Attacking Neural Networks

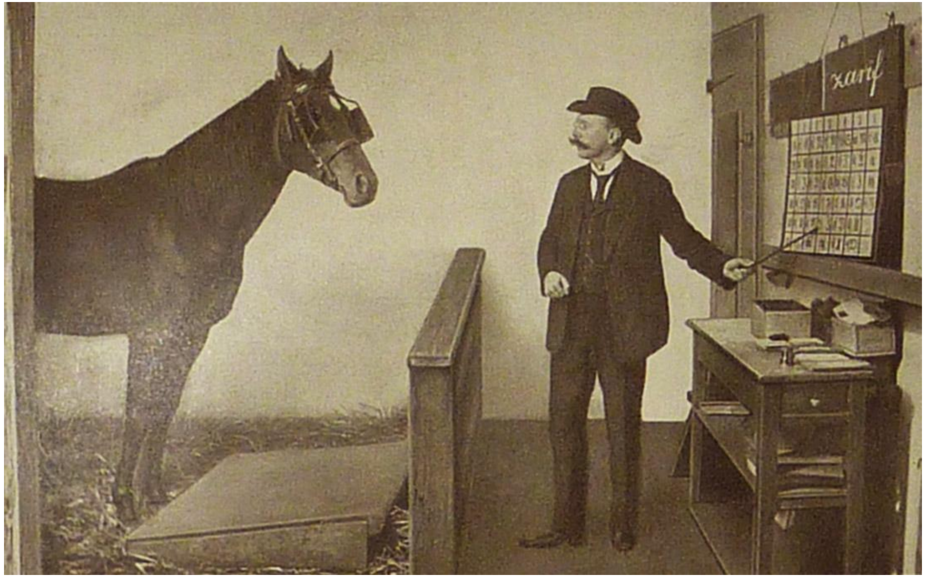
Fooling image classification models with adversarial inputs

Outline

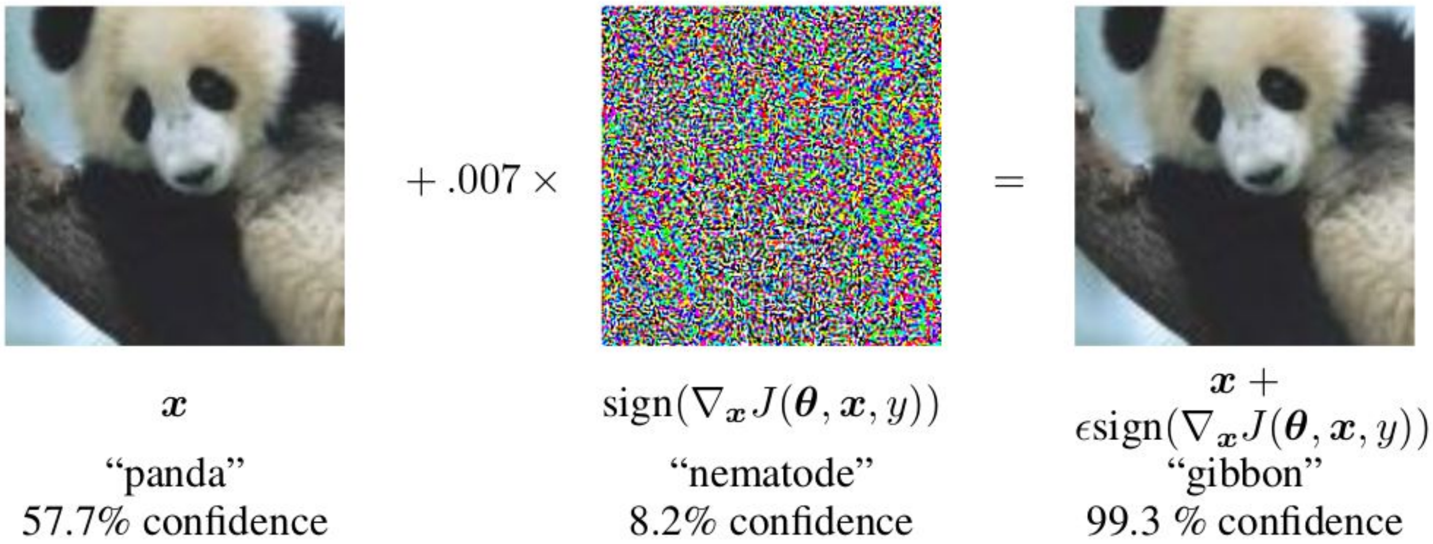
- What is an adversarial input?
- Review of neural networks and gradients
- Attack methods
- Defense methods
- Physical world
- Code

Clever Hans

- Able to perform basic arithmetic, but only when trainer asked the questions
- Learned to read involuntary body language from trainer
- Machine learning models may achieve high accuracy from test set from same distribution of training data
- Models can perform poorly when exposed to data outside that distribution



What is an adversarial input?



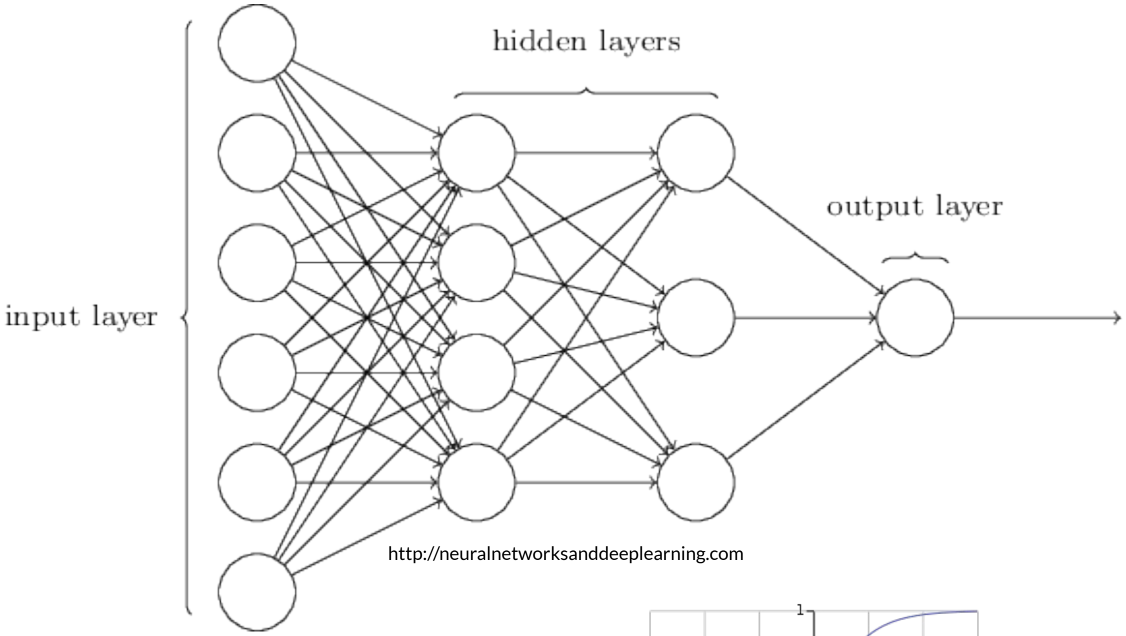
[I. Goodfellow, J Shlens & C. Szegedy. Explaining and Harnessing Adversarial Examples]

