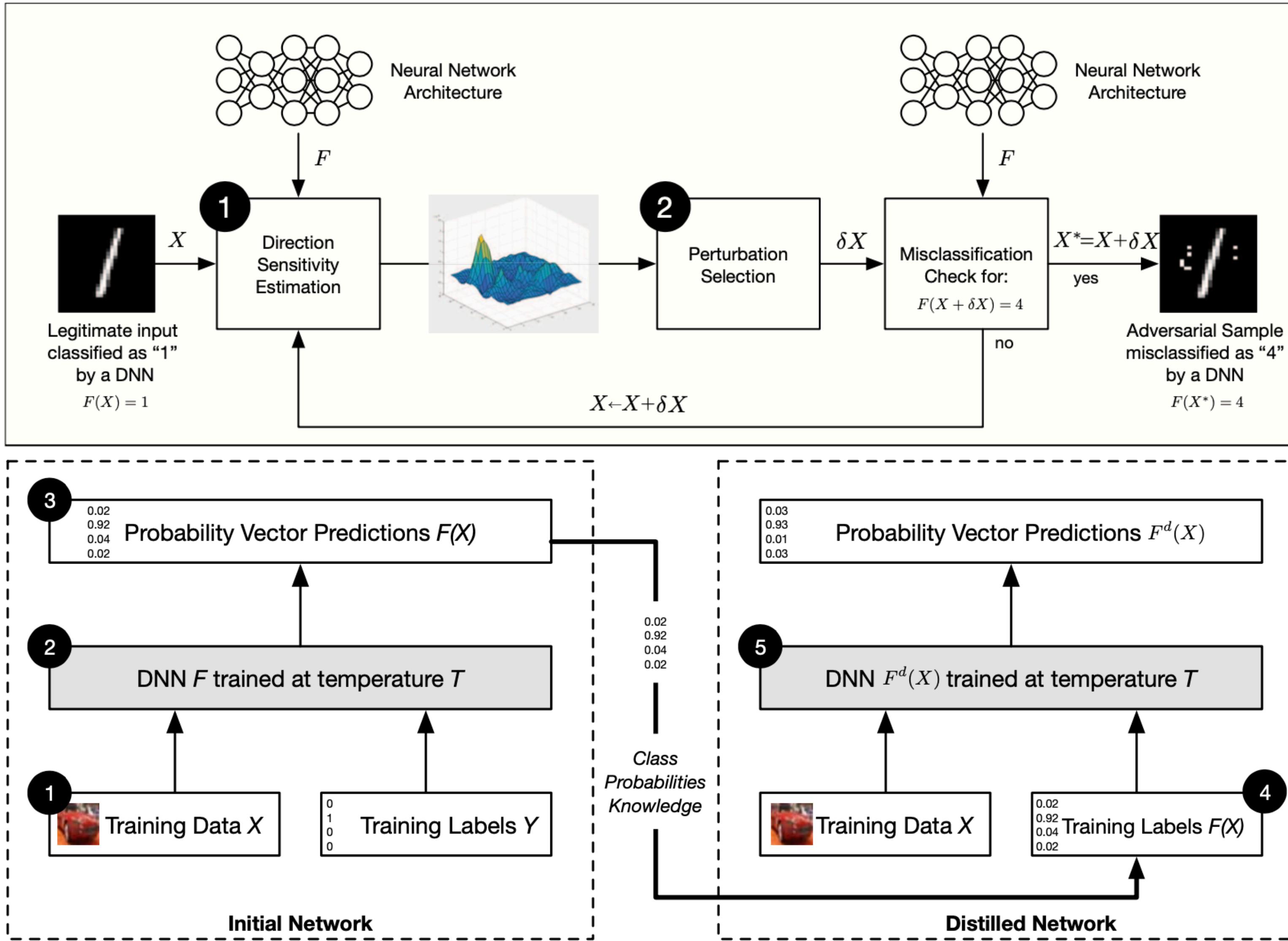
Adversarial Sample Crafting

- Fast Gradient Sign Method
- Adversarial Saliency Maps



Boulder

Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks



Robustness

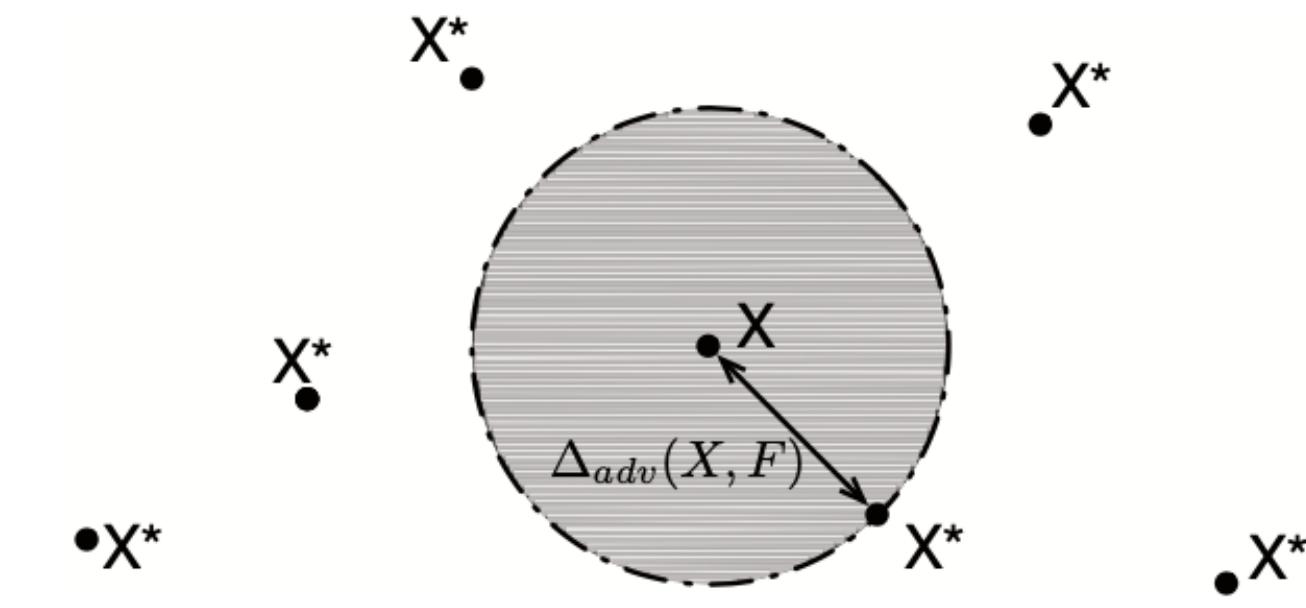
Robustness is the average minimal perturbation required to produce an adversarial sample from the distribution modeled by F .

$$\rho_{adv}(F) = E_{\mu}[\Delta_{adv}(X, F)]$$

$$\Delta_{adv}(X, F) = \arg \min_{\delta X} \{ \|\delta X\| : F(X + \delta X) \neq F(X) \}$$

$X \rightarrow$ inputs drawn from distribution μ that DNN architecture is modeling with F

$\Delta_{adv}(X, F) \rightarrow$ minimum perturbation required to misclassify sample X in each of the other classes



$$\rho_{adv}(F) \simeq \frac{1}{|\mathcal{X}|} \sum_{X \in \mathcal{X}} \min_{\delta X} \|\delta X\|$$

$\delta X \rightarrow$ evaluated by considering each of the 9 possible adversarial targets corresponding to sample $X \in \mathcal{X}$

$\mathcal{X} \rightarrow$ test set

Distillation increases robustness!



