

Adversarial Images

Dr. Konda Reddy Mopuri
Deep Learning for Computer Vision (DL4CV)
IIT Guwahati
Aug-Dec 2022

Adversarial Attacks

- What are they ?
- Why are they dangerous ?
- How are they generated ?
- Why do they exist ?
- How to handle them ?

What are they ?

What are Adversarial attacks ?

- ❑ Perturbed inputs that fool the learned models
 - ❑ Negligible/small perturbation



Shetland sheepdog



Paintbrush

What are Adversarial attacks ?

- Altered with specific objectives
 - E.g., in case of vision: Visually quasi-imperceptible (to humans)
 - Misleading the classifiers → wrong prediction



Mortar



Komodo Dragon

Why are they dangerous ?

Why are they dangerous ?




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

Image from **Goodfellow** et al. 2014

Even the SOTA models will misclassify them with high confidence

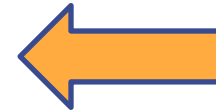
Not convinced ?

- What if we can make them predict what we want ?



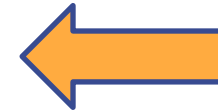
Not convinced ?

❑ What if we can make them predict what we want ?



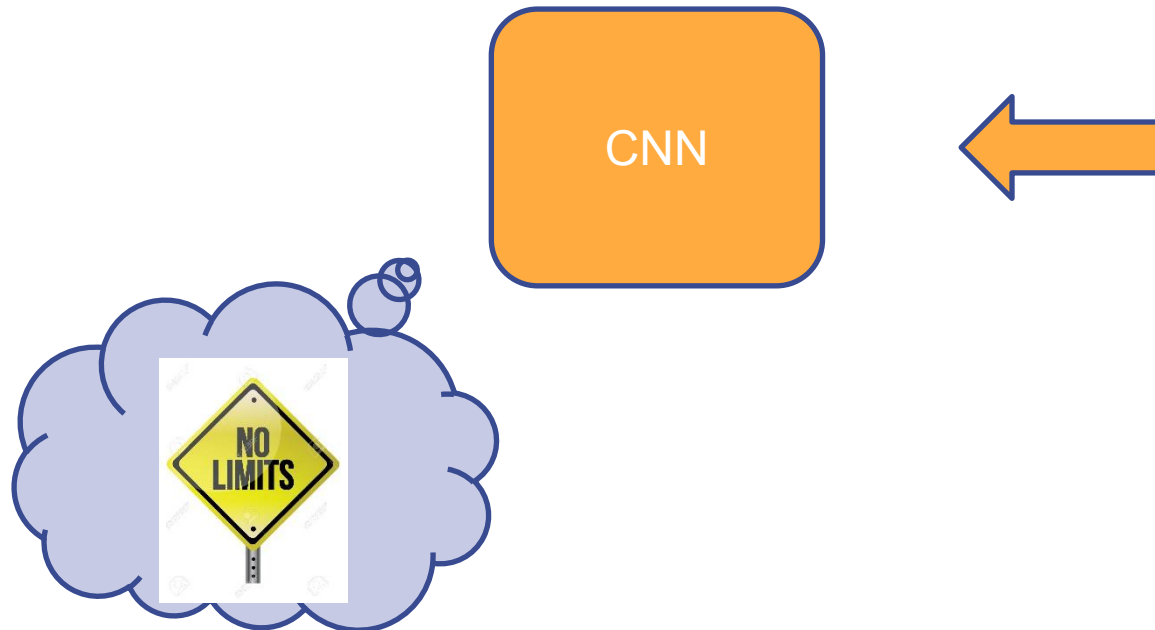
Not convinced ?

❑ What if we can make them predict what we want ?



Not convinced ?

❑ What if we can make them predict what we want ?



Not convinced ?

❑ What if we can make them predict what we want ?



Can't Deploy models

- ☐ Simple access granting
- ☐ Defence applications
- ☐ Autonomous driving
- ☐ Medical diagnosis etc.

What is more dangerous ?

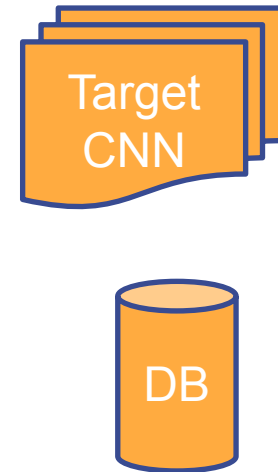
Transferability

Transferability

- ☐ You don't need to
 - ☐ Know the target system (say, the architecture of target CNN)
 - ☐ Have the same data on which the target is trained

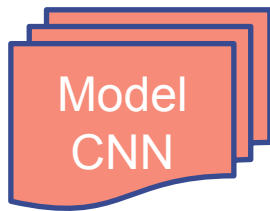
Transferability

- ❑ Generate on your own model and data → they can attack many other systems (to significant effect)



Transferability

- Generate on your own model and data → they can attack many other systems (to significant effect)



Need not be the same architectures

Need not be the same set of train images



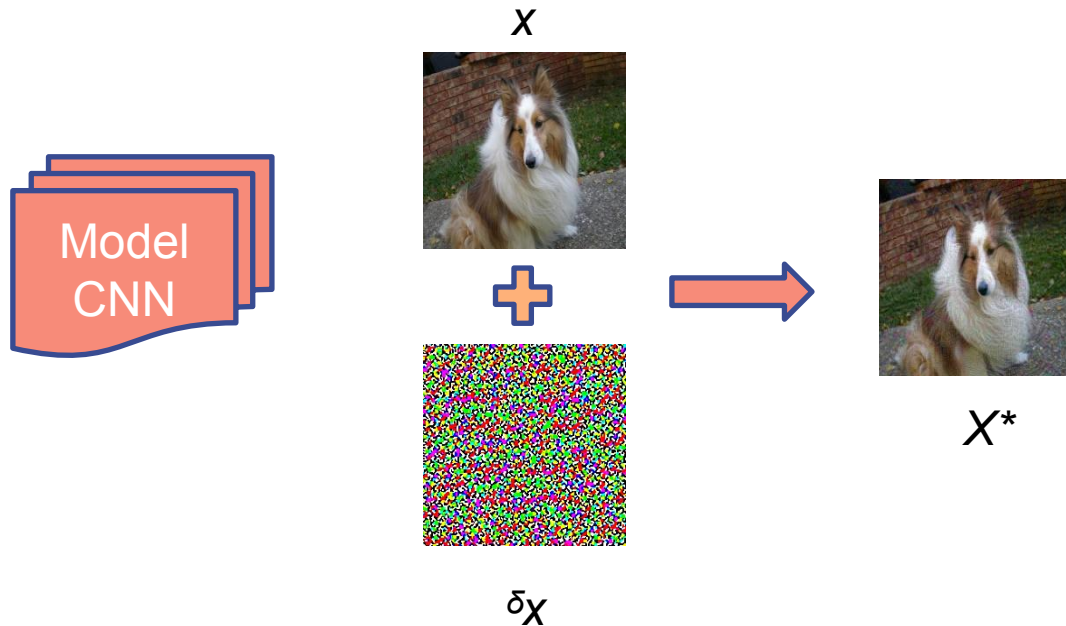
Transferability

- ❑ Generate on your own model and data → they can attack many other systems (to significant effect)



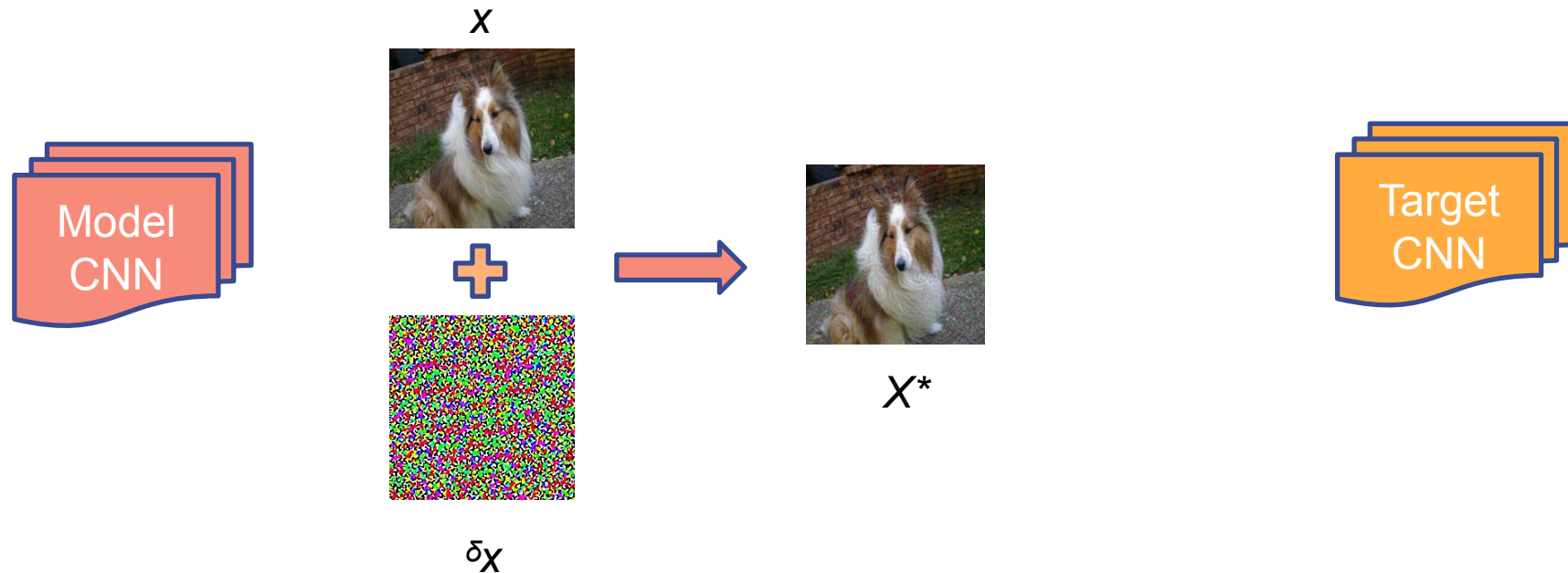
Transferability

- Generate on your own model and data → they can attack many other systems (to significant effect)



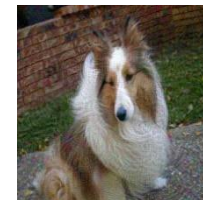
Transferability

- Generate on your own model and data → they can attack many other systems (to significant effect)



Transferability

- Generate on your own model and data → they can attack many other systems (to significant effect)



X^*



Transferability

- Generate on your own model and data → they can attack many other systems (to significant effect)



X^*



Paintbrush

How are they generated ?

