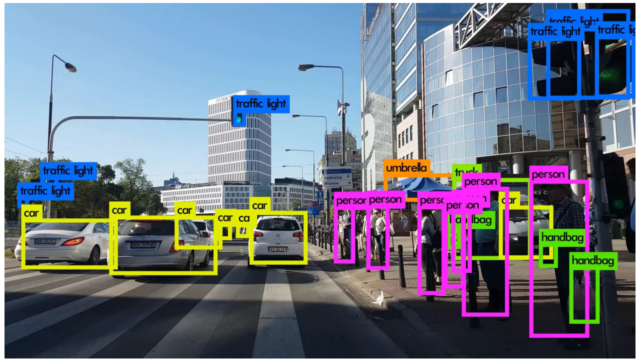
Seminar

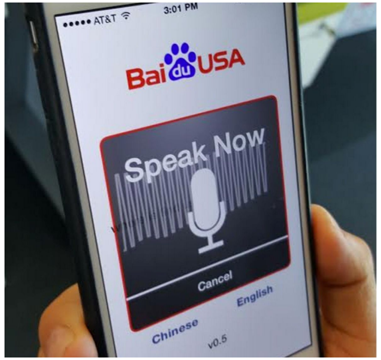
Adversarial Examples: Attacks and Defences

Marco Ciccone

Department of Electronics, Informatics and Bioengineering
Politecnico di Milano

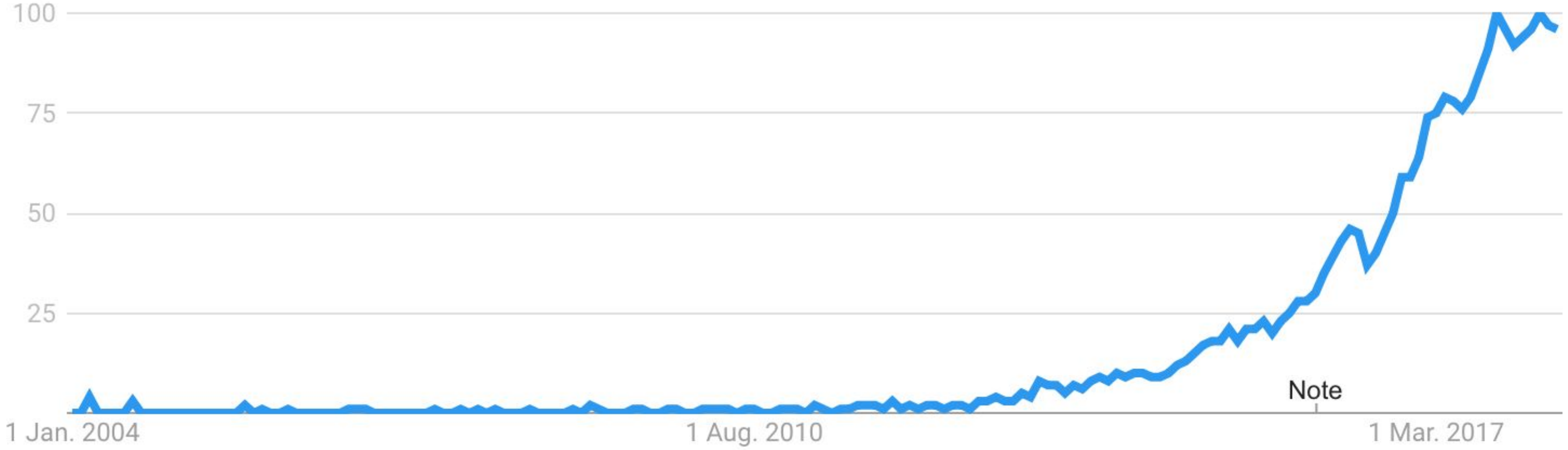
Deep Learning at scale since 2013...



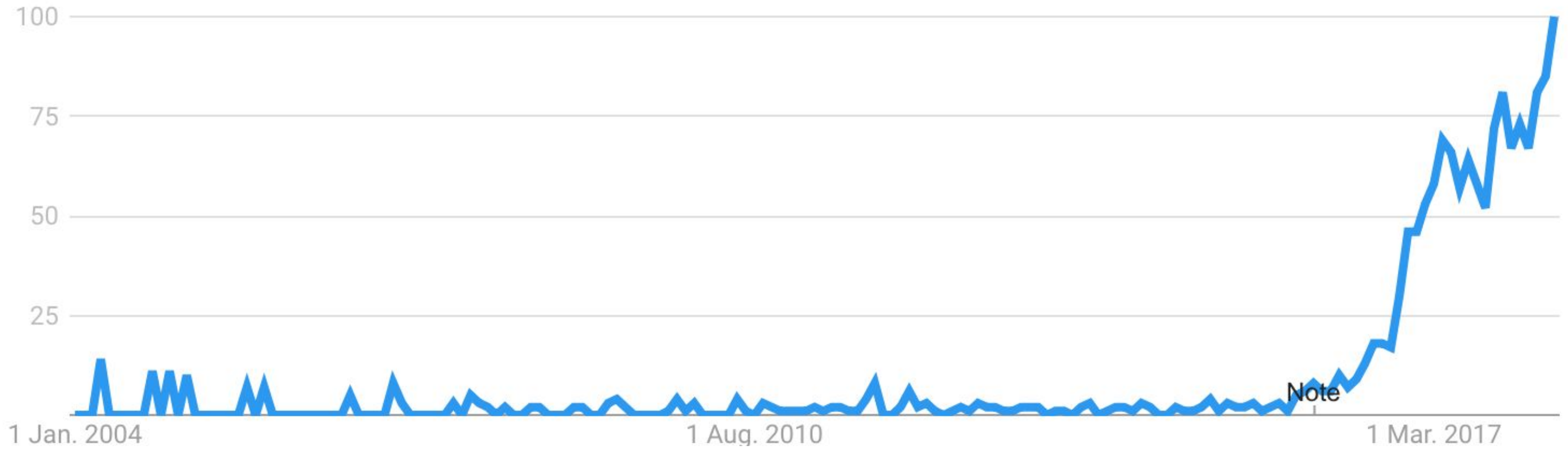


RNN handwriting generation demo

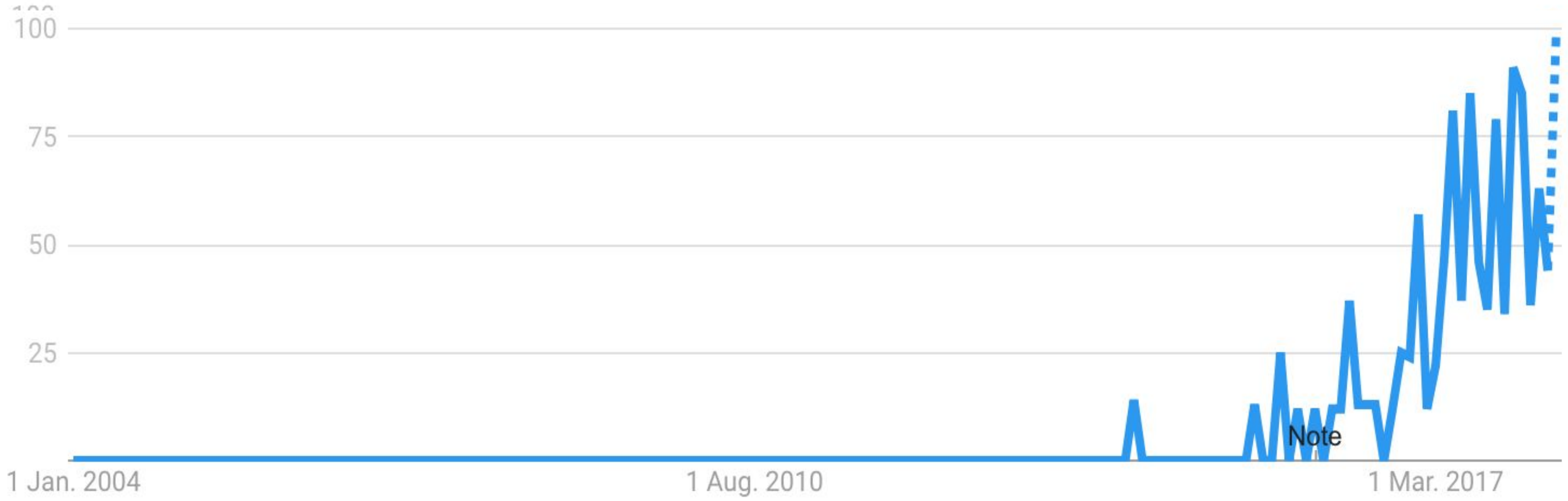
“Deep Learning” Trend



“Adversarial” Trend

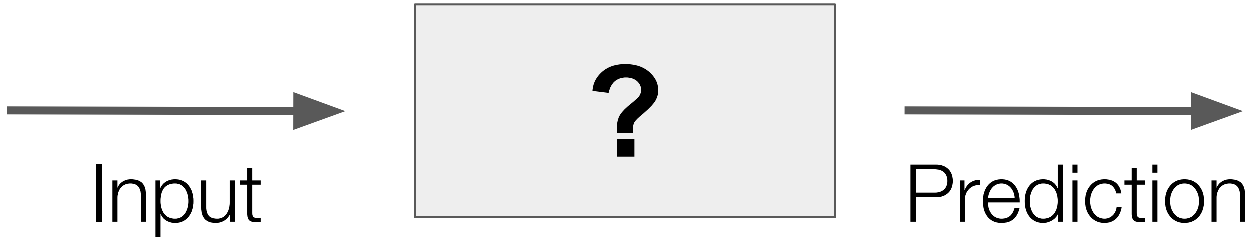


“Adversarial Examples” Trend



Why Deep Learning works so well?

Black-box models



Model Parameters θ
learnt from data

How can these model parameters be interpreted?

We would like to know how NNs compute their final answer.

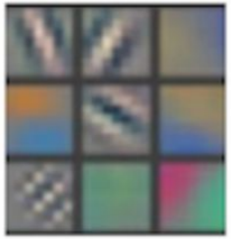
Feature/Weights Visualization is important

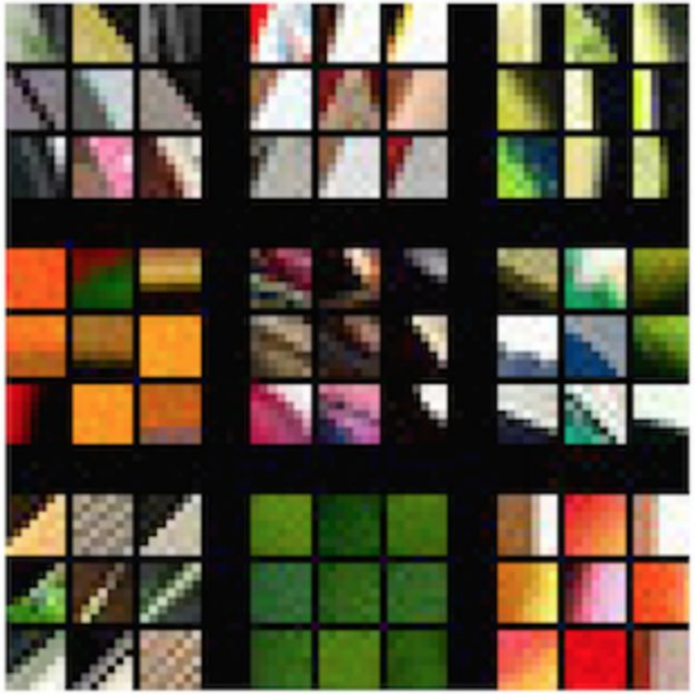
Mapping between a neuron in a layer to the features in the image.

- Understand how and why neural networks work
- Observe the evolution of features during training
- Aid the development of better models (rather than just trial-and-error)
- Diagnose potential problems with the model

Visualize the weights: First Layer

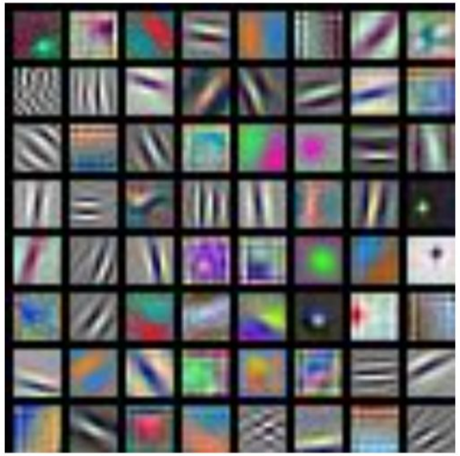
For Neurons in the first hidden layer, we can visualize the weights.





A neuron would be activated the most if the input looks like the weight matrix.

Visualize the weights: First Layer


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

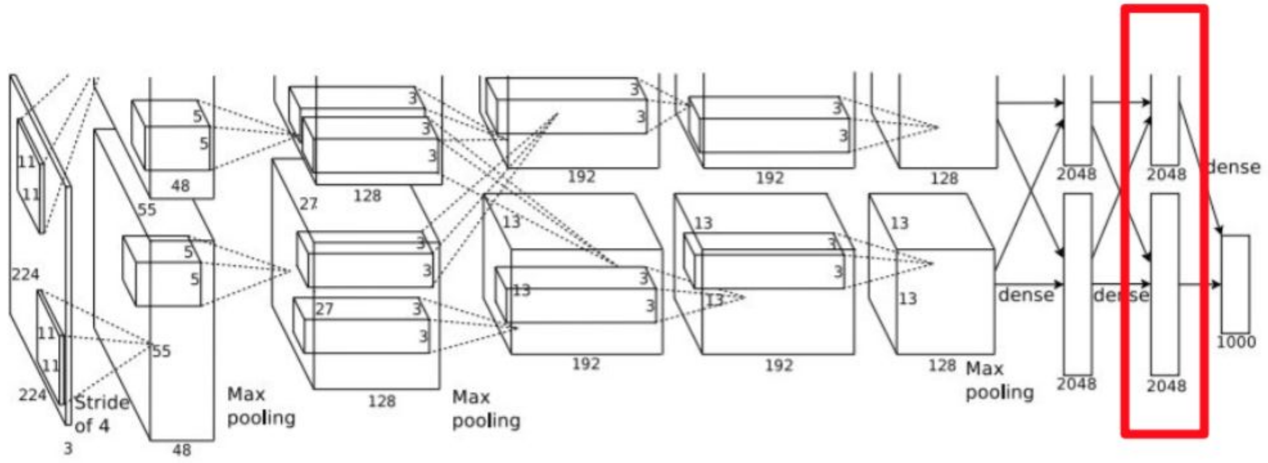
ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

Very interpretable, applied directly to the pixels. They indicates edge/blob detectors.
We can also visualize filters at higher layers, but not that interesting.

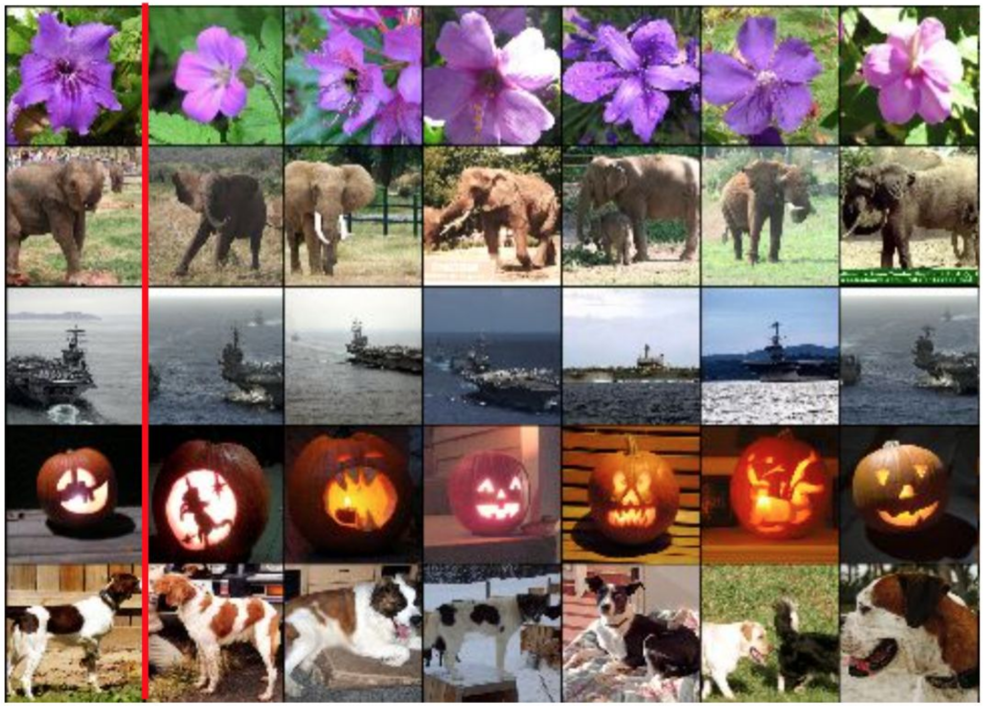
Visualize the weights: Last Layer

- Take the 4096-dimensional feature vector for an image (layer immediately before the classifier)
- Run the network on many images, collect the feature vectors.



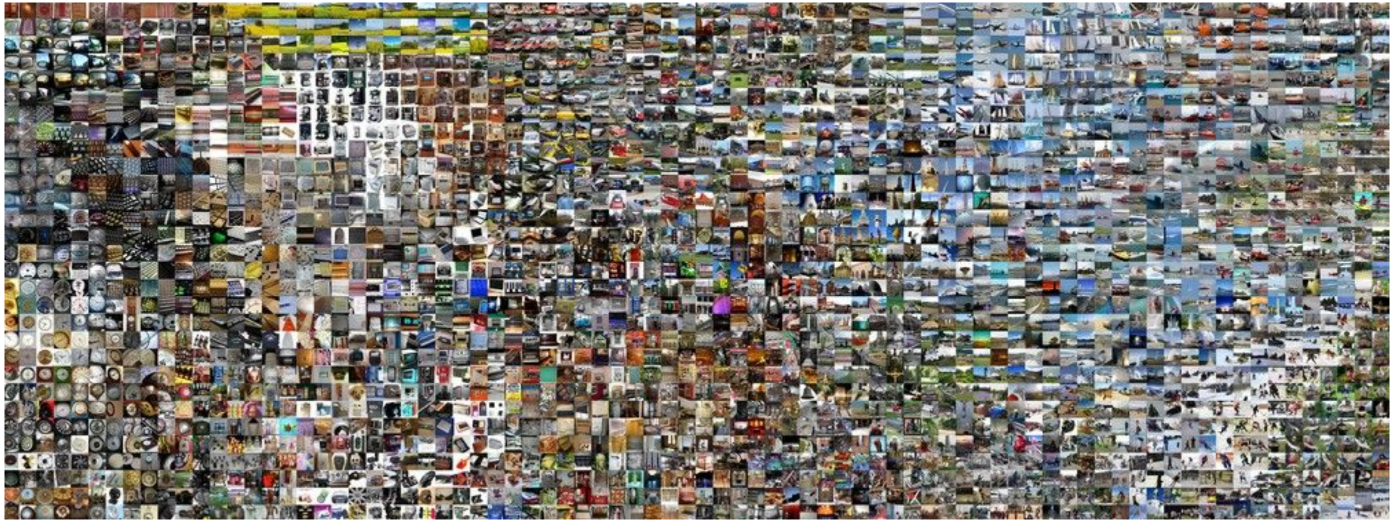
Last Layer: Nearest Neighbours

Test image L2 Nearest Neighbours in **Feature Space**



Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions with Principal Component Analysis (PCA) or t-SNE



Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based.

What about intermediate layers?