Neural networks and gradients

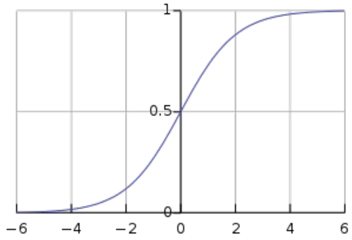
Matrix dimensions: 6x4 -> 4x3 -> 3x1

Neural Networks

- Sequence of matrices (weights) and activation functions
- Input vector fed through the network by taking dot product with weights, and feeding product through activation functions, then repeat for each layer
- Output layer usually a 1 dimensional sigmoid function (range of [0,1]) or n dimension softmax function (sum of dimensions = 1, give probabilities for labels)



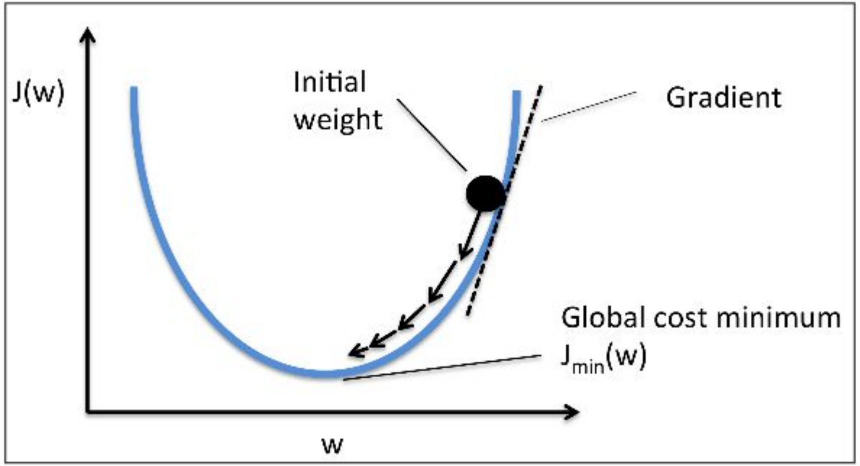
http://neuralnetworksanddeeplearning.com



https://en.wikipedia.org/wiki/Sigmoid_function

Neural Networks

- Training process optimizes weights to minimize **loss function** with **gradient descent**
- **Loss function** - measures how correct a prediction is
- **Gradient descent** - move parameters in direction of negative gradient until minimum found
- **Gradient** - vector of partial derivatives
- Weights are moved in direction of gradient of loss function with respect to weights



Raschka, Sebastian. Python Machine Learning

Gradients and Jacobians

- Gradients used to see how loss function changes
- Jacobians used to see how output (softmax or logits) change

when $f : \mathbb{R}^n \to \mathbb{R}$, then for x in $\mathbb{R}^n$,

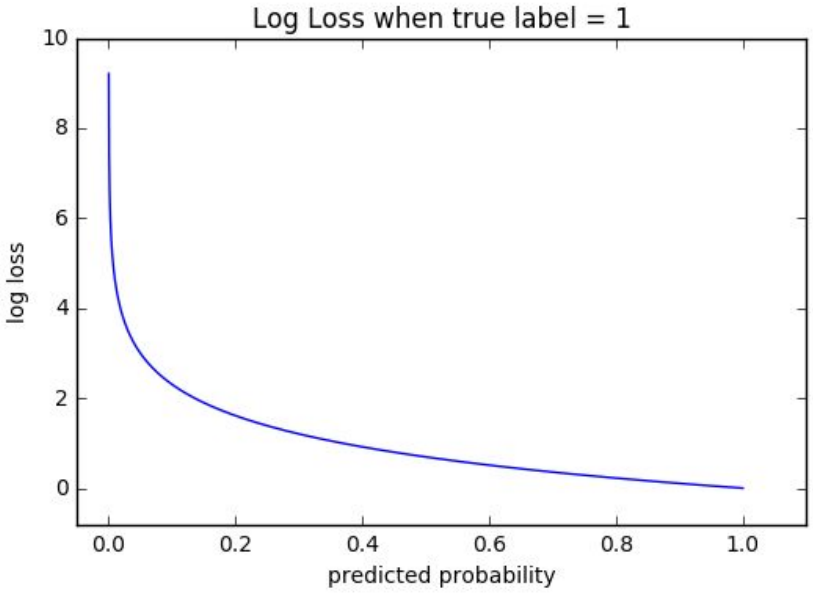
$$\mathrm{grad}_x(f) := \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n}\right]\Big|_x$$

when $f : \mathbb{R}^n \to \mathbb{R}^m$, then for x in $\mathbb{R}^n$,

$$\mathrm{Jac}_x(f) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}\Big|_x$$

https://math.stackexchange.com/questions/1519367/difference-between-gradient-and-jacobian

Cross Entropy



https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

$$H(p,q) = -\sum_{\forall x} p(x)\log(q(x))$$

$$L = -\mathbf{y} \cdot \log(\hat{\mathbf{y}})$$

$$L = -(1 \times log(0.1) + 0 \times \log(0.5) + \dots)$$

$$L = -log(0.1) \approx 2.303$$

https://datascience.stackexchange.com/questions/20296/cross-entropy-loss-explanation

- Common loss function for classification
- Smaller when Prob(y_hat) closer to Prob(y_true)

Softmax

- Used to map input to a probability distribution of classes
- Used as output activation function
- **Logits** - input to softmax layer, or non normalized output of final hidden layer