Linear model case

❑ Dot product between 'w' and 'x*'

$$w^T x^* = w^T x + w^T \delta_x$$

❑ Activation grows by $w^T \delta_x$

❑ We can maximize this “perturbation” by making $\delta_x = \text{sign}(w)$

Linear model case

- ❑ Max norm constraint \mathcal{E} on δ_x
- ❑ Average magnitude of 'w' is 'm'
- ❑ Dimension of 'w' is 'n'

$$w^T x^* = w^T x + w^T \delta_x$$

Linear model case

□ Max norm constraint ϵ on δ_x

$$w^T x^* = w^T x + w^T \delta_x$$

□ Average magnitude of 'w' is 'm'

□ Dimension of 'w' is 'n'

Activation grows by **$n m \epsilon$**

Can make many infinitesimal changes to the input \rightarrow large change to the output

Nonlinear Models

Gradient

□ Till now we have computed gradient of cost wrt the weights

▸ To update the weights during training

□ Input – x , output – y and params - Θ

$$\Delta\theta = -\eta \frac{\partial}{\partial\theta} J(\theta, x, y)$$

Gradient wrt image

- ❑ What does that mean?

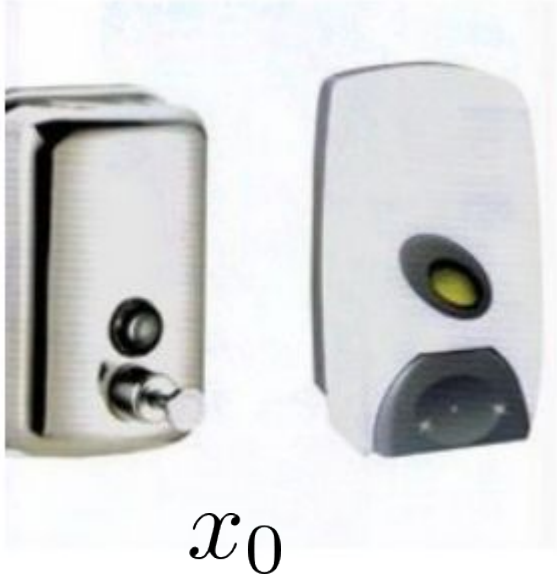
$$\Delta x = \frac{\partial}{\partial x} J(\theta, x, y)$$

- ❑ Will be of same size as image

- ❑ Direction in image space in which the cost increases → predicted confidence decreases

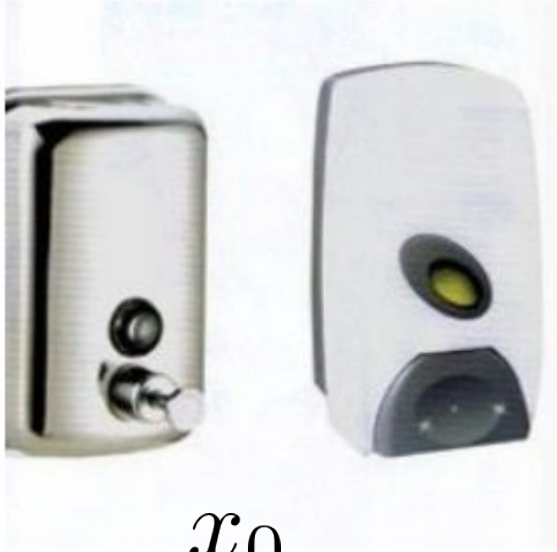
The idea is simple, rather than working to minimize the loss by adjusting the weights based on the backpropagated gradients, the attack adjusts the input data to maximize the loss based on the same backpropagated gradients. In other words, the attack uses the gradient of the loss w.r.t the input data, then adjusts the input data to maximize the loss.

Gradient wrt image

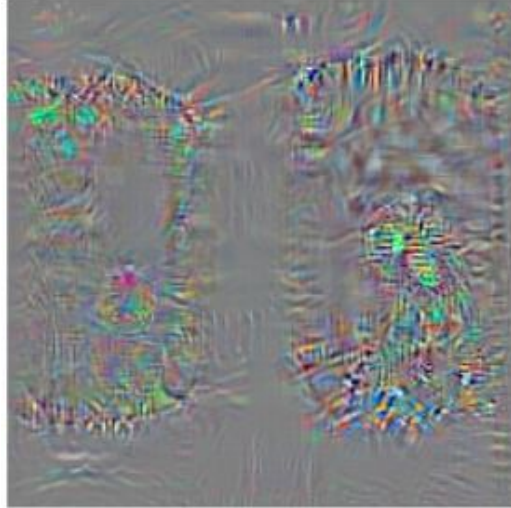


$$J = 0.76$$
$$C_{\text{soap-dispencer}} = 0.546$$

Gradient wrt image

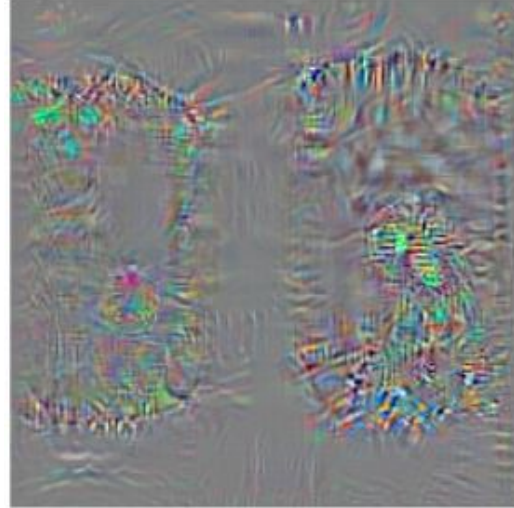
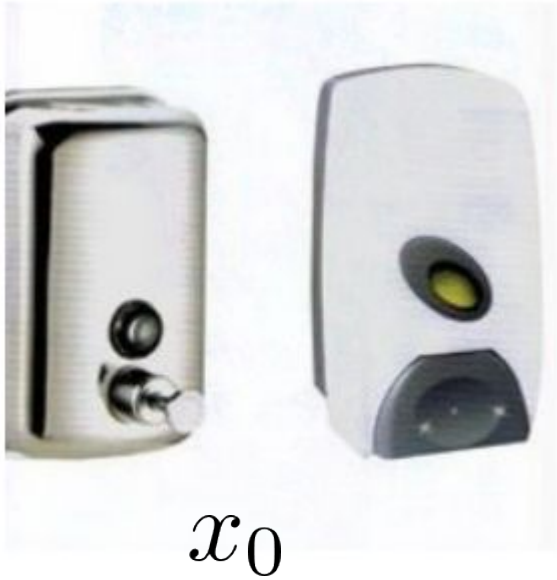


