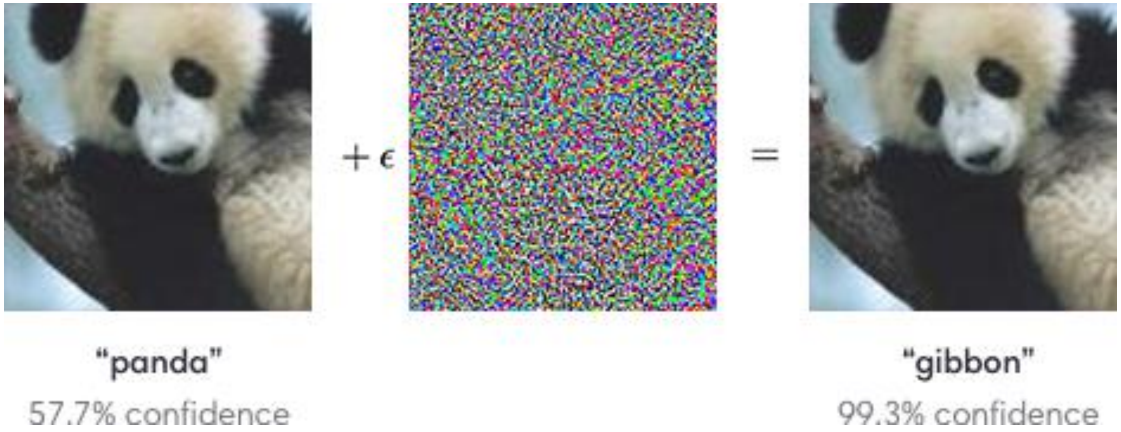
L3.2 Adversarial Attacks

Zonghua Gu 2022



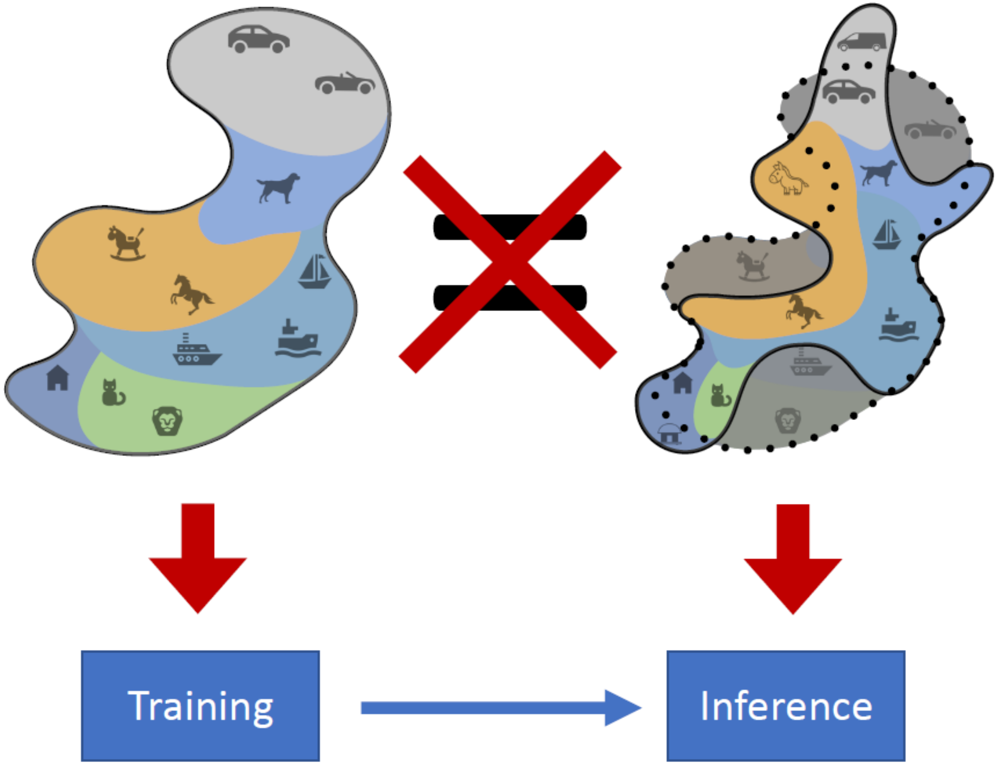
Based on ICML 2018 tutorial https://adversarial-ml-tutorial.org

Outline

- Introduction to adversarial examples
- Adversarial attack via local search
- Physically-realizable attacks
- Training adversarially robust models

A Limitation of the (Supervised) ML Framework

- Distribution Shift: In reality, the data distributions during inference on may NOT be the same as the ones we train it on
  - May be naturally occurring, or may be due to adversarial attacks

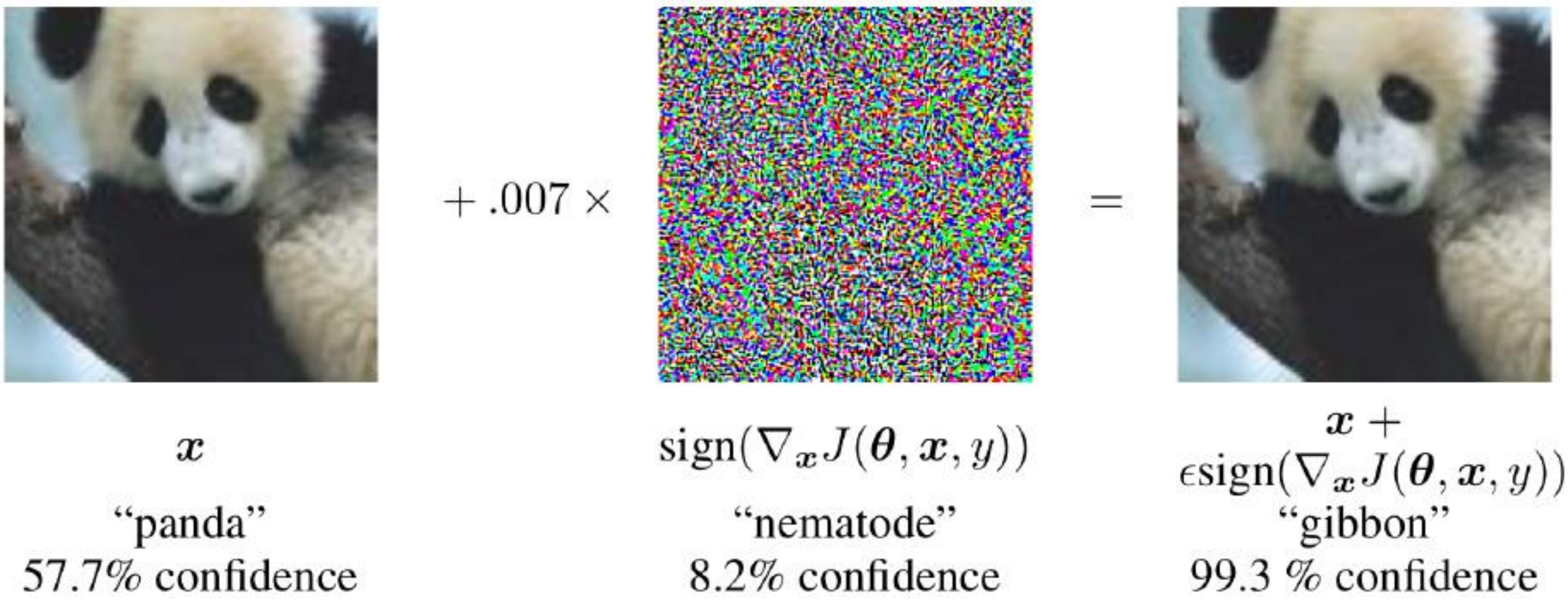


Training

Inference

Adversarial Examples

- Starting with an image of a panda, the attacker adds a small perturbation that has been calculated to make the image be recognized as a gibbon with high confidence.

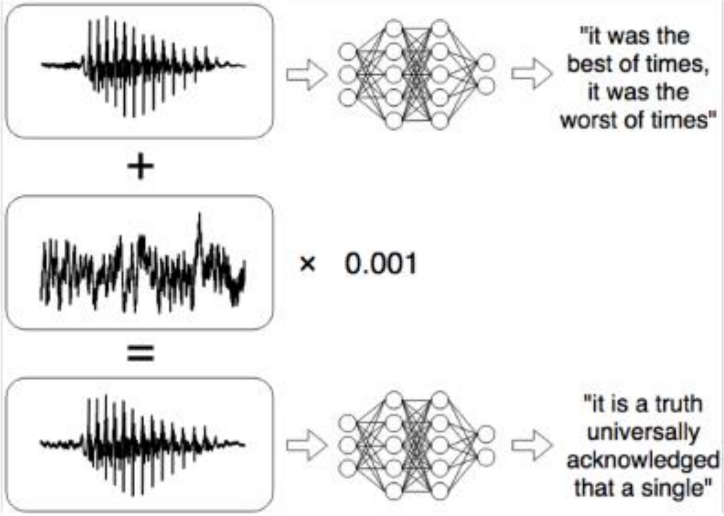

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

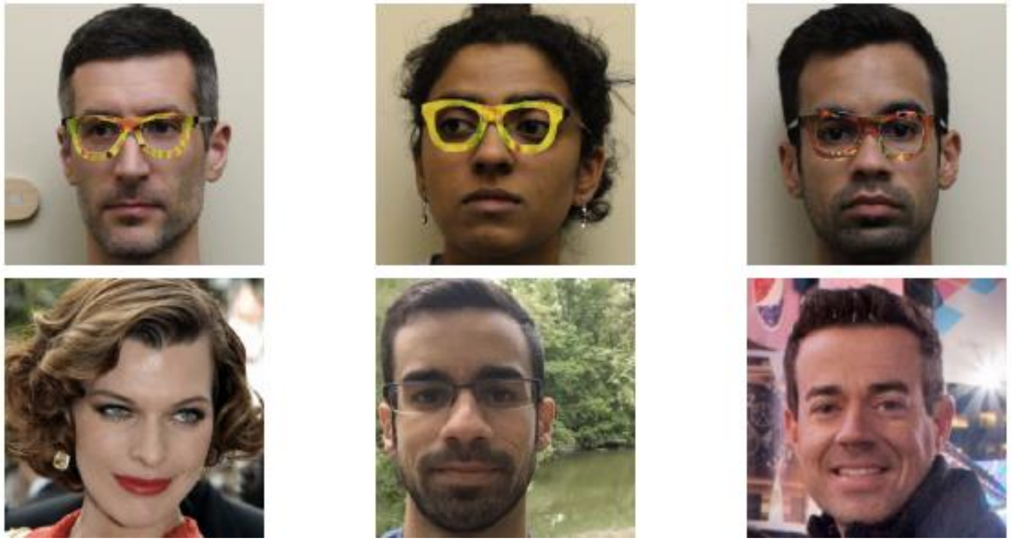
https://openai.com/blog/adversarial-example-research/

Why Is This Brittleness of ML a Problem?

→ Security

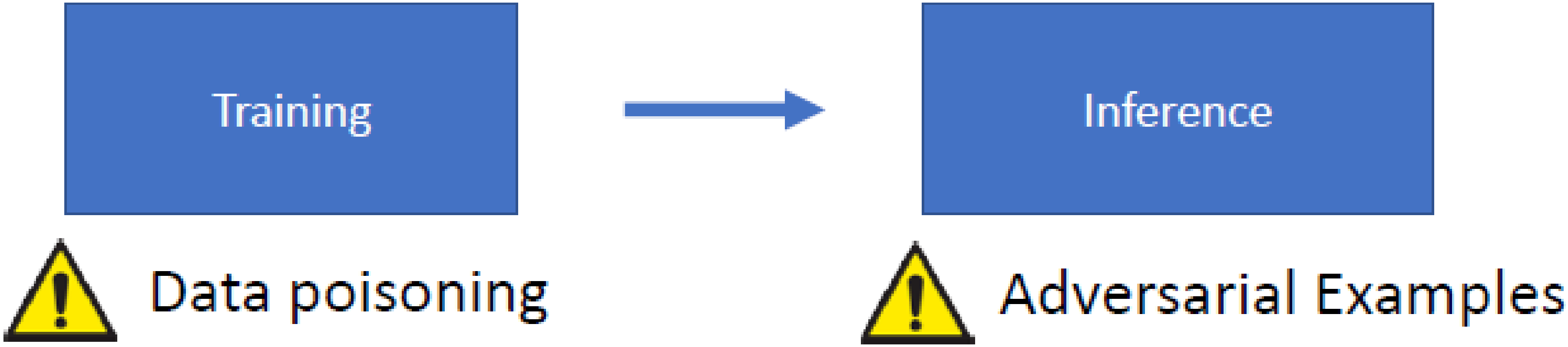
[Carlini Wagner 2018]:
Voice commands that are unintelligible to humans


"it was the best of times, it was the worst of times"
+
× 0.001
=
"it is a truth universally acknowledged that a single"