Visualize patches that maximally activate neurons

1) Pick a layer and a channel (**unit** of the network):

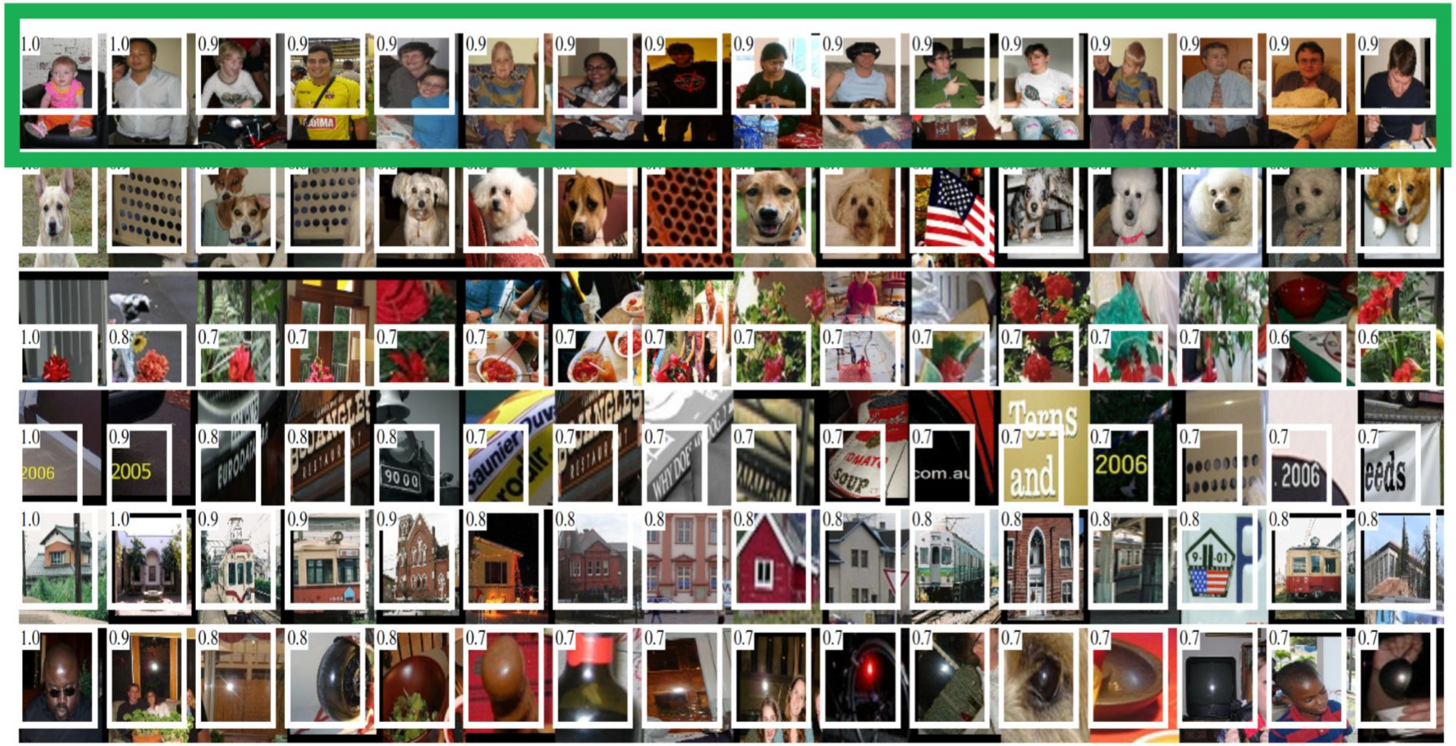
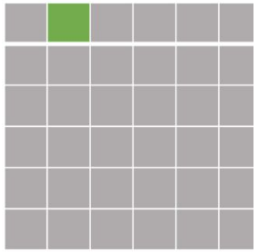
   e.g. conv5 is 128 x 13 x 13, pick channel 17/128

2) Run many images through the network and record activations values of chosen channel
3) Sort the proposals from highest to lowest activation, perform NMS
4) Visualize image patches that correspond to maximal activations

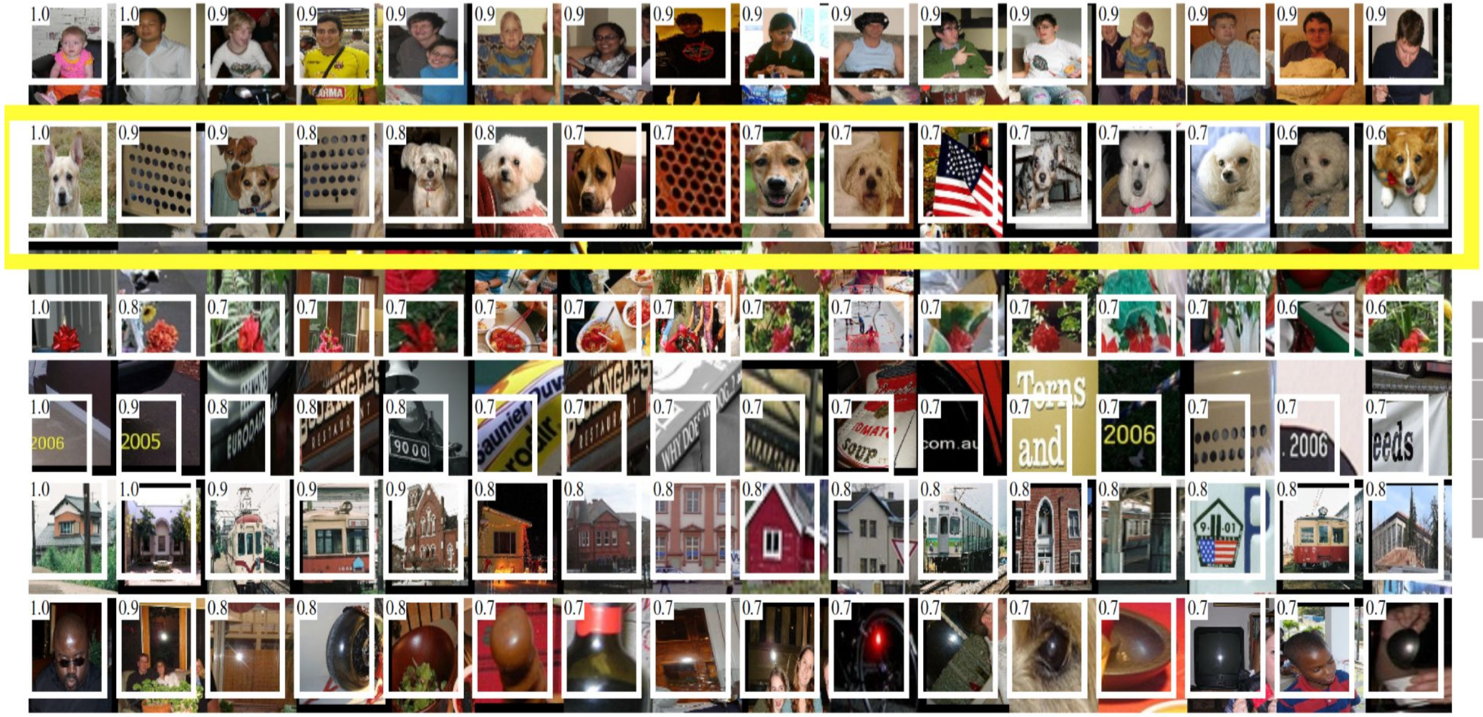
This method **lets the selected unit “speak for itself”** by showing exactly which inputs it fires on.

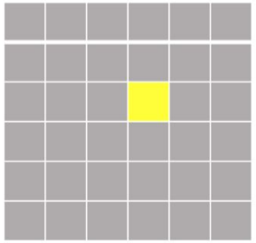
Rich feature hierarchies for accurate object detection and semantic segmentation, Girshick et al

Visualize patches that maximally activate neurons

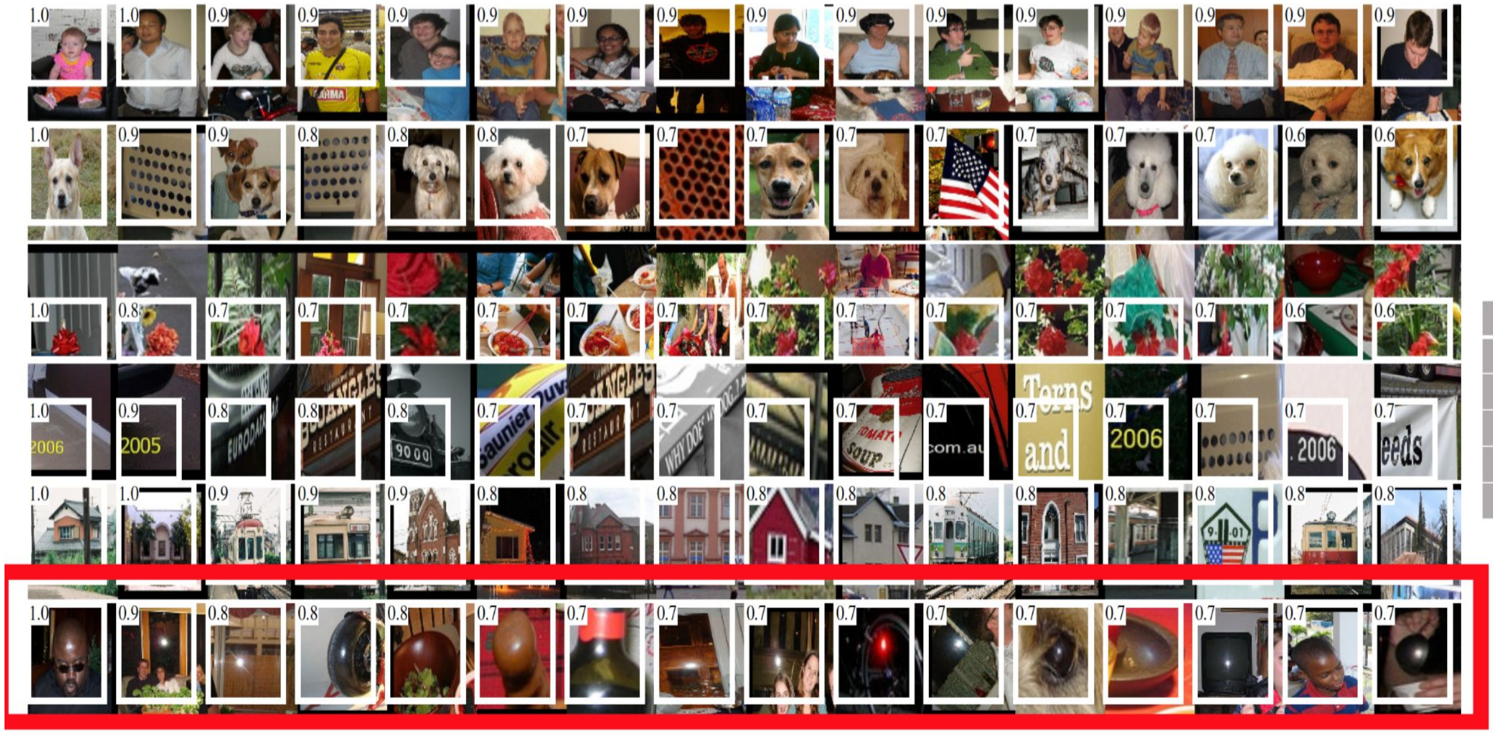



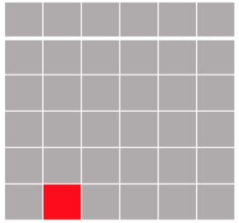
Visualize patches that maximally activate neurons





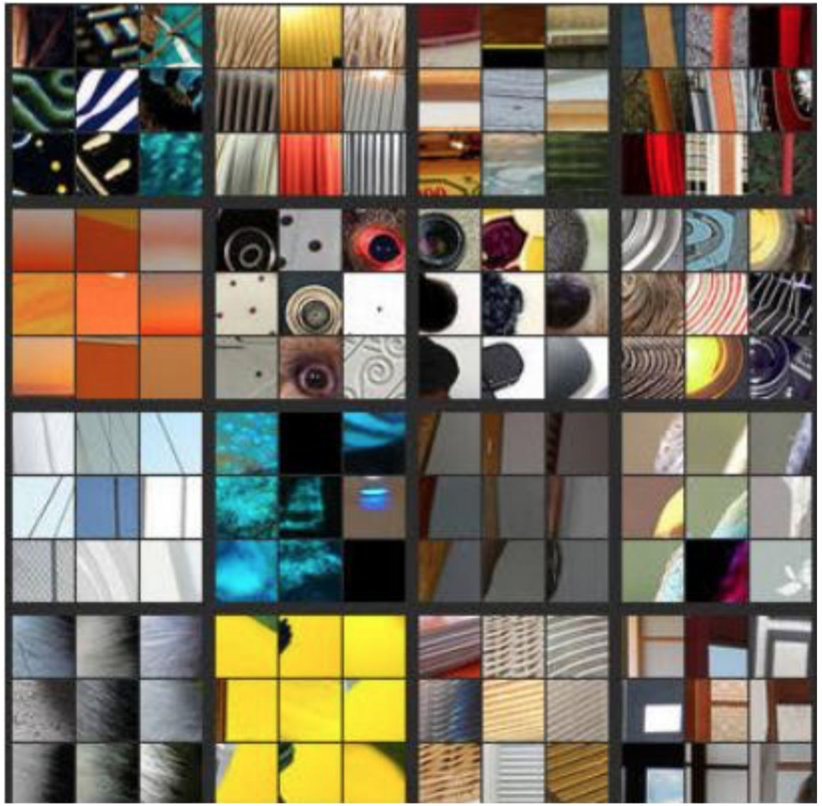
Visualize patches that maximally activate neurons





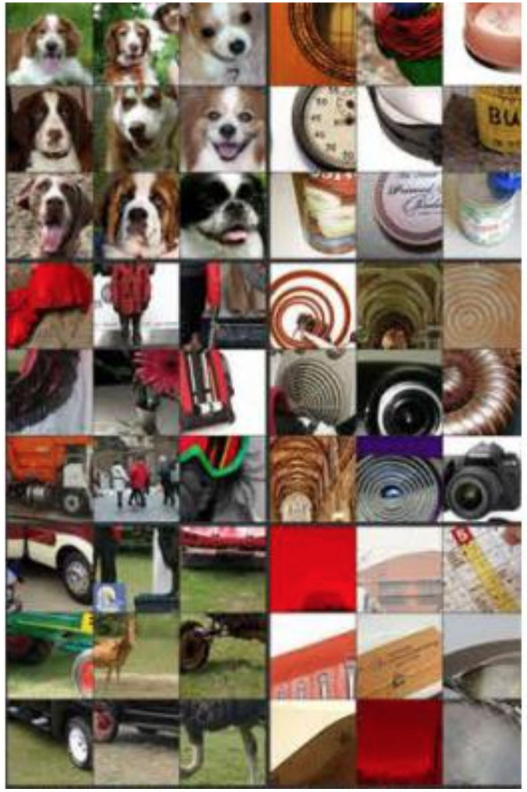
Example: Layer 2

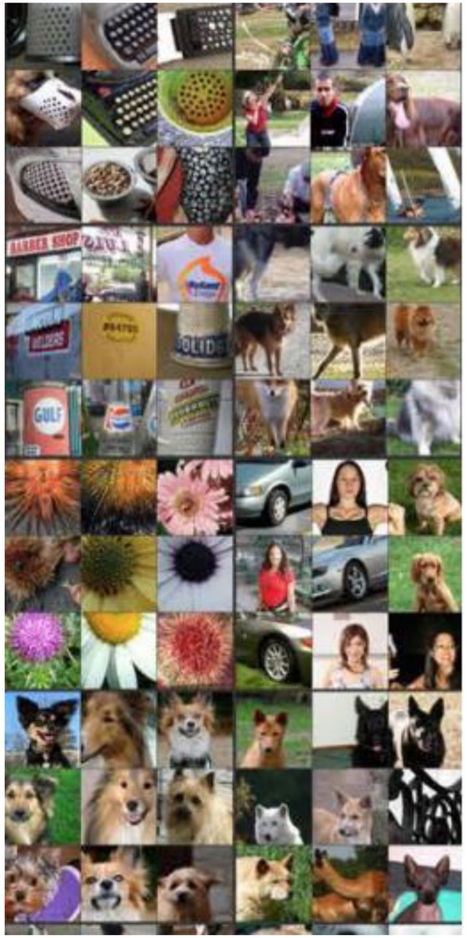
- We are interested in the original images, not in the actual input volumes at each layer processed by the filters.
- The effective “receptive field” for a neuron is larger than the filter itself (POOL + CONV), so it doesn’t make sense to directly compare the filters to the original images as in the first layer



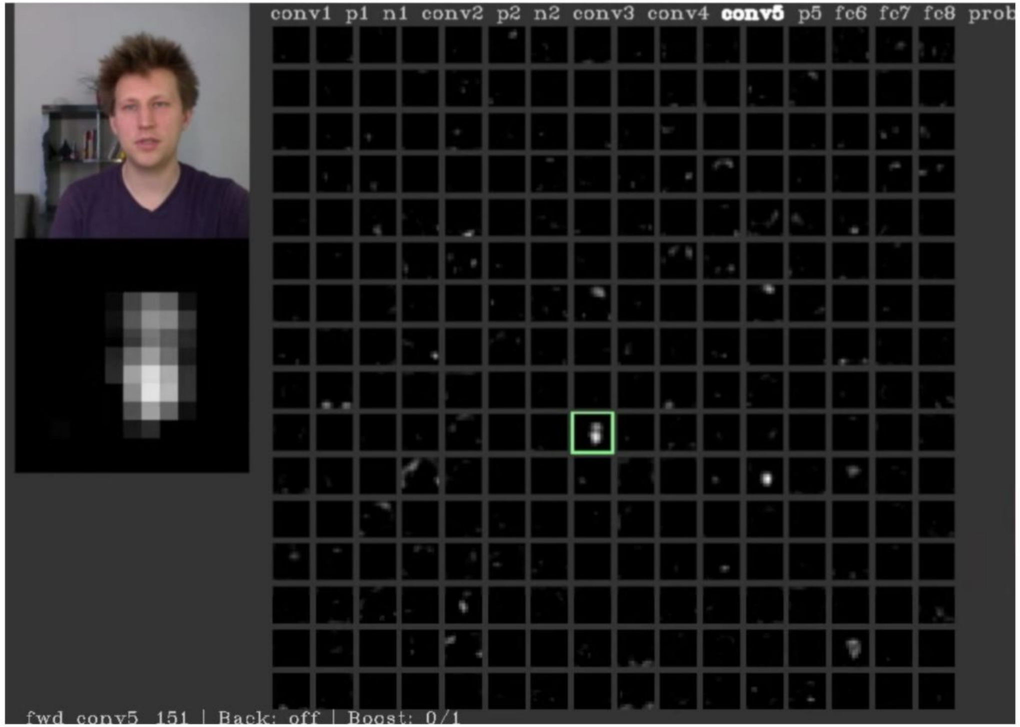
Maximally activated patches within each cell have much in common.

Example: Layers 4-5





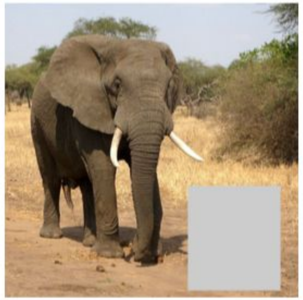
Visualize the activations

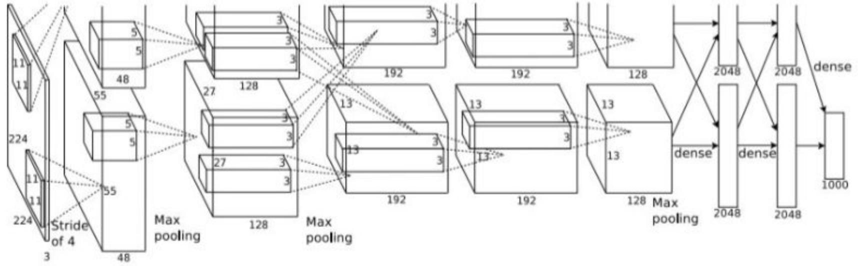

conv1 p1 n1 conv2 p2 n2 conv3 conv4 conv5 p5 fc6 fc7 fc8 prob
fwd conv5 151 | Back: off | Boost: 0/1

conv5 feature map is 128x13x13; visualized as 128 13x13 grayscale images

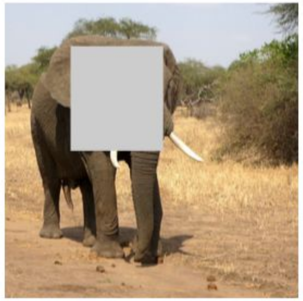
Which pixels matter: Occlusions

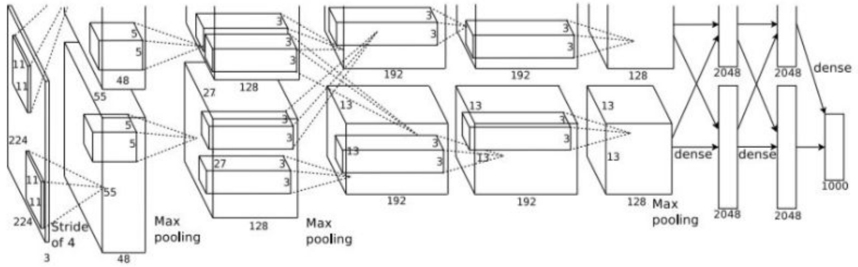
Mask part of the image before feeding to CNN, check how much predicted probabilities change.