x_0

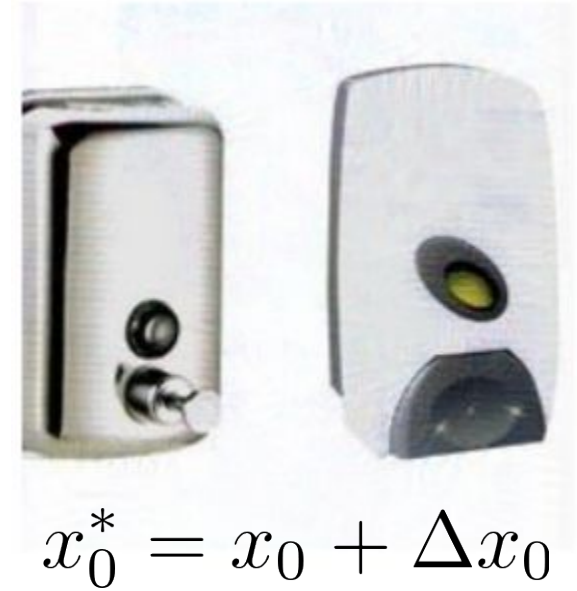


$$\Delta x_0 = \left. \frac{\partial J}{\partial x} \right|_{x_0}$$

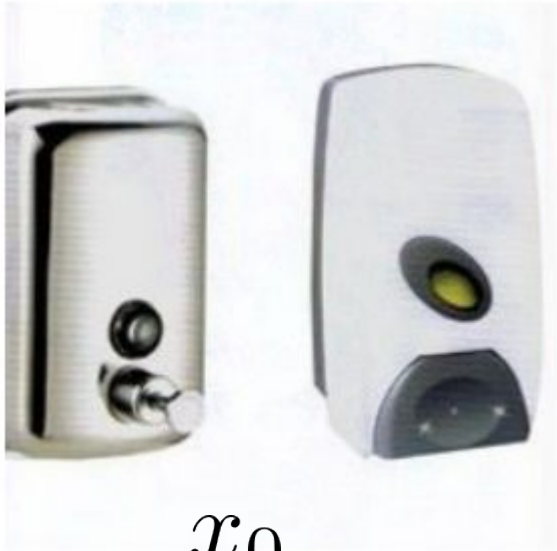
Gradient wrt image



$$\Delta x_0 = \left. \frac{\partial J}{\partial x} \right|_{x_0}$$



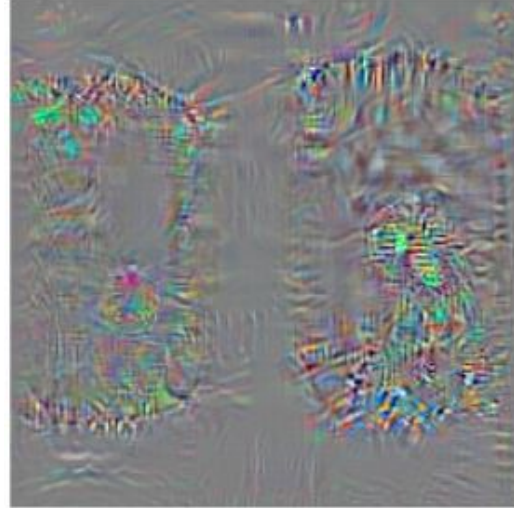
Gradient wrt image



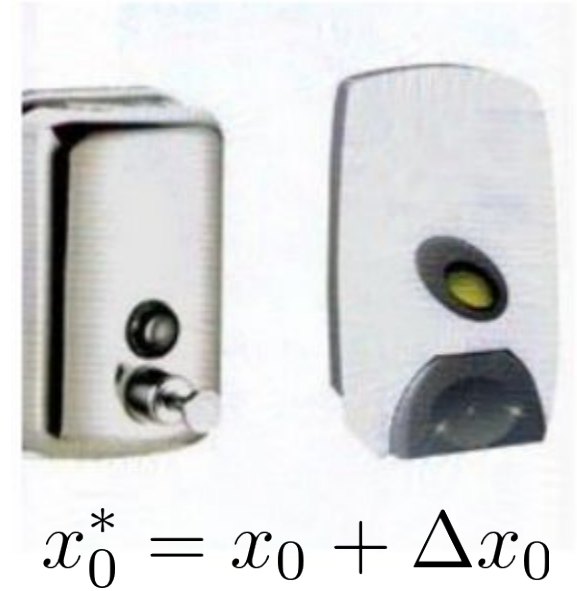
$$J = 0.76$$

$$C_{\text{soap-dispenser}} = 0.546$$

$$C_{\text{ostrich}} = 0.007$$



$$\Delta x_0 = \frac{\partial J}{\partial x} \bigg|_{x_0}$$

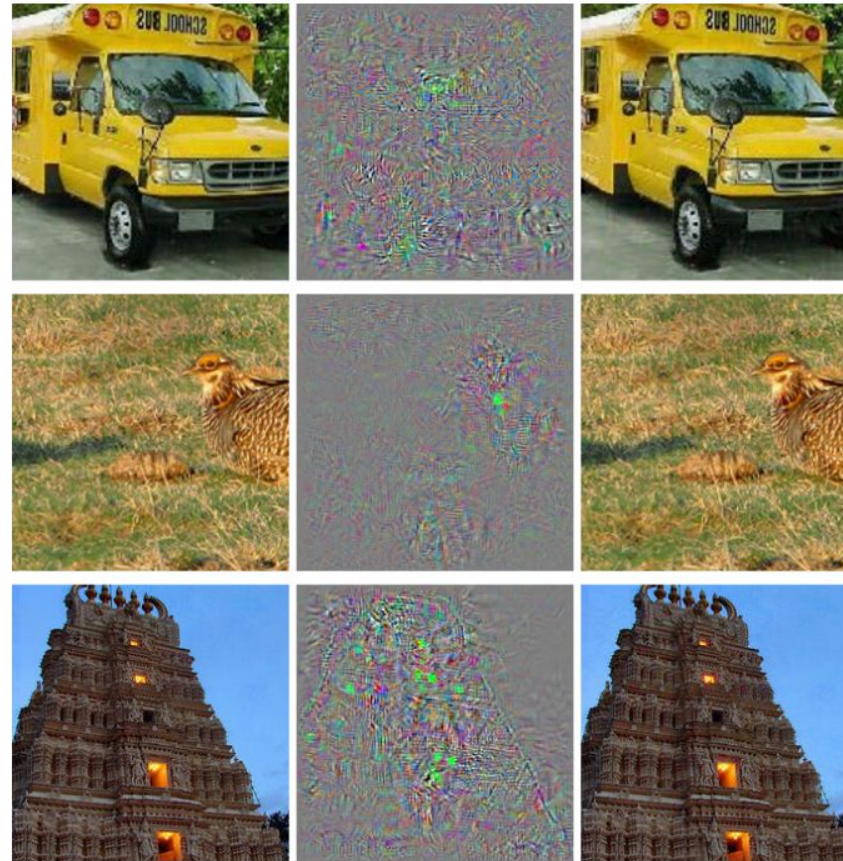


$$J = 1.18$$

$$C_{\text{soap-dispenser}} = 0.046$$

$$C_{\text{ostrich}} = 0.39$$

Gradient wrt image



Nonlinear models

□ Max-norm perturbation is given by $\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

$$x^{\text{adv}} = x + \epsilon \text{sgn}(\nabla_x \mathbf{J}_{\text{cls}}(x, y_{\text{true}}(x)))$$

Nonlinear models

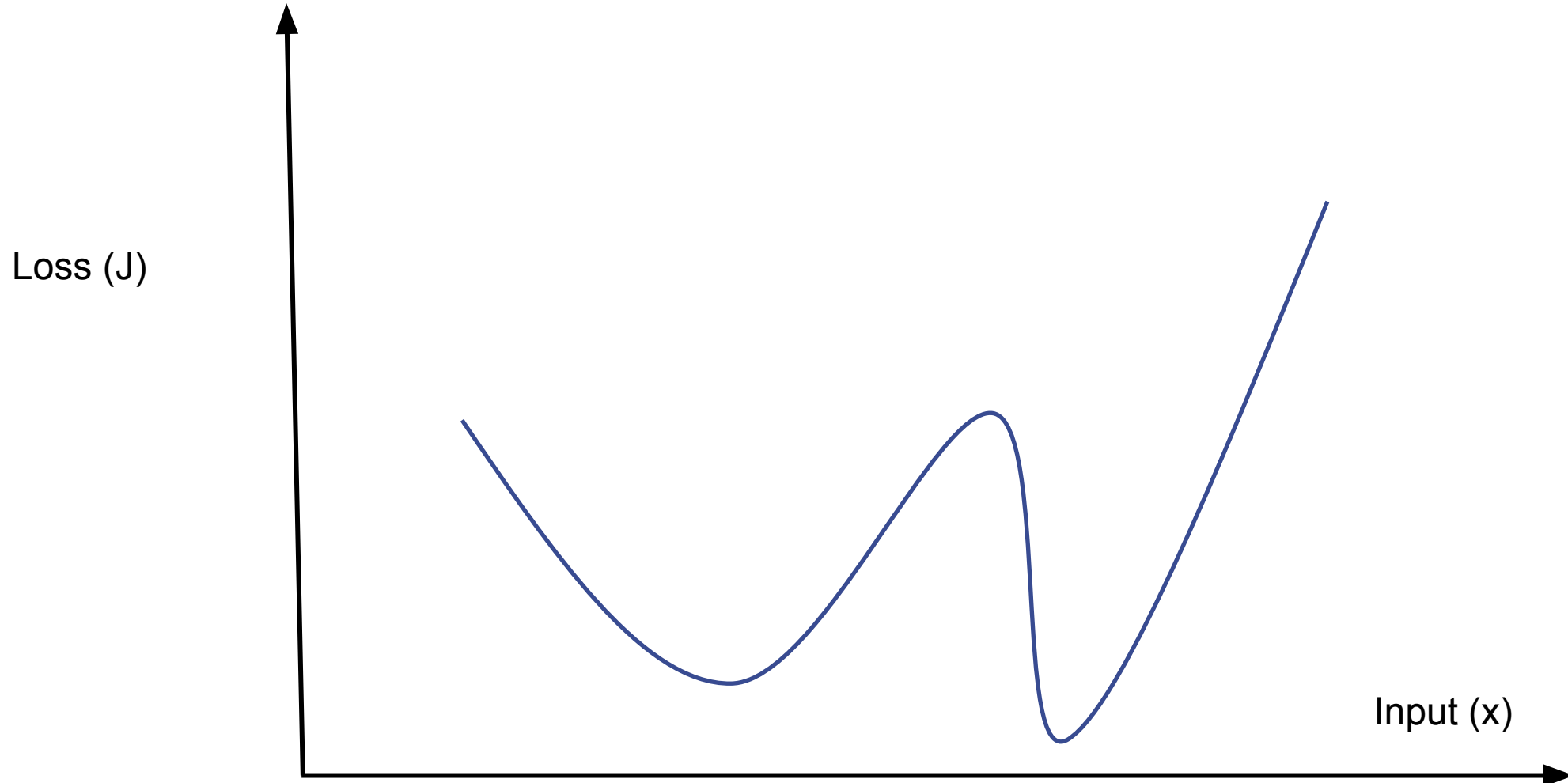
□ Max-norm perturbation is given by $\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

$$x^{\text{adv}} = x + \epsilon \text{sgn}(\nabla_x J_{\text{cls}}(x, y_{\text{true}}(x)))$$

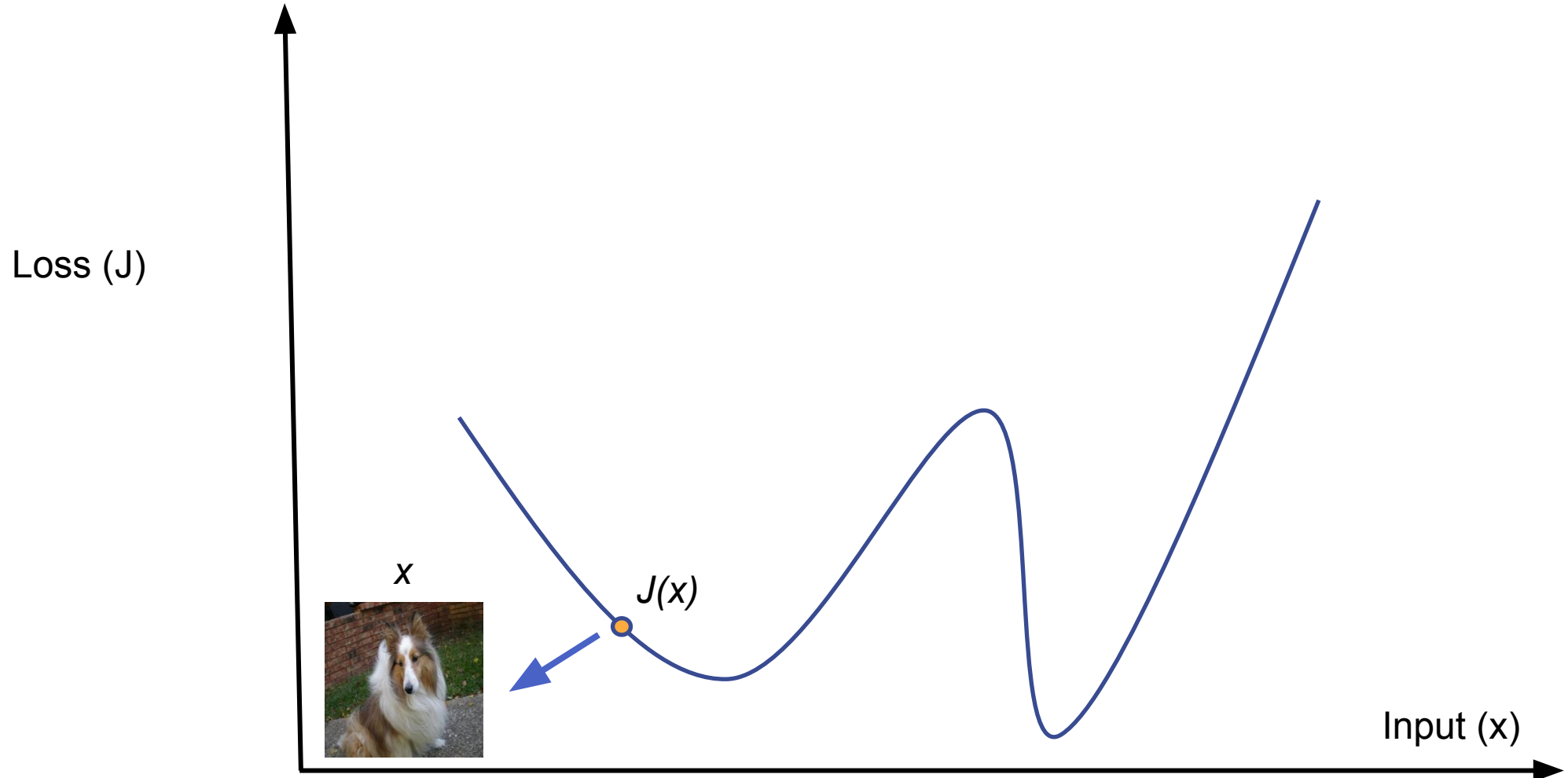
The idea is simple, rather than working to minimize the loss by adjusting the weights based on the backpropagated gradients, the attack adjusts the input data to maximize the loss based on the same backpropagated gradients. In other words, the attack uses the gradient of the loss w.r.t the input data, then adjusts the input data to maximize the loss.