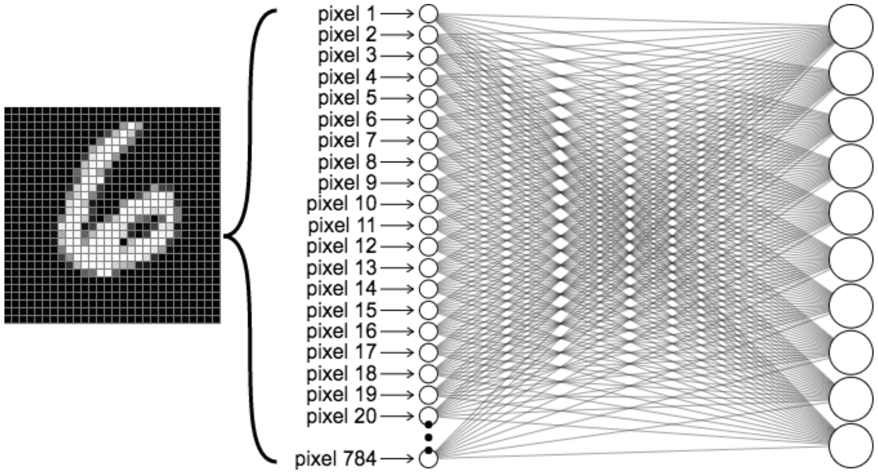
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

https://en.wikipedia.org/wiki/Softmax_function

- x = [-0.2, 0.3, 0.1]
- F(x) with T = 1: [0.250, 0.413, 0.337]

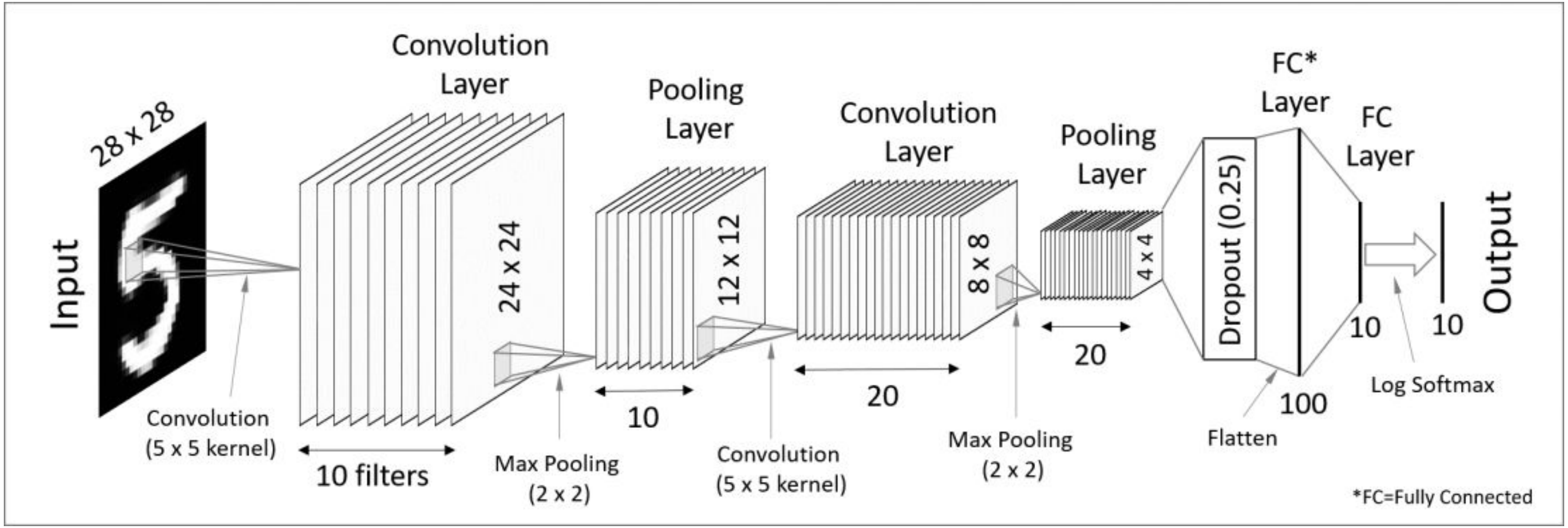
Image Classification

- Each pixel value of an image is a feature
- For greyscale: Integer values in [0,255]
- RGB: one 8 bit value per channel