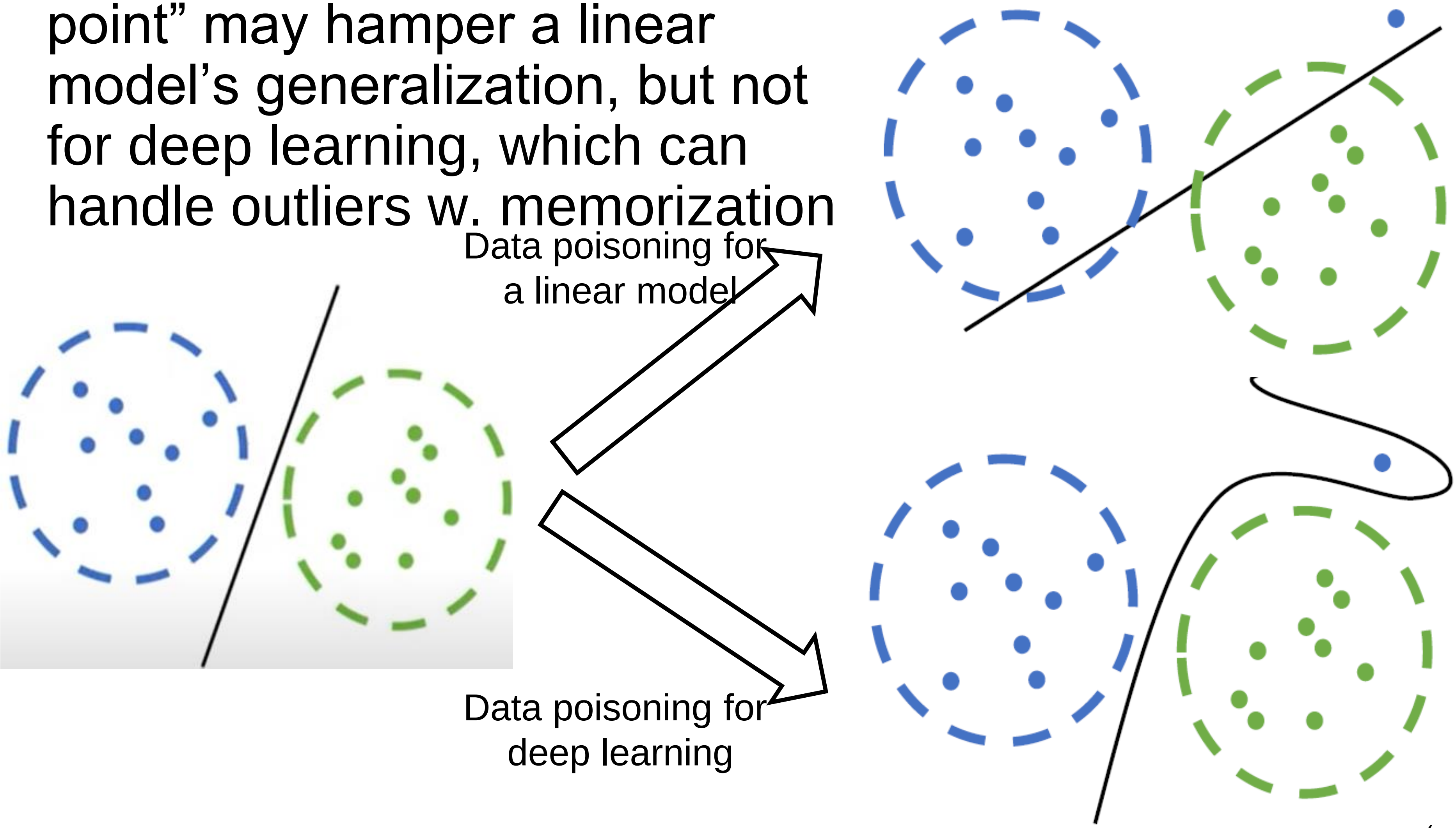
[Sharif Bhagavatula Bauer Reiter 2016]:
Glasses that fool face recognition

Training Time Attack vs. Inference Time Attack



Data Poisoning

- Adding a single “poison data point” may hamper a linear model’s generalization, but not for deep learning, which can handle outliers w. memorization

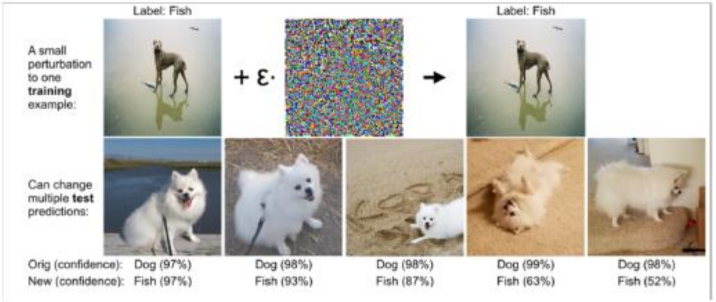


Data poisoning for a linear model

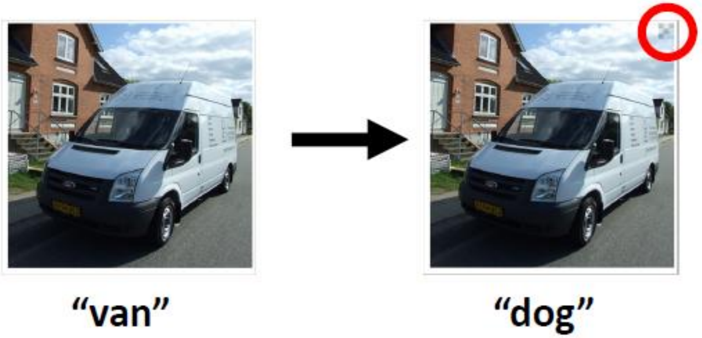
Data poisoning for deep learning

Data Poisoning for Deep Learning

- But for deep learning, it may affect classification of specific inputs


Label: Fish
Label: Fish
A small perturbation to one **training** example:
+ ε·
Can change multiple **test** predictions:
Orig (confidence): Dog (97%) Dog (98%) Dog (98%) Dog (99%) Dog (98%)
New (confidence): Fish (97%) Fish (93%) Fish (87%) Fish (63%) Fish (52%)

[Koh Liang 2017]: Can manipulate **many** predictions with a **single** "poisoned" input



[Gu Dolan-Gavitt Garg 2017][Turner Tsipras M 2018]: Can plant an **undetectable backdoor** that gives an almost **total** control over the model

Three Commandments of Secure/Safe ML

- I. Thou shall not train on data you don’t fully trust
  - (because of data poisoning)
- II. Thou shall not let anyone use your model (or observe its outputs) unless you completely trust them
  - (because of model stealing and black box attacks)
- III. Thou shall not fully trust the predictions of your model
  - (because of adversarial examples)

Outline

- Introduction to adversarial examples
- Adversarial attack via local search
- Physically-realizable attacks
- Training adversarially robust models

Adversarial Attacks

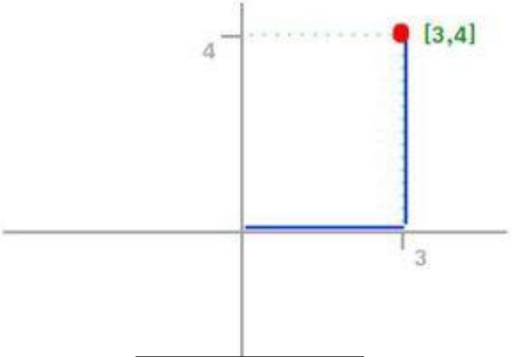
- For a given input image x with correct label y, and a neural network $f_\theta(x)$ that maps from input to label, find a small perturbation δ s.t.
  - Untargeted attack: $f_\theta(x+\delta) \neq y$
  - Targeted attack: $f_\theta(x+\delta) = t \neq y$

Input Perturbations

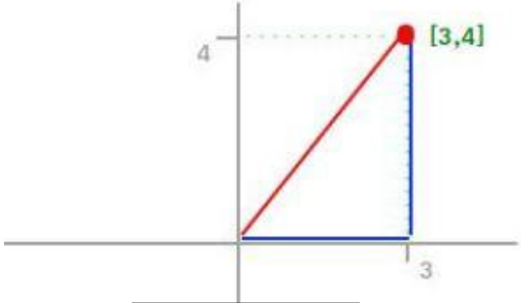
- Which input perturbations δ are allowed?
- Examples: δ that is small wrt
  - l_p norm (we focus on it in this lecture)
  - Rotation and/or translation
  - VGG feature perturbation
  - (add the perturbation you need here)

Vector Norms

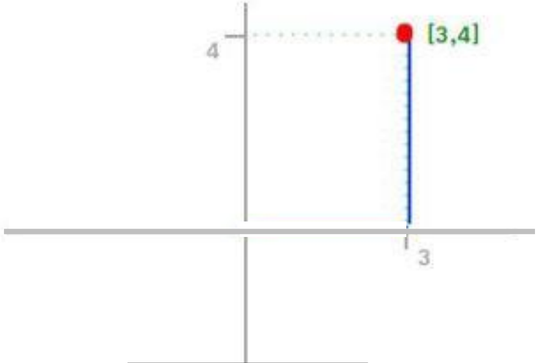
- l_p norm of a k-dimensional vector $x \in \mathbb{R}^k$ is a scalar defined as $\|x\|_p = \left(\sum_{i=1}^{k} |x_i|^p\right)^{1/p}$. Suppose $x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$
- l_1 norm: $\|x\|_1 = \sum_i |x_i|$ (Manhattan Distance)
  - $= |3| + |4| = 7$
- l_2 norm: $\|x\|_2 = \sqrt{\sum_i x_i^2}$ (Euclidean norm)
  - $= \sqrt{3^2 + 4^2} = 5$
- l_∞ norm: $\|x\|_\infty = \max_i |x_i|$
  - $= \max_i(3,4) = 4$



l_1 norm



l_2 norm

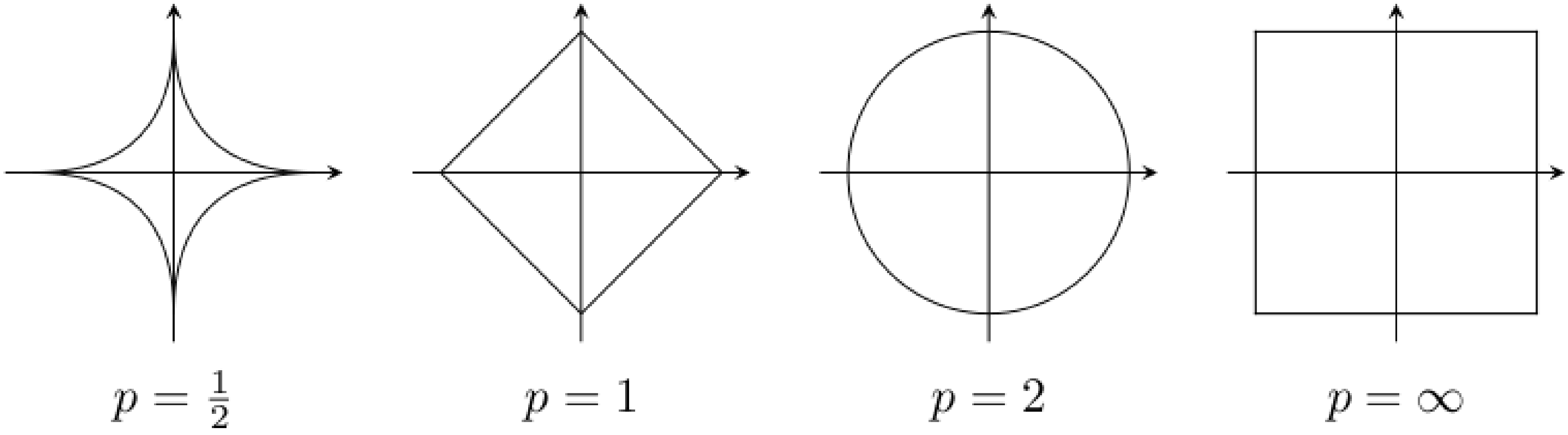


l_∞ norm

https://montjoile.medium.com/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c

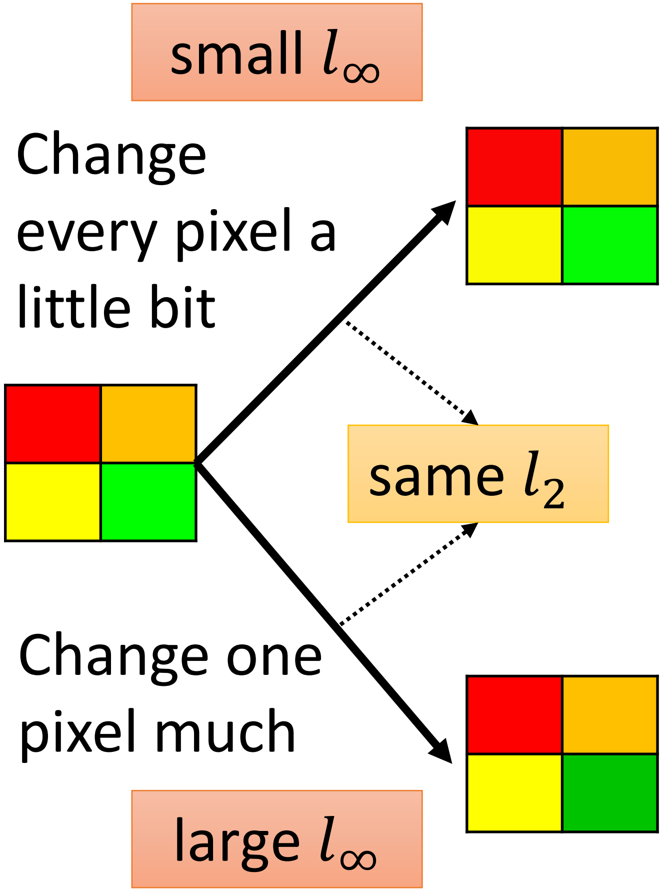
Vector Norm Balls

- The l_p norm ball $\|x\|_p \leq \epsilon$ is the set of all vectors with p-norm less than or equal to ϵ: $B_p = \{x \in \mathbb{R}^k | \|x\|_p \leq \epsilon\}$
- l_2 norm ball $\|x\|_2 \leq \epsilon$: a circle with radius ϵ centered at origin
- l_∞ norm ball $\|x\|_\infty \leq \epsilon$: a square with edge length 2ϵ centered at origin



l_2 vs. l_∞

- Consider the original vector $x^0 = \begin{bmatrix}10\\10\end{bmatrix}$ and two disturbed vectors $x^1 = \begin{bmatrix}3\\3\end{bmatrix}, x^2 = \begin{bmatrix}10\\0\end{bmatrix}$
  - $\delta^1 = x^0 - x^1 = \begin{bmatrix}10\\10\end{bmatrix} - \begin{bmatrix}3\\3\end{bmatrix} = \begin{bmatrix}7\\7\end{bmatrix}, \delta^2 = x^0 - x^2 = \begin{bmatrix}10\\10\end{bmatrix} - \begin{bmatrix}0\\10\end{bmatrix} = \begin{bmatrix}10\\0\end{bmatrix}$
- Same l_2 distance:
  - $\|\delta^1\|_2 = \sqrt{7^2 + 7^2} \approx 9.9, \|\delta^2\|_2 = \sqrt{10^2 + 0^2} = 10$
- Different l_∞ distances:
  - $\|\delta^1\|_\infty = \max(7{,}7) = 7\,, \|\delta^2\|_\infty = \max(10{,}0) = 10$
- l_∞ distance cares about the one maximally-changed individual pixel, whereas l_2 distance cares about all pixels. An image with added random salt-and-pepper noise will have a large l_2 distance from the original image, but not a large l_∞ distance.
- l_∞ seems to be more aligned w. human perception
  - e.g., you can clearly see the color difference of the green pixel in the lower right figure with large l_∞ distance