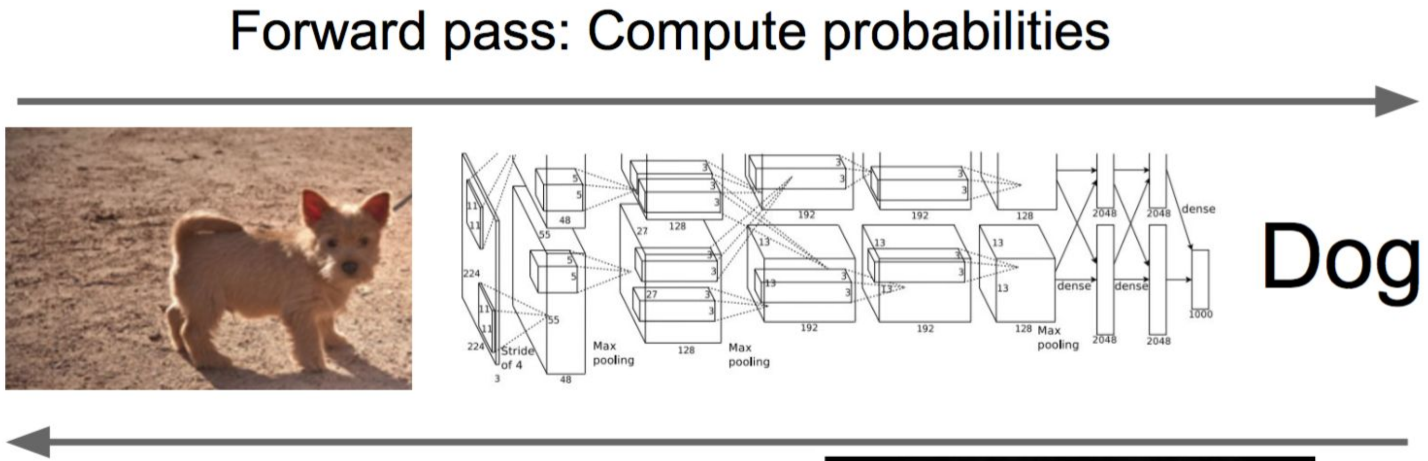
P(elephant) = 0.95

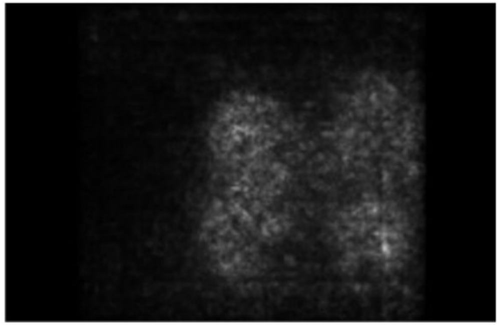




P(elephant) = 0.75

Which pixels matter: Saliency via Backprop

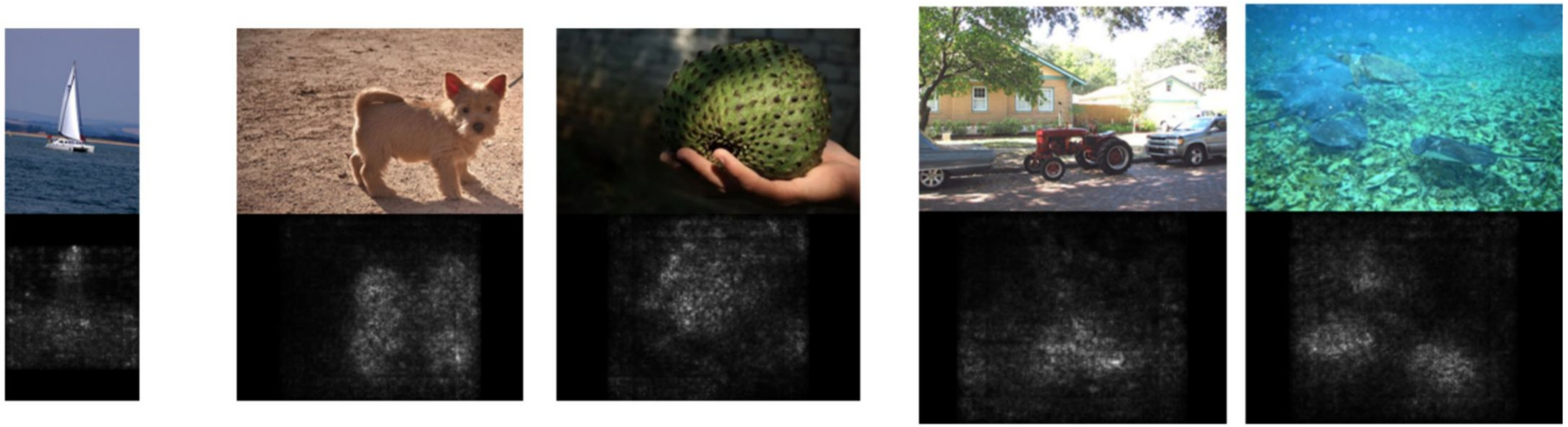




Compute **gradient of (unnormalized) class score with respect to image pixels**, take absolute value and max over RGB channels

[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al]

Saliency Maps



Deep Inside Convolutional Networks:
Visualising Image Classification Models and Saliency Maps, Simonyan et al

Which Pixels in the Input Affect the Neuron the Most?

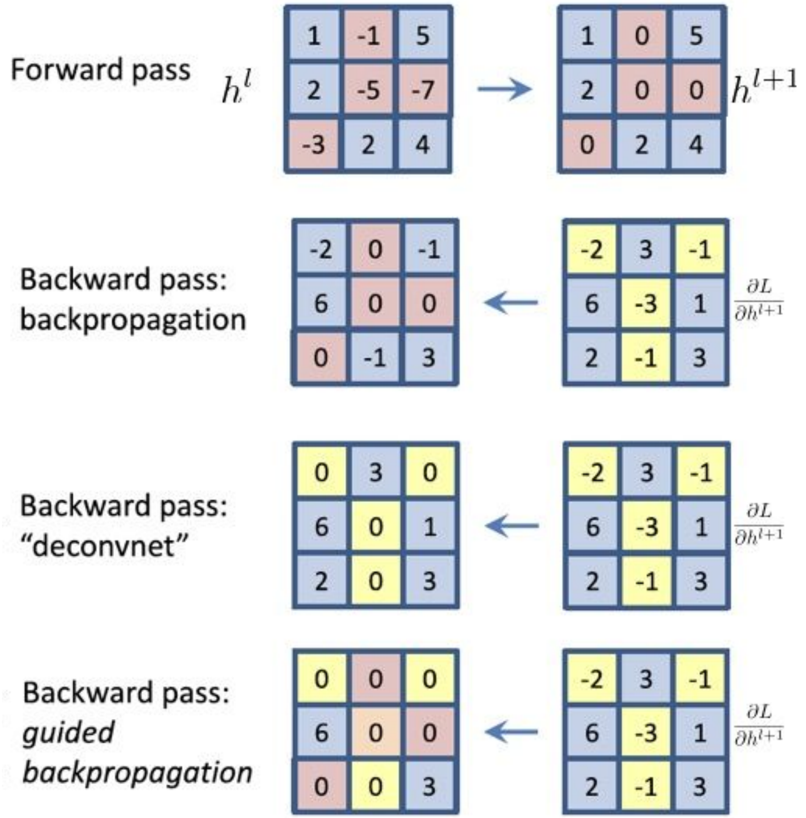
Pick a **single intermediate neuron**,
e.g. one value in 128 x 13 x 13 conv5 feature map.

Compute gradient of neuron value with respect to image pixels:

$$\frac{\partial neuron}{\partial x_i}$$

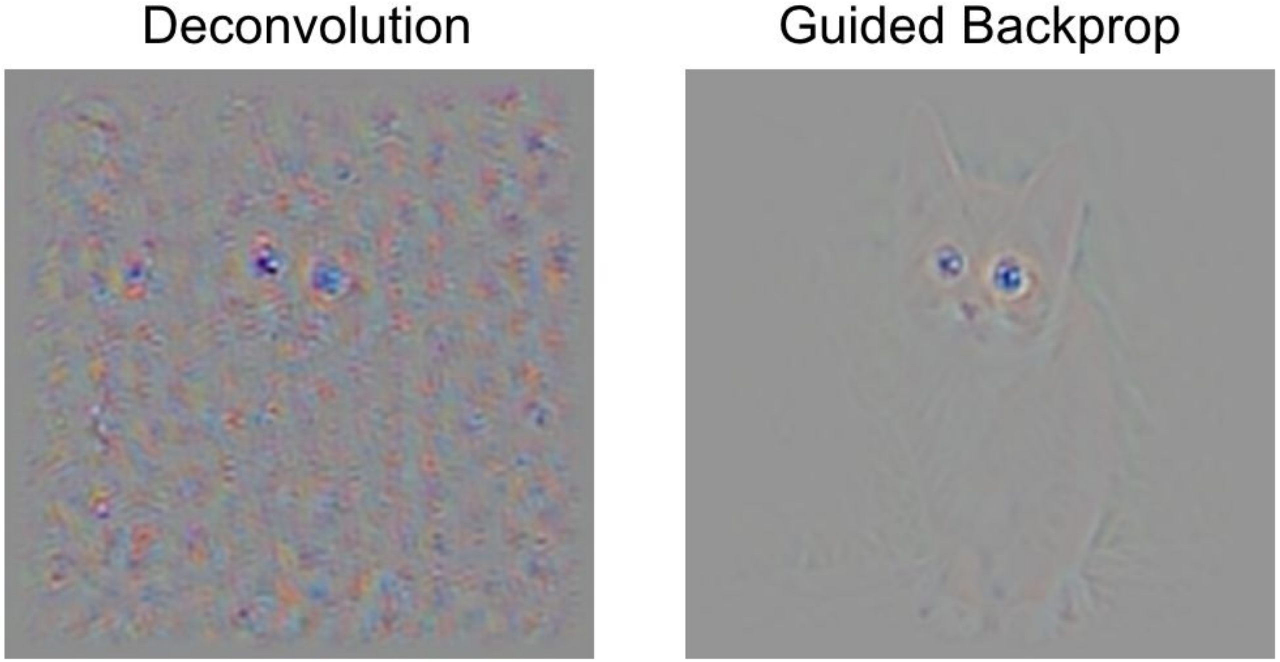
Gradients tell us how changes in x affect y. We can interpret the gradient as:

how does changing a pixel in the input image change the network's behaviour for that input.



Forward pass
h^l
1 -1 5
2 -5 -7
-3 2 4
1 0 5
2 0 0
0 2 4
h^{l+1}
Backward pass: backpropagation
-2 0 -1
6 0 0
0 -1 3
-2 3 -1
6 -3 1
2 -1 3
$\frac{\partial L}{\partial h^{l+1}}$
Backward pass: "deconvnet"
0 3 0
6 0 1
2 0 3
-2 3 -1
6 -3 1
2 -1 3
$\frac{\partial L}{\partial h^{l+1}}$
Backward pass: *guided backpropagation*
0 0 0
6 0 0
0 0 3
-2 3 -1
6 -3 1
2 -1 3
$\frac{\partial L}{\partial h^{l+1}}$

“Deconvolution” vs Guided Backpropagation



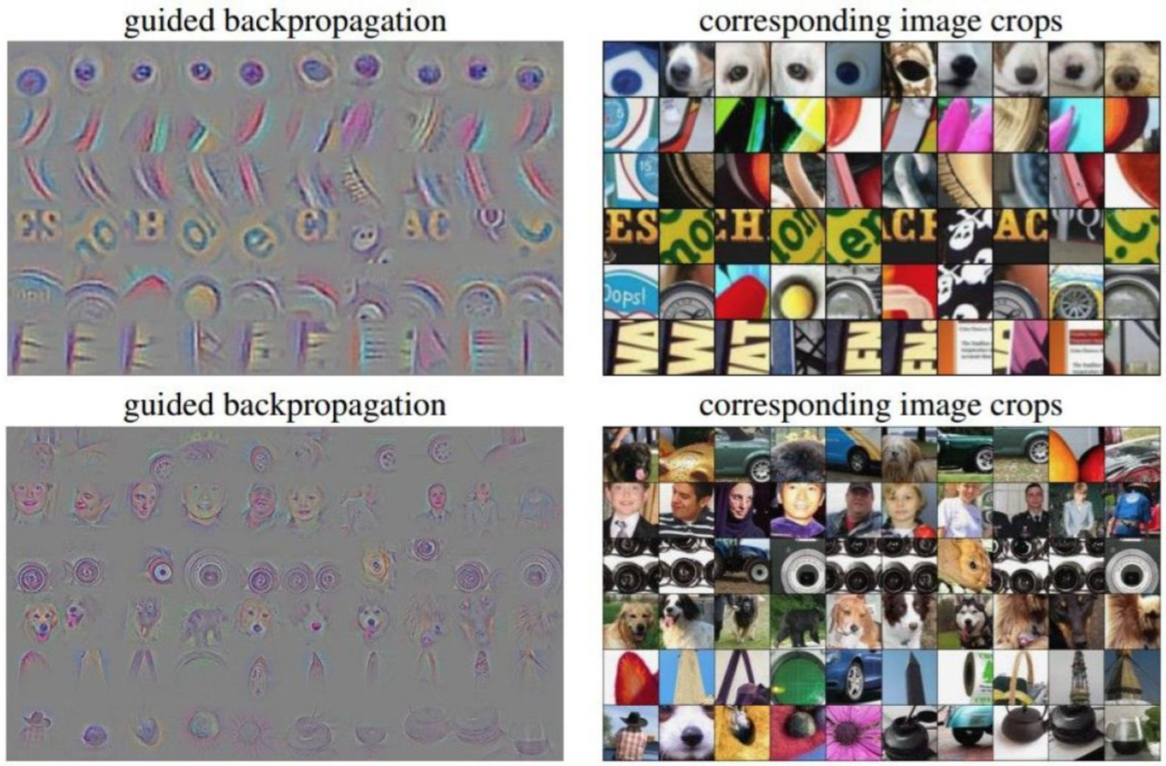
Guided Backpropagation

Idea: neurons act like detectors of particular image features

- We are only interested in what image features the neuron detects (Positive Gradients), not in what it doesn't detect (Negative Gradients).
- We don't care if a pixel "suppresses" a neuron somewhere along the part to our neuron.
- **Guided Backpropagation**, keeps only paths that lead to positive influence on the class score, and suppress the ones that have negative influence, leading to much cleaner looking images.

So when propagating the gradient, **all the negative gradients are set to 0.**

Guided Backpropagation



Can we synthesize images that maximize a specific unit activation?

Of course, with Gradient Optimization!

Gradient Ascent

(Guided) backprop: Find the part of an image that a neuron responds to

Gradient ascent: Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I \boxed{f(I)} + \boxed{R(I)}$$

Neuron value   Natural image regularizer

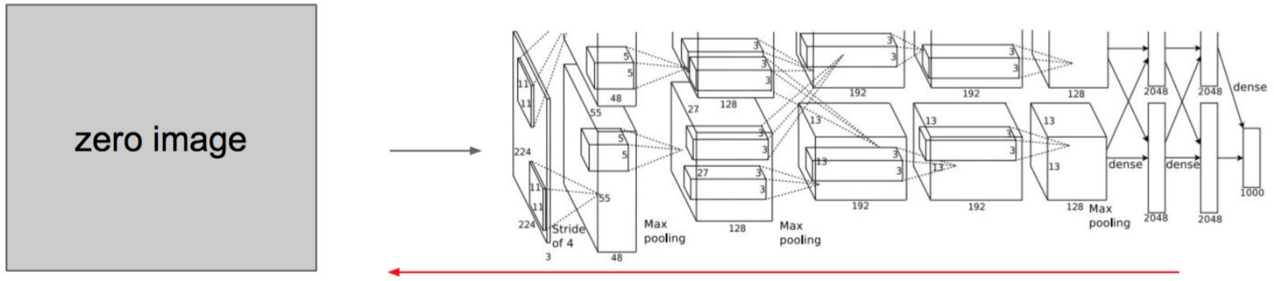
$$x \leftarrow x + \alpha \cdot \partial a_i(x)/\partial x$$

Gradient Ascent

$$\arg\max_{I} \boxed{S_c(I)} - \lambda\|I\|_2^2$$

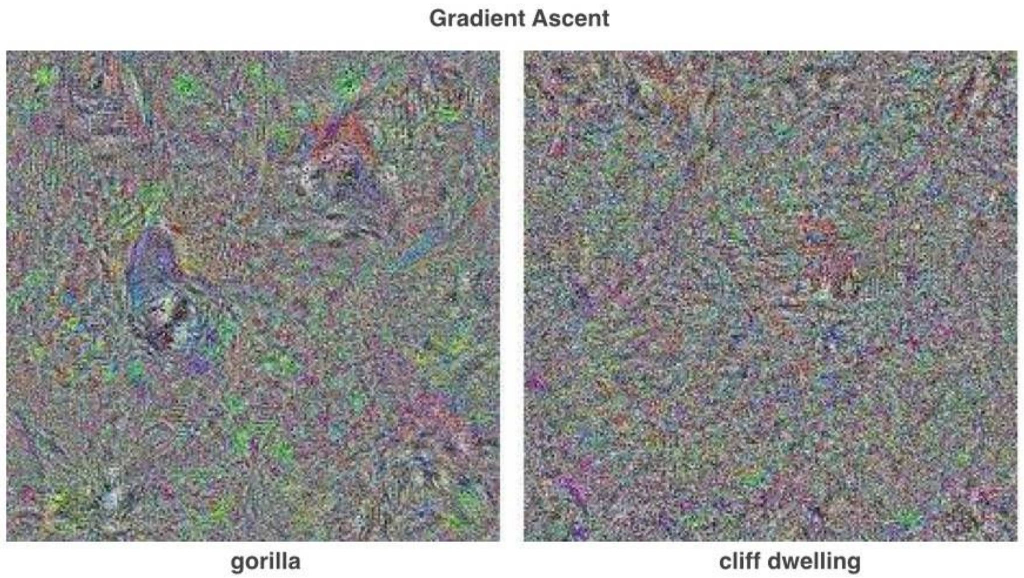
score for class c (before Softmax)

1. Initialize image to zeros



Repeat:
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

Visualizing Neurons without Regularization/Priors

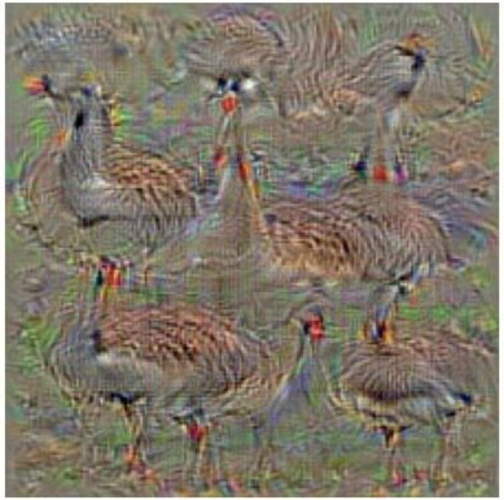


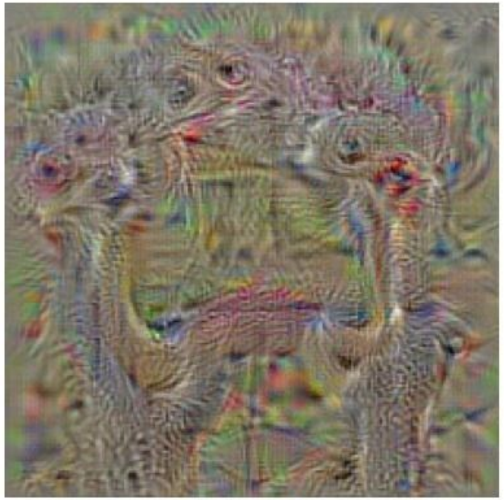
Deep Neural Networks are Easily Fooled, Nguyen et al.