https://ml4a.github.io/ml4a/looking_inside_neural_nets/

Convolutional Neural Networks



https://codetolight.files.wordpress.com/

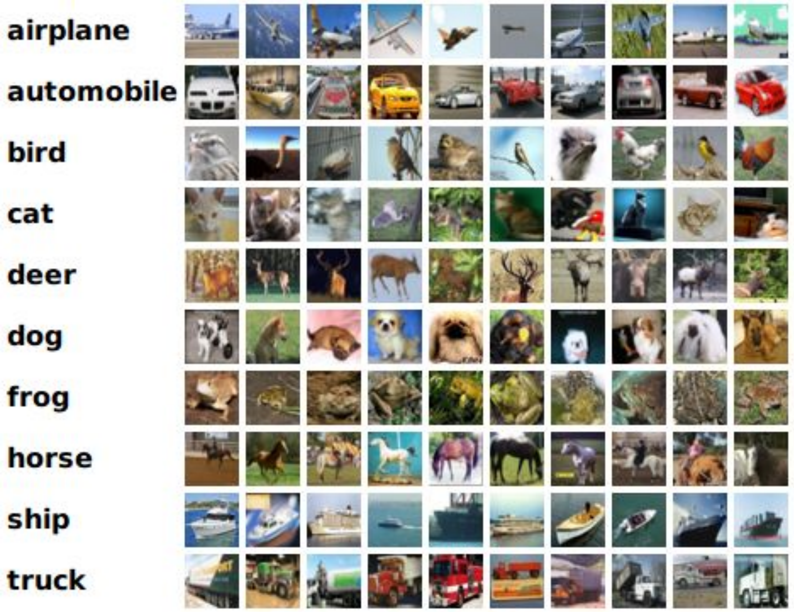
Image Datasets: MNIST

- Handwritten digits
- 28x28 greyscale images

https://en.wikipedia.org/wiki/MNIST_database

Image Datasets: CIFAR10

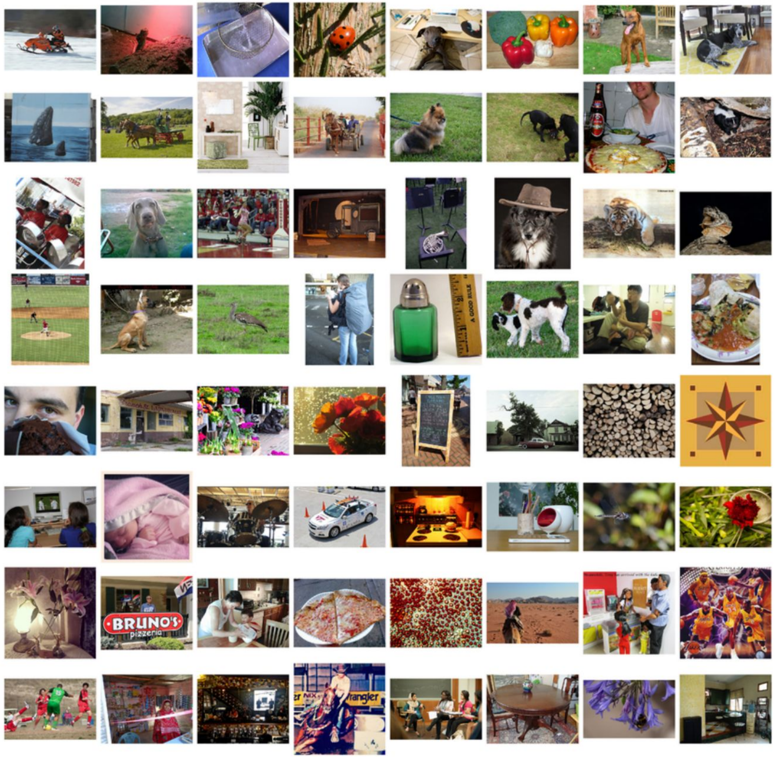
- 32x32 RGB images
- 10 classes (vehicles, animals)



https://www.cs.toronto.edu/~kriz/cifar.html

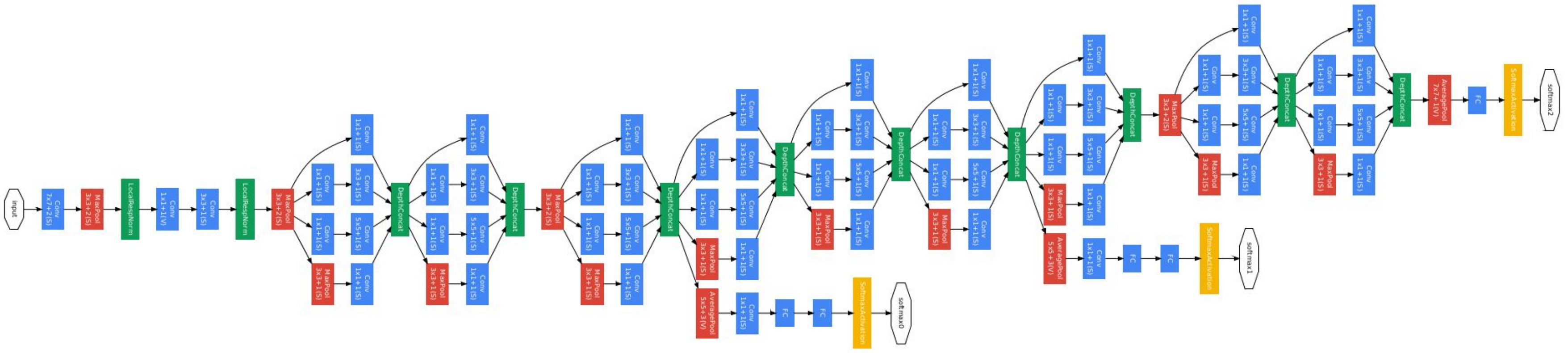
Image Datasets: ImageNet

- 1000 classes
- ImageNet challenge introduced breakthrough in computer vision performance

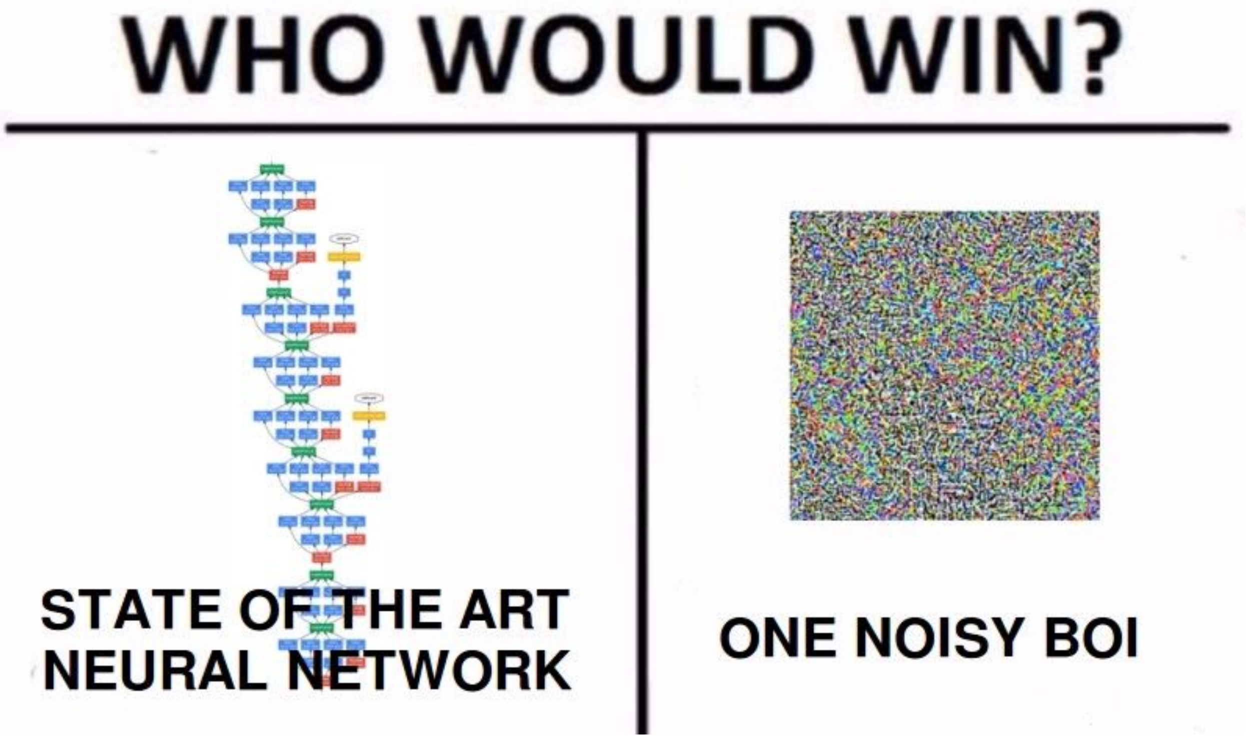


https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/

Inception



Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke: “Going Deeper with Convolutions”, 2014


WHO WOULD WIN?
STATE OF THE ART NEURAL NETWORK
ONE NOISY BOI

Attack Methods

Most are gradient-based optimization methods

- Take gradient of loss function with respect to input to find direction to shift pixels
- Multiple optimization methods can be used to minimize perturbation

