Accepted to the 1st IEEE European Symposium on Security & Privacy, IEEE 2016. Saarbrucken, Germany.

The Limitations of Deep Learning in Adversarial Settings

Nicolas Papernot*, Patrick McDaniel*, Somesh Jha†, Matt Fredrikson‡, Z. Berkay Celik*, Ananthram Swami§
*Department of Computer Science and Engineering, Penn State University
†Computer Sciences Department, University of Wisconsin-Madison
‡School of Computer Science, Carnegie Mellon University
§United States Army Research Laboratory, Adelphi, Maryland
{ngp5056,mcdaniel}@cse.psu.edu, {jha,mfredrik}@cs.wisc.edu, zbc102@cse.psu.edu, ananthram.swami.civ@mail.mil

Presented by: Hengtao Guo
08/026/2020

Motivation

1. The increasing use of deep learning is creating incentives for adversaries to manipulate DNNs to force misclassification of inputs.
2. Imperfections in the training phase of deep neural networks make them vulnerable to adversarial samples: inputs crafted by adversaries with the intent of causing deep neural networks to misclassify.
3. Example: If slightly altering “STOP” signs causes DNNs to misclassify them, the car would not stop, thus subverting the car’s safety.
4. Study adversarial samples can help DNN’s robust training.

Intriguing properties of neural networks

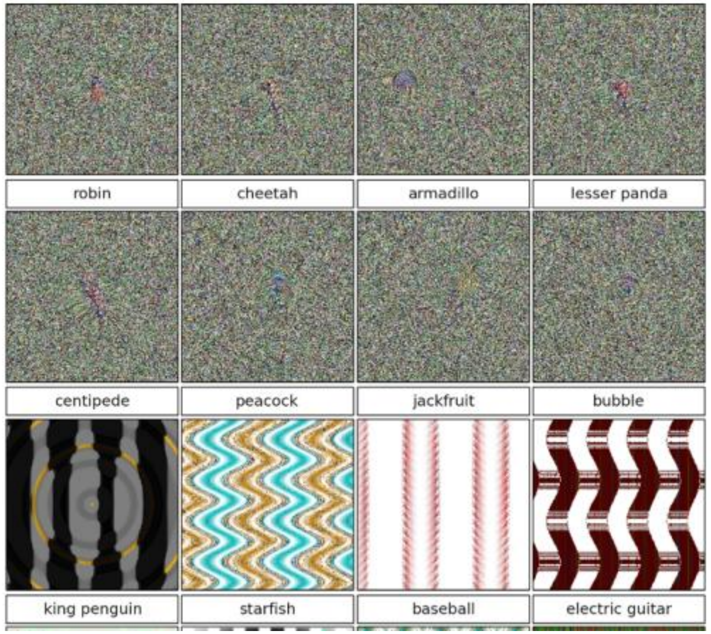
Christian Szegedy Google Inc.
Wojciech Zaremba New York University
Ilya Sutskever Google Inc.
Joan Bruna New York University
Dumitru Erhan Google Inc.
Ian Goodfellow University of Montreal
Rob Fergus New York University, Facebook Inc.

Small but carefully-crafted perturbations to an image of a vehicle resulted in the DNN classifying it as an ostrich.

Adversary's goal is to generate inputs that are correctly classified (or not distinguishable) by humans or other classifiers, but are misclassified by the targeted DNN.

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

Anh Nguyen, University of Wyoming, anguyen8@uwyo.edu
Jason Yosinski, Cornell University, yosinski@cs.cornell.edu
Jeff Clune, University of Wyoming, jeffclune@uwyo.edu



Producing images that are unrecognizable to humans, but are nonetheless labeled as recognizable objects by DNNs.

Summary

1. In this paper, we describe a new class of algorithms for adversarial sample creation against any feedforward (acyclic) DNN and formalize the threat model space of deep learning with respect to the integrity of output classification.
2. Unlike previous approaches mentioned above, we compute a direct mapping from the input to the output to achieve an explicit adversarial goal.
3. Furthermore, our approach only alters a (frequently small) fraction of input features leading to reduced perturbation of the source inputs. It also enables adversaries to apply heuristic searches to find perturbations leading to input targeted misclassifications (perturbing inputs to result in a specific output classification
4. We construct an adversarial sample X∗ from a benign sample X by adding a perturbation vector δX solving the following optimization problem

$$\arg\min_{\delta_{\mathbf{X}}} \|\delta_{\mathbf{X}}\| \text{ s.t. } \mathbf{F}(\mathbf{X} + \delta_{\mathbf{X}}) = \mathbf{Y}^* \tag{1}$$

Specifically, an adversary of a deep learning system seeks to provide an input $X*$ that results in an incorrect output classification.