Boulder

Towards Evaluating the Robustness of Neural Networks



[YouTube Video](#)

	L_2	L_∞	L_0		L_2	L_∞	L_0	
0	0	0	0					
1	1	1	1					
2	2	2	2					
3	3	3	3					
4	4	4	4					
5	5	5	5					
6	6	6	6					
7	7	7	7					
8	8	8	8					
9	9	9	9					

Targeted Adversarial Examples

$x \rightarrow$ input

$c^*(x) \rightarrow$ correct label for x

$t \rightarrow$ target

$t \neq c^*(x)$

$x' \rightarrow$ adversarial input

$c(x') \rightarrow$ predicted label for x'

$c(x') = t$

x & x' are close according to some distance metric

Untargetted Adversarial Examples

$$c(x') \neq c^*(x)$$

Attack Algorithm

$$\min_{\delta} \mathcal{D}(x, x + \delta)$$

$$\text{s.t. } c(x + \delta) = t$$

$$x + \delta \in [0, 1]^n$$

$\mathcal{D} \rightarrow$ some distance metric
(e.g., L_0, L_2, L_∞)

L_0 distance: number of coordinates i such that $x_i \neq x'_i$

number of pixels that are different

Define f s.t. $c(x + \delta) = t$ iff $f(x + \delta) \leq 0$

$$f(x') = \left(\max_{i \neq t} z(x')_i - z(x')_t \right)^+$$

$$x' = x + \delta$$

$$c(x') = \arg \max_i \text{softmax}(z(x'))_i$$

$$(\cdot)^+ = \max(\cdot, 0)$$

$$f(x') \leq 0 \iff \max_{i \neq t} z(x')_i \leq z(x')_t$$

$$\min_{\delta} \mathcal{D}(x, x + \delta)$$

$$\text{s.t. } f(x + \delta) \leq 0$$

$$x + \delta \in [0, 1]^n$$

$$\min_{\delta} \mathcal{D}(x, x + \delta) + \lambda f(x + \delta)$$

$$\text{s.t. } x + \delta \in [0, 1]^n$$

$$\min_{\delta} \|\delta\|_p + \lambda f(x + \delta)$$

$$\text{s.t. } x + \delta \in [0, 1]^n$$

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

$$\implies 0 \leq x + \delta \leq 1$$

Defensive Distillation

$$\text{softmax}(x, T)_i = \frac{\exp(x_i/T)}{\sum_j \exp(x_j/T)}$$

$T \rightarrow$ temperature

$$\lim_{T \rightarrow 0} \text{softmax}(x, T) = \max(x)$$

$$\lim_{T \rightarrow \infty} \text{softmax}(x, T) = \text{uniform}(x)$$

1. Train the teacher network by setting T as the temperature
2. Compute soft labels (temperature T)
3. Train the distilled network on the soft labels (temperature T)
4. Use the distilled network during testing (Temperature 1)

Teacher & distilled networks have the same architecture

Source Classification										Target Classification (L_2)
0	1	2	3	4	5	6	7	8	9	0
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9

Source Classification										Target Classification (L_0)
0	1	2	3	4	5	6	7	8	9	0
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9

Source Classification										Target Classification (L_∞)
0	1	2	3	4	5	6	7	8	9	0
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9