Notation and symbols

- **x** - original input
- **x'** - adversarial input
- η - perturbation
- **c** or ε - constant to reduce perceptibility
- **l** - original label
- **l'** - target label
- **J_θ(x', l'),** - loss function (usually cross entropy)
- **f( )** - image classifier network, map x -> l

Norms

- L_0 - number of non-zero values
- L_2 - Euclidean distance
- L_∞ - absolute max

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$

$$\|\mathbf{x}\|_\infty := \max_i |x_i|.$$

https://en.wikipedia.org/wiki/Norm_(mathematics)

L-BFGS method

- First method proposed (2014)
- **L-BFGS** - second order optimization method, more computationally intensive than gradient descent, but can perform better
- Use line or binary search to find minimal c
  - initial c at 1e-5
  - double c and run L-BFGS with x as initial guess until find f(x') = l'
  - binary search from 0 to c to find smaller c to reduce perceptibility
- Slower than most methods
- Can find examples with very little perceptibility

$$\min_{x'} \quad c\|\eta\| + J_\theta(x', l')$$
$$s.t. \quad x' \in [0, 1].$$

Xiaoyong Yuan, Pan He, Qile Zhu: “Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017

Fast Gradient Sign Method

- Second method proposed
- Not targeted
- "One-step" method (no optimization)
- Tries to increase cost with correct label, rather than decrease cost with targeted label
- Often not very successful but was used for famous panda image
- Very fast

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

Xiaoyong Yuan, Pan He, Qile Zhu:
“Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017

Projected Gradient Descent

- aka "Basic Iterative" and "Iterative Least Likely"
- Clip pixels from 0-255
- Least likely class can give very interesting results
- Faster than L-BFGS but creates larger perturbations

$$x_0 = x,$$
$$x_{n+1} = Clip_{x,\xi}\{x_n + \epsilon sign(\nabla_x J(x_n, y))\}.$$

$$x_0 = x,$$
$$y_{LL} = arg\min_y\{p(y|x)\},$$
$$x_{n+1} = Clip_{x,\epsilon}\{x_n - \epsilon sign(\nabla_x J(x_n, y_{LL}))\}.$$

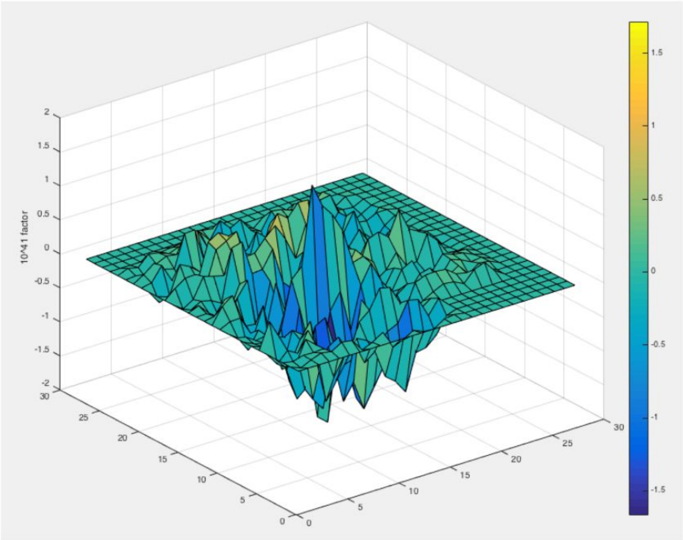
Xiaoyong Yuan, Pan He, Qile Zhu: “Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017

$$J_F(x) = \frac{\partial F(x)}{\partial x} = \left[\frac{\partial F_j(x)}{\partial x_i}\right]_{i \times j}.$$

Xiaoyong Yuan, Pan He, Qile Zhu: "Adversarial Examples: Attacks and Defenses for Deep Learning", 2017

Jacobian-based Saliency Map Attack (JSMA)

- **Saliency map -** shows each pixel's impact on output when perturbed
- At each iteration, calculate saliency map and perturb pixel with highest saliency by given amount θ
- Repeat until f(x') = l' or x' reaches a given distortion threshold
- Perturbs smaller areas but often in higher amounts



Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik: "The Limitations of Deep Learning in Adversarial Settings", 2015

Carlini & Wagner's Attack

- In general most powerful against current defenses
- **g(x + η)** - <= 0, only if f(x') = l'
  - distance/penalty better optimized
  - **Z** - softmax
  - **k** - confidence (usually set to 0)
  - difference between prediction and target probability or 0 if predicted target
- η - defined directly with range of [0,1] (no more clipping)

$$\min_{\eta} \quad \|\eta\|_p + c \cdot g(x+\eta)$$
$$s.t. \quad x + \eta \in [0,1]^n,$$

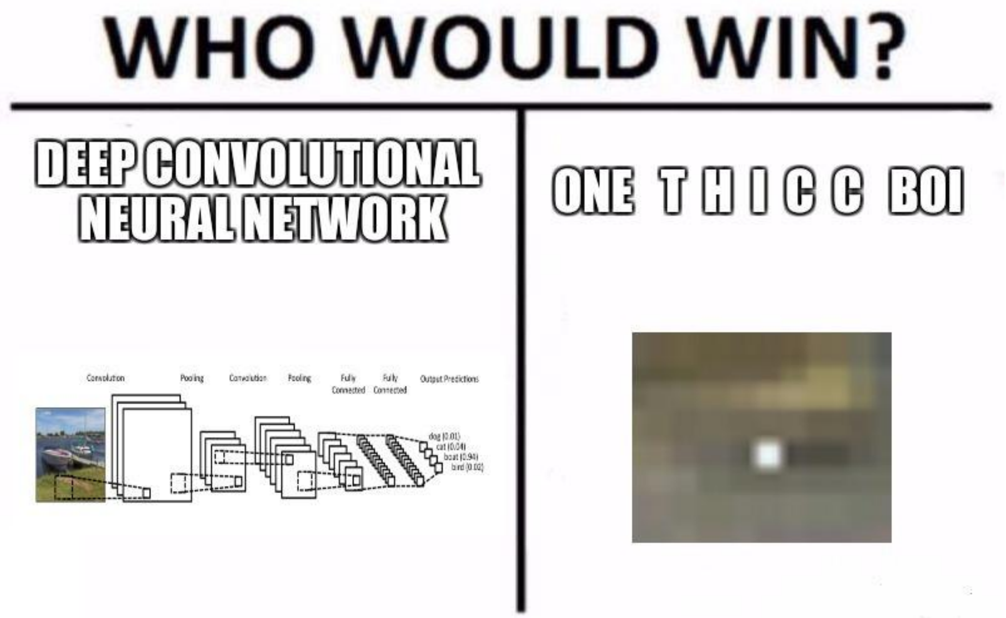
$$g(x') = \max(\max_{i \neq l'}(Z(x')_i) - Z(x')_t, -\kappa),$$

$$\eta = \tfrac{1}{2}(\tanh(w)+1) - x$$

$$\min_{w} \|\frac{1}{2}(\tanh(w)+1)\|_2 + c \cdot g(\frac{1}{2}\tanh(w)+1).$$

Xiaoyong Yuan, Pan He, Qile Zhu: “Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017