Fast Gradient Sign Method (**FGSM**) by Goodfellow et al. ICLR 2015

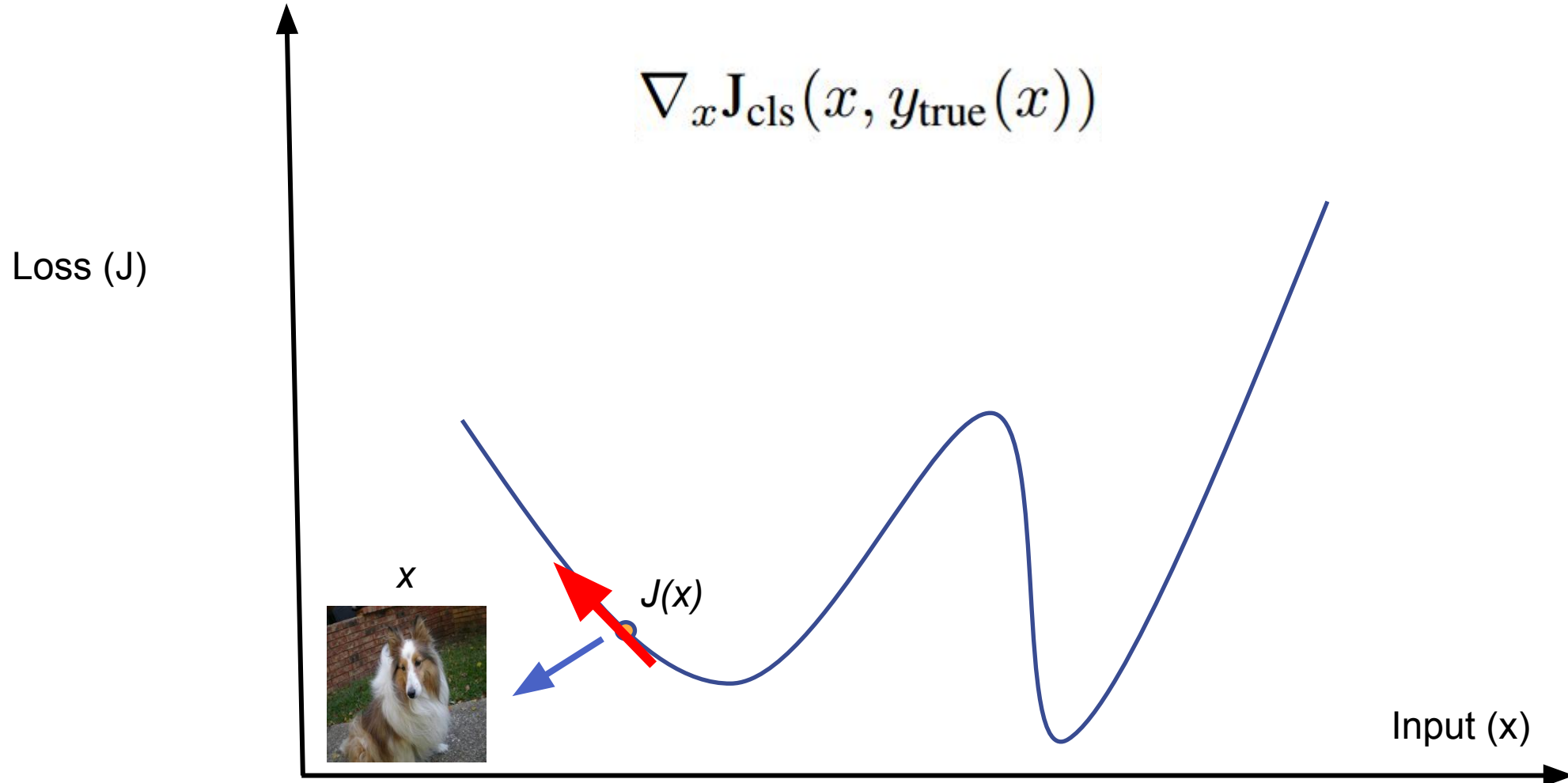
Simple gradient ascent on loss



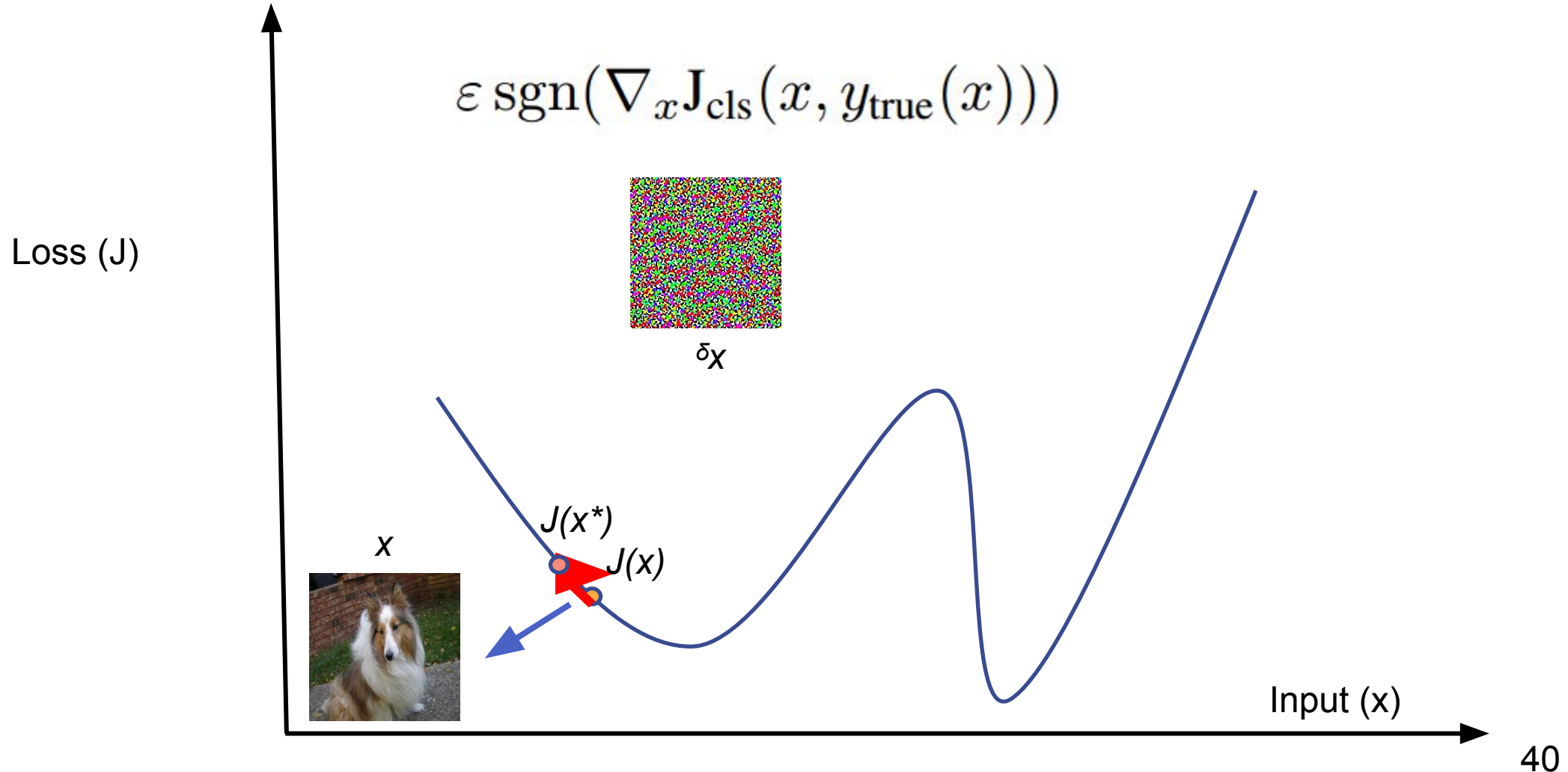
Simple gradient ascent on loss



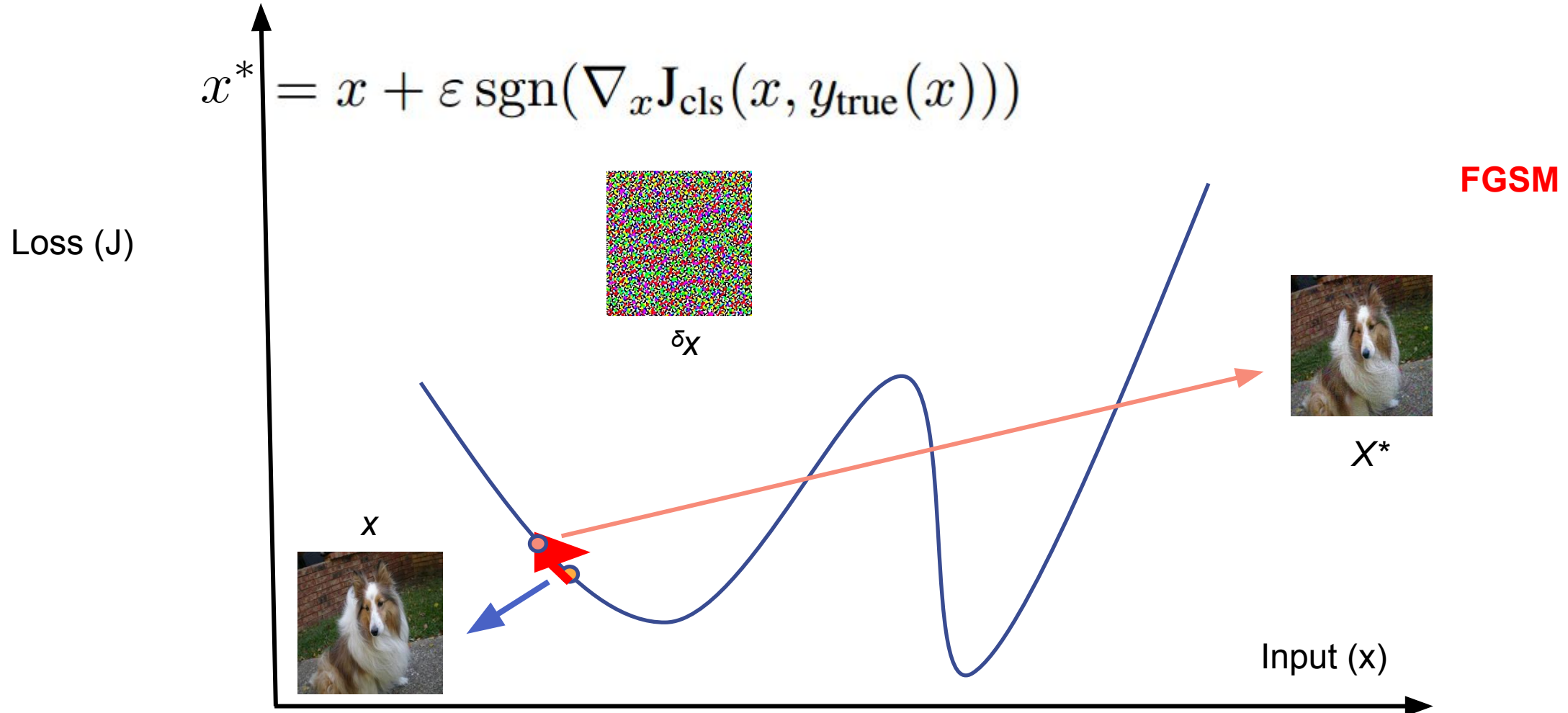
Simple gradient ascent on loss



Simple gradient ascent on loss



Simple gradient ascent on loss



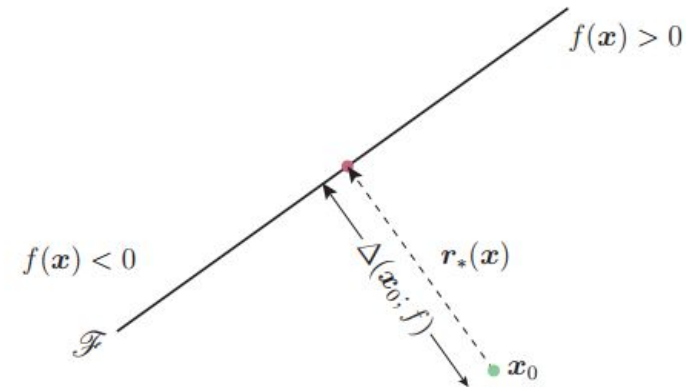
Other ways to generate

- ☐ Multiple other ways are proposed/observed
- ☐ Most of them carefully manipulate the input image to fool the classifier