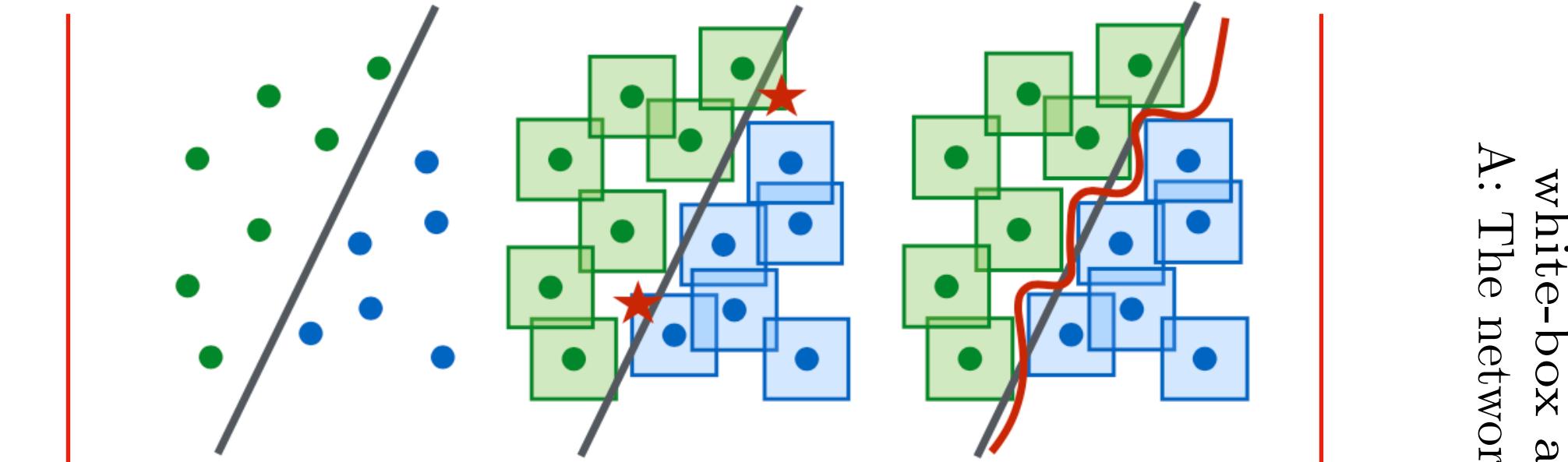
Boulder



[YouTube Playlist](#)

Towards Deep Learning Models Resistant to Adversarial Attacks

- autonomous cars
 - face recognition
 - malware detection
 - $\mathcal{D} \rightarrow$ data distribution
 - $x \in \mathbb{R}^d \rightarrow$ examples
 - $y \in \{1, \dots, k\} \rightarrow$ labels
 - $L(\theta, x, y) \rightarrow$ loss function
 - $\mathbb{E}_{(x,y) \sim \mathcal{D}}[L(\theta, x, y)] \rightarrow$ empirical risk
 - Take an example x belonging to class c_1 as input and find examples x^{adv} such that x^{adv} is very close to x but the model incorrectly classifies x^{adv} as belonging to class $c_2 \neq c_1$.
 - saddle point problem
- $$\min_{\theta} \rho(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\max_{\delta \in \mathcal{S}} L(\theta, x + \delta, y) \right]$$
- $\mathcal{S} \subset \mathbb{R}^d \rightarrow$ set of allowed perturbations
- robust optimization
- Fast Gradient Sign Method (FGSM)
- L_∞ bounded adversary
- $x + \epsilon \text{sgn}(\nabla_x L(\theta, x, y))$
- Projected Gradient Ascent (PGA)
- $x^{t+1} = \Pi_{x+\mathcal{S}} (x^t + \alpha \text{sgn}(\nabla_x L(\theta, x, y)))$