One-Pixel



WHO WOULD WIN?
DEEP CONVOLUTIONAL NEURAL NETWORK
Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)
ONE T H I C C BOI

One-Pixel

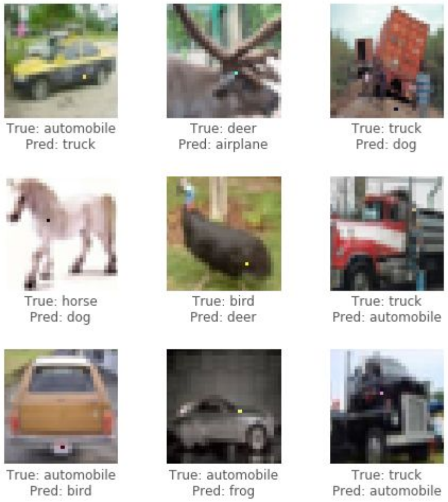
- Uses evolutionary algorithm to find adversarials:
  - A candidate solution consists of an xy coordinate and RGB pixel value
  - Initialize 400 candidate solutions (parents)
  - Generate 400 candidate solutions for next generation by combining parent positions and color values (children)
  - Children compete with corresponding parents, best are kept for next parent set
  - 100 iterations or early-stop when reaching threshold (given probability of target class)
- Weaker on ImageNet models

$$\min_{x'} \quad J(f(x'), l')$$

$$s.t. \quad \|\eta\|_0 \leq \epsilon_0,$$

Xiaoyong Yuan, Pan He, Qile Zhu: "Adversarial Examples: Attacks and Defenses for Deep Learning", 2017

$\epsilon_0 = 1$ for modifying only one pixel

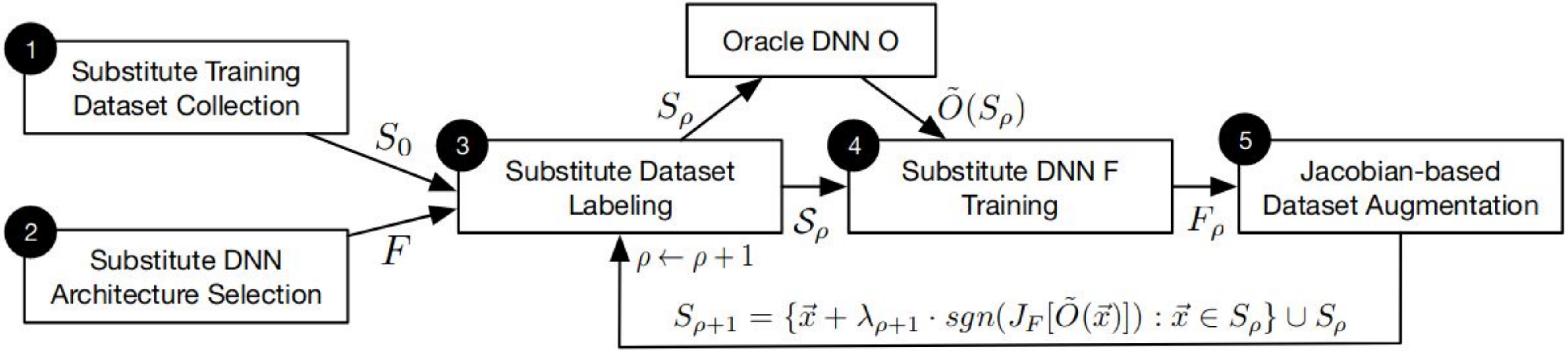


https://github.com/Hyperparticle/one-pixel-attack-keras

Black-box method

- All previous methods require access to model to get gradient (or at least probabilities)
- Many consumer/commercial ML services don't provide anything except predicted labels
- Can learn a substitute model to approximate decision boundaries in target model
- Jacobian-based augmentation used to synthesize and augment dataset to teach substitute model the target's decision boundary

Black-box method



- Identifies sensitive direction of the model's decision boundary

Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik: “Practical Black-Box Attacks against Machine Learning”, 2016

Defense Methods

Adversarial Training

- Generate adversarial examples and train network with these
- Can improve robustness against one-step method adversarial inputs and black box attacks, but in general weak against iterative methods
- Can also add regularization to reduce overfitting

Defensive Distillation

- **Distillation** - method used to reduce size of DNN architectures by training a smaller model with the probability outputs from larger model as labels
    - knowledge acquired during training also encoded in probability outputs (relative difference between classes)
- **Defensive Distillation** - rather than reduce size, we want to increase robustness and smooth decision boundaries
- Increasing temperature increases ambiguity between probabilities
- Train with high temperature, reset to 1 during test time