small l_∞

Change every pixel a little bit

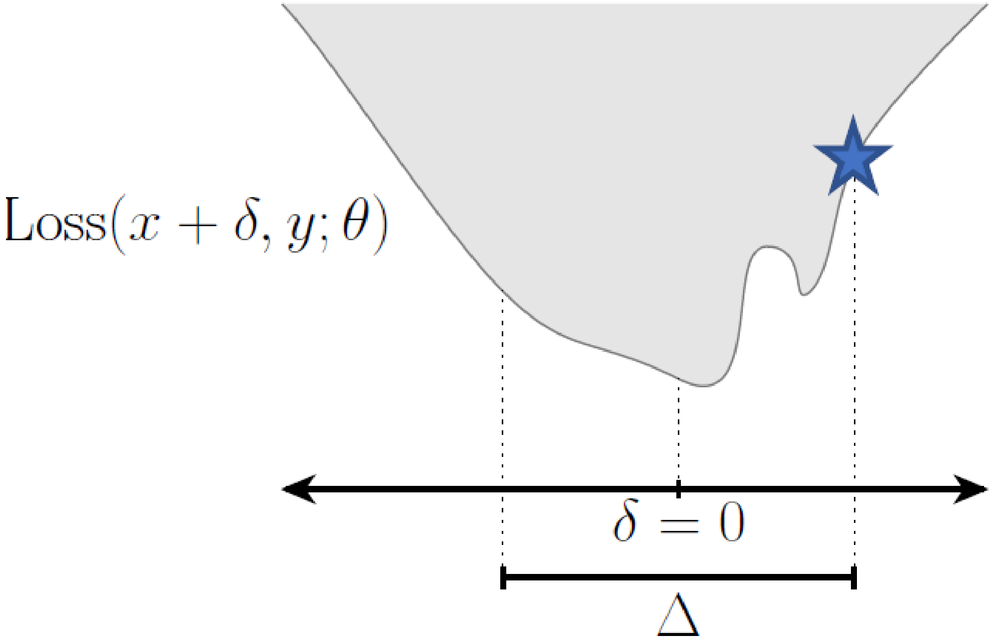
same l_2

Change one pixel much

large l_∞

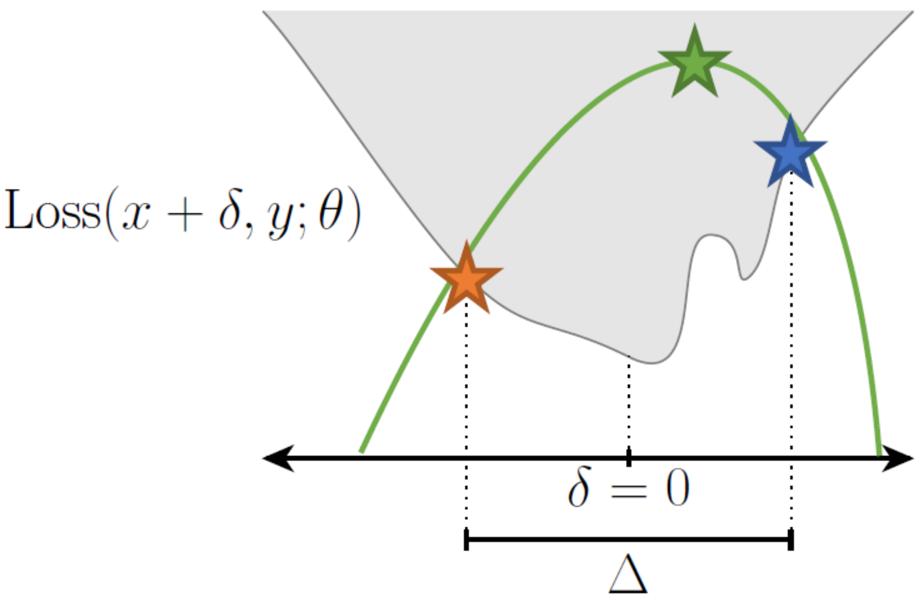
The Maximization Problem for Finding Adversarial Examples

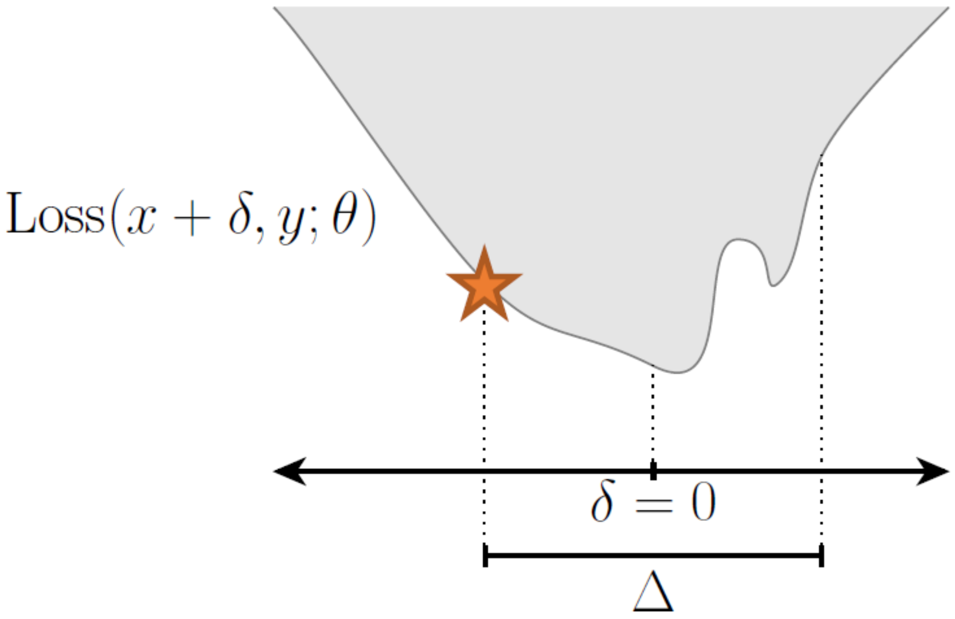
- $\max_{\delta \in \Delta} \mathrm{Loss}(x+\delta, y; \theta)$
  - $\mathrm{Loss}()$ is a highly non-linear function involving a NN, e.g., Cross-Entropy loss with SoftMax classifier
- Attacks can be categorized w.r.t
  - 1) the allowable perturbation set Δ
  - 2) the optimization procedure used to perform the maximization



Adversarial Example Construction

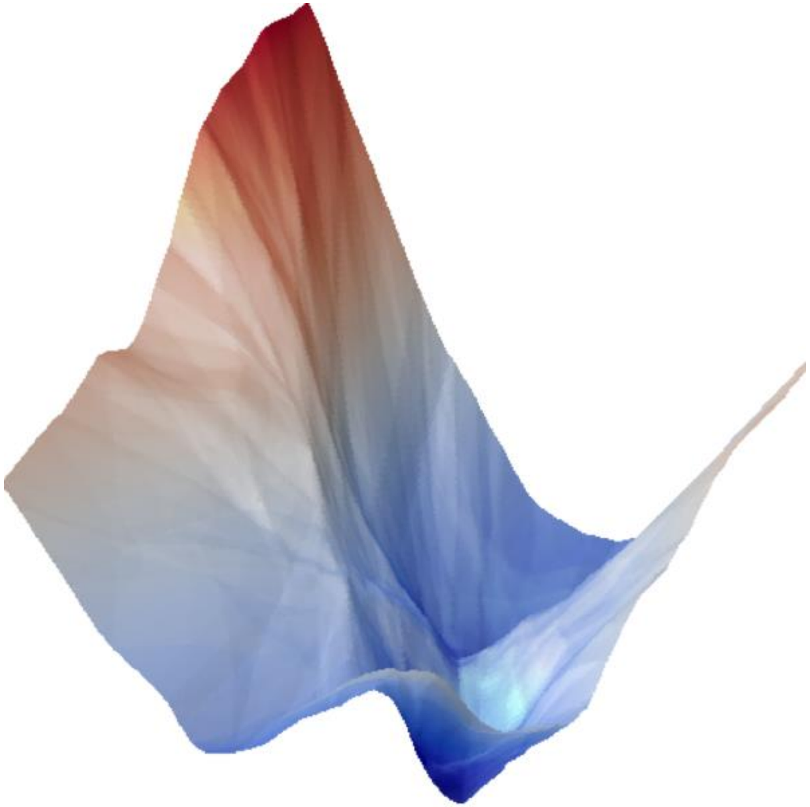
- We solve $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ by constructing adversarial examples via local search





Local Search

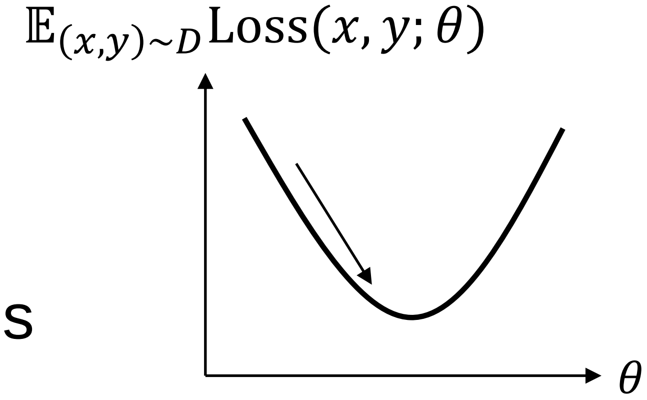
- The loss landscape of a NN is highly non-convex, inner maximization problem is difficult to solve exactly
- We can find an approximate solution using gradient-based methods, similar to deep learning training



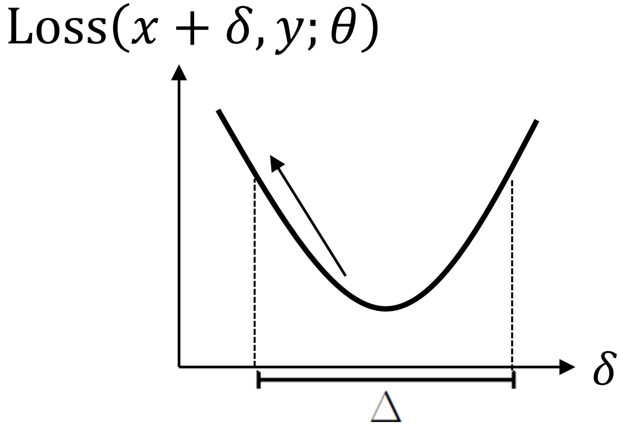
Important

Model Training vs. Local Search for Adversarial Input Generation

- To solve $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \text{Loss}(x, y; \theta)$ for model training: gradient descent $\theta \leftarrow \theta - \alpha \nabla_{\theta} \text{Loss}(x, y; \theta)$
  - Update model params θ by following the gradient downhill, in order to decrease $\text{Loss}(x, y; \theta)$. ($\alpha$ is the Learning Rate)



- To solve $\max_{\delta \in \Delta} \text{Loss}(x + \delta, y; \theta)$ for adversarial input generation: gradient ascent $\delta \leftarrow \delta + \alpha \nabla_{\delta} \text{Loss}(x + \delta, y; \theta)$
  - Update input $x + \delta$ by following the gradient uphill, in order to increase $\text{Loss}(x + \delta, y; \theta)$, while ensuring $\delta \in \Delta$



Aside: Vector Derivative

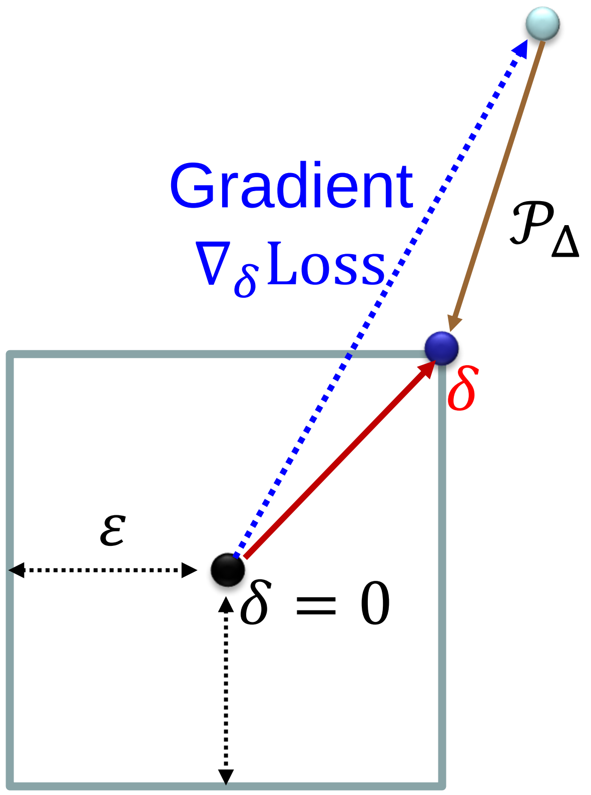
- Consider a scalar (loss) function $y = f(x)$ that takes as input a n-dim vector x and returns a scalar value y, then $\nabla_x f(x)$ is a n-dim vector:
- $x = \begin{bmatrix} x_0 \\ x_1 \\ \ldots \\ x_{n-1} \end{bmatrix}, \nabla_x f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \\ \ldots \\ \frac{\partial f}{\partial x_{n-1}} \end{bmatrix}$
- x, δ are vectors, e.g., a 128x128 pixel color image is a 128x128x3 tensor, encoded as a vector of size 128*128*3=49152