Visualizing Neurons with Weak Regularization/Priors

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Simple regularizer: Penalize L2 norm of generated image


goose


ostrich

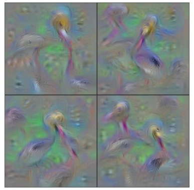
[Deep Inside Convolutional Networks: Visualising Image Classification, Simonyan at al]

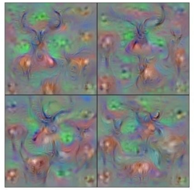
Visualizing Neurons with Better Regularization/Priors

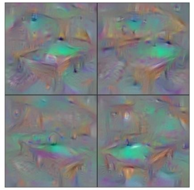
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

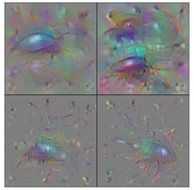
Better regularizer: Penalize L2 norm of image; also during optimization periodically
(1) Gaussian blur image or Total Variation
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0

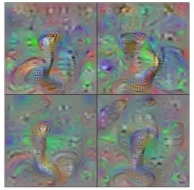
Flamingo

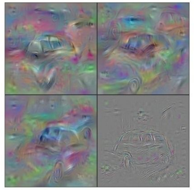

Pelican

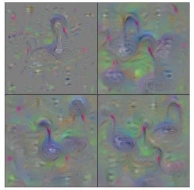

Hartebeest

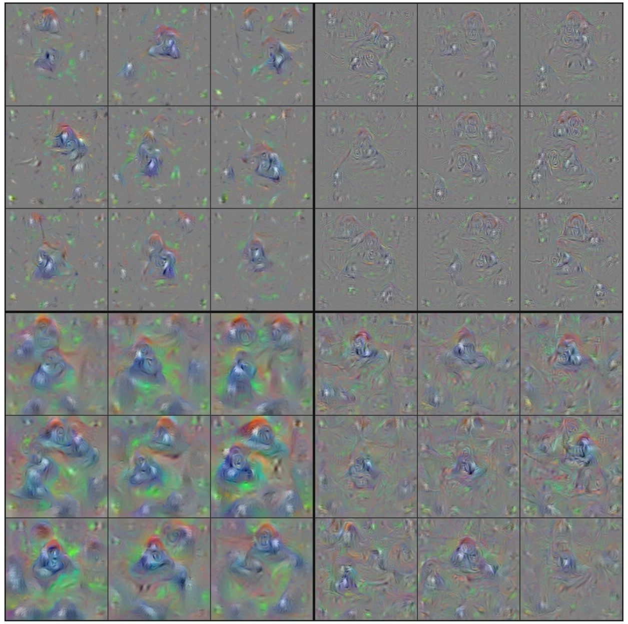

Billiard Table


Ground Beetle


Indian Cobra

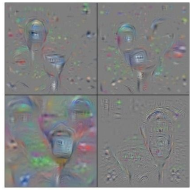

Station Wagon


Black Swan


Gorilla

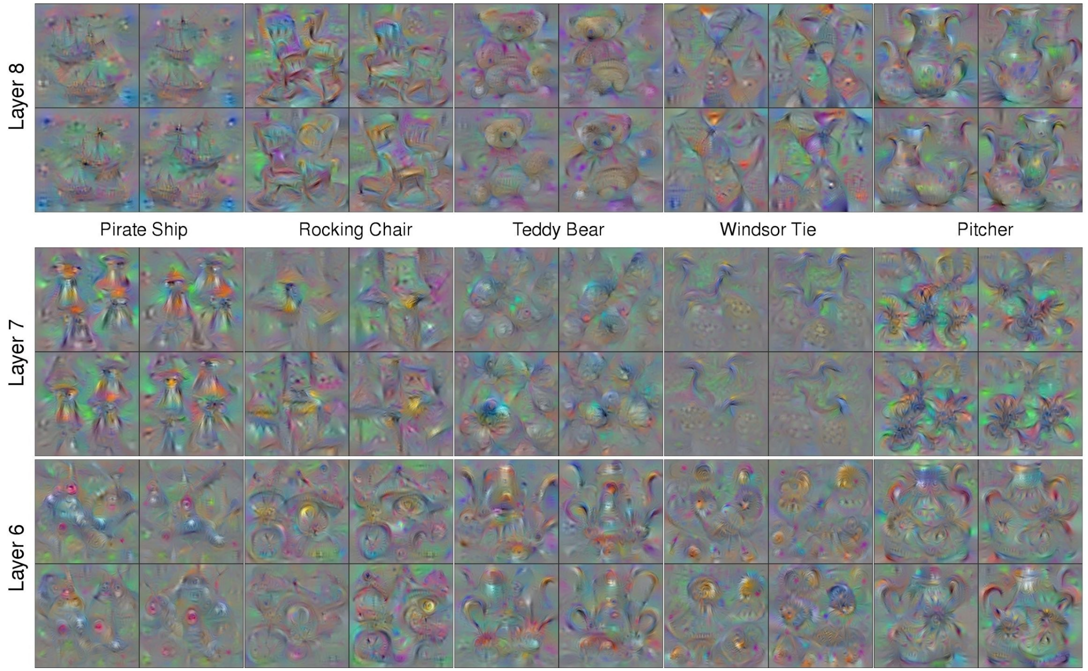
School Bus

Tricycle

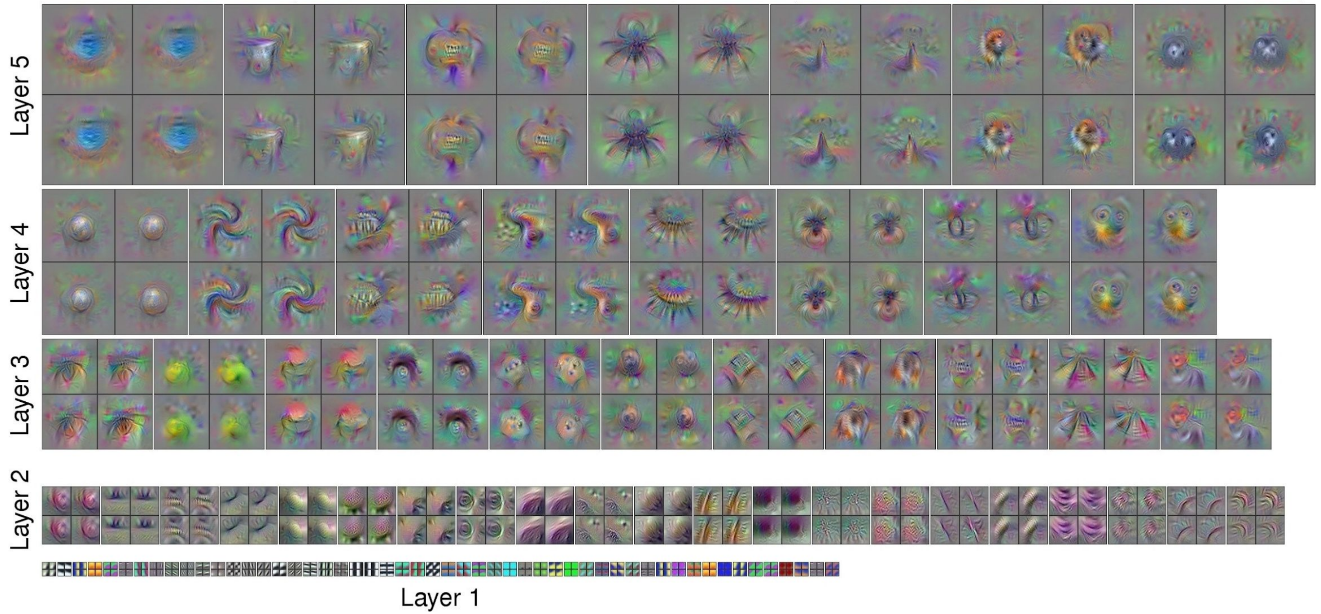

Parking Meter

[Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.]

Visualizing Neurons with Better Regularization/Priors



Visualizing Neurons with Better Regularization/Priors



https://distill.pub/2017/feature-visualization/

Can we use these generated images to fool Neural Networks?

Unfortunately, yes.

Adversarial Examples

Szegedy et al. showed that despite their high performances in terms of accuracy, modern DNNs are **surprisingly susceptible to adversarial attacks in the form of small perturbations** to images that remains (almost) imperceptible to human vision.

That means, adding a well designed noise to an image:

- A Neural Network classifier can completely change its prediction.
- The attacked model report high confidence on the wrong prediction.
- **The same perturbation can fool different classifiers.**

Siberian Husky

Husky

Dog

Alaskan Malamute

Husky

Dog

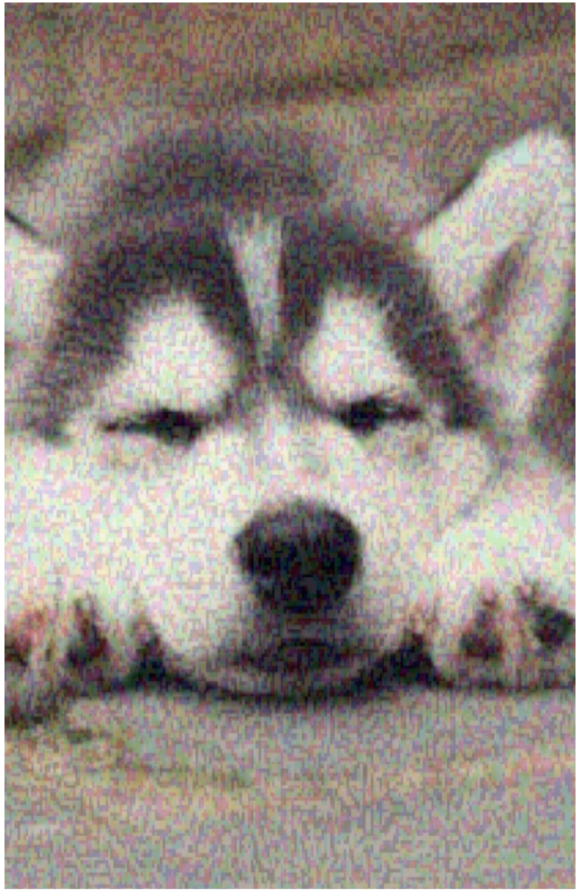
Alaskan Husky

Husky

Dog

Adversarial Noise

+



Husky

Bananaaa!

Timeline

- “Adversarial Classification” Dalvi et al 2004: fool spam filter
- “Evasion Attacks Against Machine Learning at Test Time” Biggio 2013: fool neural nets
- Szegedy et al 2013: fool ImageNet classifiers imperceptibly
- Goodfellow et al 2014: cheap, closed form attack

Many other works from this point …

DL in safety-critical environments

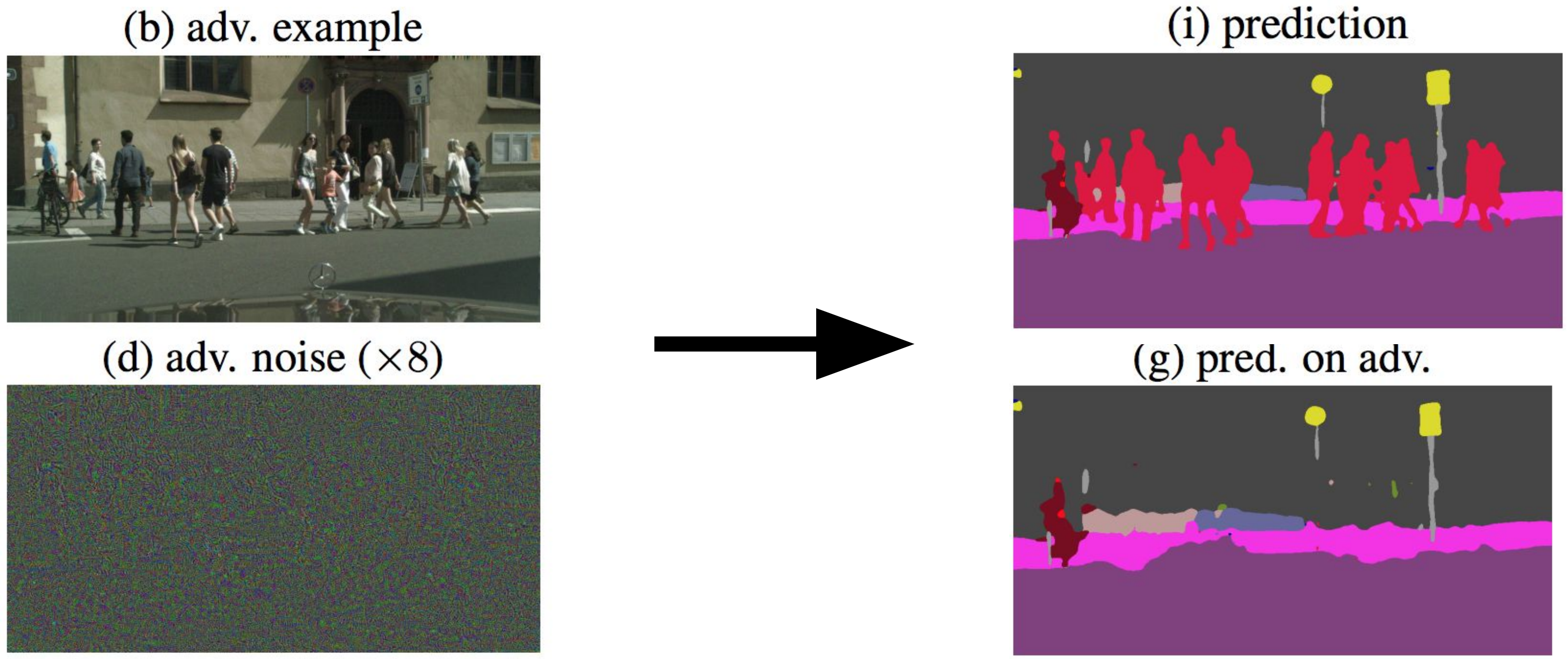
DL is mature enough to be deployed and playing a major role in safety-critical environments:

- Self-driving cars
- Surveillance
- Malware Detection
- Drones and Robotics
- Voice Command Recognition
- Facial Recognition in ATM or FaceID in mobile phones

Examples of issues in real world applications

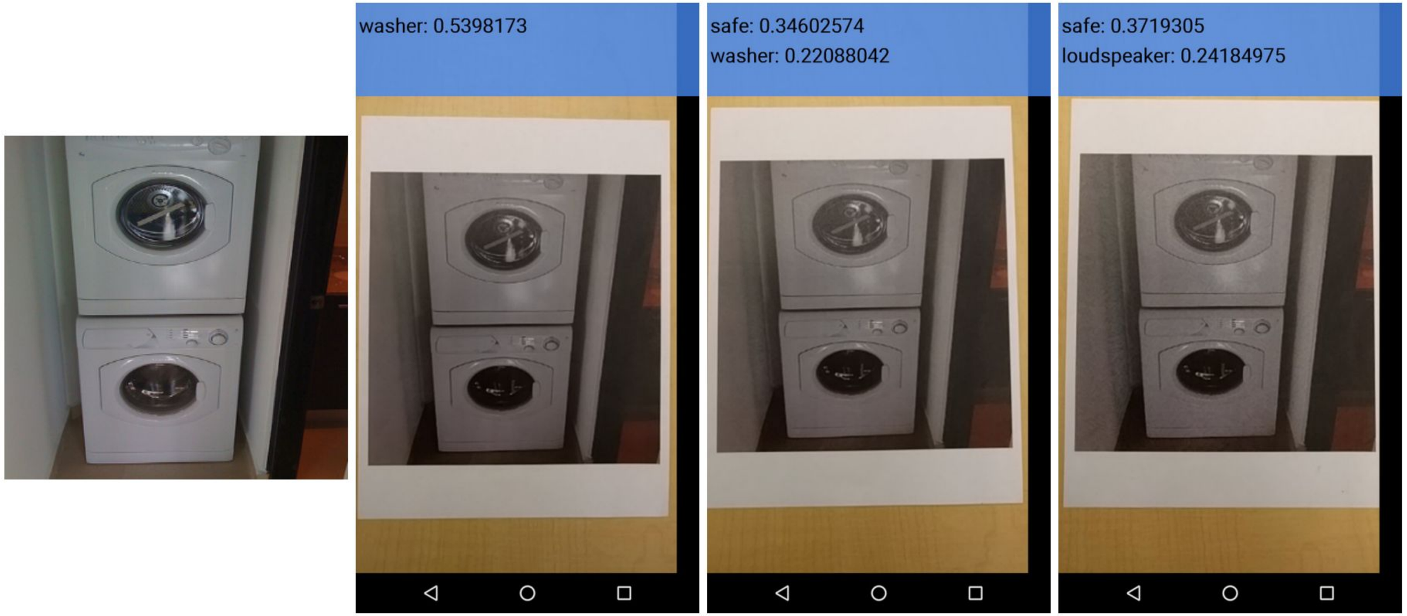
- Manipulate traffic signs to confuse autonomous vehicles
- Removing segmentation of pedestrians in an object detection system
- Adversarial commands for ASR models and VCS (Siri, Alexa, Cortana)

Adversarial Example for Autonomous Driving



[Adversarial Example for Semantic Image Segmentation, Fischer et al]

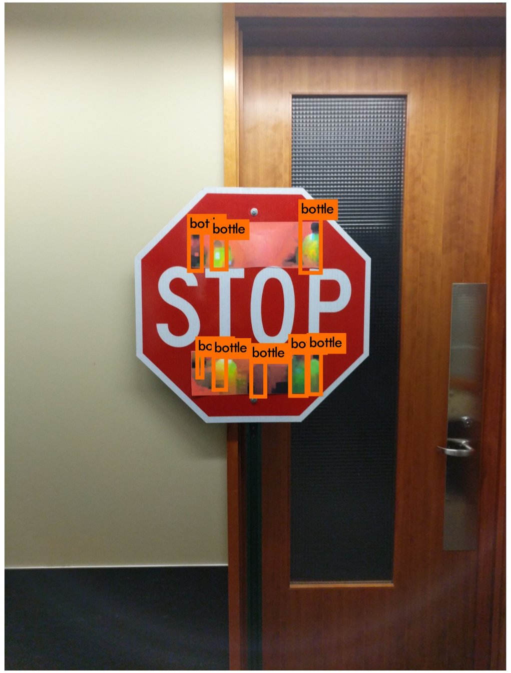
Physical Adversarial Examples

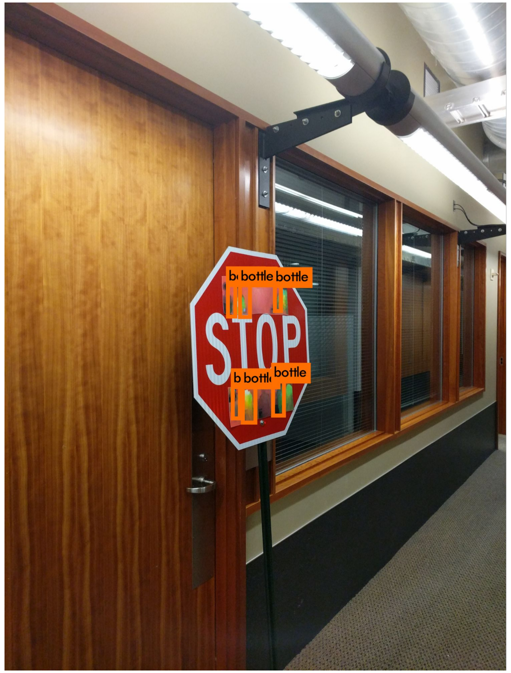


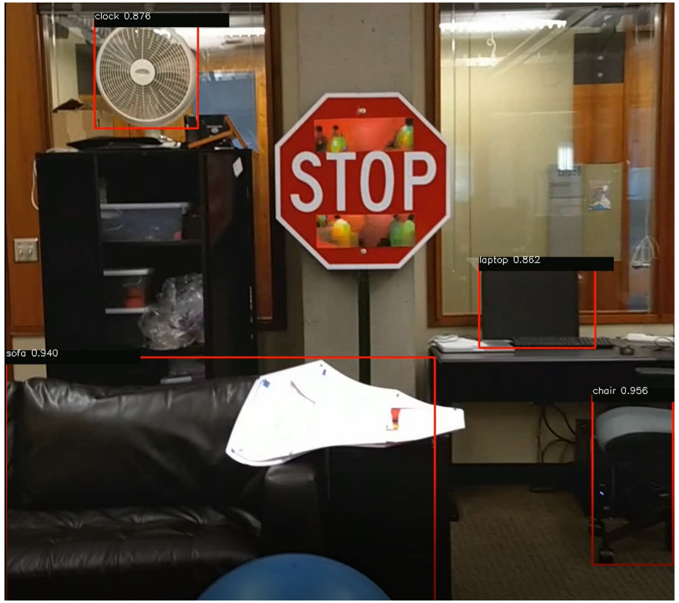
(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

[Adversarial examples in the physical world, Kurakin et al]()