- ☐ Use optimization techniques
 - ☐ Visually imperceptible
 - ☐ Misleading the classifiers

Other ways to generate

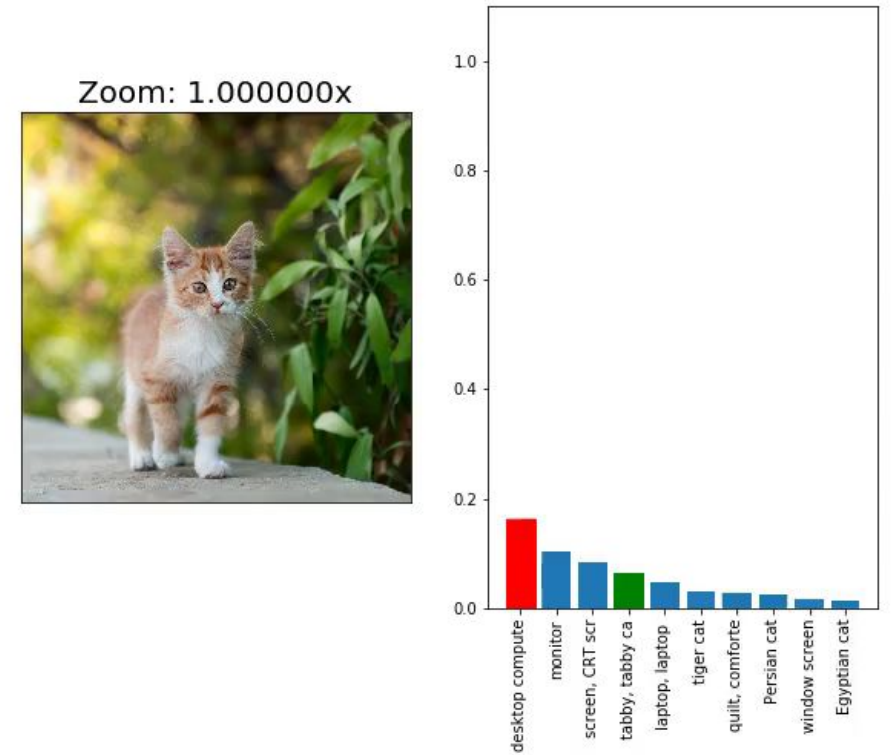
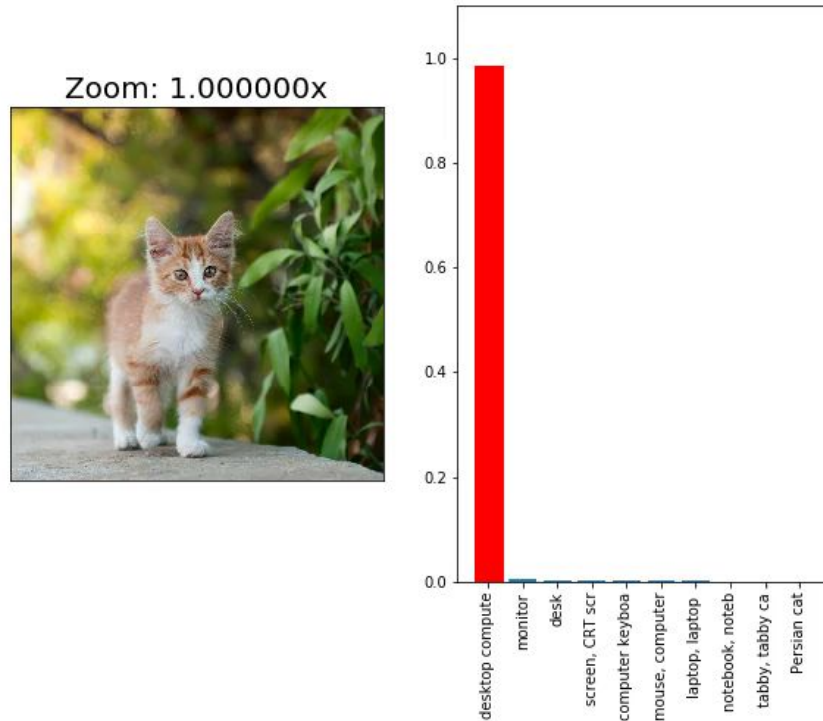
$$\begin{aligned} \mathbf{r}_*(\mathbf{x}_0) &:= \arg \min \|\mathbf{r}\|_2 \\ &\text{subject to } \text{sign}(f(\mathbf{x}_0 + \mathbf{r})) \neq \text{sign}(f(\mathbf{x}_0)) \\ &= -\frac{f(\mathbf{x}_0)}{\|\mathbf{w}\|_2^2} \mathbf{w}. \end{aligned}$$

For binary classifier
[Moosavi-Dezfooli CVPR 2016]



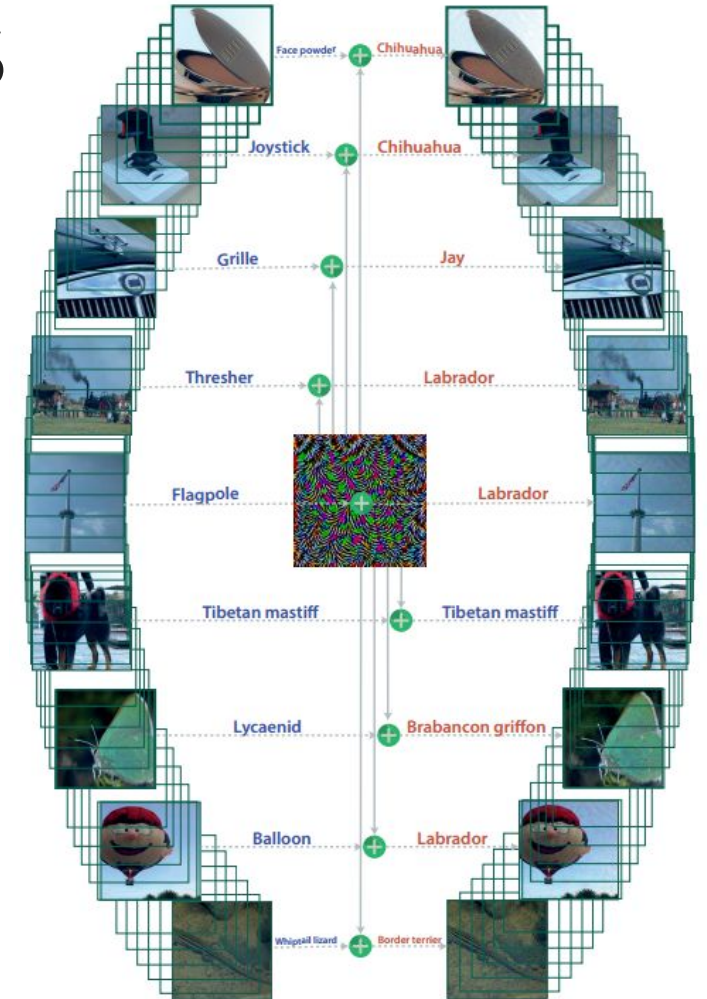
Fooling is getting easier ..