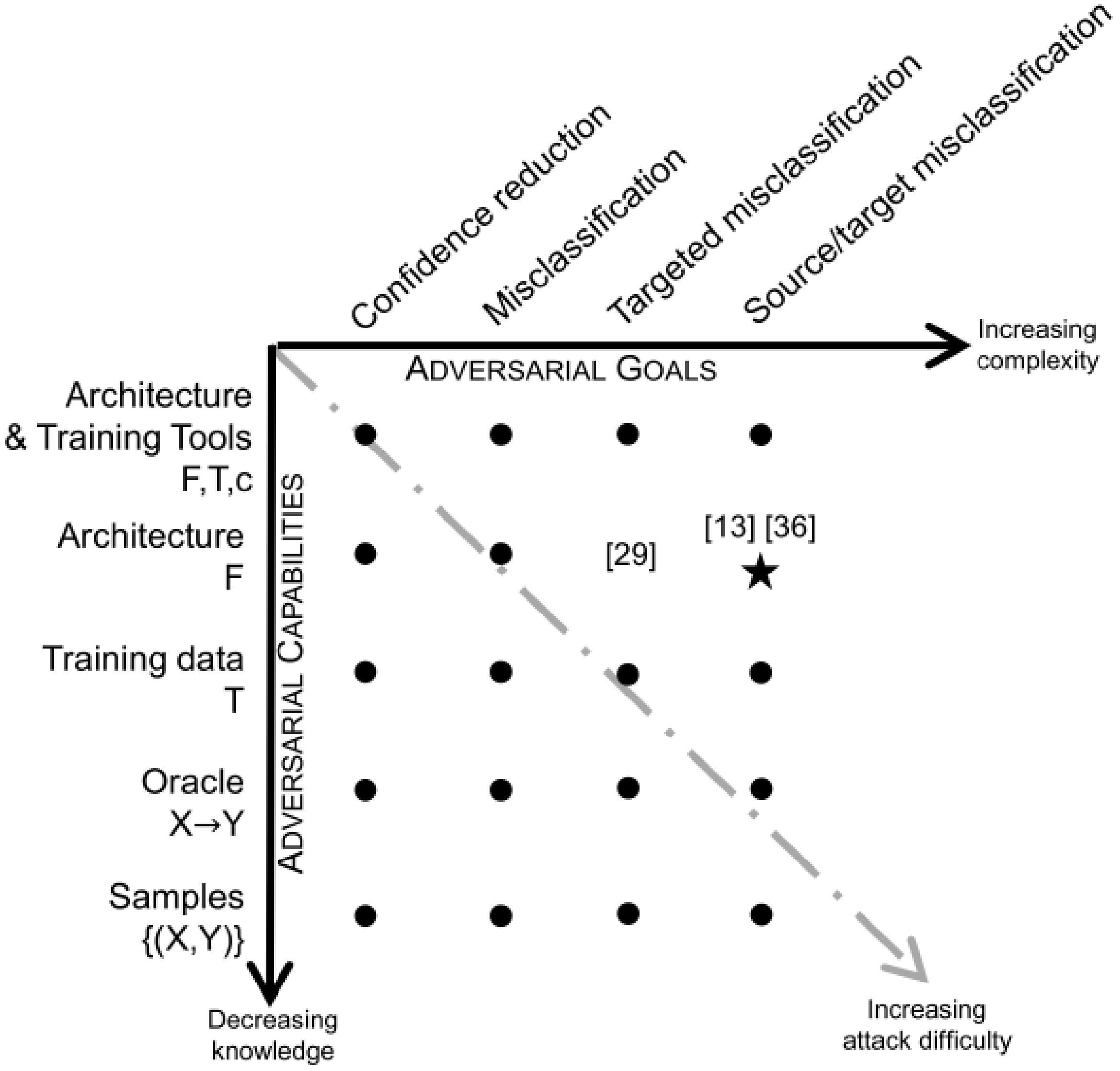


Fig. 2: Threat Model Taxonomy: our class of algorithms operates in the threat model indicated by a star.