Physical Adversarial Examples

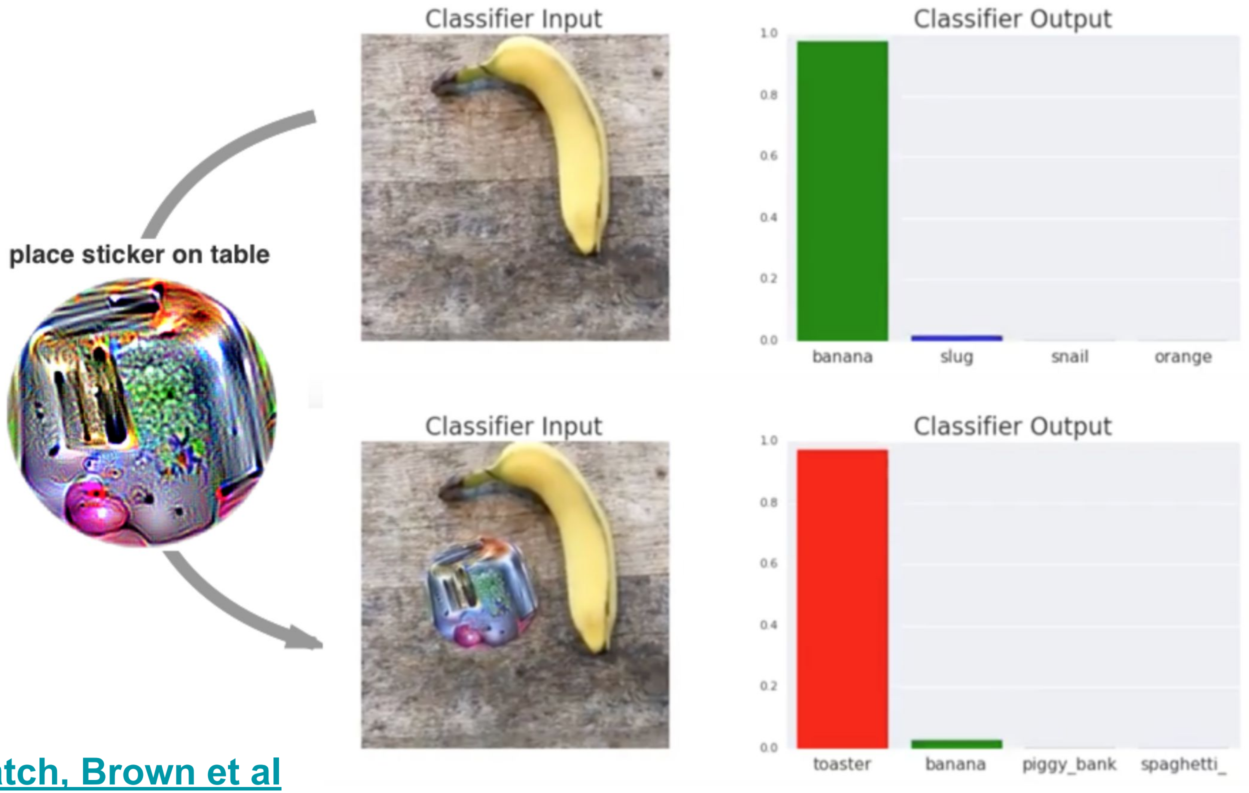






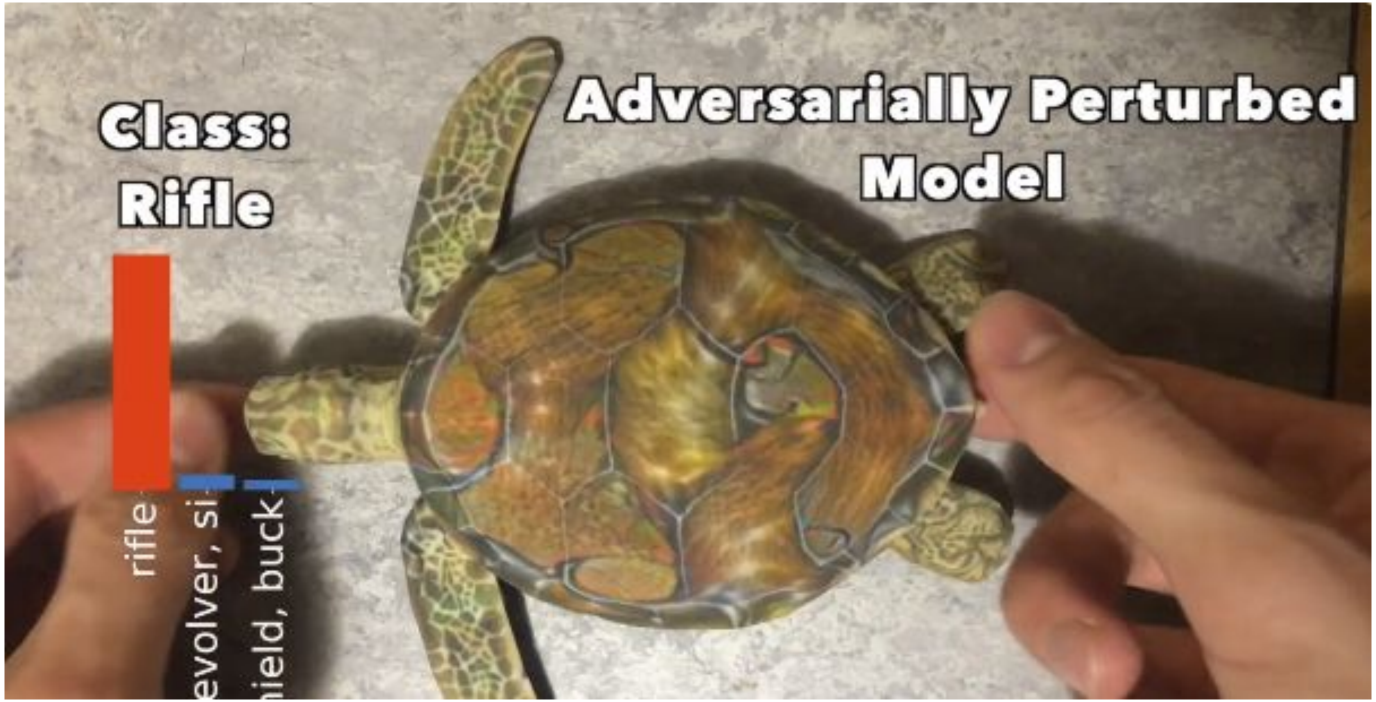
http://bair.berkeley.edu/blog/2017/12/30/yolo-attack/

Adversarial Patches



Adversarial Patch, Brown et al

3D Adversarial Objects



http://www.labsix.org/physical-objects-that-fool-neural-nets/

Computer Vision community believed that perceptual distances are well approximated by Euclidean distances in feature space of deep learning models.

Adversarial examples reveal some contradictions...

Additive perturbations

Additivite perturbations: $T(x) = x + r$

The magnitude of the perturbation can be measured with the $\ell_p\text{-norm}$ of the minimal perturbation that is necessary to change the label of a classifier

$$\min_{r \in \mathcal{R}} \| r \|_p \text{ subject to } f(x + r) \neq f(x).$$

$$\ell_p\text{-norm} \quad ||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

Distance Metrics

1) ℓ_0 **distance** (0-norm counts the number of nonzero coordinates)

It measures the number of coordinates i such that $x_i \neq x'_i$. Thus it corresponds to the **number of pixels that have been altered** in an image.

2) ℓ_2 **distance**

It measures the Euclidean (root mean-square) distance between x_i and x'_i . It can remain small when there are many small changes to many pixels.

3) ℓ_∞ **distance** $||x - x'||_\infty = \max(|x_1 - x'_1|, \ldots, |x_n - x'_n|)$

It measures the maximum change to any of the coordinates. For images, we can image there is a maximum budget, and each pixel is allowed to be changed up to this limit.

Adversarial Perturbation

Structured noise carefully designed to fool a classifier causing a minimal change in the input.

The perturbations can be characterized along different dimensions:

- Type of perturbation
- Type of error
- Adversarial Specificity
- Adversary Knowledge

Types of perturbations

Depending on the conditions that one sets on the support of the perturbations $\mathcal{R}$ the additive model leads to different forms of robustness:

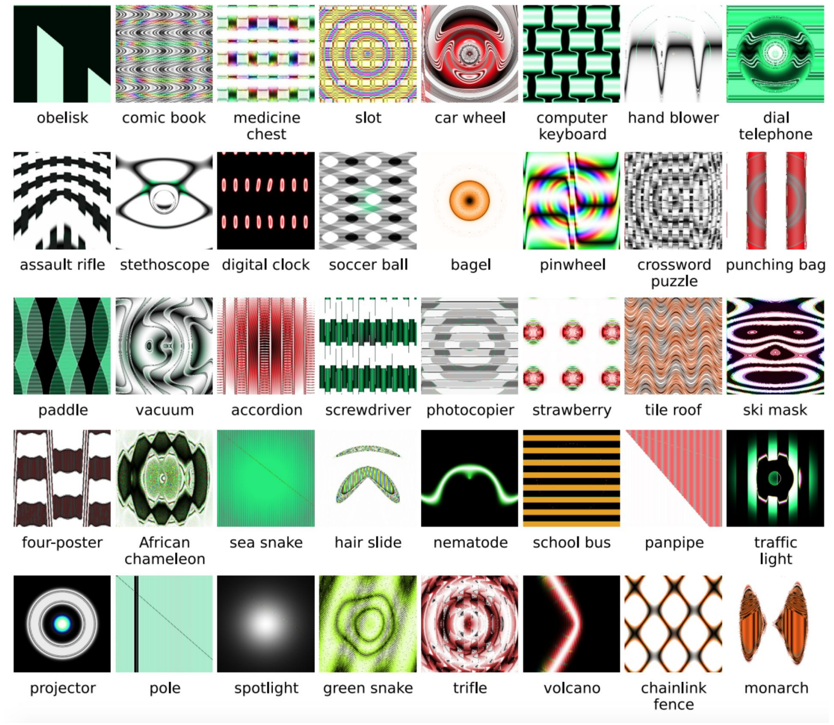
- **Adversarial**
- Random Noise
- Semi-random Noise
- **Universal (image agnostic)**

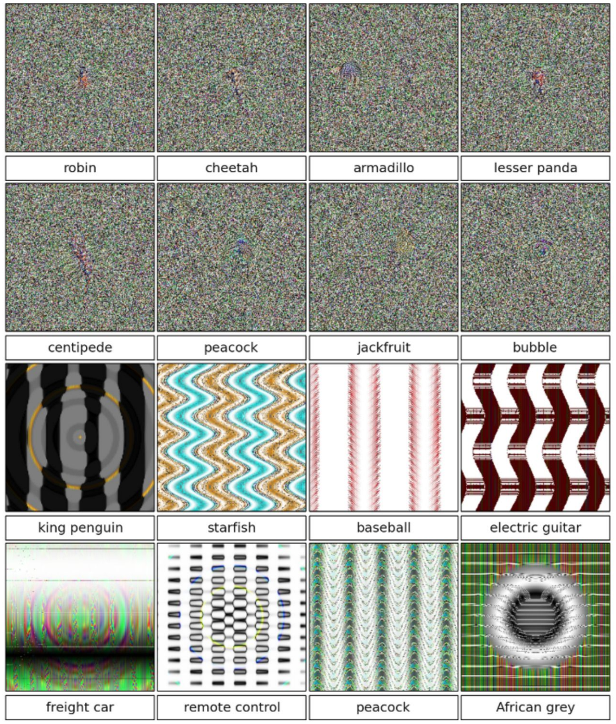
Types of error

- **False Positive (Type I Error):** an adversary image unrecognizable to human, but predicted by NNs to a class with high confidence.

- **False Negative (Type II Error):** an image that can be recognized by human, but cannot be predicted by NNs (like a malware that is not identified by a detection system)

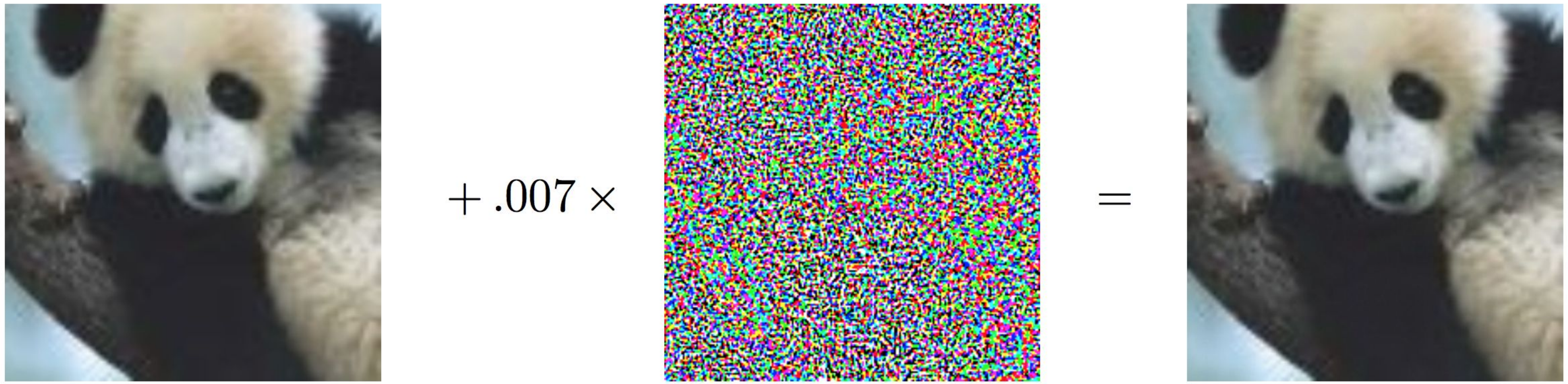
False positives (Rubbish)





[Deep Neural Networks are Easily Fooled, Nguyen et al]

False negatives



$$x + .007 \times \operatorname{sign}(\nabla_{x} J(\theta, x, y)) = x + \epsilon \operatorname{sign}(\nabla_{x} J(\theta, x, y))$$

"panda"
57.7% confidence

"nematode"
8.2% confidence

"gibbon"
99.3 % confidence

Adversarial Specificity

- **Targeted attacks** misguide the model to a specific prediction. Usually, in multi-class classification problems, the attacker aim at fooling the model to predict a specific class. (E.g., Face recognition tries to disguise a face to an authorized person)

- **Non-targeted attacks** do not assign assign a specific class to the neural network output. The adversary class of output can be arbitrary expect the original one.

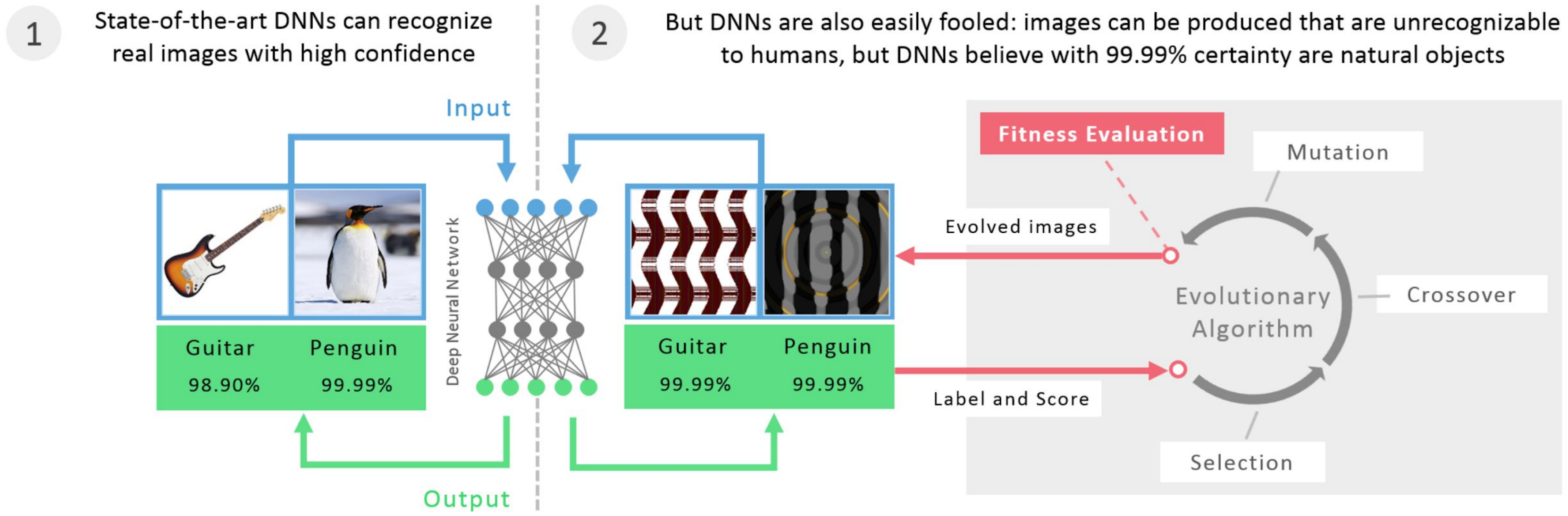
Adversary Knowledge

- **White-box attacks** assume the adversary knows everything about the trained model (trained data, architecture, hyper-parameters, weights…)
- **Black-box attack** assume the adversary has no access to the trained model. This assumption is common for production systems such as ML web services (e.g., ML on AWS, Google Cloud AI, Clarify etc…)

Transferability Property: Most AE attacks are white-box. However, attacks can be transferred among models…. (see more later)

Attacks methods

False Positive Attacks (EA)


1
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
Input
Deep Neural Network
Guitar
98.90%
Penguin
99.99%
Output
Guitar
99.99%
Penguin
99.99%
Fitness Evaluation
Mutation
Evolved images
Evolutionary Algorithm
Crossover
Label and Score
Selection

[Deep Neural Networks are Easily Fooled, Nguyen et al. (2014)]

False Negative Attacks (Targeted)

Given an image x, find a different image $x' = x + \rho$ that is similar under ℓ_2 distance, yet is *labeled differently* by the classifier:

$$\min ||x - x'||_2^2$$
$$s.t.\ C(x') = l$$
$$x' \in [0, 1]^n$$

$C(\cdot)$ is the network classifier, l is the targeted class

This problem can be very difficult to solve because **the constraint is highly nonlinear,** so different techniques exist to approximate adversarial perturbations.

False Negative Attacks (Non-targeted)

Non-targeted attacks are less constrained and hence easier to implement compared to targeted attacks since there are more options and space to redirect the output.

$$\min ||x - x'||_2^2$$
$$s.t.\ C(x') \neq y$$
$$x' \in [0,1]^n$$

Two ways of generating Non-targeted attacks:

1) Running several targeted attacks and taking the one with the smallest perturbation.
2) Minimizing the probability of the correct class.

Box-constrained L-BFGS attack (ℓ_2-norm)

Szegedy et al. (2013) first generated small perturbations on images for classification problem and fooled state-of-the-asrt DNNs with high probability.

$$\min c \cdot ||x - x'||_2^2 + \mathcal{L}(C(x'), l) \qquad s.t.\ x' \in [0,1]^n$$

"Line search" is performed to find the constant **c > 0** that yields an adversarial example of minimum distance, in other words:

- Repeatedly solve this optimization problem for multiple values of **c**,
- Adaptively updating **c** using **binary search** or any other method for one-dimensional optimization.

It was observed that the robustness of DNNs could be improved by “*adversarial training*”.

L-BFGS attacks are slow, so they can not be used for this purpose.

Goodfellow et al. (2014) proposed a method to efficiently compute an adversarial perturbation.

Fast Gradient Sign Method (FGSM, ℓ_∞-norm)

Taking the first-order approximation of the loss function around the true training example x with a **small perturbation Δx:**

$$J_{\theta,y}(x+\Delta_x) \approx J_{\theta,y}(x) + \langle \nabla J_{\theta,y}(x), \Delta_x \rangle,$$

Maximizing the right hand side with respect to Δx restricted to an **∞-ball** of radius ϵ

$$\max_{\Delta_x} \quad J_{\theta,y}(x) + \langle \nabla J_{\theta,y}(x), \Delta_x \rangle$$

$$\text{s.t} \; \|\Delta_x\|_\infty < \epsilon$$

Fast Gradient Sign Method (FGSM, ℓ_∞-norm)

The perturbation that maximizes the optimization problem can be expressed choosing its magnitude as:

$$\eta = \epsilon sign(\nabla_x J_\theta(x, l)),$$

The attack **only performs one step gradient update** along the direction of the sign of gradient at each pixel.

Fast Gradient Sign Method (FGSM, $\ell_\infty\text{-norm}$)

- FGSM perturbs an image to increase the loss of the classifier on the resulting image.
- Ths **sign** function ensures that the magnitude of the loss is maximized
- ϵ essentially restrict the $\ell_\infty\text{-norm}$ of the perturbation.

Fast Gradient Value Method (FGVM)

Rozsa et al replace the sign of the gradient with the raw gradient:

$$\eta = \nabla_x J_\theta(x, l)$$

FGVM has been developed to seek *greater adversarial diversity.*

It has no constraints in each pixel and can generate images with a larger local difference.

Adversarial Diversity and Hard Positive Generation, Rozsa et al

Fast Gradient Value Method (FGVM)

The computed gradient can be normalized with the ℓ_2, ℓ_∞-norms

$$\eta = \epsilon \frac{\nabla_x J_\theta(x, l)}{||\nabla_x J_\theta(x, l)||_p},$$

$$p = \{2, \infty\}$$

Distributional Smoothing with Virtual Adversarial Training, Myhato et al

Adversarial Machine Learning at Scale, Kurakin et al

One shot / One step methods

Broadly speaking all of these methods in are seen as ‘one-shot’ or ‘one-step’.

Intuitively:

- For each pixel they uses the gradient of the loss function to determine in which direction the pixel’s intensity should be changed to maximize the loss function;

- Then, shift all pixels simultaneously in one step.

Iterative methods

- One step attacks perturb the images by taking a single large step in the direction that increases the loss of the classifier (i.e. one-step gradient ascent)
- One step attacks are easy to transfer, but also easy to defend.
- Also, it is important to note that methods such as FGSM attack were designed to be fast, rather than optimal.
- They are not meant to produce the minimal adversarial perturbations.

An intuitive extension of this idea is to **iteratively take multiple steps** while adjusting the direction after each step.

Iterative Momentum FGSM

Momentum can be applied to compute the perturbation in a more iterative way.

The magnitude of the perturbation can be the same of the one generated by the one-shot FGSM, but the directions are more precise and the attack is more effective.

$$g_{t+1} = \mu g_t + \frac{\nabla_x J_\theta(x'_t, l)}{||\nabla_x J_\theta(x'_t, l)||_p},$$

$$x'_{t+1} = x'_t + \epsilon \; sign(g_{t-1})$$

NIPS 2017 Adversarial Targeted and Non-targeted Competition Winner

Basic Iterative Method (BIM)

$$Clip_{x,\delta}\{x'\} = \{255, x + \delta, \max\{0, x - \epsilon, x'\}\}$$

$$x_0 = x,$$

$$x_{t+1} = Clip_{x,\delta}\{x_t + \epsilon sign(\nabla_x J_\theta(x_n, y))\}$$

The number of iterations is determined by the formula $\min(\delta + 4, 1.25\epsilon)$

At each iteration value of the pixels are clipped to avoid large changes.

Madry et al pointed out that BIM is equivalent to the (ℓ_∞ version of) Projected Gradient Descent (PGD), a standard convex optimization method.

Iterative Least-Likely-Class Methods (ILLC)

Extension of BIM where the label of the the image is replaced by the **target label of the least likely class predicted by the classifier.**

Very effective on SOTA DNNs, also for very small values of epsilon.

$$x_0 = x,$$

$$Y_{LL} = \underset{y}{\operatorname{argmin}}\ p(y|x),$$

$$x_{t+1} = Clip_{x,\delta}\{x_t + \epsilon sign(\nabla_x J_\theta(x_n, y_{LL}))\}$$

Jacobian-based Saliency Map Attack (JSMA)

The algorithm modifies pixels one at a time and monitors the effects of the change on the performance by computing a saliency map using the gradients of the outputs.

$$\mathbf{J}(x) = \frac{\partial f(x)}{\partial x} = \left[\frac{\partial F_j(x)}{\partial x_i}\right]_{i \times j}$$

Find the feature/pixels that induce the largest change in the output. In the saliency map, a large value indicates a higher likelihood to fool the network.