Robust perturbations



Input-agnostic perturbations

- ❑ One perturbation to fool across images and models
- ❑ “Universal” Adversarial Perturbation (UAP)
- ❑ Exhibit cross model generalizability



Moosavi-Dezfooli et al. CVPR 2017

Universal perturbations (UAP)

- ❑ Imperceptibility and fooling objectives
- ❑ Accumulate the DeepFool perturbations for representative set of training images

$$\|v\|_p \leq \xi,$$
$$\mathbb{P}_{x \sim \mu} \left(\hat{k}(x + v) \neq \hat{k}(x) \right) \geq 1 - \delta.$$

Fooling rates for UAP

	VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19	ResNet-152
VGG-F	93.7%	71.8%	48.4%	42.1%	42.1%	47.4 %
CaffeNet	74.0%	93.3%	47.7%	39.9%	39.9%	48.0%
GoogLeNet	46.2%	43.8%	78.9%	39.2%	39.8%	45.5%
VGG-16	63.4%	55.8%	56.5%	78.3%	73.1%	63.4%
VGG-19	64.0%	57.2%	53.6%	73.5%	77.8%	58.0%
ResNet-152	46.3%	46.3%	50.5%	47.0%	45.5%	84.0%

Data-free objectives to craft UAPs

- ❑ No need to have any samples from training data
- ❑ Effective objectives to misfire and misclassify → unstable representations

	VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19
VGG-F	81.59%	48.20%	38.56%	39.31%	39.19%
CaffeNet	56.18%	80.92%	39.38%	37.22%	37.62%
GoogLeNet	49.73%	46.84%	56.44%	40.91%	40.17%
VGG-16	46.49%	43.31%	34.33%	47.10%	41.98%
VGG-19	39.91%	37.95%	30.71%	38.19%	43.62%

Generalizable and Data-free objectives to craft UAPs

❑ Typically, objective to craft UAP is task specific

▸ Fooling is task dependent

❑ Generic objective of “Activation loss”

▸ Single objective across various vision tasks

$$Loss = -\log \left(\prod_{i=1}^K \|l_i(\delta)\|_2 \right), \quad \text{such that} \quad \|\delta\|_\infty < \xi.$$

Generalizable and Data-free objectives to craft UAPs

- ❑ Locally linear assumption: for a feature extractor f , $f(x + \delta) \approx f(x) + f(\delta)$
- ❑ Hence it is sufficient to maximize the change $f(\delta)$ for a vector delta of bounded max-norm
- ❑ Is it a valid assumption ?

Generalizable and Data-free objectives to craft UAPs

- Can exploit minimal prior about the target data distribution
 - Mean value and dynamic range of the samples → Range prior
 - Minimal number of actual samples (not using fooling loss) → data prior

$$Loss = -\log \left(\prod_{i=1}^K \|l_i(d + \delta)\|_2 \right),$$

such that $\|\delta\|_\infty < \xi$, and $d \sim \mathcal{N}(\mu, \sigma)$.

GD-UAP Fooling rates

Model	Baseline	No prior	Range prior	Data prior	FFF [9]	UAP [8]
CaffeNet	12.9	84.88	87.02	91.54	80.92	93.1
VGG-F	12.62	85.96	91.81	92.64	81.59	93.8
Googlenet	10.29	58.62	71.44	83.54	56.44	78.5
VGG-16	8.62	45.47	63.08	77.77	47.10	77.8
VGG-19	8.40	40.68	64.67	75.51	43.62	80.8
Resnet-152	8.99	29.78	37.3	66.68	-	84.0

Why do they exist ?

- ☐ Linear behaviour
- ☐ Insufficient model averaging/regularization
 - ☐ Over-fitting/Under-fitting
- ☐ Discontinuous mappings learned by models
- ☐ Fundamental blind-spots in learning algorithms

Why do they exist ?

- ☐ Linear behaviour
- ☐ Insufficient model averaging/regularization
 - ☐ Over-fitting/Under-fitting
- ☐ Discontinuous mappings learned by models
- ☐ Fundamental blind-spots in learning algorithms

Currently, most of them are speculations! And need to be studied further.

How to overcome them

- ❖ Reconstruction as pre-processing
 - ❖ Foveation (Object cropping), Image blurring, Jpeg Compression, etc.
 - ❖ Auto-encoding, GAN based re-generating, etc.
- ❖ Learn a robust model via Adversarial training

Simple Methods

- ❖ CNNs are robust to scale, occlusion, translation, etc.
- ❖ Adversarial noise need not be
- ❖ Exploits the robustness of CNNs
- ❖ However, the recent attacks are also ROBUST!

Before Foveation

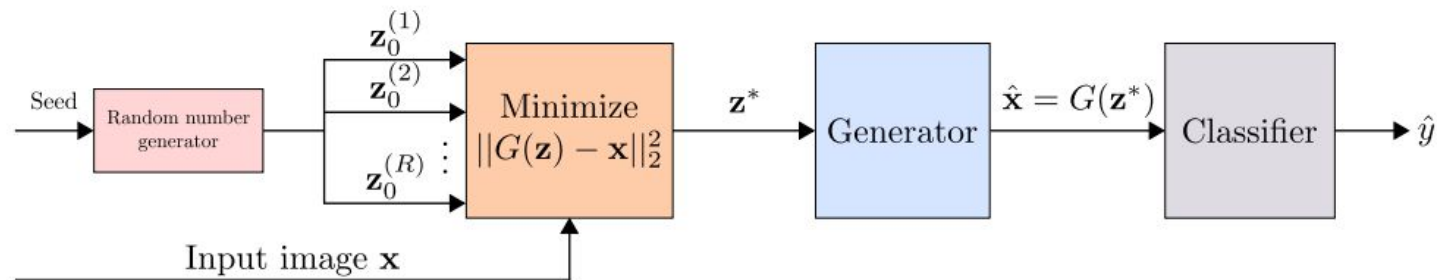


After Foveation



Defense GAN

- ❑ At inference time, finds the GAN-generated sample that is nearest to the (adversarial) input example
- ❑ Generator was trained to model the unperturbed training data \rightarrow a substantial reduction of any potential adversarial noise



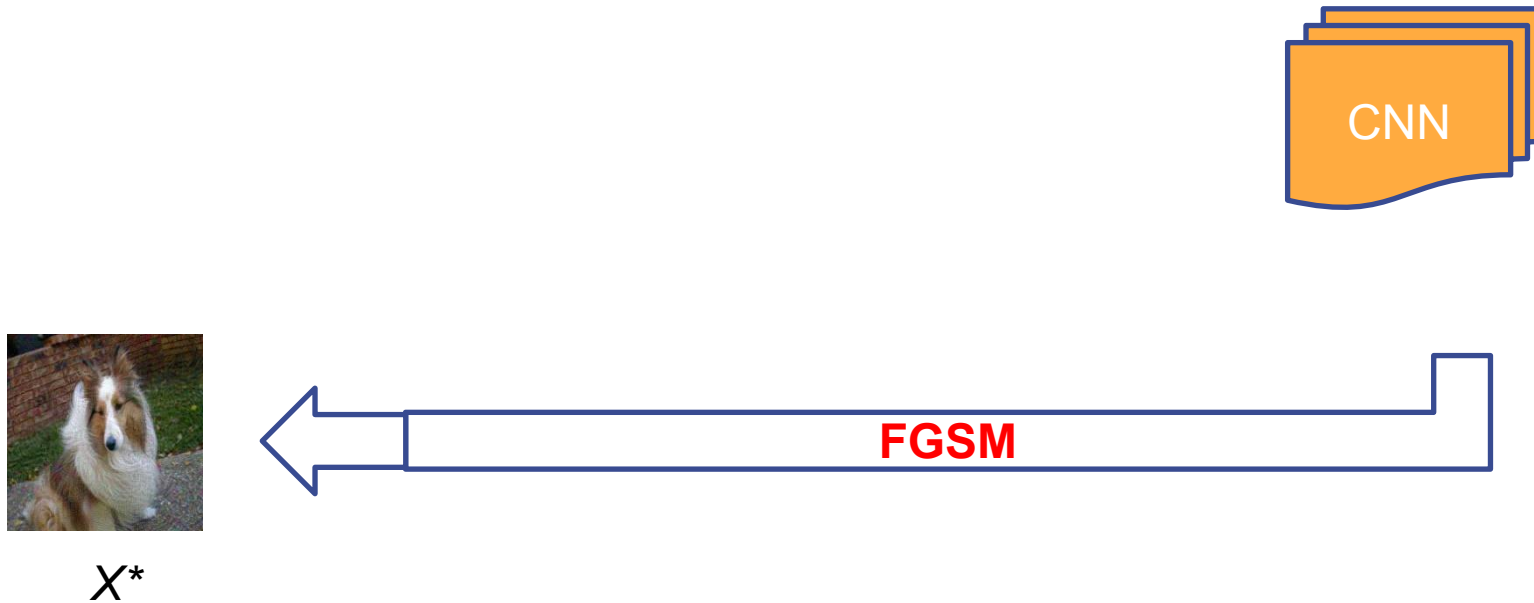
Adversarial Training

- ❖ Best known method to learn robust models **till now!**
- ❖ Train using adversarial images (x^*) also with the GT label
- ❖ At each iteration
 - ❖ Compute adversarial images for some batch members
 - ❖ Train using both normal and adversarial images