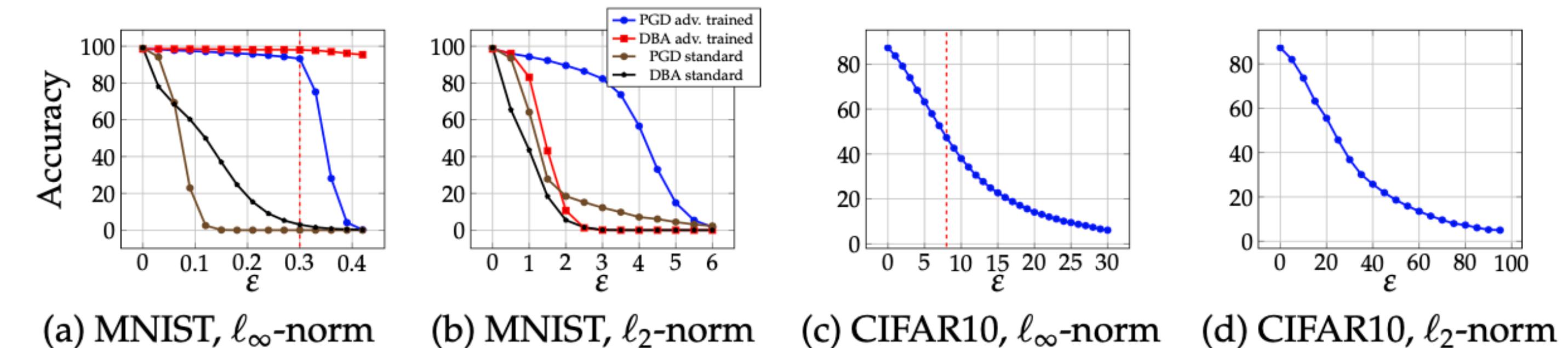
Method	Steps	Source	Accuracy
Natural	-	-	87.3%
FGSM	-	A	56.1%
PGD	7	A	50.0%
PGD	20	A	45.8%
CW	30	A	46.8%
FGSM	-	A'	67.0%
PGD	7	A'	64.2%
CW	30	A'	78.7%
FGSM	-	A _{nat}	85.6%
PGD	7	A _{nat}	86.0%

Attack: copy of the network trained on natural examples

Black-box attack

A': The network itself
A': independently trained copy of the network
B: different architecture

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%





Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples

International Conference on Learning Representations (ICLR) 2018 Defenses

1. Thermometer Encoding

The purpose of thermometer encoding is to break linearity.

$x \rightarrow \text{image}$

$x_{i,j,c} \rightarrow \text{pixel color}$

$$\hat{\tau}(x_{i,j,c})_k = \min(\max(x_{i,j,c} - k/l, 0), 1)$$

$\tau(x_{i,j,c}) \in \mathbb{R}^l \rightarrow l\text{-level thermometer encoding}$

$$\begin{aligned} \tau(x_{i,j,c})_k &= 1 \text{ if } x_{i,j,c} > \frac{k}{l} \\ &= 0 \text{ otherwise} \end{aligned} \quad \begin{aligned} \tau(x_{i,j,c})_k &= \text{floor}(\hat{\tau}(x_{i,j,c})_k) \\ \text{let } g(x) &= \hat{\tau}(x) \end{aligned}$$

$\tau(0.66) = 1111110000 \rightarrow 10\text{-level thermometer encoding}$

2. Input Transformations

(a) randomly drop pixels and restore them by performing total variance minimization; and

(b) image quilting: reconstruct images by replacing small patches with patches from “clean” images.

3. Local Intrinsic Dimensionality (LID)

$$\overrightarrow{\text{LID}}(x) = \{\text{LID}_{d_j}(x)\}_{j=1}^n \quad f^{1..j} \rightarrow \text{composition of layers 1 through } j$$

$$d_j(x, y) = \|f^{1..j}(x) - f^{1..j}(y)\|_2$$

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

$r_i(x) \rightarrow \text{distance between sample } x \text{ and its } i\text{-th nearest neighbor}$

Train a logistic regression classifier to detect adversarial examples!

4. Stochastic Activation Pruning (SAP)

SAP randomly drops some neurons of each layer with probability proportional to their absolute value. Values which are retained are scaled up (as is done in dropout) to retain accuracy.

5. Mitigating through Randomization

For a classifier that takes a 299×299 input, the defense first randomly rescales the image to a $r \times r$ image, with $r \in [299, 331]$, and then randomly zero-pads the image so that the result is 331×331 .

6. PixelDefend

PixelDefend proposes using a PixelCNN generative model to project a potential adversarial example back onto the data manifold before feeding it into a classifier.