97% adversarial success rate by modifying only 4.02% of the image, but the method is slow because its significant computational cost.

DeepFool

DeepFool iteratively finds the **optimal** direction in which we need to travel the minimum distance to cross the decision boundary of the target model.

(Closest distance from original input to the decision boundary)

Although in non-linear cases, this optimality is not guaranteed, DeepFool works well in practice performing an iterative attack by linear approximation, and usually generates very subtle noise.

DeepFool

we assume a classifier $\hat{k}(x) = \mathbf{sign}(f(x)),$

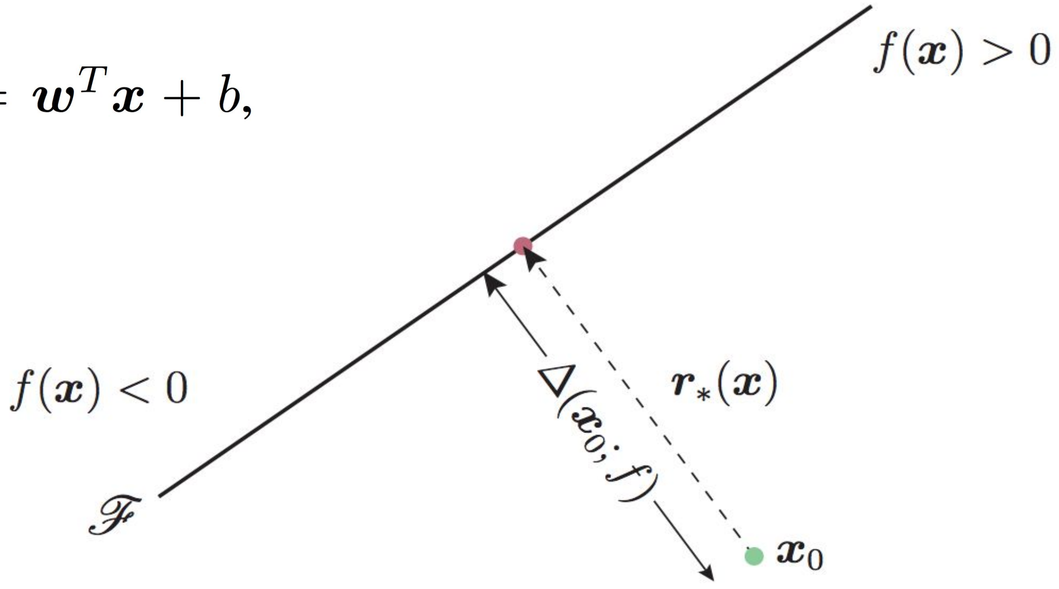
Where f is an arbitrary scalar-valued image classification $f : \mathbb{R}^n \to \mathbb{R}$

$$\mathscr{F} \triangleq \{x : f(x) = 0\}$$

DeepFool: Linear Classifier

Consider first an affine classifier $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$,
The robustness of f at point x_0 is equal to the distance from x_0 to the separating affine hyperplane

$$\mathscr{F} = \{\boldsymbol{x} : \boldsymbol{w}^T\boldsymbol{x} + b = 0\}$$

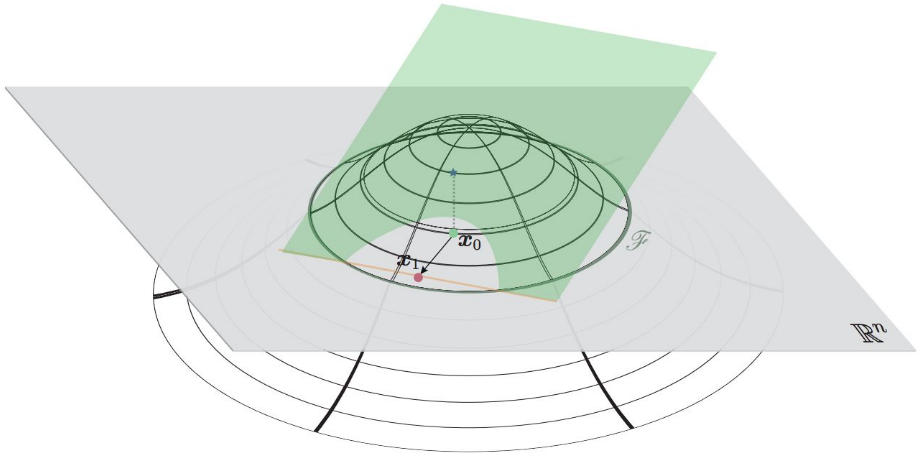


The **minimal perturbation** to change the classifier's decision corresponds to the orthogonal projection of x_0 onto $\mathscr{F}$:

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

DeepFool: General Classifier

For a general binary differentiable classifier, an iterative procedure can be used to estimate its robustness $\Delta(x_0; f)$



Specifically, at each iteration, f is linearized around the current point x_i and the minimal perturbation of the linearized classifier is computed as:

$$\underset{\boldsymbol{r}_i}{\arg\min} \|\boldsymbol{r}_i\|_2 \text{ subject to } f(\boldsymbol{x}_i) + \nabla f(\boldsymbol{x}_i)^T \boldsymbol{r}_i = 0.$$

The perturbation r_i at iteration of the algorithm is computed using the closed form solution in of the linear case, and the next iterate x_{i+1} is updated. The algorithm stops when x_{i+1} changes sign of the classifier.

DeepFool

Algorithm 1 DeepFool for binary classifiers

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

Algorithm 2 DeepFool: multi-class case

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

Multi-class case: Find the closest hyperplane

Optimization Problem (Recall)

Given an image x, find a different image $x' = x + \rho$ that is similar under ℓ_2 distance, yet is *labeled differently* by the classifier:

$$\begin{aligned} \min ||x - x'||_2^2 \\ s.t.\ C(x') = l \\ x' \in [0,1]^n \end{aligned}$$

$C(\cdot)$ is the network classifier, l is the targeted class

This problem can be very difficult to solve because **the constraint is highly nonlinear,** so different techniques exist to approximate adversarial perturbations.

Carlini & Wagner’s Attack

Define an optimization problem easier to solve where:

$$C(x') = l \text{ if and only if } f(x') \leq 0$$

They proposed several objective functions:

$$\begin{aligned}
f_1(x') &= -\text{loss}_{F,t}(x') + 1 \\
f_2(x') &= (\max_{i \neq t}(F(x')_i) - F(x')_t)^+ \\
f_3(x') &= \text{softplus}(\max_{i \neq t}(F(x')_i) - F(x')_t) - \log(2) \\
f_4(x') &= (0.5 - F(x')_t)^+ \\
f_5(x') &= -\log(2F(x')_t - 2) \\
f_6(x') &= (\max_{i \neq t}(Z(x')_i) - Z(x')_t)^+ \\
f_7(x') &= \text{softplus}(\max_{i \neq t}(Z(x')_i) - Z(x')_t) - \log(2)
\end{aligned}$$

Towards Evaluating the Robustness of Neural Networks, Carlini & Wagner

Carlini & Wagner's Attack

$$\begin{aligned} &\min ||x - x'||_p \\ &s.t.\ f(x') \le 0 \\ &x' \in [0,1]^n \end{aligned} \longrightarrow \begin{aligned} &\min ||x - x'||_p + c \cdot f(x') \\ &s.t.\ x' \in [0,1]^n \\ &\ell_0, \ell_2, \ell_\infty\text{-norms} \end{aligned}$$

LBFGS-B is slow but supports box constrained optimization natively.

C&W investigated Projected Gradient Descent and other methods to clip the coordinates when after each step to be inside the box.

This allows them to use first-order methods such as SGD or Adam to generate high quality solutions for the optimization problem converging faster.

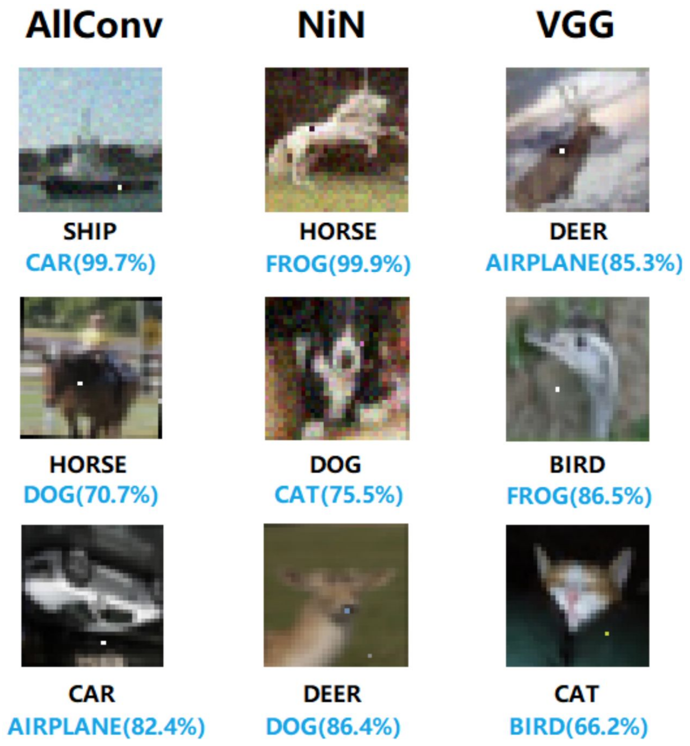
One pixel Attack

Extreme case, **only a single pixel** is changed!

Fooling Success Rate of 70.97%

Avg Confidence on the wrong labels 97,47%

Differential evolution is used to generate the perturbation, so no information about the model parameters or gradients is needed. **(Black-box)**



One pixel attack for fooling deep neural networks, Su et al

Can we find a single perturbation that fools the classifier on every image?

Universal additive perturbations

One might be interested to understand if classifiers are also vulnerable to generic (data and network agnostic) perturbations because:

1) They might not require the precise knowledge of the classifier under test
2) They might capture important security and reliability properties of classifiers
3) They show important properties on the geometry of the decision boundary of the classifier.

Universal additive perturbations

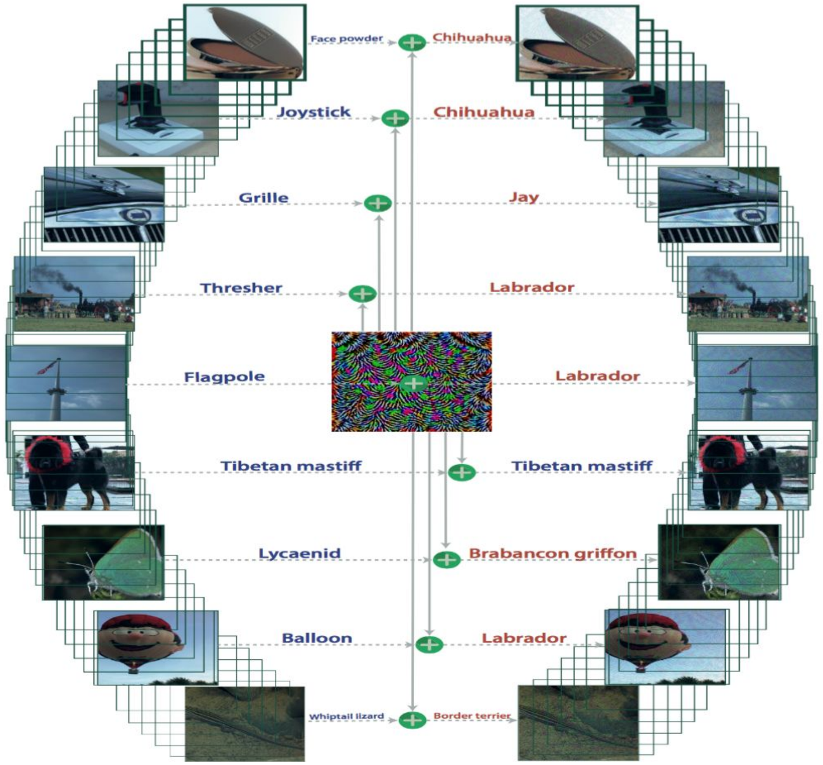
A universal perturbation v can be defined as the minimal perturbation that fools a large fraction of the data points sampled from the data distribution mu, i.e.,

$$v = \operatorname*{argmin}_{r} \| r \|_p \text{ subject to } \mathop{\mathbb{P}}_{x \sim \mu}(f(x + r) \neq f(x)) \geq 1 - \epsilon,$$

where ϵ controls the fooling-rate of the universal perturbation.

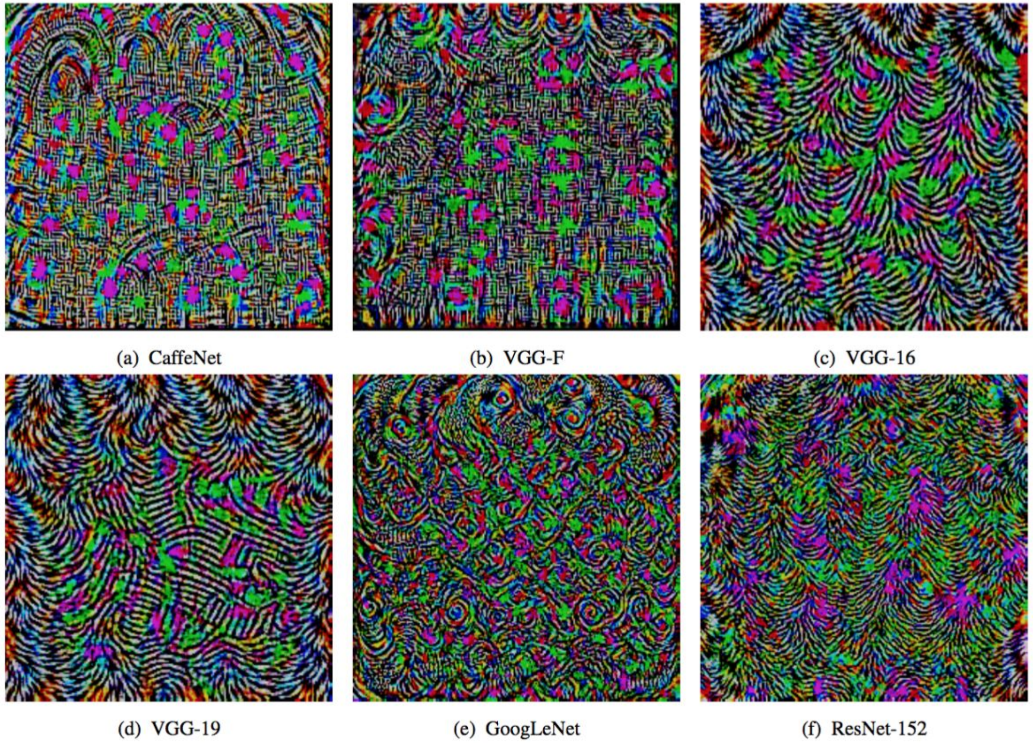
Unlike adversarial perturbations that target to fool a specific data point, universal perturbations **attempt to fool most images sampled from the natural images distribution**.

Specifically, by adding this single (image-agnostic) perturbation to a natural image, the label estimated by the deep neural network will be changed with high probability.



Universal Adversarial Perturbations – Moosavi-Dezfooli et al - 2016

Examples of Universal Perturbations


(a) CaffeNet (b) VGG-F (c) VGG-16 (d) VGG-19 (e) GoogLeNet (f) ResNet-152

Universal Adversarial Perturbations, Moosavi-Dezfooli et al

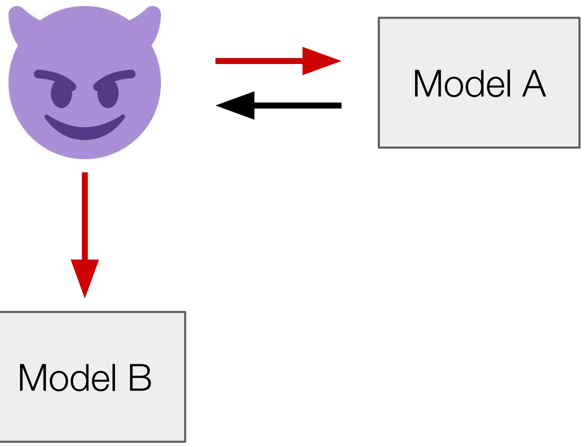
Black-box attacks and transferability

Adversarial Examples Transferability

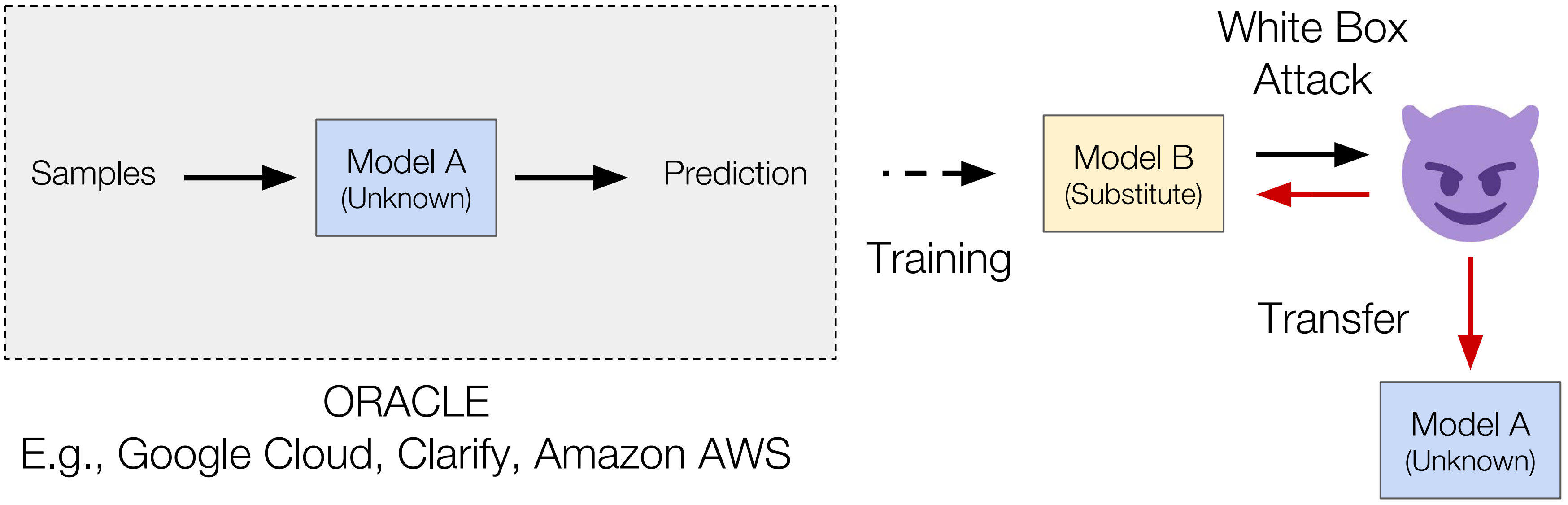
Adversarial examples have a **transferability** property:

"Samples crafted to mislead a model A are likely to mislead also a model B"

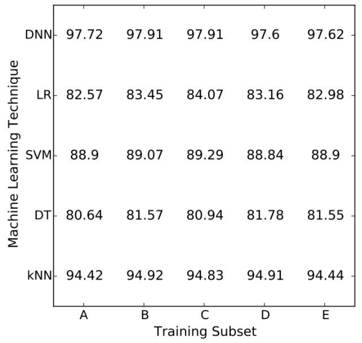
(on the same task)



Substitute model attack



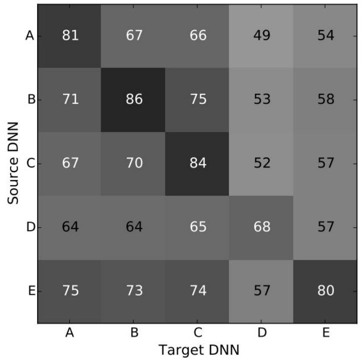
Intra-technique Transferability



Machine Learning Technique	A	B	C	D	E
DNN	97.72	97.91	97.91	97.6	97.62
LR	82.57	83.45	84.07	83.16	82.98
SVM	88.9	89.07	89.29	88.84	88.9
DT	80.64	81.57	80.94	81.78	81.55
kNN	94.42	94.92	94.83	94.91	94.44

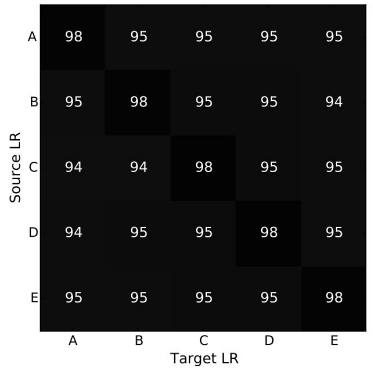
Training Subset

(a) Model Accuracies



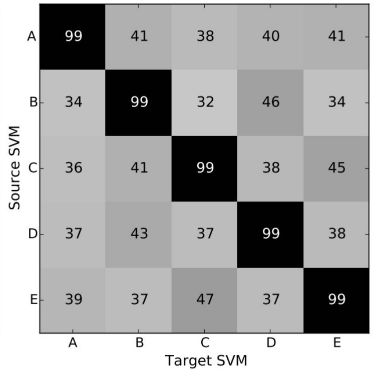
Source DNN \ Target DNN	A	B	C	D	E
A	81	67	66	49	54
B	71	86	75	53	58
C	67	70	84	52	57
D	64	64	65	68	57
E	75	73	74	57	80