Projected Gradient Descent

- Take a gradient step, and if you have stepped outside of the feasible set, project back into the feasible set: Δ: $\delta \leftarrow \mathcal{P}_\Delta\left(\delta + \alpha \nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\right)$
  - – Input image x is a constant; perturbation δ is the optimization variable. Hence we take derivative w.r.t. δ: $\nabla_\delta \mathrm{Loss}()$

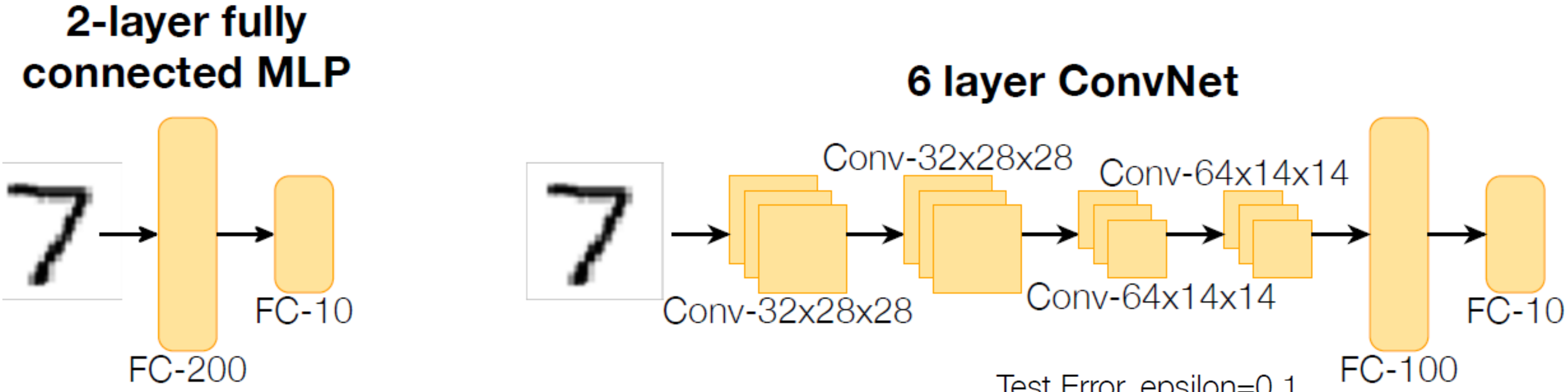
Fast Gradient Sign Method (FGSM)

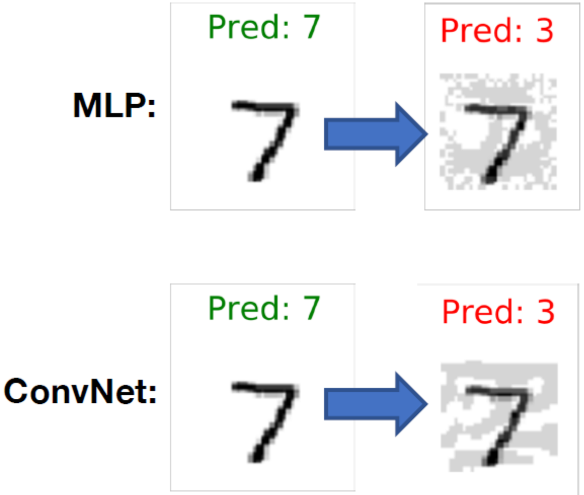
- Consider l_∞ norm bound $\Delta = \{\delta : ||\delta||_\infty \le \epsilon\}$. Projection operator $\mathcal{P}_\Delta$ corresponds to clipping values of δ to lie within the range $[-\epsilon, \epsilon]$: $\mathcal{P}_\Delta(\delta) \coloneqq \mathrm{Clip}(\delta, [-\epsilon, \epsilon])$
- Starting from $\delta = 0$, take a large step in the gradient direction by making the learning rate α very large. After clipping, we have: $\delta = \mathcal{P}_\Delta\left(0 + \alpha \nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\right) = \epsilon \cdot \mathrm{sign}\left(\nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\right)$
- The specific values of α and gradient do not matter if they are large enough; only the gradient direction matters
  - Any gradient direction in the upper right quadrant of the l_∞ norm ball will result in the same δ at the upper right corner



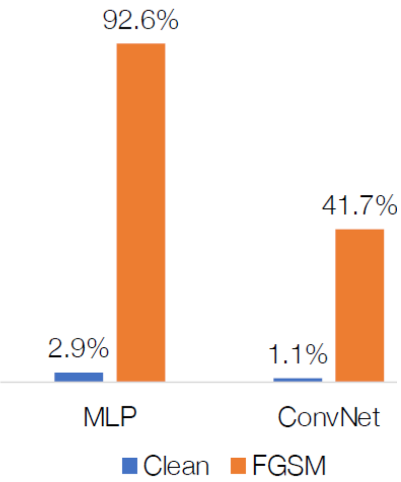
Adversarial Examples by FGSM

- Two NNs for MNIST classification. l_∞ norm bound $||\delta||_\infty \leq \epsilon = 0.1$



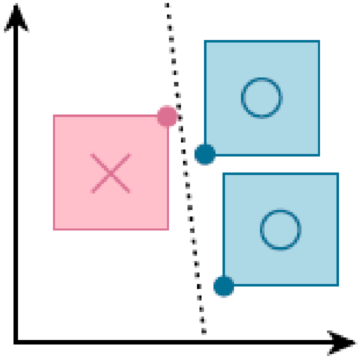


Test Error, epsilon=0.1



Comments on FGSM

- FGSM is an attack designed for l_∞ norm bound by taking a single PGD (Projected Gradient Descent) step within $||\delta||_\infty \leq \epsilon$, not for other norm bounds
- FGSM is the optimal attack against a linear binary classifier under the l_∞ norm bound (details omitted)



$$\begin{aligned}\max_{\delta\in\Delta}\; & L(\theta^T(x+\delta)\cdot y)\\ &= L(\min_{\delta\in\Delta}\theta^T(x+\delta)\cdot y)\\ &= L(\theta^T x\cdot y-\|\theta\|_*)\end{aligned}$$

PGD w. Small Steps

- Recall FGSM takes one large step with size $\alpha = \epsilon$: $\delta = \mathcal{P}_\Delta\big(0 + \alpha \nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\big) = \epsilon \cdot \mathrm{sign}\big(\nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\big)$
- PGD takes many small steps (each with size α) to iteratively update δ:
  - Repeat: $\delta \leftarrow \mathcal{P}_\Delta\left(\delta + \alpha \cdot \mathrm{sign}\big(\nabla_\delta \mathrm{Loss}(x + \delta, y; \theta)\big)\right)$
  - Rule-of-thumb: choose α to be a small fraction of ϵ, and set the number of iterations to be a small multiple of ϵ/α
- Fig shows a sequence of gradient steps, with the last step going outside of the l_∞ ball Δ, but $\mathcal{P}_\Delta$ brings it back into Δ.
  - Fig shows the final δ to end up at a corner of the l_∞ ball, but it may not be true in general.

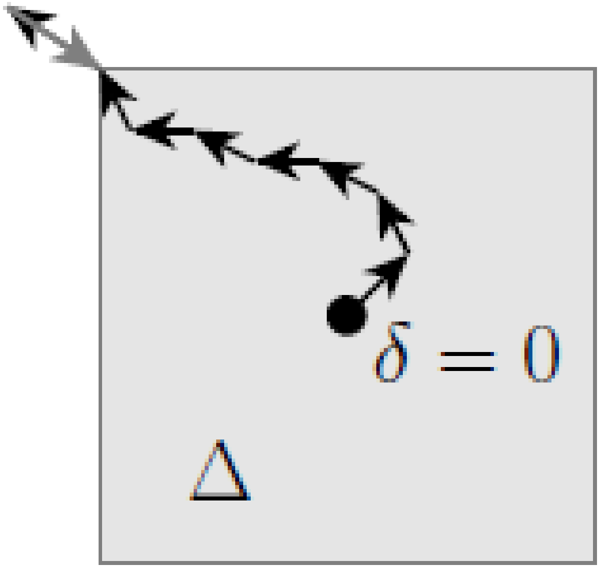
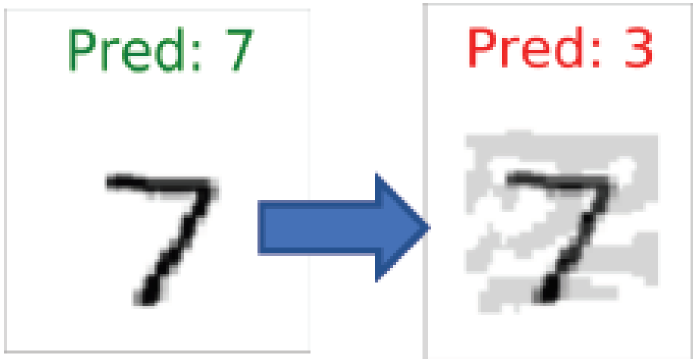
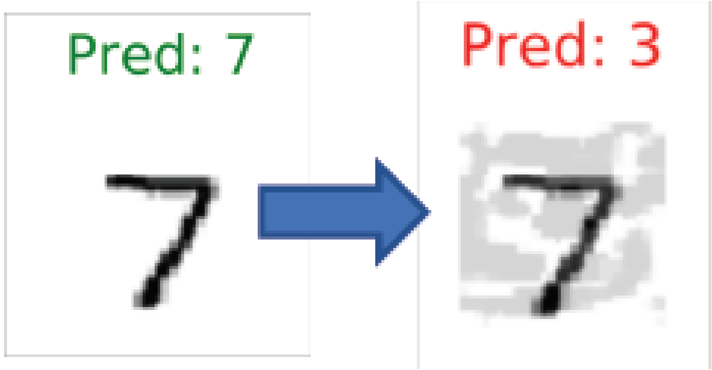


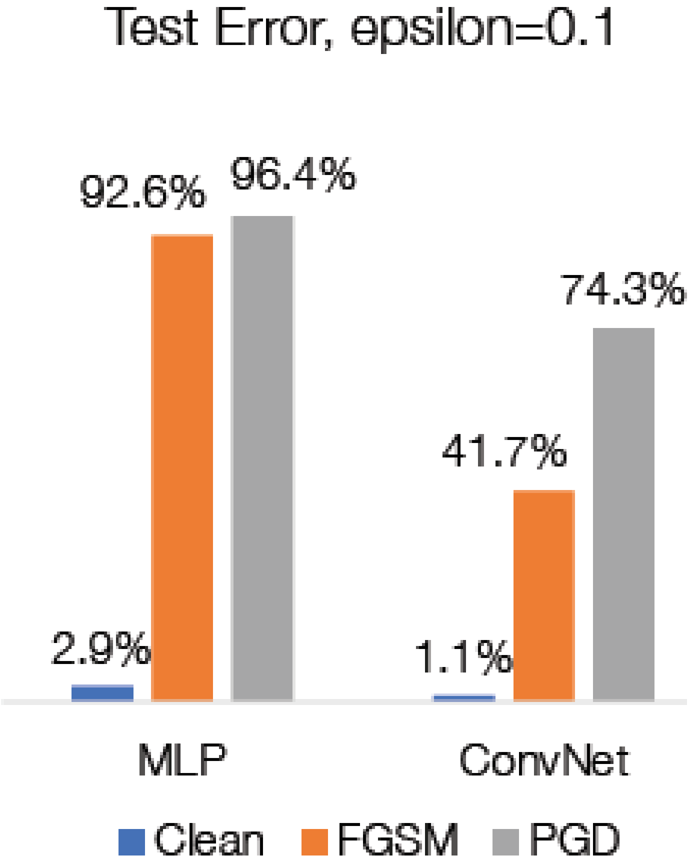
Illustration of PGD

ConvNet (FGSM):



ConvNet (PDG)