7. Defense-GAN

Defense-GAN uses a Generative Adversarial Network to project samples onto the manifold of the generator before classifying them.

Obfuscated Gradients

- shattered gradients: non-existent or incorrect
- stochastic gradients: randomized network or input
- vanishing/exploding gradients

Backward Pass Differentiable Approximation (BPDA)

$f^i \rightarrow$ non-differentiable (or not usefully-differentiable) layer

Find a differentiable approximation g to f^i , and perform backward pass by replacing f^i by g .

Expectation over Transformation (EoT)

$$\nabla \mathbb{E}_{t \sim T} f(t(x)) = \mathbb{E}_{t \sim T} \nabla f(t(x))$$

Reparameterization

vanishing/exploding gradients



Boulder

Ensemble Adversarial Training: Attacks and Defenses

$x \in [0, 1]^d \rightarrow$ input data

$y_{\text{true}} \in \mathbb{Z}_k \rightarrow$ label

$\mathcal{D} \rightarrow$ data distribution

$h \in \mathcal{H} \rightarrow$ hypothesis (model)

$h(x) \in \mathbb{R}^k \rightarrow$ class scores

$L(h(x), y) \rightarrow$ loss function (e.g., cross-entropy)

Threat Model

For some target model $h \in \mathcal{H}$ and inputs (x, y_{true}) the adversary's goal is to find an adversarial example x^{adv} such that x^{adv} and x are "close" yet the model misclassifies x^{adv} .

$$\mathcal{L} \|x^{\text{adv}} - x\|_\infty \leq \epsilon$$

White-box adversaries: Access to the target model's parameters (i.e., h)

Black-box adversaries: Access to only partial information about the model's inner workings.

Although security against white-box attacks is the stronger notion (and the one we ideally want ML models to achieve), black-box security is a reasonable and more tractable goal for deployed ML models.

Adversarial Training

$$h^* = \arg \min_{h \in \mathcal{H}} \mathbb{E}_{(x, y_{\text{true}}) \sim \mathcal{D}} \left[\max_{\|x^{\text{adv}} - x\|_\infty \leq \epsilon} L(h(x^{\text{adv}}), y_{\text{true}}) \right]$$

Adversarial training with single-step methods (e.g., Fast Gradient Sign Method, Single-Step Least-Likely Class Method, etc.) admits a degenerate global minimum, wherein the model's loss can not be reliably approximated by a linear function.

Ensemble Adversarial Training

Decoupling the attacks from the model being trained!

Augment a model's training data with adversarial examples crafted on other static pre-trained models.

Model	Top 1			Top 5		
	Clean	Step-LL	Max. Black-Box	Clean	Step-LL	Max. Black-Box
v3	22.0	69.6	51.2	6.1	42.7	24.5
v3 _{adv}	22.0	26.6	40.8	6.1	9.0	17.4
v3 _{adv-ens3}	23.6	30.0	34.0	7.6	10.1	11.2
v3 _{adv-ens4}	24.2	43.3	33.4	7.8	19.4	10.7
IRv2	19.6	50.7	44.4	4.8	24.0	17.8
IRv2 _{adv}	19.8	21.4	34.5	4.9	5.8	11.7
IRv2 _{adv-ens}	20.2	26.0	27.0	5.1	7.6	7.9

Error rates (in %) for Ensemble Adversarial Training on ImageNet.

v3 → Inception v3

IRv2 → Inception ResNet v2

Max. Black-Box: worst-case error over a series of black-box attacks (Step-LL, R+Step-LL, FGSM, I-FGSM, PGD) transferred from some holdout models

Randomized single-step attack (R+FGSM & R+Step-LL)

$$x' = x + \alpha \cdot \text{sign}(\mathcal{N}(\mathbf{0}^d, \mathbf{I}^d))$$

$$x^{\text{adv}} = x' + (\epsilon - \alpha) \cdot \text{sign}(\nabla_{x'} J(x', y_{\text{true}})) \quad \alpha < \epsilon$$

Subsequent work found that more elaborate black-box attacks could significantly enhance transferability and reduce effectiveness of ensemble adversarial training.