(b) DNN models



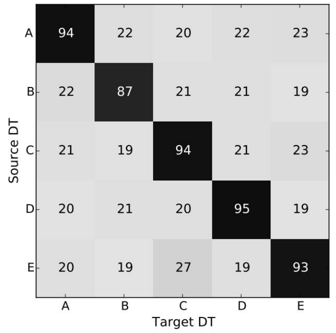
Source LR \ Target LR	A	B	C	D	E
A	98	95	95	95	95
B	95	98	95	95	94
C	94	94	98	95	95
D	94	95	95	98	95
E	95	95	95	95	98

(c) LR models



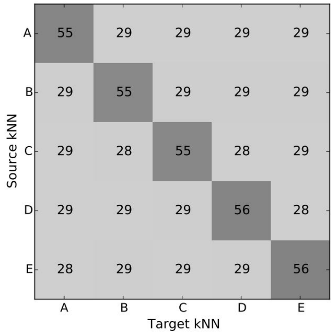
Source SVM \ Target SVM	A	B	C	D	E
A	99	41	38	40	41
B	34	99	32	46	34
C	36	41	99	38	45
D	37	43	37	99	38
E	39	37	47	37	99

(d) SVM models



Source DT \ Target DT	A	B	C	D	E
A	94	22	20	22	23
B	22	87	21	21	19
C	21	19	94	21	23
D	20	21	20	95	19
E	20	19	27	19	93

(e) DT models

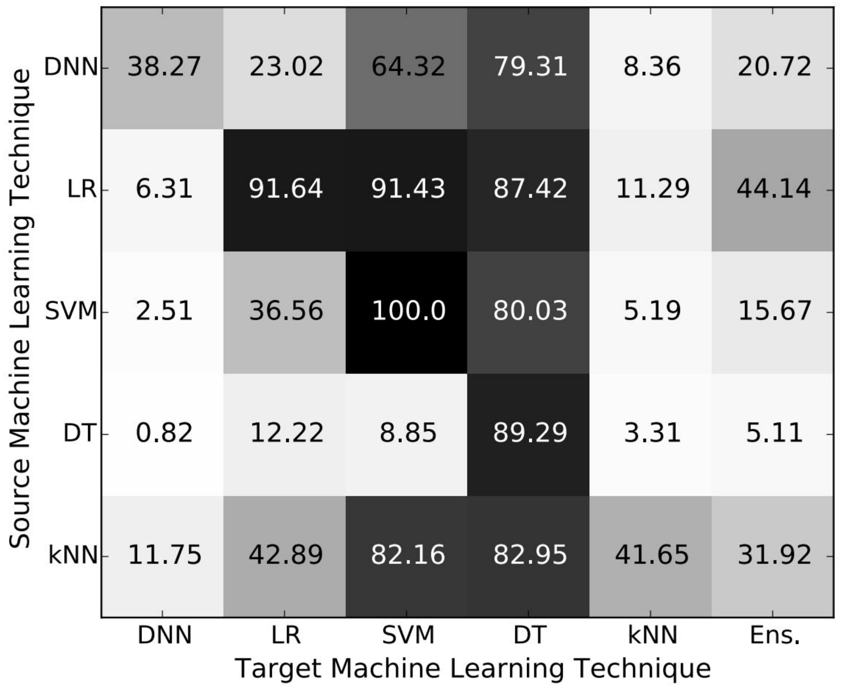


Source kNN \ Target kNN	A	B	C	D	E
A	55	29	29	29	29
B	29	55	29	29	29
C	29	28	55	28	29
D	29	29	29	56	28
E	28	29	29	29	56

(f) kNN models

Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples

Cross-technique Transferability



Source Machine Learning Technique \ Target Machine Learning Technique	DNN	LR	SVM	DT	kNN	Ens.
DNN	38.27	23.02	64.32	79.31	8.36	20.72
LR	6.31	91.64	91.43	87.42	11.29	44.14
SVM	2.51	36.56	100.0	80.03	5.19	15.67
DT	0.82	12.22	8.85	89.29	3.31	5.11
kNN	11.75	42.89	82.16	82.95	41.65	31.92

Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples

(Failed) Defenses Techniques

Gradient defenses

- Most adversarial example construction techniques use the gradient of the model to make an attack.

- **But what if there were no gradient?** what if an infinitesimal modification to the image caused no change in the output of the model?

This seems to provide some defense because the attacker does not know which way to “push” the image.

Knowledge Distillation

Knowledge Distillation was originally introduced in *Distilling the Knowledge in a Neural Network* as a technique for model compression.

At high level:

1) Train a teacher network on the hard labels provided from the training dataset.
2) Collect softmax probabilities.
3) Distill the knowledge into a smaller student network by training on the soft probabilities.

Distillation can potentially increase accuracy on the test and reduce overfitting.

Softmax (Recall)

$$\mathrm{softmax}(x, T)_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$$

The temperature constant T controls the **smoothness of the prediction.**

- Usually in Neural Network is set to 1.
- When T is large the output will be vague (uniform $\text{if } T \to \infty, \text{ then } p(k) \to \frac{1}{|K|}$) ("Softer predictions")
- When T is small, only one class is close to 1 while the rest is close to zero. ("Harder prediction")

Defensive Distillation
increases the softmax temperature T to reduce the sensitivity to perturbations

Defensive distillation

1) Train the **teacher network**, by setting the temperature of the softmax to T (~40-50) during the training phase.
2) **Compute soft labels** by applying the teacher network to each instance in the training set, evaluating the softmax at temperature T.
3) Train the **distilled student network** (with the same capacity) on the soft labels, again using softmax at temperature T.
4) Finally, at **test time** the distilled network is evaluated using temperature **T=1**.

This creates a model whose **surface is smoothed in the directions an adversary will typically try to exploit,** making it difficult for them to discover adversarial input tweaks that lead to incorrect categorization.

Carlini & Wagner attack defeated Defensive Distillation.

Adversarial Training i.e. Robust Optimization

The most common defence consists of introducing adversarial images to train a more robust network, which are generated using the target model:

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon \operatorname{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

Brute force solution where we simply generate online a lot of adversarial examples and explicitly train the model not to be fooled by each of them.

Adversarial Training

Madry et al. (2018) considered an adversarial variant of standard **Empirical Risk Minimization (ERM)**, where the aim is to minimize the risk over adversarial examples:

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

Adversarial training has a natural interpretation in this context, where a given (single or iterative step) attack is used to approximate solutions to the inner maximization problem, and the outer minimization problem corresponds to training over these examples.

Adversarial Training

When performing adversarial training with a single-step attack the ERM Equation is approximated by replacing the solution to the inner maximization problem:

$$x_{\text{FGSM}}^{\text{adv}} := x + \varepsilon \cdot \mathbf{sign}\left(\nabla_x L(h(x), y_{\text{true}})\right) .$$

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

$$h^* = \underset{h \in \mathcal{H}}{\arg\min} \underset{(x, y_{\text{true}}) \sim \mathcal{D}}{\mathbb{E}} \left[L(h(x_{\text{FGSM}}^{\text{adv}}), y_{\text{true}}) \right]$$

Degenerate Global Minima for Single-Step AdvTrain

The minimizer h^* could be a model for which the approximation method underlying the attack (i.e., linearization in our case) poorly fits the model's loss function. That is:

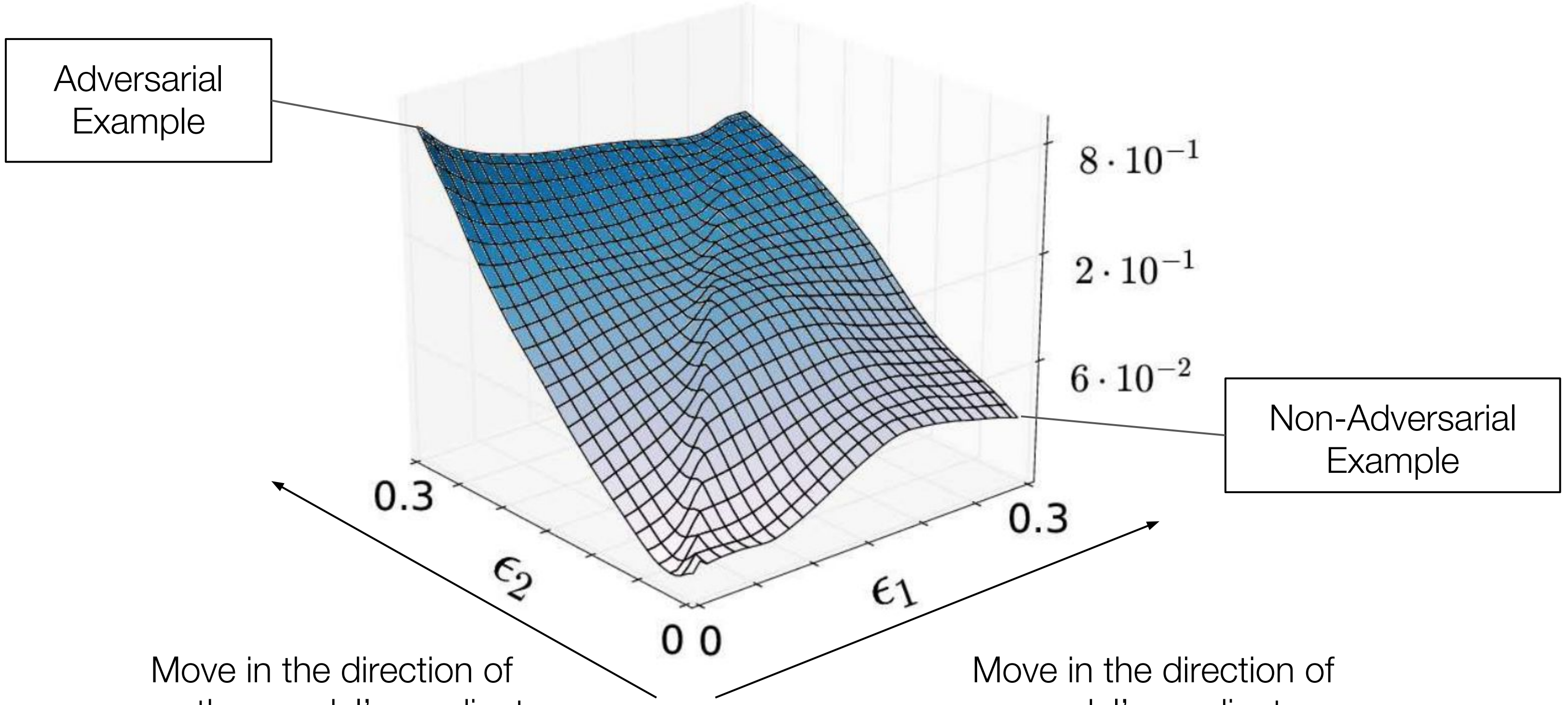
$$L(h^*(x^{\text{adv}}_{\text{FGSM}}), y_{\text{true}}) \ll \max_{\|x^{\text{adv}}-x\|_\infty \leq \epsilon} L(h^*(x^{\text{adv}}), y_{\text{true}})] \,.$$

The attack when applied to h^* produces samples x_{adv} **that are far from optimal!**

Degenerate Global Minima for Single-Step AdvTrain

Note that this second “degenerate” minimum can be more subtle than a simple case of overfitting to samples produced from single-step attacks.

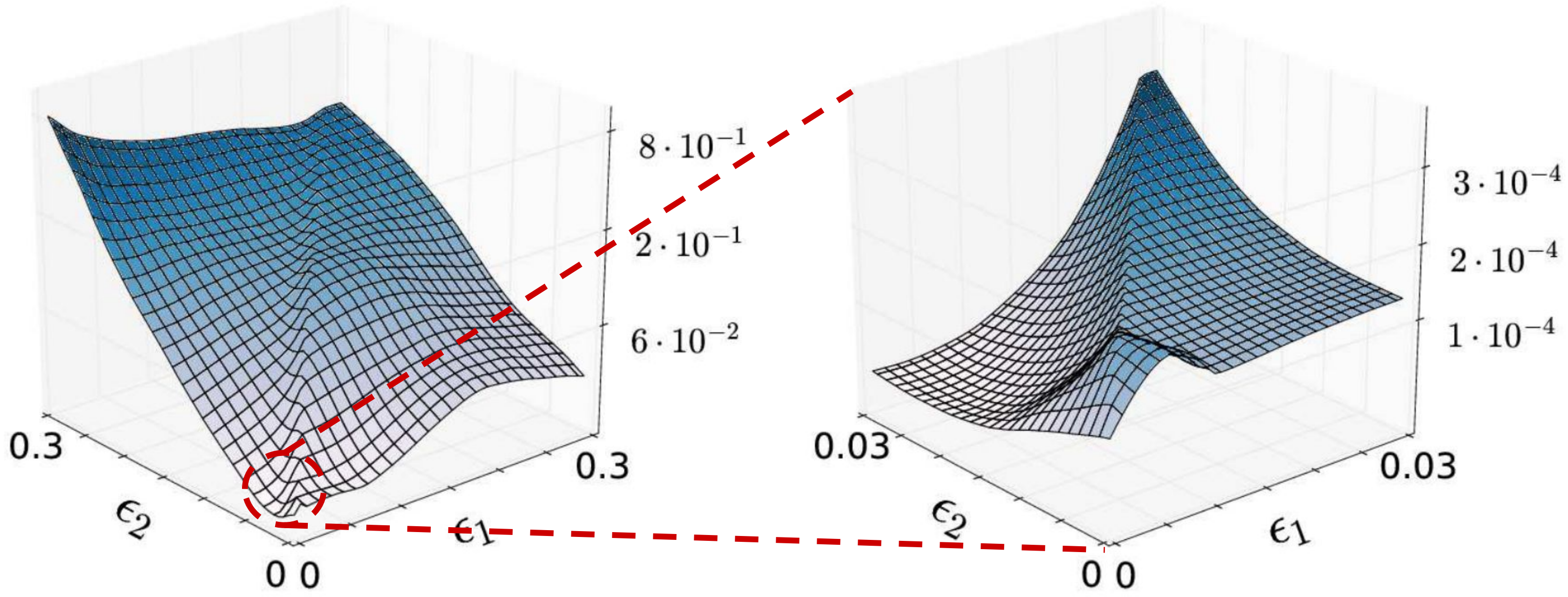
- That single-step attacks applied to adversarially trained models create “adversarial” examples that **are easy to classify even for undefended models.**
- Thus, adversarial training does not simply learn to resist the particular attack used during training, but actually to **make that attack perform worse overall.**



Move in the direction of <u>another model's</u> gradient (black-box attack)

Move in the direction of model's gradient (white-box attack)

Gradient masking



Small curvature artifacts near the data points **obfuscate** a linear approximation of the loss.

A failed defence: “Gradient masking”

- Instead of learning parameters that make the model robust, it learns parameter that make the attacker weaker.
- The degenerate minimum is attainable because **the learned model’s parameters influence the quality of both the minimization and maximization** in the adv-ERM problem.
- We haven’t made the model more robust; we have just given the attacker fewer clues to figure out where the holes in the models defense are.

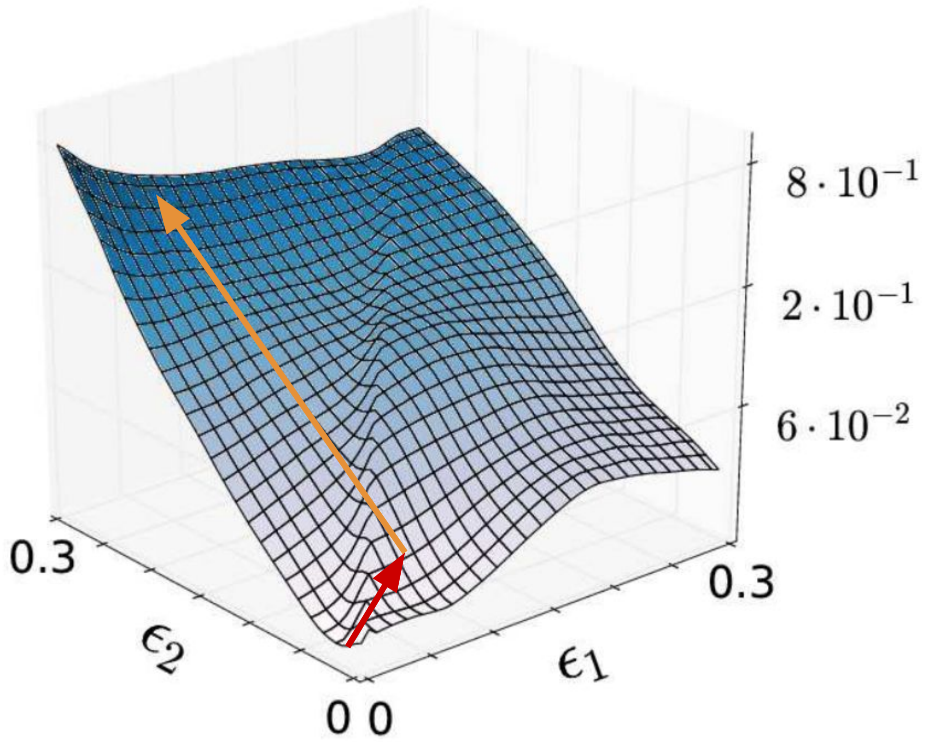
A failed defence: “Gradient masking”

Both adversarial training and defensive distillation accidentally perform a kind of gradient masking. Neither algorithm was explicitly designed to perform gradient masking, but gradient masking is apparently a defense that machine learning algorithms can invent relatively easily when they are trained to defend themselves and not given specific instructions about how to do so.

A perturbation in the gradients is introduced, making the white box attacks less effective, but **the decision boundary remains mostly unchanged** after the defense.

RAND-FGSM

1) Small random step
2) Step in the direction of gradient



RAND-FGSM

Adding random noise when updating the adversarial example to defeat gradient masking issue.

$$\tilde{x} = x + \alpha \cdot sign(\mathcal{N}(\mathbf{0}^d, \mathbf{I}^d)),$$
$$x' = \tilde{x} + (\epsilon - \alpha) \cdot sign(\nabla_{\tilde{x}} J_\theta(\tilde{x}, l))$$

$$\alpha < \epsilon$$

[Ensemble Adversarial Training: Attacks and Defenses, Tramèr et al]

Degenerate Global Minima for Single-Step AdvTrain

Solutions:

- One solution is to use a stronger adversarial example generation process, at a high performance cost (Madry et al., 2018).

- Alternatively, Baluja & Fischer (2018) suggest training an adversarial generator model as in the GAN framework (Goodfellow et al., 2014). The power of this generator is likely to require careful tuning, to avoid similar degenerate minima (where the generator or classifier overpowers the other).

Ensemble Adversarial Training

Decoupling the generation of adversarial examples from the model being trained, while simultaneously drawing an explicit connection with robustness to black-box adversaries.

Ensemble Adversarial Training, augments a model's training data with adversarial examples **crafted on other static pre-trained models**. Intuitively, as adversarial examples transfer between models, perturbations crafted on an external model are good approximations for the maximization problem in (1).

Ensemble Adversarial Training

The learned model **can not influence the “strength”** of the attacker.

As a result, minimizing the “Ensemble Adversarial Training Loss” implies increased robustness to black-box attacks from some set of models.

Parseval Networks

Problem: DNNs are a composition of many nonlinear layers so a small perturbation can be amplified very much before reaching the final classification layer.

Idea: constrain the **lipschitz constants** of each layer to be smaller that 1 so that the network lipschitz constant does not explode exponentially (use Parseval Tight frame parametrization).

Lipschitz constant: magnitude of the smallest upper bound on the derivative of a function. It limits how much a function changes fast with respect to variations of the input.

Parseval Networks: Improving Robustness to Adversarial Examples

Parseval Networks

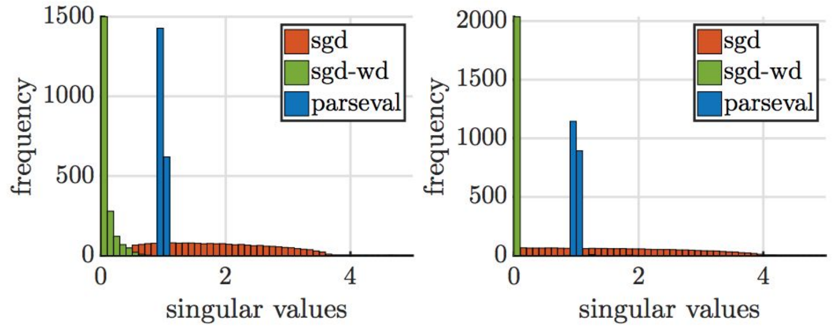


Figure 2. Histograms of the singular values of the weight matrices at layers 1 and 4 of our network in CIFAR-10.