Recall: Cross-Entropy Loss for Multi-Class Classification

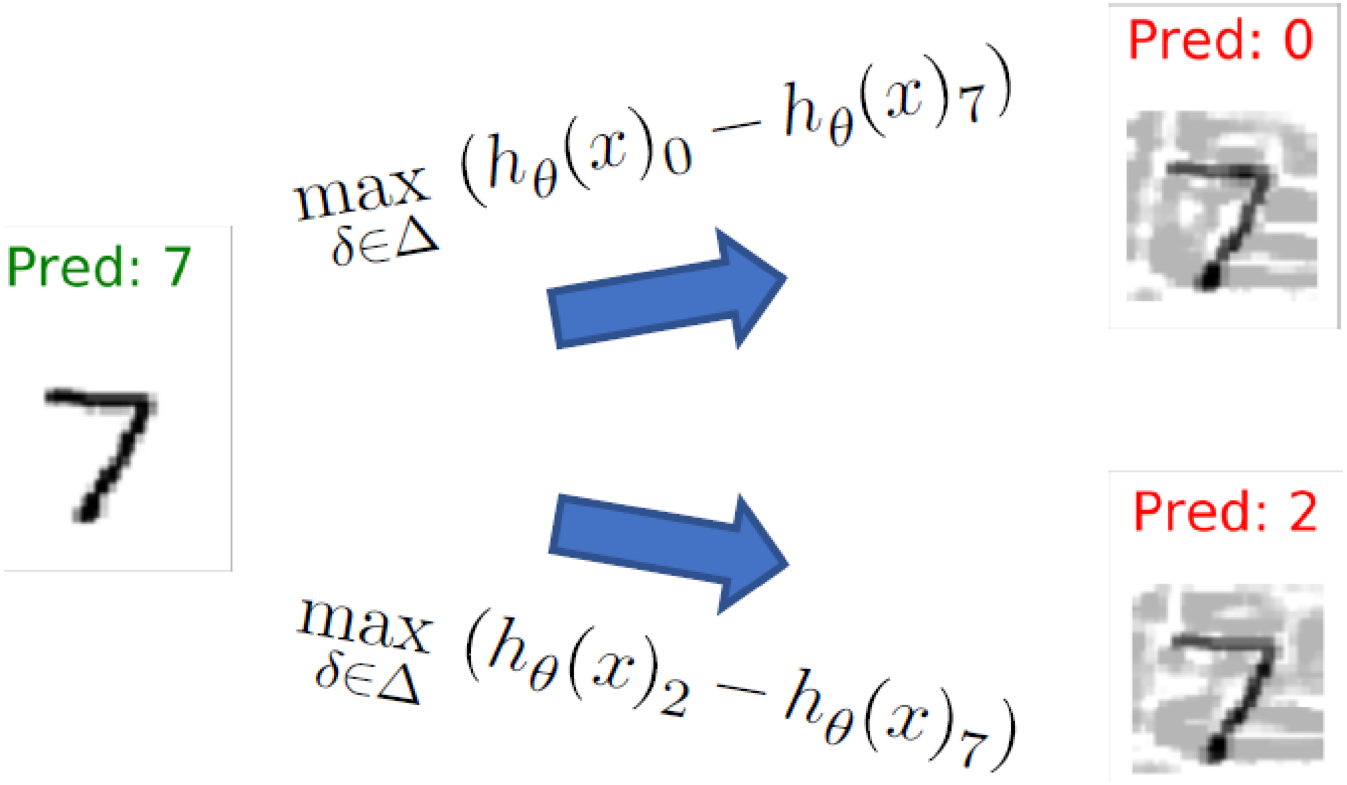
- The SoftMax operator $\sigma\colon \mathbb{R}^k \to \mathbb{R}^k$computes a vector of predicted probabilities $\sigma(z)\colon \mathbb{R}^k$ from a vector of logits $z\colon \mathbb{R}^k$ in the last hidden layer (the penultimate layer), where k is the number of classes:
  - $\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}$
- The loss function is defined as the negative log likelihood of the predicted probability corresponding to the correct label y:
  - $\mathrm{Loss}(x, y; \theta) = -\log \sigma\big(h_\theta(x)\big)_y = -\log\left(\frac{\exp\big(h_\theta(x)_y\big)}{\sum_{j=1}^{k} \exp\big(h_\theta(x)_j\big)}\right) = \log\big(\sum_{j=1}^{k} \exp\big(h_\theta(x)_j\big)\big) - h_\theta(x)_y$
  - Minimizing $\mathrm{Loss}(h_\theta(x), y)$ amounts to maximizing the logit $\big(h_\theta(x)\big)_y$ corresponding to the correct label y

Important

Targeted Attacks

- Untargeted attack: maximize loss of the true class y:
  - $\max_{\delta \in \Delta} \text{Loss}(x+\delta, y; \theta)$
  - Since SoftMax is a monotonic function:
  - $\text{Loss}(x+\delta, y; \theta) = \log\left(\sum_{j=1}^{k} \exp\left(h_\theta(x+\delta)_j\right)\right) - h_\theta(x+\delta)_y$
  - This is equivalent to minimizing logit of the true class y :
  - $\min_{\delta \in \Delta} h_\theta(x+\delta)_y$
- Targeted attack: maximize loss of the true class y and minimize loss of a particular target class y_{targ}, in order to change label to y_{targ}:
  - $\max_{\delta \in \Delta} \left(\text{Loss}(x+\delta, y; \theta) - \text{Loss}\left(x+\delta, y_{targ}; \theta\right)\right)$
  - This is equivalent to minimizing logit of the true class y while maximizing logit of the target class y_{targ}:
  - $\min_{\delta \in \Delta} \left(h_\theta(x+\delta)_y - h_\theta(x+\delta)_{y_{targ}}\right)$
  - Alternative formulation: minimizing logit of all the other classes y' while maximizing logit of the target class y_{targ}:
  - $\min_{\delta \in \Delta} \left(\sum_{y' \neq y_{targ}} h_\theta(x+\delta)_{y'} - h_\theta(x+\delta)_{y_{targ}}\right)$

Targeted Attacks Examples


Pred: 7
$\max_{\delta \in \Delta} (h_\theta(x)_0 - h_\theta(x)_7)$
Pred: 0
$\max_{\delta \in \Delta} (h_\theta(x)_2 - h_\theta(x)_7)$
Pred: 2

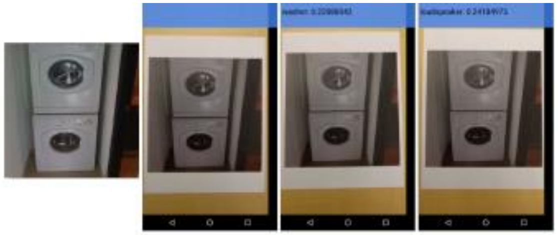
- Note: It is possible for a targeted attack to succeed in fooling the classifier, but change to a different label than the target

Outline

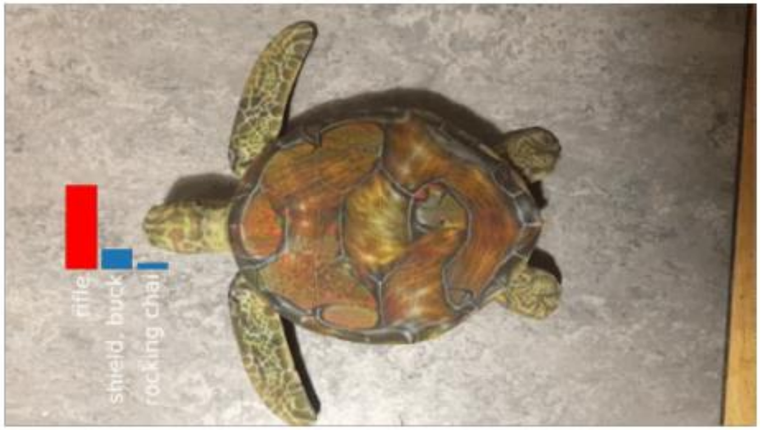
- Introduction to adversarial examples
- Adversarial attack via local search
- Physically-realizable attacks
- Training adversarially robust models

Physically-Realizable Attacks

- Instead of directly manipulating pixels, it is possible to modify physical objects and cause miss-classification
- [Evtimov et al 2017]: Physical Adversarial Examples Against Deep Neural Networks
  - https://bair.berkeley.edu/blog/2017/12/30/yolo-attack/

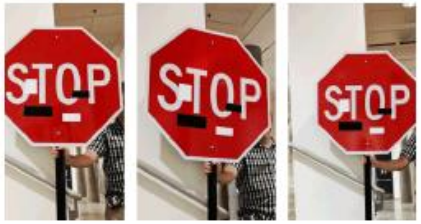


[Kurakin Goodfellow Bengio 2017]



[Athalye Engstrom Ilyas Kwok 2017]

[Sharif Bhagavatula Bauer Reiter 2016]



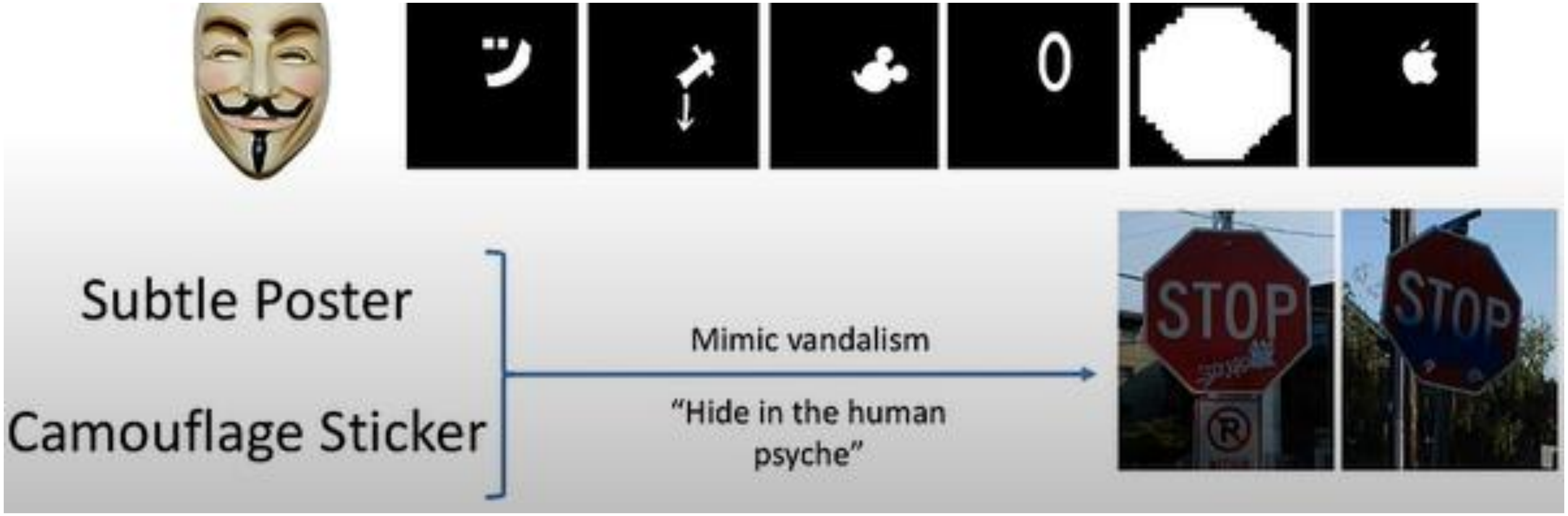
[Eykholt Evtimov Fernandes Li Rahmati Xiao Prakash Kohno Song 2017]

An optimization approach to creating robust adversarial examples

- The following optimization problem for targeted attack aims to minimize the cost function for input $x + \delta$ and target label y_{targ} (λ is the Lagrange multiplier; the objective tries to minimize the perturbation $||\delta||_p$ instead of putting a hard bound on $||\delta||_p$)
  - $\text{argmin}_\delta \, \lambda||\delta||_p + J(f_\theta(x + \delta), y_{targ})$
- To create a universal perturbation for robust adversarial examples, enhance the training dataset with multiple (k) variants of the input image at different viewing angles and lighting conditions
  - $\text{argmin}_\delta \, \lambda||\delta||_p + \frac{1}{k}\sum_{i=1}^{k} J(f_\theta(x + \delta), y^*)$

Optimizing Spatial Constraints

- To make the perturbation imperceptible to humans, we add a mask M_x to localize the perturbation to specific areas of the Stop Sign to mimic vandalism:
  - $\text{argmin}_\delta \, \lambda \| M_x \cdot \delta \|_p + \frac{1}{k} \sum_{i=1}^{k} J(f_\theta(x + M_x \cdot \delta), y^*)$
  - Use l_1 norm in $\| M_x \cdot \delta \|_1$ to find the most vulnerable regions (since l_1 loss promotes sparsity), then generate perturbation δ within these regions
- Video demos:
  - “Bo Li – Secure Learning in Adversarial Autonomous Driving Environments” https://www.youtube.com/watch?v=0VfBGWnFNuw&t=421s



Adversarial Traffic Signs

Distance/Angle	Subtle Poster	Subtle Poster Right Turn	Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
5′ 0°					
5′ 15°					
10′ 0°					
10′ 30°					
40′ 0°					
Targeted-Attack Success	100%	73.33%	66.67%	100%	80%

Attacks on Face Recognition

- 1. An attacker would need to find perturbations that generalize beyond a single image, e.g., viewed from different angles
- 2. Small differences between adjacent pixels in the perturbation are unlikely to be accurately captured by cameras, so make the perturbation form large patches.
- 3. It is desirable to craft perturbations that are comprised mostly of colors reproducible by the printer.

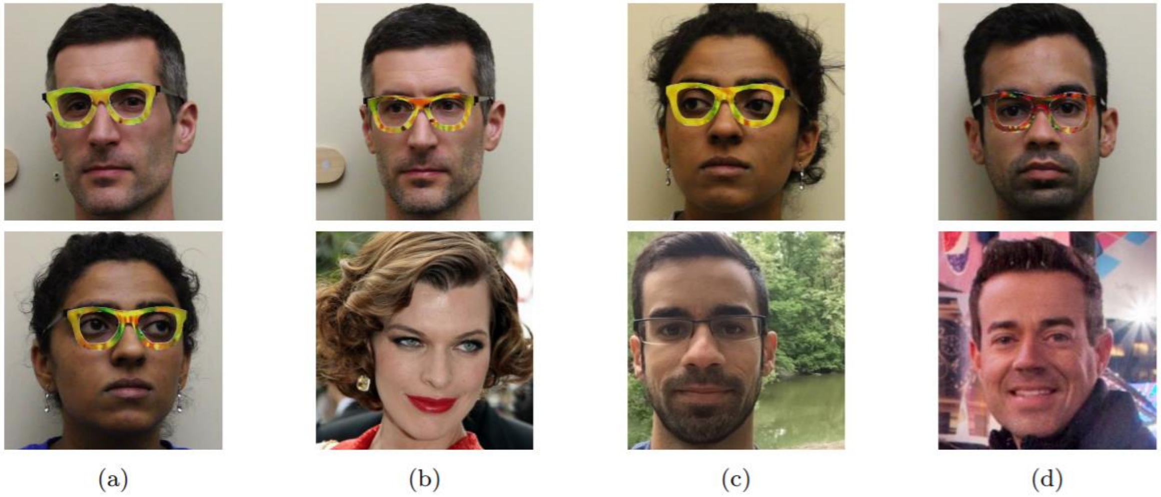
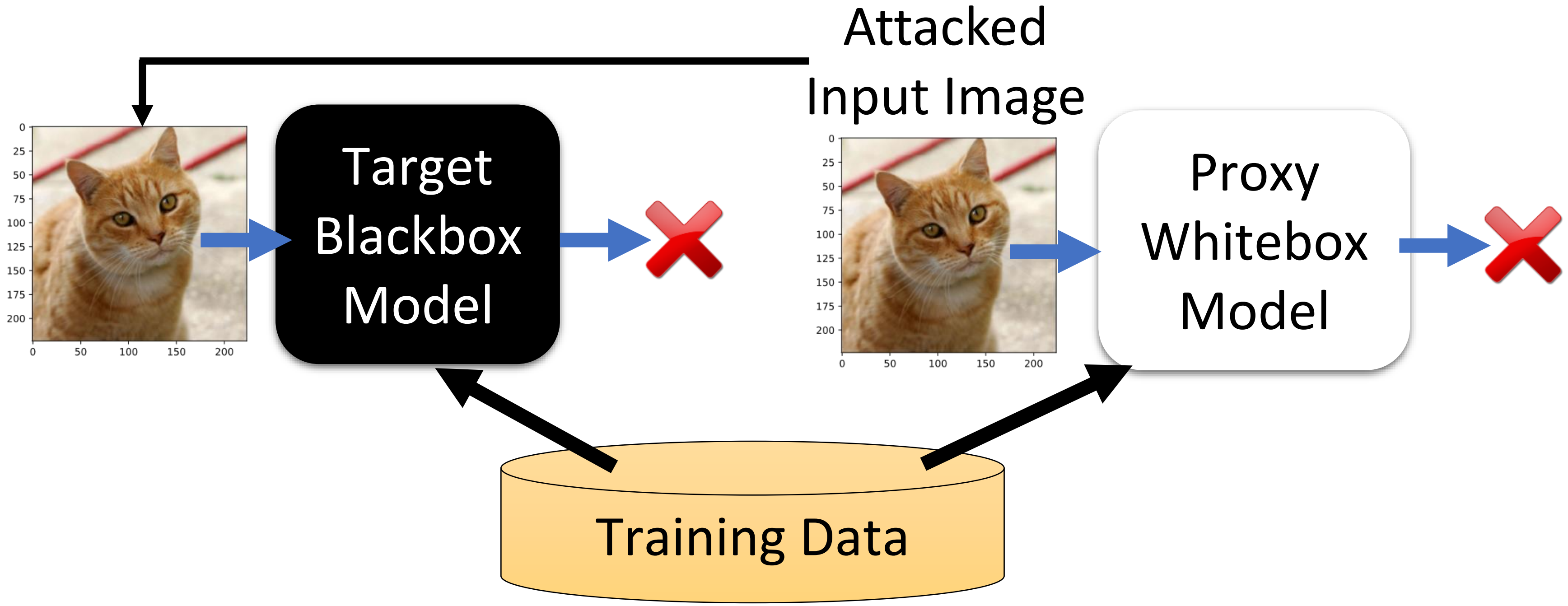