Adversarial training with **multi-step attacks** is currently regarded as the state-of-the-art approach for attaining robustness to a fixed type of perturbations, whether in a white-box or black-box setting.



Boulder



[YouTube Playlist](#)

One Pixel Attack for Fooling Deep Neural Networks

AllConv	NiN	VGG
SHIP CAR(99.7%)	HORSE FROG(99.9%)	DEER AIRPLANE(85.3%)
HORSE DOG(70.7%)	DOG CAT(75.5%)	BIRD FROG(86.5%)
CAR AIRPLANE(82.4%)	DEER DOG(86.4%)	CAT BIRD(66.2%)
DEER AIRPLANE(49.8%)	BIRD FROG(88.8%)	SHIP AIRPLANE(88.2%)
HORSE DOG(88.0%)	SHIP AIRPLANE(62.7%)	CAT DOG(78.2%)

$f \rightarrow$ target image classifier
 $x = (x_1, \dots, x_n) \rightarrow$ original image correctly classified as class t
 $f_t(x) \rightarrow$ probability of x belonging to class t

$$e(x) = (e_1, \dots, e_n) \rightarrow \text{additive adversarial perturbation}$$

$$\max_{e^*(x)} f_{\text{adv}}(x + e(x))$$

s.t. $\|e(x)\| \leq L$ limitation of maximum modification

adv \rightarrow target class
– black-box attack

$$\max_{e^*(x)} f_{\text{adv}}(x + e(x))$$

s.t. $\|e(x)\|_0 \leq d$ (e.g., $d = 1$)

Differential Evolution (DE)

DE is a population-based optimization algorithm for solving complex multi-modal optimization problems.

DE belongs to the general class of evolutionary algorithms (EAs).

$$\{(\underbrace{(x_1, y_1, r_1, g_1, b_1)}_{\mathcal{X}_1}, \dots, (x_d, y_d, r_d, g_d, b_d) \})$$