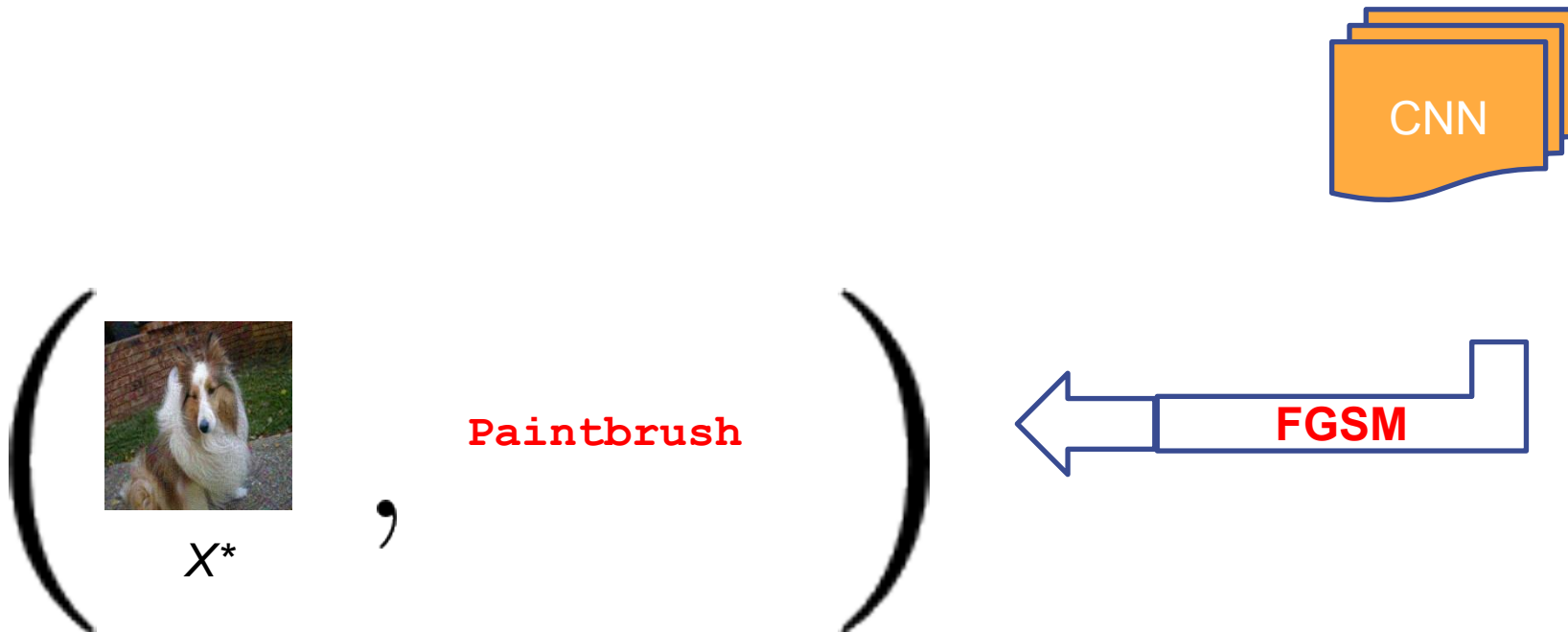
Adversarial Training



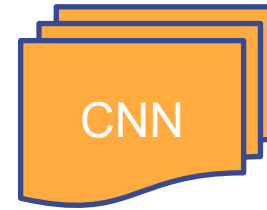
Adversarial Training



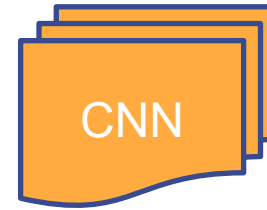
Adversarial Training



Adversarial Training



Adversarial Training



Adversarial Training



Adversarial Training – drawbacks

- ❖ Doubles train time
- ❖ Adv. Training over-fits
- ❖ Gradient masking
 - ❖ Pseudo robustness, Susceptible to black-box attacks

Ensemble Adversarial Training

- ❖ Diversity in the adversarial samples \rightarrow robustness to black-box attacks
- ❖ Have an ensemble of N pre-trained models \rightarrow seed x^* for Adv. Training
- ❖ Now, train $N+1^{\text{st}}$ model with x^* from itself and ensemble
- ❖ Increases robustness compared to x^* from single model

Ensemble Adversarial Training – Cons

- ❖ Highly inefficient
- ❖ Need to train N models first \rightarrow can't scale

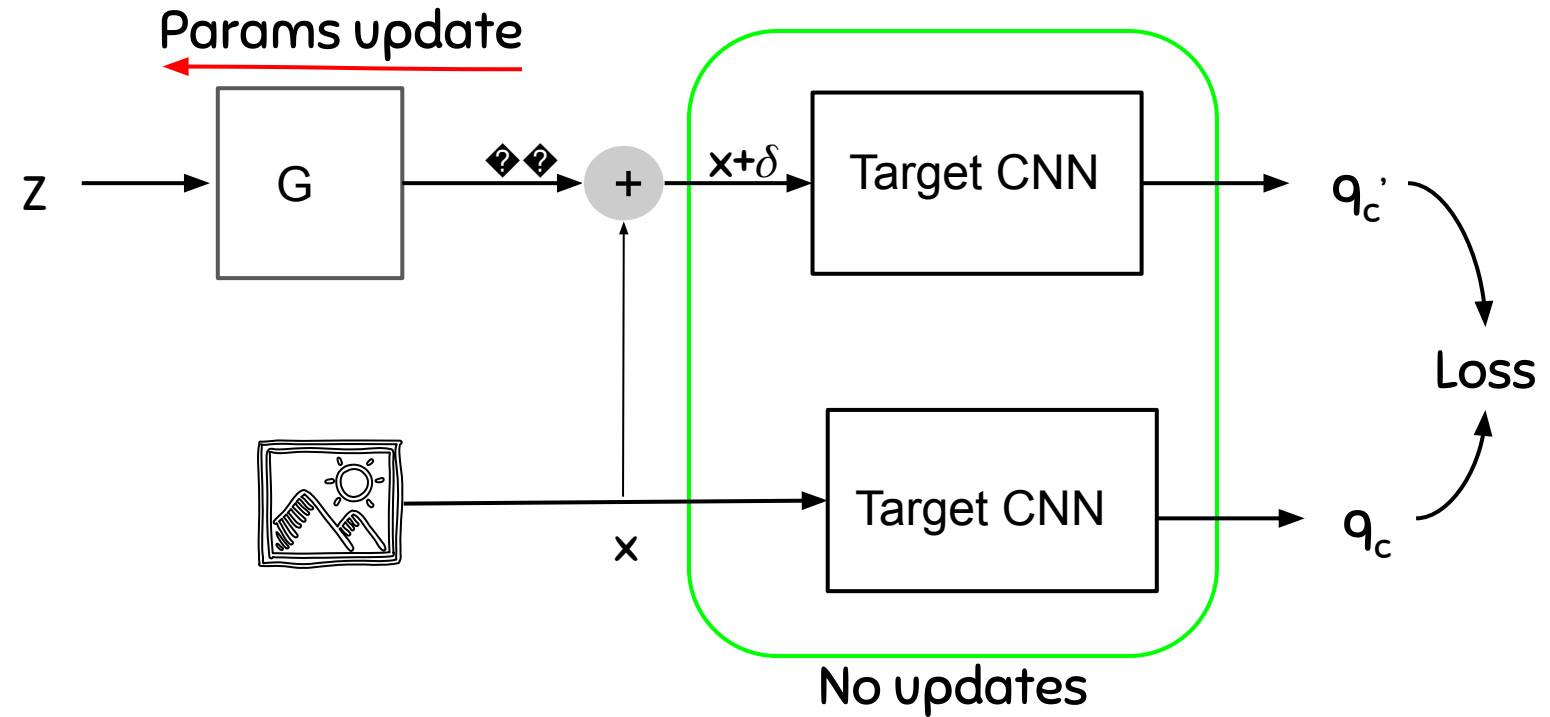
Can we model the distribution
of adversaries ?

Modelling the perturbations

- Can we learn a generative model that models the perturbations for a given classifier ?
- Generative model \rightarrow Infinite perturbations \rightarrow lot of diversity \rightarrow better adv. Training \rightarrow robust models

$$\Delta = \{\delta : \hat{k}(x + \delta) \neq \hat{k}(x) \text{ for } x \sim \mathcal{X} \text{ and } \|\delta\|_{\infty} < \xi\}$$

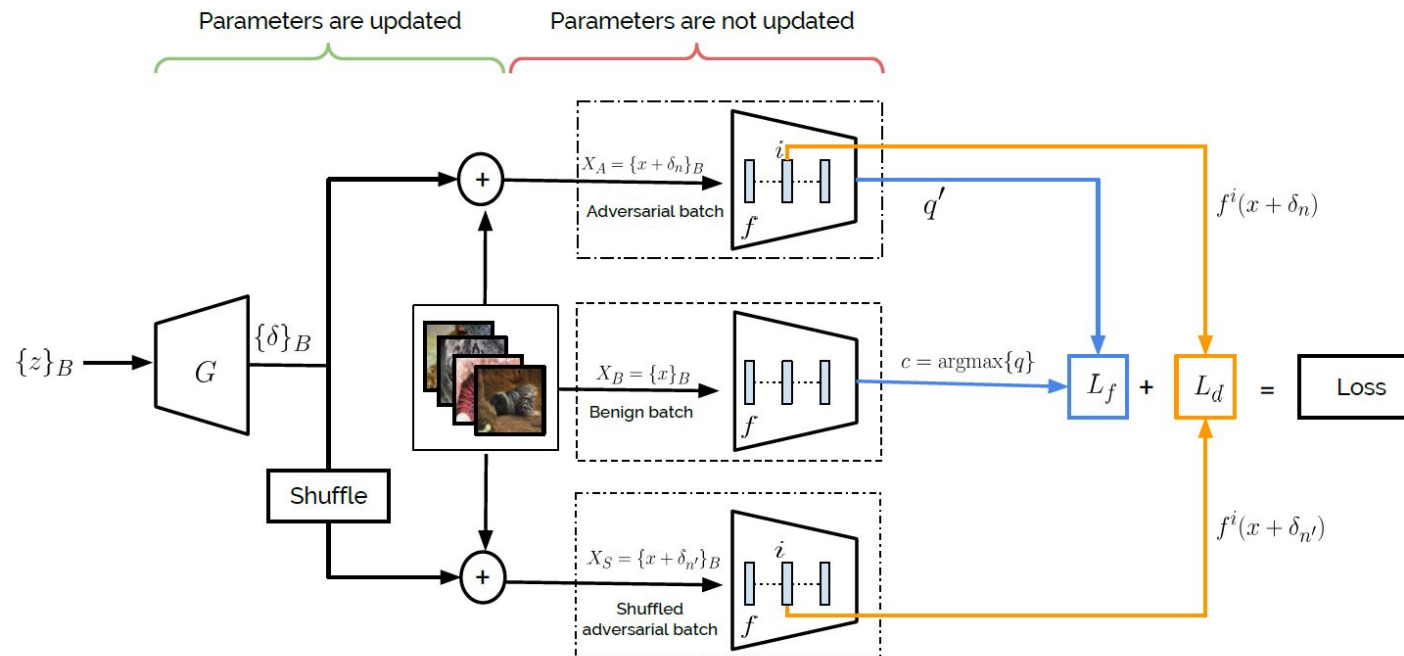
NAG: Network for Adversary Generation



NAG: Network for Adversary Generation

□ First ever generative approach to model the UAPs

□ Inspired from GANs



NAG: Network for Adversary Generation

- ❑ Fooling loss \rightarrow ability to fool the classifier
- ❑ Imposes diversity \rightarrow explores the manifold and learns to generate

$$L_f = -\log(1 - q'_c) \qquad L_d = - \sum_{n=1}^B d(f^i(x_n + \delta_n), f^i(x_n + \delta_{n'}))$$