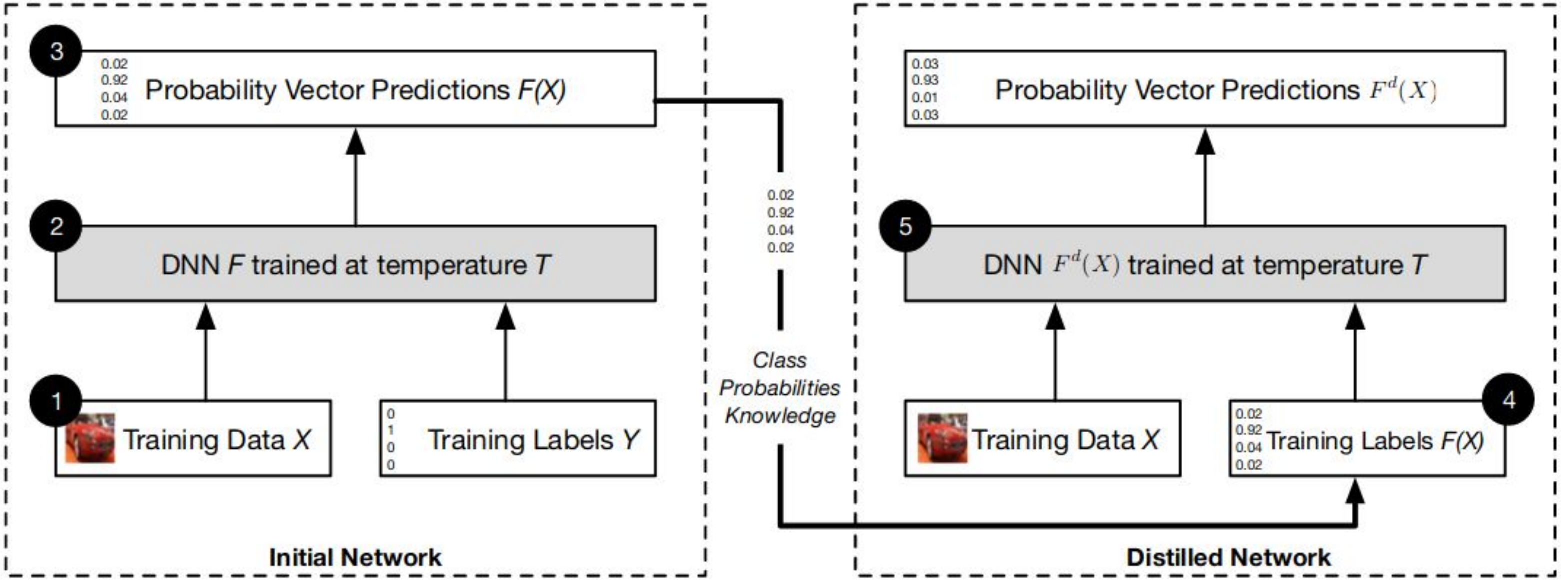
$$F(X) = \left[\frac{e^{z_i(X)/T}}{\sum_{l=0}^{N-1} e^{z_l(X)/T}} \right]_{i \in 0..N-1}$$

Softmax with temperature parameter

- x = [-0.2, 0.3, 0.1]
- F(x) with T = 1: [0.250, 0.413, 0.337]
- F(x) with T = 100:
  [0.3324, 0.3341, 0.3335]

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha: "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks", 2015

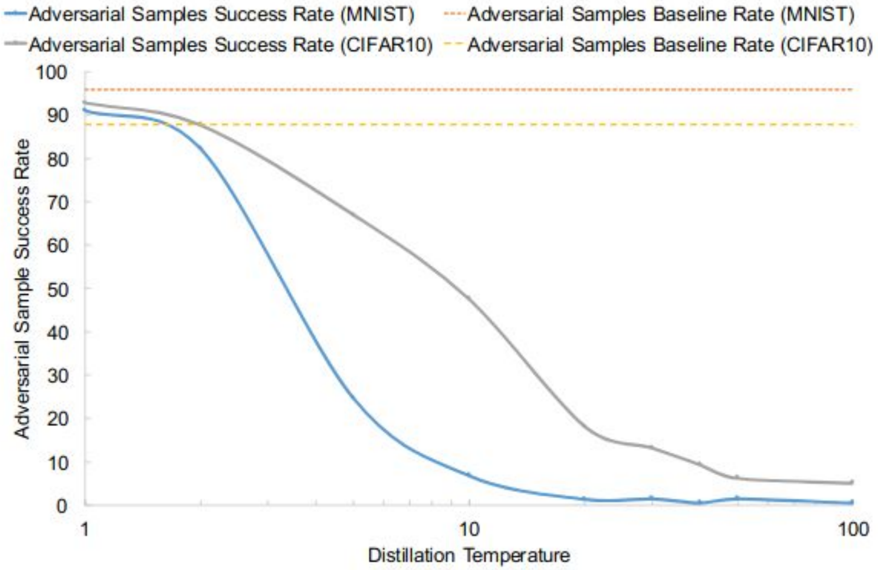
Defensive Distillation



Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha: “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”, 2015

Defensive Distillation

*JSMA method used for attacks



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

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha: “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”, 2015

Adversarial Detecting

- Train secondary neural networks to detect adversarials given input or layer outputs of target model
- Use PCA to detect properties of inputs or network parameters
- Compare distribution with standard statistical methods such as maximum mean discrepancy or kernel density estimation
- **KDE** - compare differences of final hidden layer outputs with training instances of same class

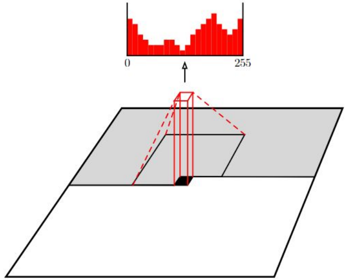
$$KDE(x) = \frac{1}{|X_t|} \sum_{s \in X_t} \exp\left(\frac{|F^{n-1}(x) - F^{n-1}(s)|^2}{\sigma^2}\right)$$

Xiaoyong Yuan, Pan He, Qile Zhu: “Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}).$$

Reconstruction/Purification: PixelDefend

- **PixelCNN** - generative model that learns conditional probability of a pixel based on all previous pixels
- **PixelDefend** - purify image by replacing pixels with expected values within range



1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0

Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves: "Conditional Image Generation with PixelCNN Decoders", 2016

Algorithm 1 PixelDefend

Input: Image $\mathbf{X}$, Defense parameter ϵ_{defend}, Pre-trained PixelCNN model p_{CNN}
Output: Purified Image $\mathbf{X}^*$

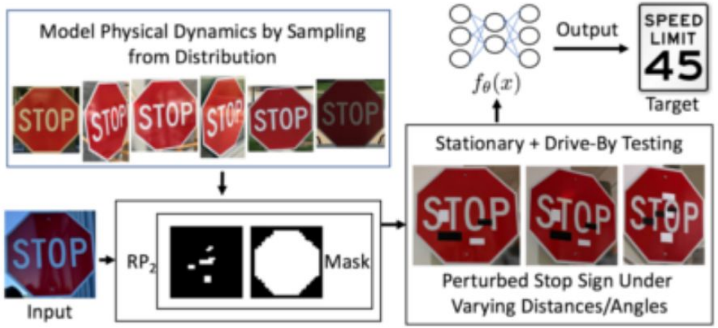
$\mathbf{X}^* \leftarrow \mathbf{X}$
for each row i **do**
  for each column j **do**
    for each channel k **do**
      $x \leftarrow \mathbf{X}[i, j, k]$
      Set feasible range $R \leftarrow [\max(x - \epsilon_{\text{defend}}, 0), \min(x + \epsilon_{\text{defend}}, 255)]$
      Compute the 256-way softmax $p_{\text{CNN}}(\mathbf{X}^*)$.
      Update $\mathbf{X}^*[i, j, k] \leftarrow \arg\max_{z \in R} p_{\text{CNN}}[i, j, k, z]$
    end for
  end for
end for

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon: "PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples", 2017