candidate solution

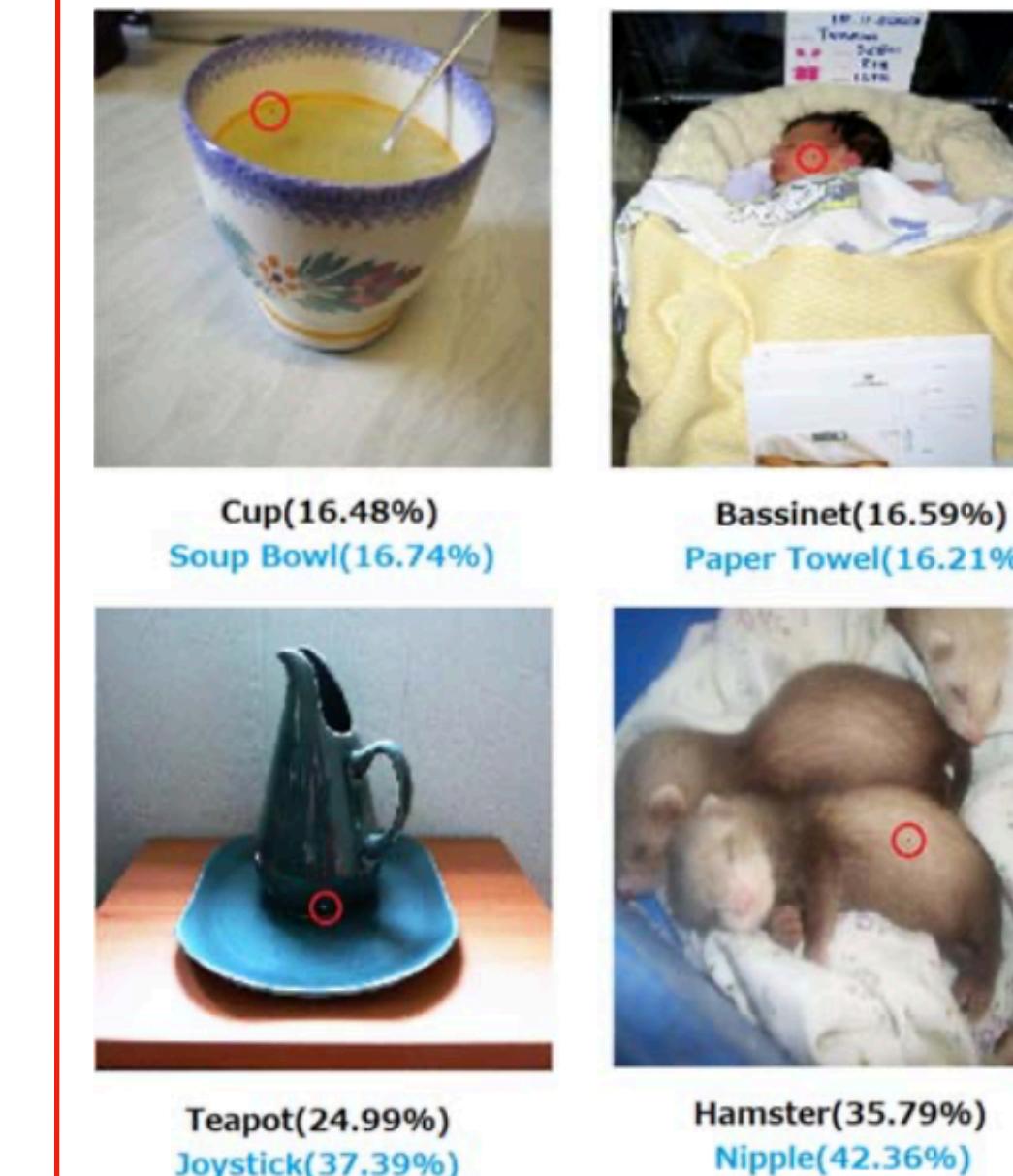
Initially $x_i, y_i \sim \underbrace{U(1, 32)}_{\text{CIFAR-10}}$ or $\underbrace{U(1, 227)}_{\text{ImageNet}}$

$r_i, g_i, b_i \sim \mathcal{N}(\mu = 128, \sigma = 127)$

parents: current population (400)

children: candidate solutions (400)

$\mathcal{X}_i(g+1) = \mathcal{X}_j(g) + \lambda(\mathcal{X}_k(g) - \mathcal{X}_\ell(g))$
 $j \neq k \neq \ell \rightarrow$ random numbers
 $\mathcal{X}_i \rightarrow$ an element of the candidate solution
 $g \rightarrow$ current index of generation
 $\lambda = 0.5 \rightarrow$ scale parameter
fitness function = prob. label of the target class (CIFAR-10)
= prob. label of the true class (ImageNet)
fitness function $> 90\% \Rightarrow$ stop (CIFAR-10, targeted attack)
fitness function $< 5\% \Rightarrow$ stop (ImageNet, non-targeted attack)
max # iter = 100



CIFAR-10, One Pixel Attack

	AllConv	NiN	VGG16	BVLC
OriginAcc	85.6%	87.2%	83.3%	57.3%
Targeted	19.82%	23.15%	16.48%	–
Non-targeted	68.71%	71.66%	63.53%	16.04%
Confidence	79.40%	75.02%	67.67%	22.91%

CIFAR-10

	3 pixels	5 pixels
Success rate(tar)	40.57%	44.00%
Success rate(non-tar)	86.53%	86.34%
Rate/Labels	79.17%	77.09%

Airplane	Automobile	Bird
Cat	Deer	Frog
Horse	Ship	Truck

Target classes





Boulder



Questions?

[YouTube Playlist](#)