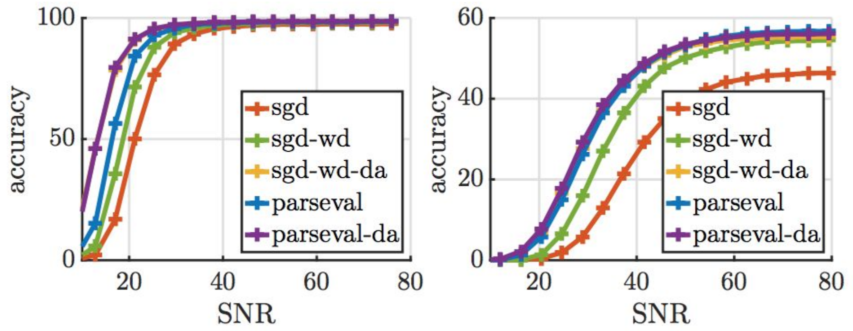


Figure 3. Performance of the models for various magnitudes of adversarial noise on MNIST (left) and CIFAR-10 (right).

Parseval Networks

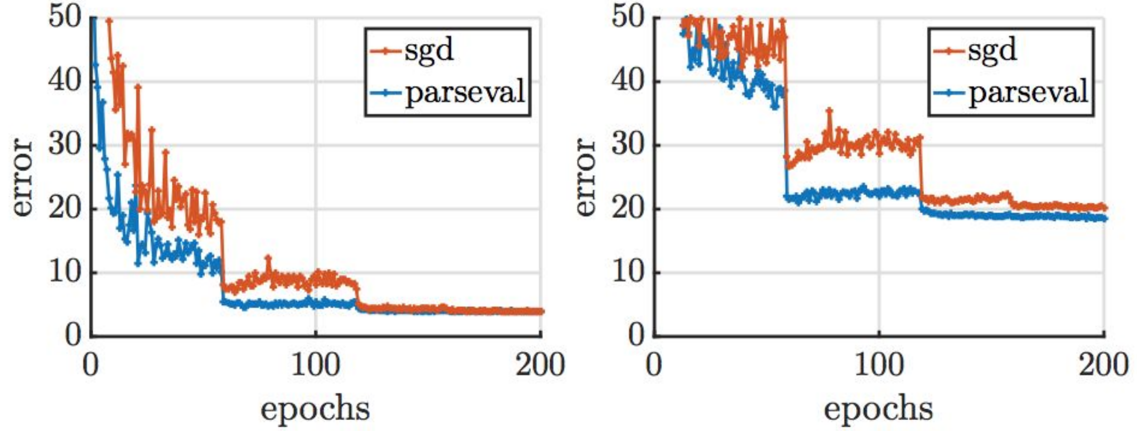


Figure 4. Learning curves of Parseval wide resnets and Vanilla wide resnets on CIFAR-10 (right) and CIFAR-100 (left). Parseval networks converge faster than their vanilla counterpart.

Geometric insights from Adversarial Examples

Geometric insights from robustness

- The study of robustness allows us to derive insights about the classifiers, more precisely, **about the geometry of the classification function acting on the high dimensional input space.**
- The high instability of classifiers to adversarial perturbations shows that natural images **lie very closely to the classifiers' decision boundary.** But this gives no insights on the shape of the decision boundary.
- A local geometric description of the decision boundary (in the vicinity of x) is rather captured by the direction of **r_adv (x)**, due to the orthogonality property of adversarial perturbations.

Robustness of Classification Regions

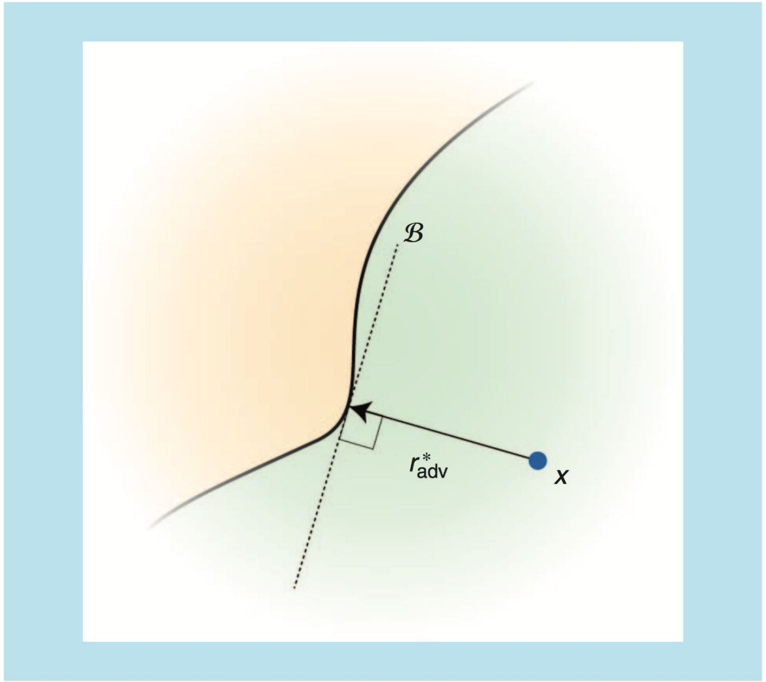


FIGURE 6. r^*_{adv} denotes the adversarial perturbation of x (with $p = 2$). Note that r^*_{adv} is orthogonal to the decision boundary $\mathcal{B}$ and $\| r^*_{\text{adv}} \|_2 = \text{dist}(x, \mathcal{B})$.

Geometric Properties of Adversarial Perturbations

Observation

Let $x \in \mathcal{X}$ and $r^*_{\text{adv}}(x)$ be the adversarial perturbation, defined as the minimizer of ✶ with $p = 2$ and $\mathcal{R} = \mathcal{X}$. Then, we have the following:

1) $\| r^*_{\text{adv}}(x) \|_2$ measures the Euclidean distance from x to the closest point on the decision boundary $\mathcal{B}$.
2) The vector $r^*_{\text{adv}}(x)$ is orthogonal to the decision boundary of the classifier, at $x + r^*_{\text{adv}}(x)$.

✶ $$\min_{r \in \mathcal{R}} \| r \|_p \text{ subject to } f(x + r) \neq f(x).$$

Specific for p=2 norm

Linearity Hypothesis

adversarial examples are a consequence of CNNs acting as a high-dimensional linear classifier.

Linearity hypothesis

- FGSM exploits the “*linearity*” of DNNs in the higher dimensional space whereas such models were commonly thought to be highly nonlinear.
- **Goodfellow et al.** hypothesized that the design of modern DNN that (intentionally) encourage linear behaviour for computational gains, also make them susceptible to cheap analytical perturbations.
- **Luo et al.** hypothesized that CNNs are “locally linear” (only for small perturbations) **only to changes on the regions of the image that contain objects recognized by the CNN,** otherwise the CNN may act non-linearly.

Foveation-based Mechanisms Alleviate Adversarial Examples, Luo et al

Explaining and Harnessing Adversarial Example, Goodfellow et al

Geometry of Classification Regions

Low curvature of the decision boundary **does not imply** that the function learned by the deep neural network is **linear,** or even approximately linear.

Figure shows illustrative examples of **highly nonlinear** functions resulting in **flat decision boundaries.**

It should be noted that, while the decision boundary of deep networks is very flat on random two-dimensional cross sections, these boundaries are not flat on all cross sections. That is, there exist directions in which the boundary is very curved.

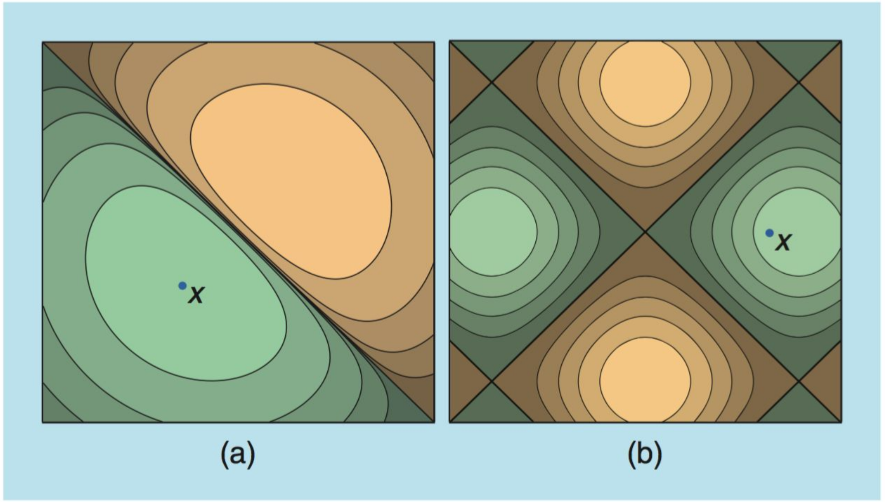
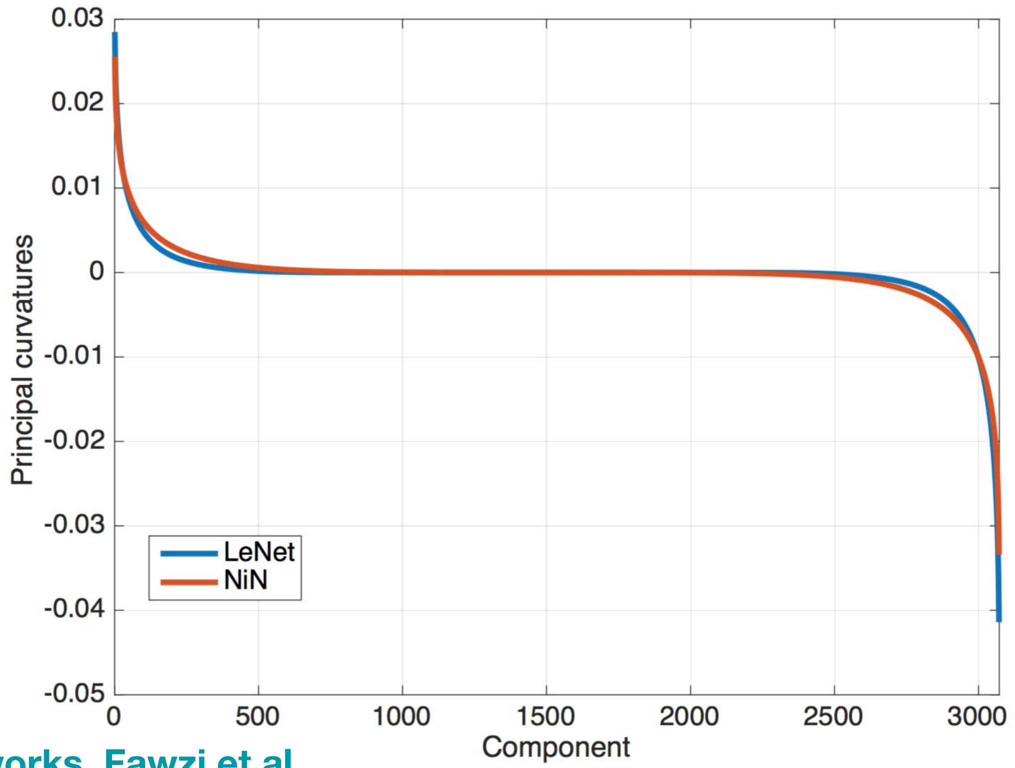


FIGURE 8. The contours of two highly nonlinear functions (a) and (b) with flat boundaries. Specifically, the contours in the green and yellow regions represent the different (positive and negative) level sets of $g(x)$ [where $g(x) = g_1(x) - g_2(x)$, the difference between class 1 and class 2 score]. The decision boundary is defined as the region of the space where $g(x) = 0$ and is indicated with a solid black line. Note that, although g is a highly nonlinear function in these examples, the decision boundaries are flat.

Geometry of Classification Regions

The curvature profile of deep networks is **highly sparse** (i.e., the decision boundaries are almost flat along most directions) but can have a very large curvature along a few directions and it is related to the depth of the network.

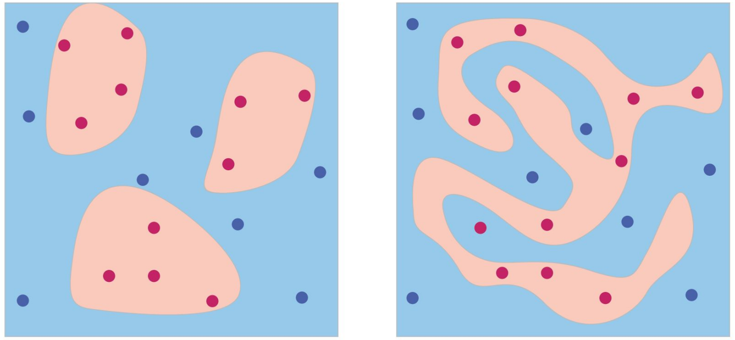
In other words: decision boundaries of DNNs **have a very low curvature** in **many directions**, and high curvature in only few directions.

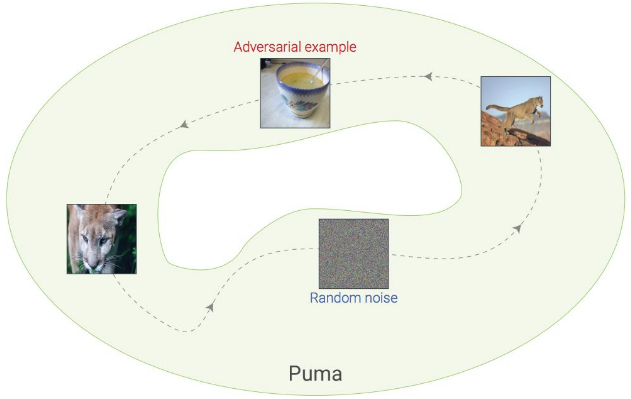


Classification regions of deep neural networks, Fawzi et al

Connected Classification regions

It has been shown empirically that the classification regions are topologically connected. In other words, **each classification region in the input space X is made up of a single connected (possibly complex) region, rather than several disconnected regions.**





Classification regions of deep neural networks, Fawzi et al

Dominant labels

The study of universal perturbations has shown the **existence of dominant labels,** with universal perturbations mostly fooling natural images into such labels. The existence of such dominant classes is attributed to the **large volumes of classification regions corresponding to dominant labels in the input space $\mathcal{X}$:**

- in fact, images sampled uniformly at random from the Euclidean sphere $\alpha\mathbf{S}^{d-1}$ of the input space $\mathcal{X}$(where the radius a is set to reflect the typical norm of natural images) are classified as one of these dominant labels.

Hence, such dominant labels represent **high-volume “oceans”** in the image space.

Dominant labels

Universal perturbations therefore tend to fool images into such target labels, as these generally result in smaller fooling perturbations.

It should be noted that these dominant labels are classifier specific and are not a result of the visual properties of the images in the class.

Universal Adversarial Perturbations, Moosavi-Dezfooli et al

“Hollow” Classification regions

The norm of the smallest adversarial perturbation needed to change the label of a random image (sampled from X) is several orders of magnitude smaller than the norm of the image itself.

- This observation suggests **that classification regions <u>are “hollow” and that most of their mass occurs at the boundaries</u>**.

Semantic insights from Adversarial Examples

Neural Networks *don't learn true concepts*!

Our best models can be making correct predictions for the wrong reasons.

Neural Networks rely heavily on proxy concepts to classify objects

as opposed to strong visual concepts typically used by humans to distinguish between objects.

NNs don’t learn true concepts

Few works nicely demonstrate that classifiers can achieve very high test accuracy without actually learning the true concepts of the classes they predict.

Rather, they can base their prediction on discriminative information, which suffices to obtain accurate predictions on test data, however does not reflect learning of the true concept that defines specific classes.

Example: distinguish vertical and horizontal stripes

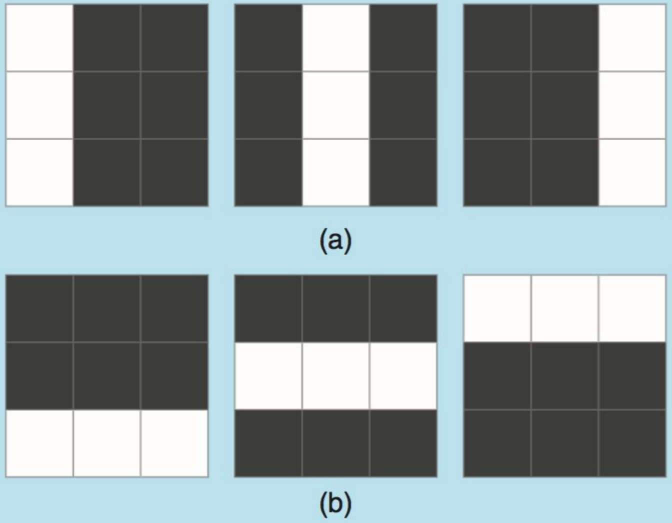


FIGURE S1. (a) The images belonging to class 1 (vertical stripe and positive bias) and (b) the images belonging to class 2 (horizontal stripe and negative bias).

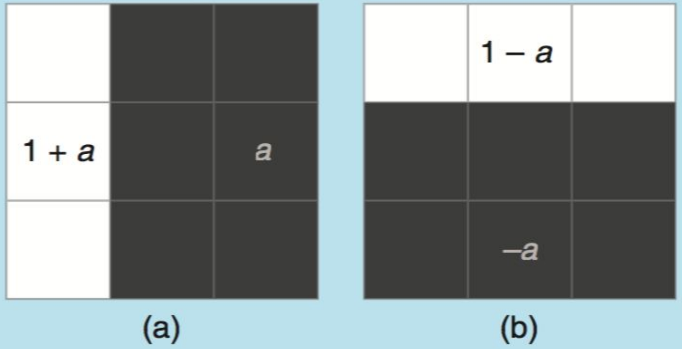


FIGURE S2. (a) An example image of class 1. White pixels have value $1 + a$, and black pixels have value a. (b) An example image of class –1. White pixels have value $1 - a$, and black pixels have value $-a$. The bias a is set to be very small, in such a way that it is imperceptible.

Example: distinguish vertical and horizontal stripes

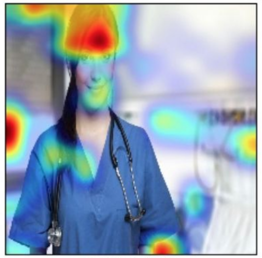
Example: the goal is to classify images based on the orientation of the stripe. In this example, linear classifiers could achieve a perfect recognition rate by exploiting the imperceptibly small bias that separates the two classes.

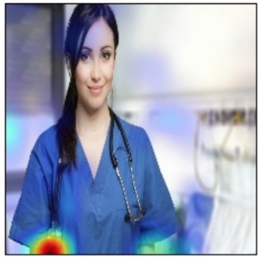
While this proxy concept achieves zero risk, **it is not robust to perturbations!**

One could design an additive perturbation that is as simple as a minor variation of the bias, which is sufficient to induce data misclassification.

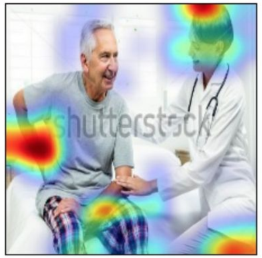
Example

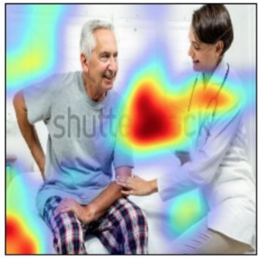
Ground-Truth: Nurse
(a) Original image


Predicted: Nurse
(b) Grad-CAM for biased model

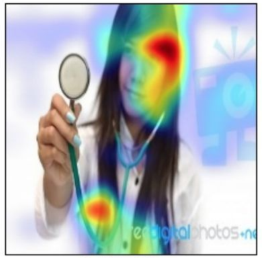

Predicted: Nurse
(c) Grad-CAM for unbiased model

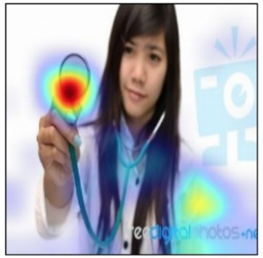
Ground-Truth: Doctor
(d) Original Image


Predicted: Nurse
(e) Grad-CAM for biased model


Predicted: Doctor
(f) Grad-CAM for unbiased model

Ground-Truth: Doctor
(g) Original Image


Predicted: Nurse
(h) Grad-CAM for biased model


Predicted: Doctor
(i) Grad-CAM for unbiased model

Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, Selvaraju et al

NNs don’t learn true concepts

The high instability of classifiers to additive perturbations observed in the literature of Adversarial Attacks suggests that deep neural networks potentially capture one of the proxy concepts that separate the different classes.

As a result, they can consistently **fail in recognizing the true class concept** in new examples or confidently give wrong predictions on specifically designed examples.

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

Analysis of classifiers' robustness to adversarial perturbations

Measuring the tendency of CNNs to Learn Surface Statistical Regularities

Recap

It is important to tackle adversarial attacks in safety-critical environments.

- Can we force the network to learn “true concepts” instead of easy proxy?
  - Design neural architectures structurally invariant to transformations (e.g. Capsule Networks)
  - Unsupervised learning: disentangling factors of variations

Attack transferability is a threat that needs to be addressed (Black box attacks)

- Can we improve our understanding of the decision boundary/input manifold?
  - Learning robust models geometry aware
  - Detecting and modeling dataset bias

Thanks for the attention!

Questions?

Acknowledgements

The introduction on feature visualization is based on slides from:

- Stanford CS231n Course Notes