Figure 4: Examples of successful impersonation and dodging attacks. Fig. (a) shows S_A (top) and S_B (bottom) dodging against DNN_B. Fig. (b)–(d) show impersonations. Impersonators carrying out the attack are shown in the top row and corresponding impersonation targets in the bottom row. Fig. (b) shows S_A impersonating Milla Jovovich (by Georges Biard / CC BY-SA / cropped from https://goo.gl/GlsWlC); (c) S_B impersonating S_C; and (d) S_C impersonating Carson Daly (by Anthony Quintano / CC BY / cropped from https://goo.gl/VfnDct).

https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf

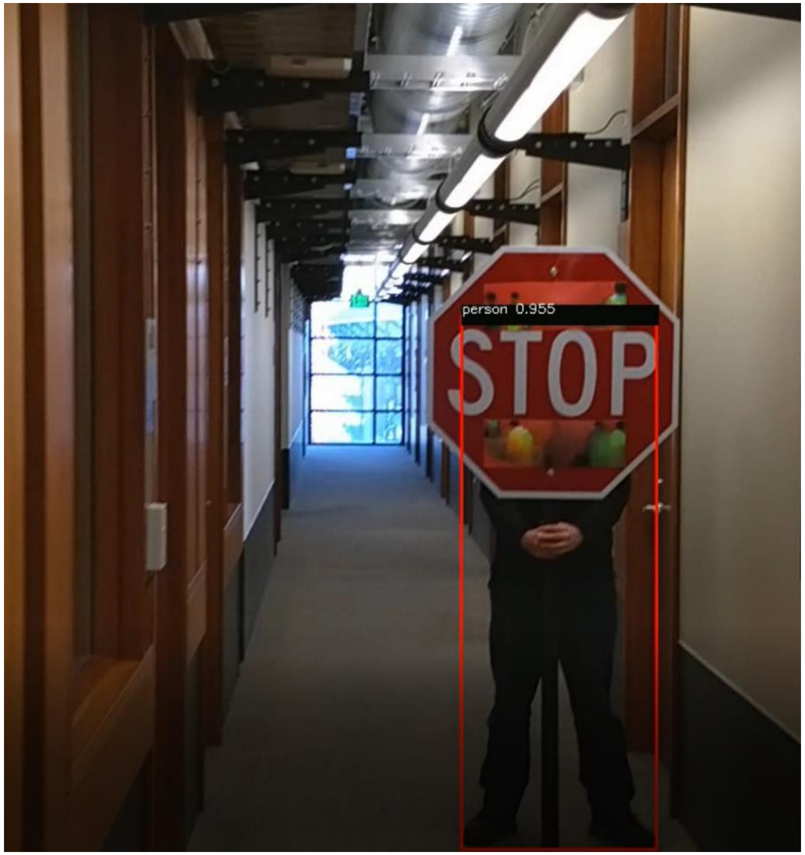
Blackbox Attacks

- We have been discussing Whitebox attacks, where we know the NN model parameters θ
- Black Box Attacks:
- If you have the training dataset of the target Blackbox model:
  - Train a proxy Whitebox model yourself
  - Generate attacked objects for the proxy model
- If you do not have the training dataset, you can obtain input-output data pairs from the target Blackbox model by invoking online cloud services
  - May get expensive if the cloud service is not free



Blackbox Attack Example

- [Evtimov et al 2017]: Physical adversarial examples generated for the YOLO object detector (the proxy Whitebox model) are also be able to fool Faster-RCNN (the Blackbox model)



DeepBillboard

- Goal: generate a single adversarial billboard image that may mislead the steering angle of an AV upon every single frame captured by onboard camera during the process of driving by a billboard.

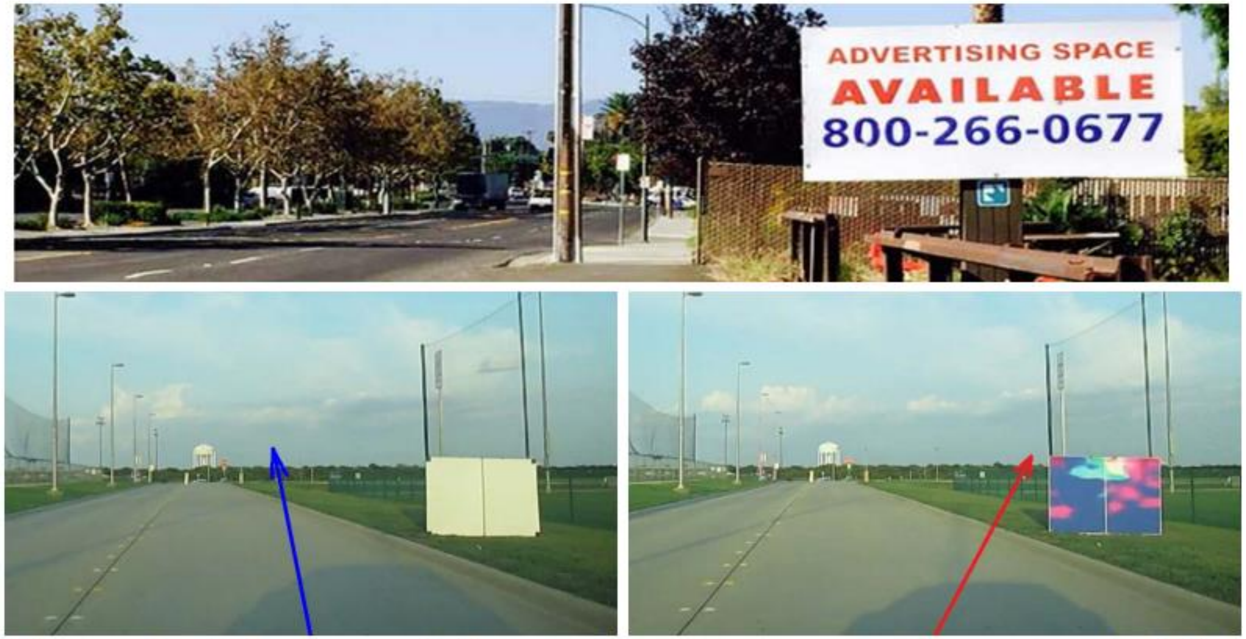
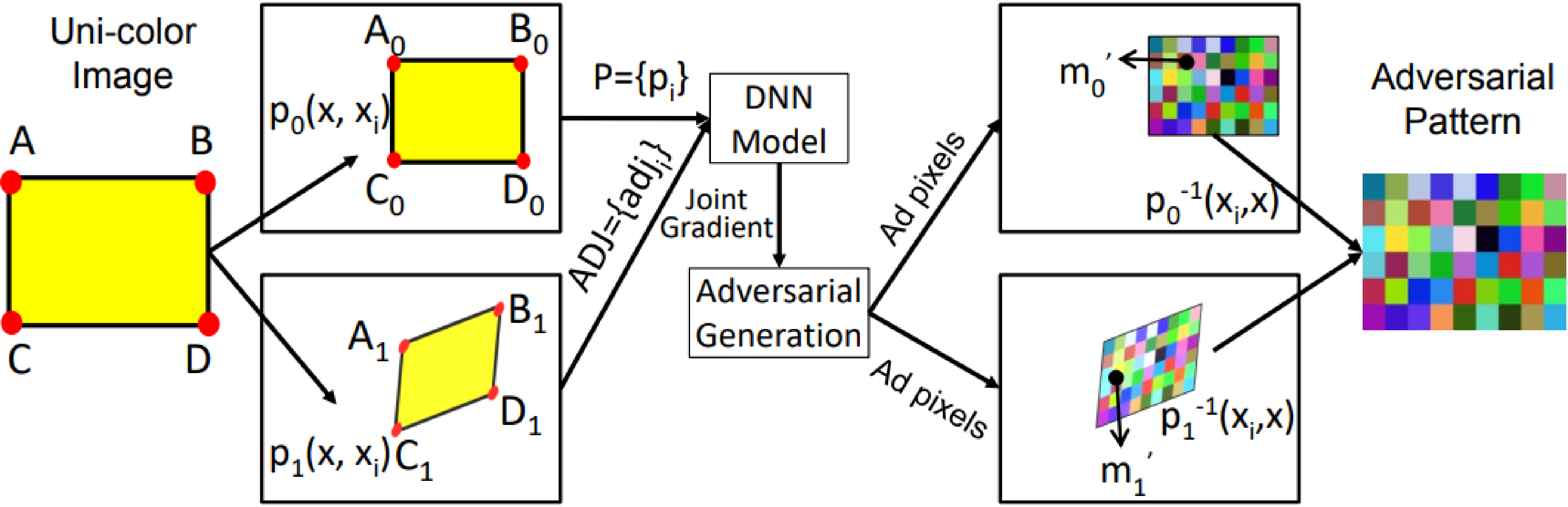


Figure 1: The top subfigure shows an example customizable roadside billboard. The bottom two subfigures show an adversarial billboard example, where the Dave [4] steering model diverges under our proposed approach.

Zhou H, Li W, Kong Z, et al. Deepbillboard: Systematic physical-world testing of autonomous driving systems[C]//2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020: 347-358.

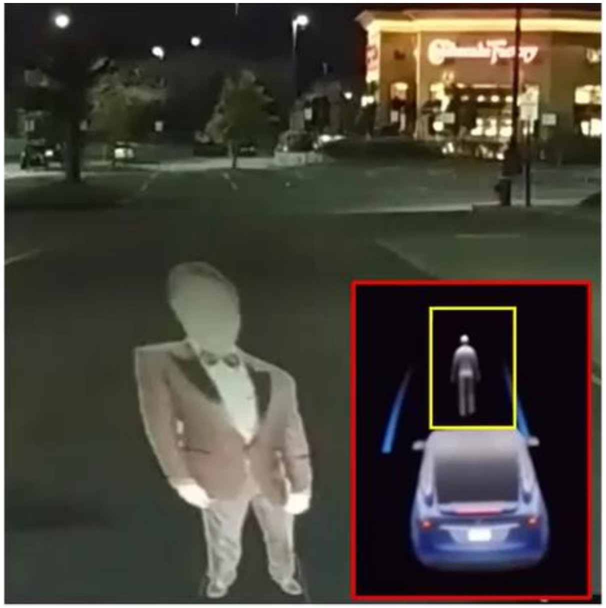
DeepBillboard Workflow

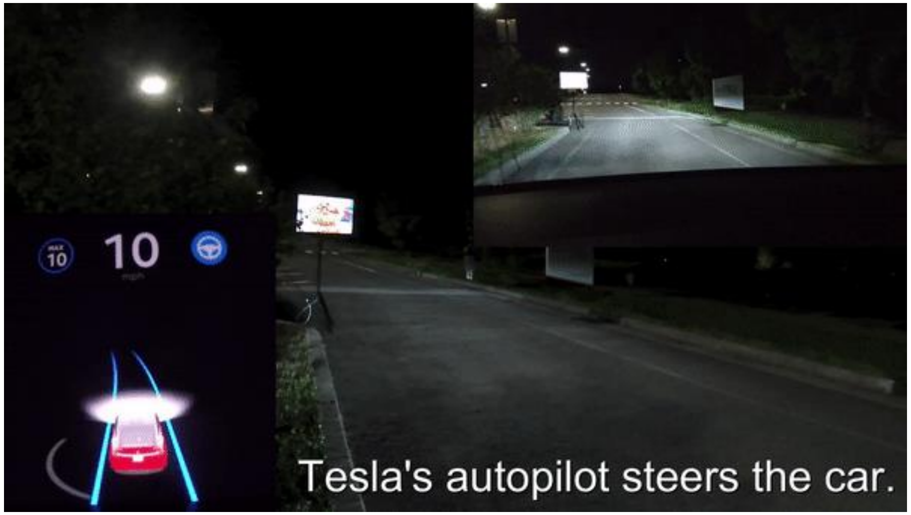
- To generate adversarial perturbations for contiguous frames,
- 1. Training dataset generation: pre-fill the billboard with unicolor, and paint its four corners with contrasting colors. Record video while driving by the billboard with different speeds and angles
- 2. Training algorithm: 1. Generate locally-best perturbation for each single frame; 2. Find a single perturbation that misleads DNNs best for multiple consecutive frames (p_i represents the i-th frame), using a joint loss optimization method; 3. Transfer RGB values to printable colors