$$Loss = L_f + \lambda L_d$$

NAG Fooling Rates

□ Fooling rates

		VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19	ResNet-50	ResNet-152	Mean FR
VGG-F	Our	94.10 \pm 1.84	81.28 \pm 3.50	64.15 \pm 3.41	52.93 \pm 8.50	55.39 \pm 2.68	50.56 \pm 4.50	47.67 \pm 4.12	63.73
	UAP	93.7	71.8	48.4	42.1	42.1	-	47.4	57.58
CaffeNet	Our	79.25 \pm 1.44	96.44 \pm 1.56	66.66 \pm 1.84	50.40 \pm 5.61	55.13 \pm 4.15	52.38 \pm 3.96	48.58 \pm 4.25	64.12
	UAP	74.0	93.3	47.7	39.9	39.9	-	48.0	56.71
GoogLeNet	Our	64.83 \pm 0.86	70.46 \pm 2.12	90.37 \pm 1.55	56.40 \pm 4.13	59.14 \pm 3.17	63.21 \pm 4.40	59.22 \pm 1.64	66.23
	UAP	46.2	43.8	78.9	39.2	39.8	-	45.5	48.9
VGG-16	Our	60.56 \pm 2.24	65.55 \pm 6.95	67.38 \pm 4.84	77.57 \pm 2.77	73.25 \pm 1.63	61.28 \pm 3.47	54.38 \pm 2.63	65.71
	UAP	63.4	55.8	56.5	78.3	73.1	-	63.4	65.08
VGG-19	Our	67.80 \pm 2.49	67.58 \pm 5.59	74.48 \pm 0.94	80.56 \pm 3.26	83.78 \pm 2.45	68.75 \pm 3.38	65.43 \pm 1.90	72.62
	UAP	64.0	57.2	53.6	73.5	77.8	-	58.0	64.01
ResNet-50	Our	47.06 \pm 2.60	63.35 \pm 1.70	65.30 \pm 1.14	55.16 \pm 2.61	52.67 \pm 2.58	86.64 \pm 2.73	66.40 \pm 1.89	62.37
	UAP	-	-	-	-	-	-	-	-
ResNet-152	Our	57.66 \pm 4.37	64.86 \pm 2.95	62.33 \pm 1.39	52.17 \pm 3.41	53.18 \pm 4.16	73.32 \pm 2.75	87.24 \pm 2.72	64.39
	UAP	46.3	46.3	50.5	47.0	45.5	-	84.0	53.27

NAG: Multi-target scenario and effect of data

- ❑ Can model distribution that can simultaneously fool multiple CNNs
- ❑ Modelling gets better with available data samples for crafting

Table 3. Mean fooling rates for 10 perturbations sampled from the distribution of adversaries modelled for multiple target CNNs. The perturbations result an average fooling rate of 80.02% across the 7 target CNNs which is higher than the best mean fooling rate of 72.62% achieved by the generator learned for VGG-19.

Network	VGG-F	CaffeNet	GoogLeNet	VGG-16	VGG-19	ResNet-50	ResNet-152
Fooling rate	83.74	86.94	84.79	73.73	75.24	80.21	75.84

Table 4. Generalizability of the perturbations learned by the ensemble generator (G_E).

	G_{VF}	G_G	G_{V16}	G_{R50}	G_E
Mean BBFR	60.63	60.15	71.26	61.87	76.40

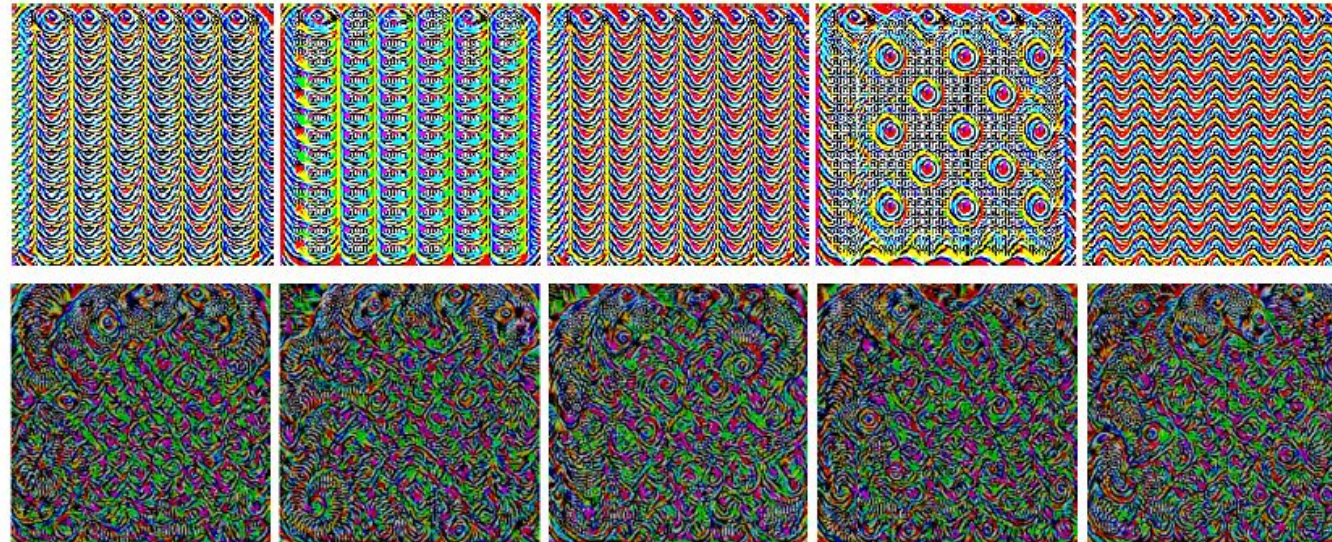
NAG: Diversity in generation

□ L_D helps to craft variety of UAPs as opposed to UAP

□ 95% of the predicted labels belong to

▸ UAP → 173 categories

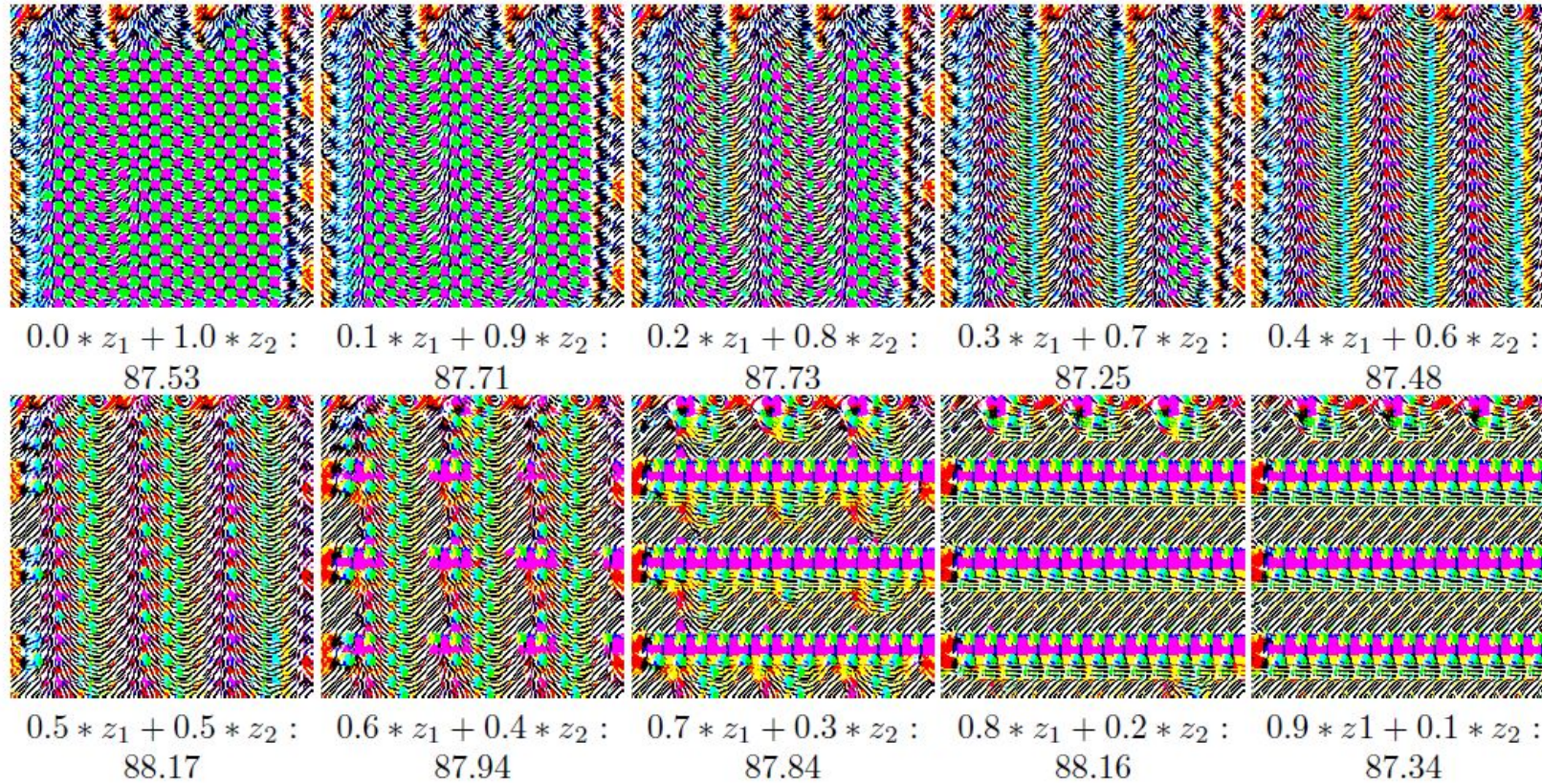
▸ NAG → 257



NAG, CVPR 2018

UAP, CVPR 2017

NAG: Traversing the manifold



Conclusion

- ❑ Adversarial perturbations are an intriguing aspect of ML and Neural nets
- ❑ Gradient is a “Double edged Knife”
- ❑ Attacking an ML system is easier than defending (?)
- ❑ We have built systems that are very good but easily broken
 - ❑ Need rigorous understanding of the models and training procedure
- ❑ The setup of current attacks seem to have issues
 - ❑ E.g., how to measure imperceptibility for humans ?

Discuss

With thanks.

Resources

- ☐ Papers and blogs from Ian Goodfellow, Nicolas Papernot and Moosavi Dez-Fooli, etc.
- ☐ Workshops and tutorials of Vision conferences such as CVPR, ICCV, NIPS, BMVC, etc.
- ☐ OpenAI blog, Reddit/MachineLearning, etc.
- ☐ <http://www.cleverhans.io/>
- ☐ <https://papernot.fr/>
- ☐ <https://arxiv.org/abs/1707.05572>
- ☐ <https://arxiv.org/pdf/1412.5068.pdf>
- ☐