Attacks in Physical World

Street Signs

- Perturbation must be within bounds of object
- Generation process accounts for physical dynamics (viewing angles)
- Mask used to define object's area
- Sample additional instances of input object from real and synthetic distribution
- **NPS** - non printability score, models printer color reproduction error
  - p hat - set of printable colors
  - p' - set of colors used in perturbation



$$\underset{\delta}{\operatorname{argmin}}\ \lambda||M_x \cdot \delta||_p + NPS$$
$$+ \mathbb{E}_{x_i \sim X^V} J(f_\theta(x_i + T_i(M_x \cdot \delta)), y^*)$$

$$NPS = \sum_{\hat{p} \in R(\delta)} \prod_{p' \in P} |\hat{p} - p'|$$

Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno: “Robust Physical-World Attacks on Deep Learning Models”, 2017

Face Recognition

- Targeted attack on facial recognition systems
- Generate perturbation that can be printed and placed on glasses
- **TV** - improve smoothness of generated image

$$softmaxloss(f(x), c_x) = -\log\left(\frac{e^{\langle h_{c_x}, f(x)\rangle}}{\sum_{c=1}^{N} e^{\langle h_c, f(x)\rangle}}\right)$$

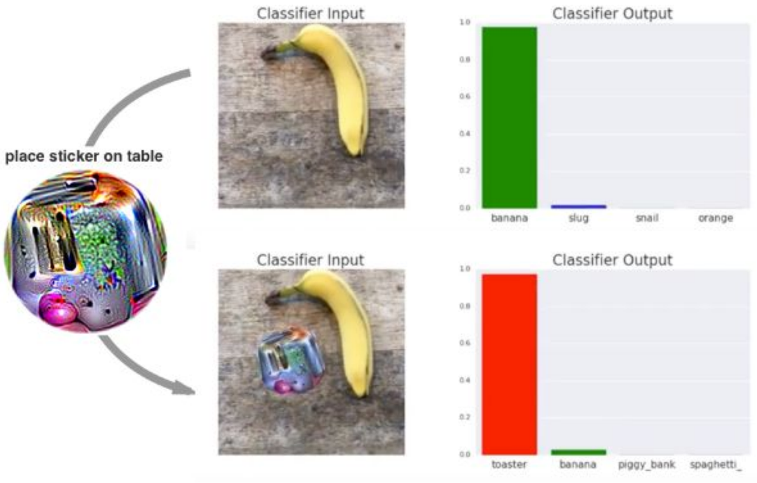
$$TV(r) = \sum_{i,j}\Big((r_{i,j} - r_{i+1,j})^2 + (r_{i,j} - r_{i,j+1})^2\Big)^{\frac{1}{2}}$$

$$\underset{r}{\operatorname{argmin}}\Big(\big(\sum_{x \in X} softmaxloss(x + r, c_t)\big) + \kappa_1 \cdot TV(r) + \kappa_2 \cdot NPS(r)\Big)$$

Sharif, Mahmood & Bhagavatula, Sruti & Bauer, Lujo & Reiter, Michael. (2016). Accessorize to a Crim
Real and Stealthy Attacks on State-of-the-Art Face Recognition.

Adversarial Patch

- Generate a "patch" that covers parts of image, can be printed out later to use in physical world
- **A(p, x, l, t)** - application operator applying patch **p**, to **x** with location **l** and translation **t**
- Optimize with gradient descent

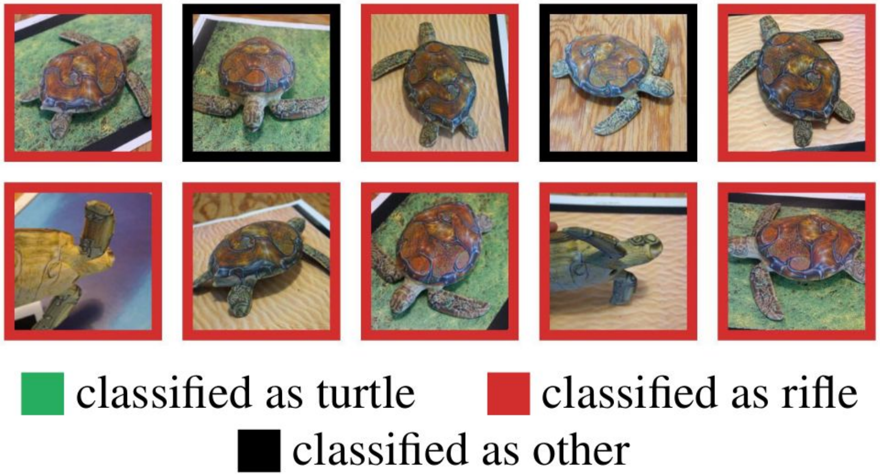


$$\widehat{p} = \arg\max_{p} \mathbb{E}_{x\sim X, t\sim T, l\sim L}\left[\log \Pr(\widehat{y}|A(p,x,l,t))\right]$$

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi: “Adversarial Patch”, 2017

3D printed adversarial objects

- Generate adversarial texture that can be applied to 3D printed objects
- **LAB** - color space in which numerical differences are proportional to perceptual differences
- **T** - set of translation functions



$$\arg\max_{x'} \mathbb{E}_{t\sim T}\Big[\log P(y_t|t(x')) - \lambda||LAB(t(x')) - LAB(t(x))||_2\Big]$$

Anish Athalye, Logan Engstrom, Andrew Ilyas: “Synthesizing Robust Adversarial Examples”, 2017

Code

Implementations / Packages

- Cleverhans
  - Implementations of most effective attacks
  - Tensorflow based, but compatible with Keras and PyTorch models
  - Maintained by authors of most methods (Goodfellow, Carlini, Papernot)
- Foolbox
  - Simpler API
  - More attacks, although some not effective
- IBM Adversarial Robustness Toolbox
  - Implementations of many attack and defense methods

Training Accuracy: 95%
Test Accuracy: 94%
Adversarial Example:

References

- Xiaoyong Yuan, Pan He, Qile Zhu: “Adversarial Examples: Attacks and Defenses for Deep Learning”, 2017
- Christian Szegedy et al: “Intriguing properties of neural networks”, 2013
- Ian J. Goodfellow, Jonathon Shlens: “Explaining and Harnessing Adversarial Examples”, 2014
- Nicholas Carlini: “Towards Evaluating the Robustness of Neural Networks”, 2016
- Anish Athalye, Logan Engstrom, Andrew Ilyas: “Synthesizing Robust Adversarial Examples”, 2017
- Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha: “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”, 2015
- Alexey Kurakin et al: “Adversarial Attacks and Defences Competition”, 2018