Phantom of the ADAS

- A phantom is a depthless presented/projected picture of a 3D object (e.g., pedestrian, traffic sign, car, truck, bicycle…), with the purpose of fooling ADAS to treat it as a real object and trigger an automatic reaction
- Phantom attacks by projecting a phantom via a drone equipped with a portable projector:
  - https://www.youtube.com/watch?v=1cSw4fXYqWI&t=85s
- or by presenting a phantom on a hacked roadside digital billboard:
  - https://www.youtube.com/watch?v=-E0t_s6bT_4





https://www.nassiben.com/phantoms

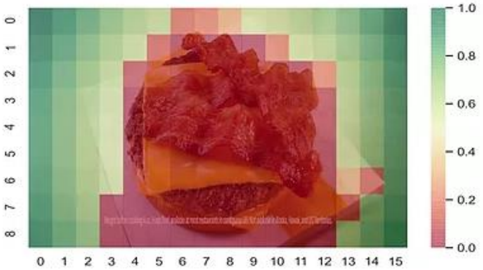
Algorithm for Disguising Phantoms

- 1. Extract key points as focus areas of human attention for every frame based on the SURF algorithm
- 2. Compute a local score for every block in a frame that represents how distant a block is from the focus areas, and embed phantoms into “dead areas” that viewers will not focus on
- 3. Display the phantom in at least t consecutive video frames (longer duration leads to higher success rate)

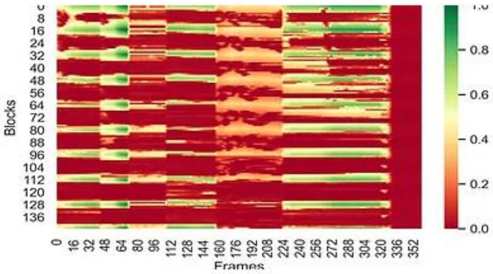
Original frame

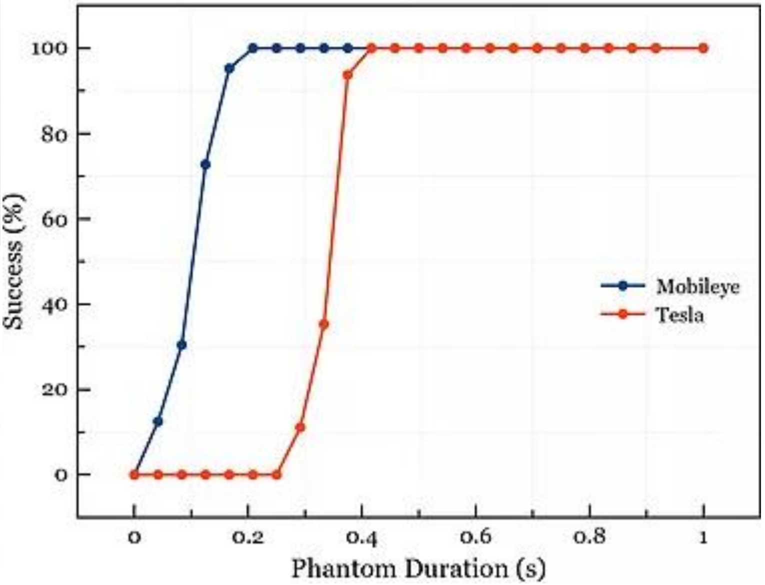
Detecting focus areas in a frame (in blue)

Detecting dead areas in a frame (in green)



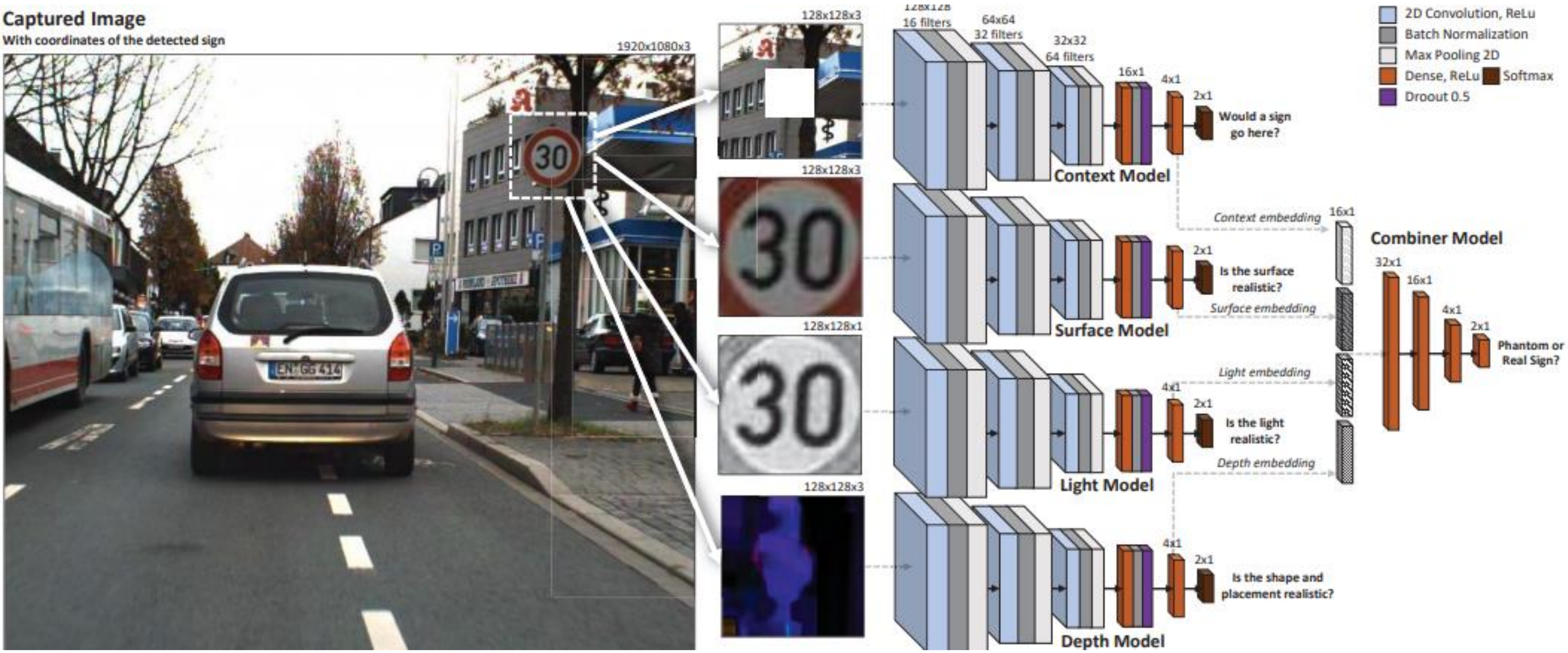
Detecting dead areas in the entire advertisement





Countermeasure - GhostBusters

- When a frame is captured, (1) the on-board object detector locates a road sign, (2) the road sign is cropped and passed to the Context, Surface, Light, and Depth models, and (3) the Combiner model interprets the models' embeddings and makes a final decision on the traffic sign (real or fake).



Constraints on Perturbations?

- In both DeepBillboard and Phantom of the ADAS attacks, there is no $\delta \in \Delta$ norm constraint on the allowable perturbations
  - DeepBillboard simply assumes human drivers don’t pay attention to roadside billboards, so the entire area of the billboard can be used for attack
  - Phantom of the ADAS embeds phantoms into “dead areas” that human viewers will not focus on
- The $\delta \in \Delta$ norm constraint may not be fully aligned with human perception

Outline

- Introduction to adversarial examples
- Adversarial attack via local search
- Physically-realizable attacks
- Training adversarially robust models

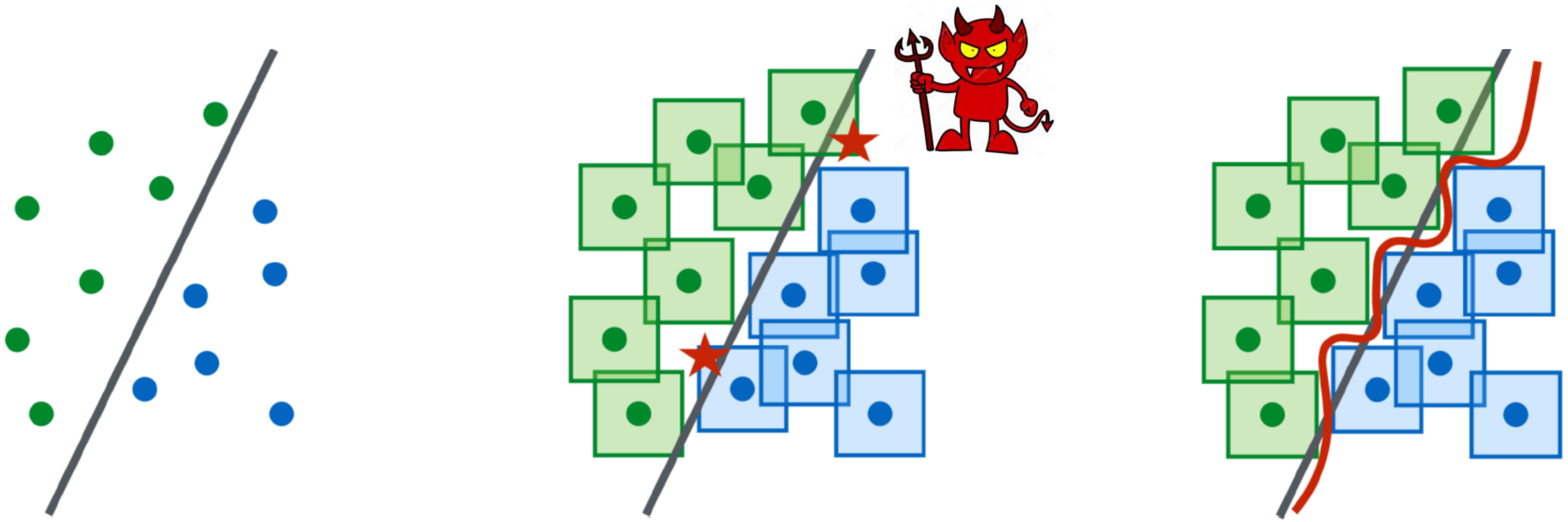
Important

Standard ML vs. Adversarial Robust ML

- Standard ML: Empirical Cost Minimization:
  $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \mathrm{Loss}(x, y; \theta)$
- Adversarial Input Generation (untargeted attack):
  $\max_{\delta\in\Delta} \mathrm{Loss}(x + \delta, y; \theta)$ (e.g., FGSM, PGD)
  - Alternative formulation: $\min_{x+\delta \text{ is adversarial}} \delta$ (e.g., Deepfool, C&W)
- Adversarial Robust ML:
  $\min_{\theta} \mathbb{E}_{(x,y)\sim D} \max_{\delta\in\Delta} \mathrm{Loss}(x + \delta, y; \theta)$
  - Inner maximization problem: generating an adversarial input by adding a small perturbation δ (or ensuring one does not exist)
  - Outer minimization problem: training a robust classifier in the presence of adversarial examples

Important

Adversarial Training w. Outer Minimization



1) Simple decision boundary

2) Adversarial examples ★ within L_∞ balls

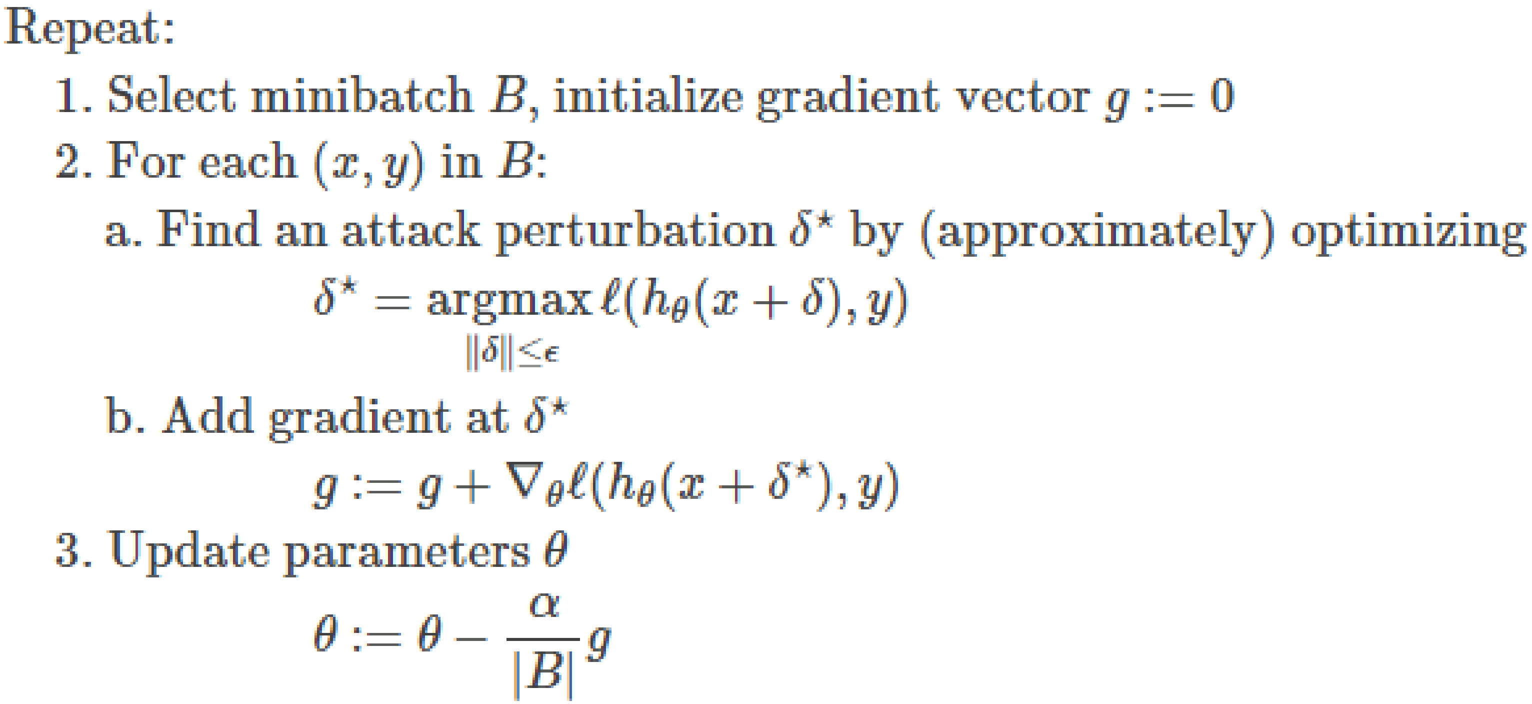
3) Training on ★ requires a complex decision boundary

- Higher network capacity enables more complex decision boundary and more robust classification

Danskin's Theorem

- How to compute the gradient of the objective with the $\max$ term inside?
- Danskin's Theorem:
  - $\nabla_y \max_x f(x, y) = \nabla_y f(x^*, y)$, where $x^* = \operatorname{argmax}_x f(x, y)$
  - (Only true when $\max$ is performed exactly)
- In our case:
  - $\nabla_\theta \max_{\delta \in \Delta} \mathrm{Loss}(x + \delta, y; \theta) = \nabla_\theta \mathrm{Loss}(x + \delta^*, y; \theta)$, where $\delta^* = \operatorname{argmax}_{\delta \in \Delta} \mathrm{Loss}(x + \delta, y; \theta)$
  - It means we can optimize through the $\max$ operator by finding the δ^* that maximizes the loss function, then taking gradient at $x + \delta^*$

Adversarial Training [Goodfellow et al., 2014]



```
Repeat:
  1. Select minibatch $B$, initialize gradient vector $g := 0$
  2. For each $(x, y)$ in $B$:
    a. Find an attack perturbation $\delta^\star$ by (approximately) optimizing
         $\delta^\star = \operatorname*{argmax}_{\|\delta\| \leq \epsilon} \ell(h_\theta(x + \delta), y)$
    b. Add gradient at $\delta^\star$
         $g := g + \nabla_\theta \ell(h_\theta(x + \delta^\star), y)$
  3. Update parameters $\theta$
         $\theta := \theta - \frac{\alpha}{|B|} g$
```

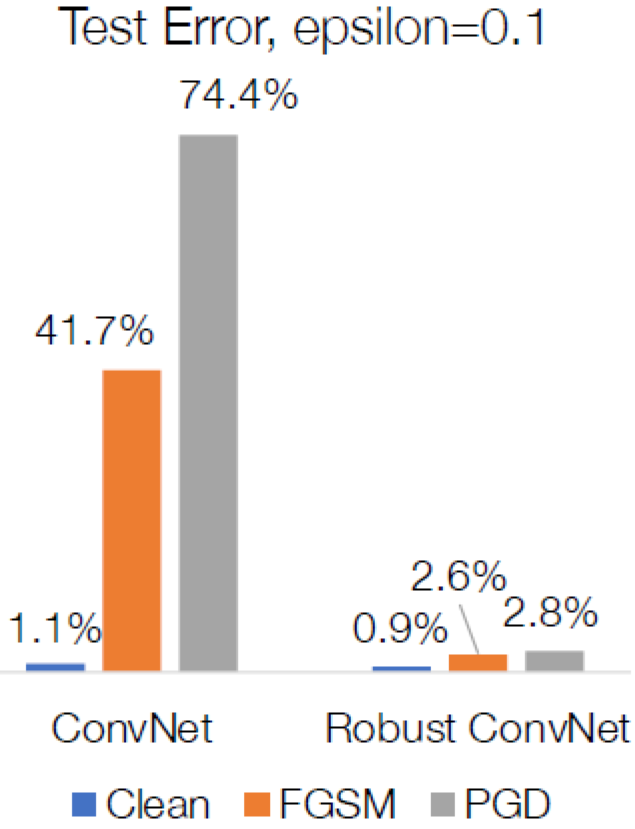
- In theory, Danskin’s theorem only applies to the case where we are able to compute the maximum exactly. In practice, the quality of the robust gradient descent procedure is tied directly to how well we are able to perform the inner maximization. In other words, the key aspect is incorporate a strong attack into the inner maximization procedure.

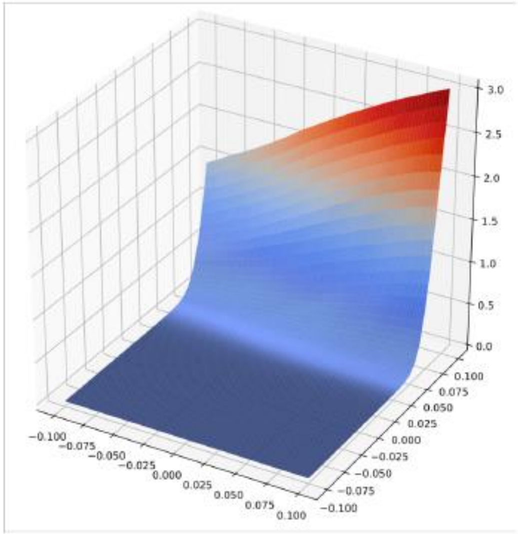
Evaluating Robust Models

- Our model looks good, but we also need to evaluate against different attacks, PGD attacks run for longer, with random restarts, etc
- Note: it is not very informative to evaluate against a different type of attack, e.g. evaluate l_∞ robust model against l_1 or l_2 attacks

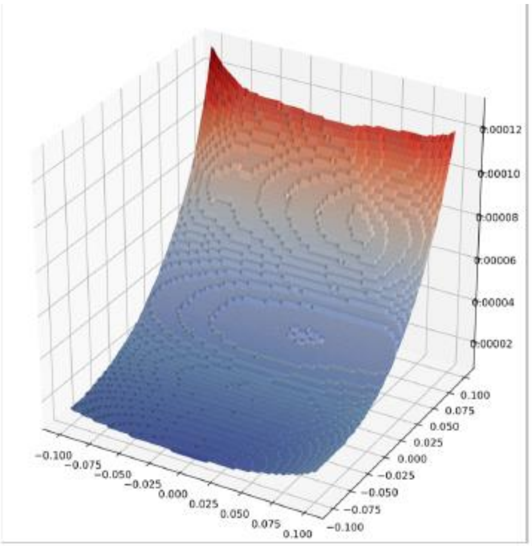
What Makes the Models Robust?

- The robust model has a smoother loss surface, making it more difficult for an attacker to change the class label with small gradient steps





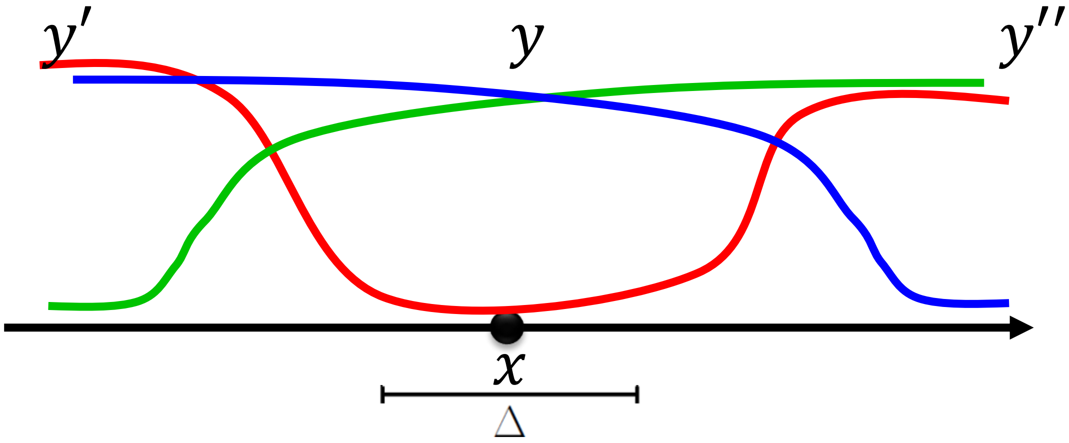
Loss surface:
standard training

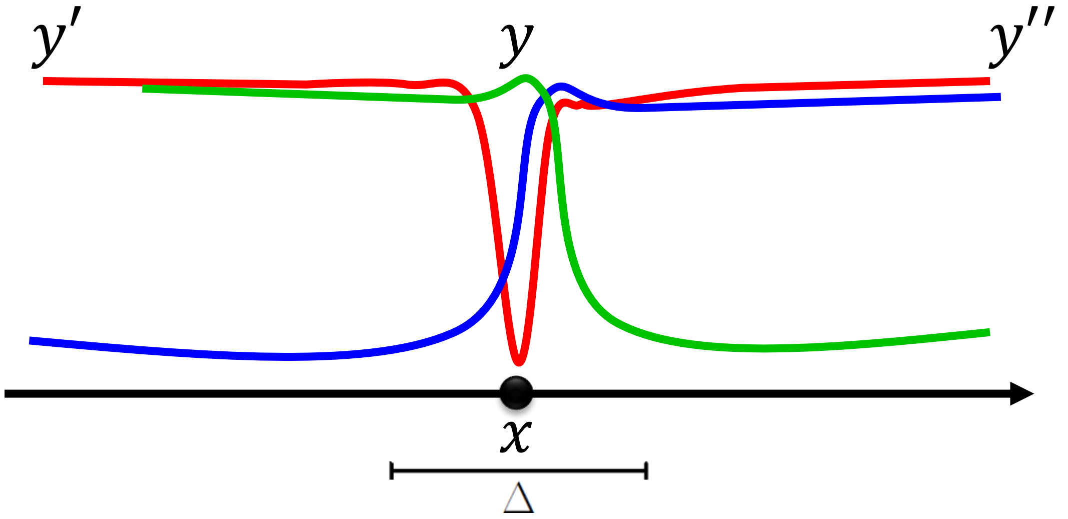


Loss surface:
robust training

Loss Surfaces Examples

- Upper right fig shows a smooth loss surface with small gradients near the correct label and large distances to other labels, which makes attacks more difficult
- Lower right fig shows a less smooth loss surface and small distances to other labels, which makes attacks easier
- You can also think of them as 2 different directions on the same loss surface, and the attacker’s goal is to find the optimal direction to change input x (e.g., by gradient ascent with FGSM or PGD)



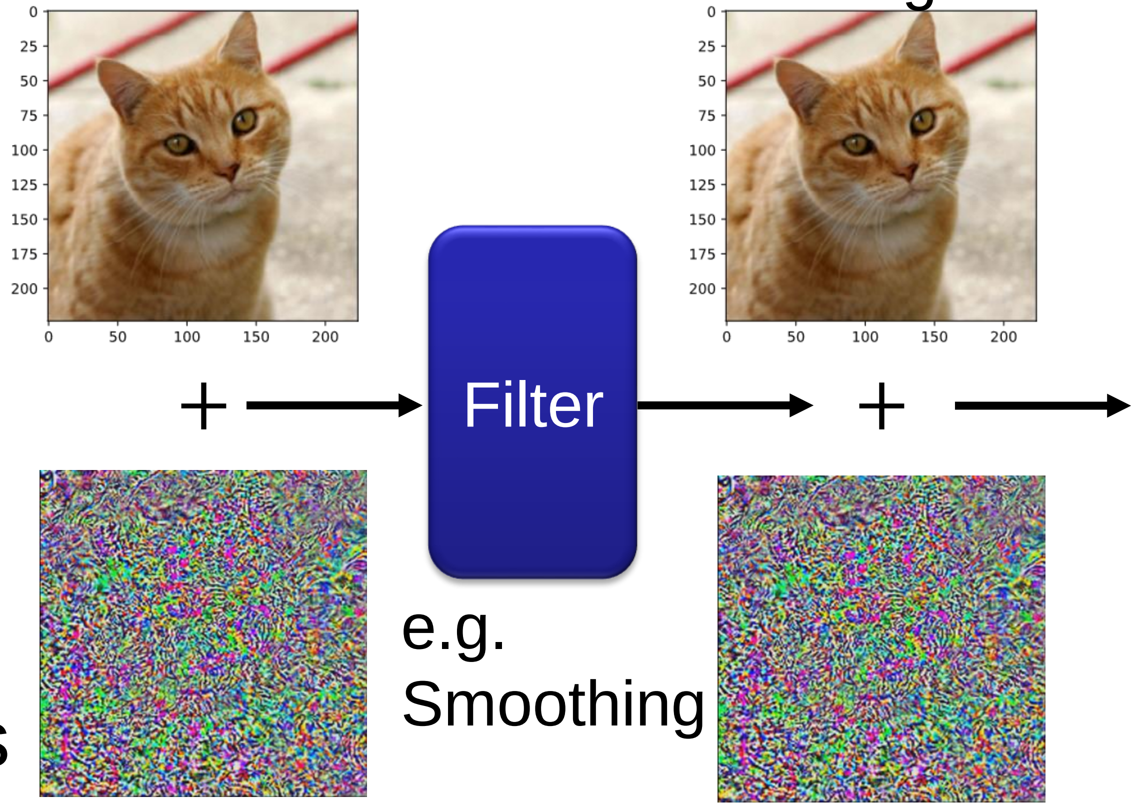


Smoothing Filter on the Input as Defense

- The filter helps make the loss surface smoother, which makes attacks more difficult
- Not a very effective defense. Furthermore, if attacker knows the filter, the defense is no longer effective

Original Image

Smoothed Image



e.g. Smoothing

Attack signal δ

Smoothed attack signal δ is less effective

Image credit: Hung-yi Lee

Conclusions

- Algorithms: Faster robust training + verification, smaller models, new architectures?
- Data: New datasets and more comprehensive set of perturbations (robust-ml.org)
- Open-source tools:
  - IBM Adversarial Robustness Toolbox (ART) https://github.com/Trusted-AI/adversarial-robustness-toolbox