Studying a Simple Neural Network

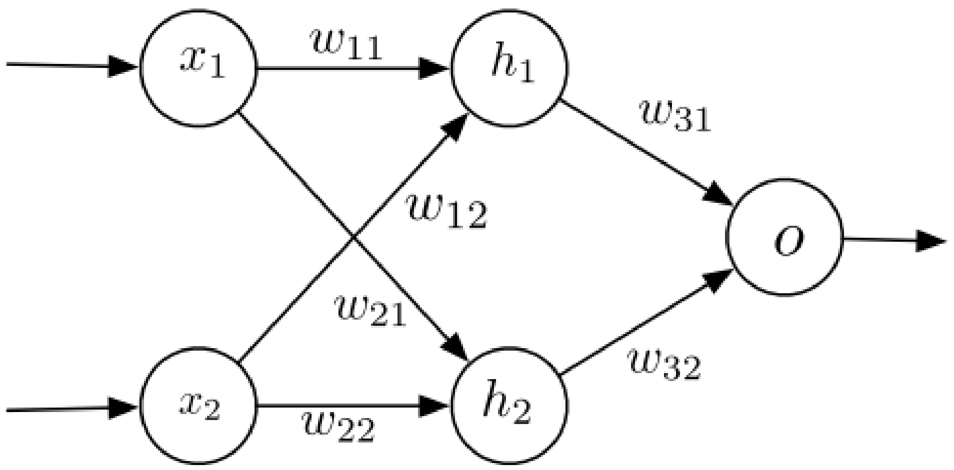


Fig. 3: Simplified Multi-Layer Perceptron architecture with input $\mathbf{X} = (x_1, x_2)$, hidden layer (h_1, h_2), and output o.

Studying a Simple Neural Network

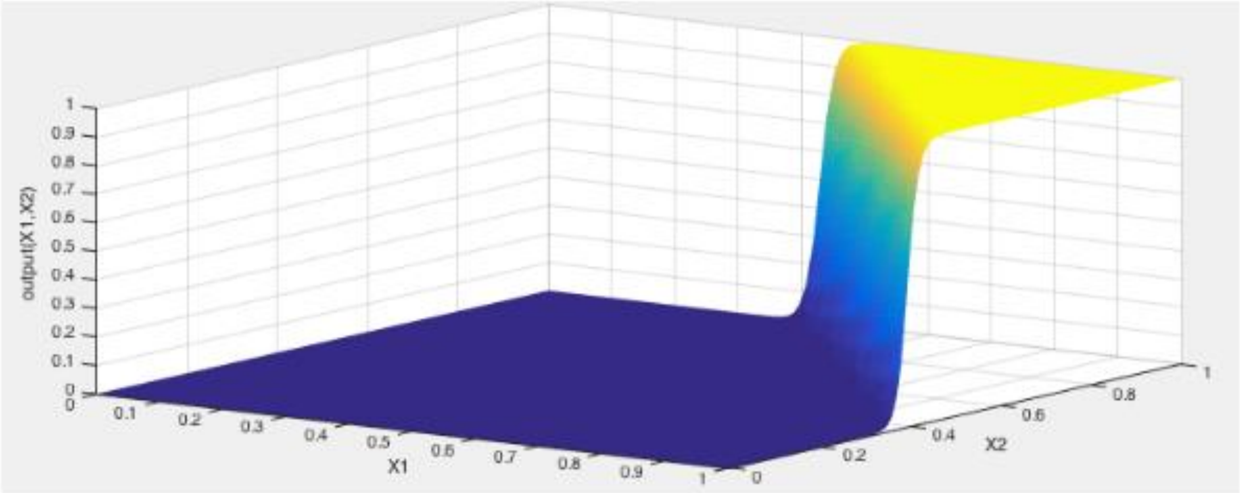


Fig. 4: The output surface of our simplified Multi-Layer Perceptron for the input domain $[0,1]^2$. Blue corresponds to a 0 output while yellow corresponds to a 1 output.

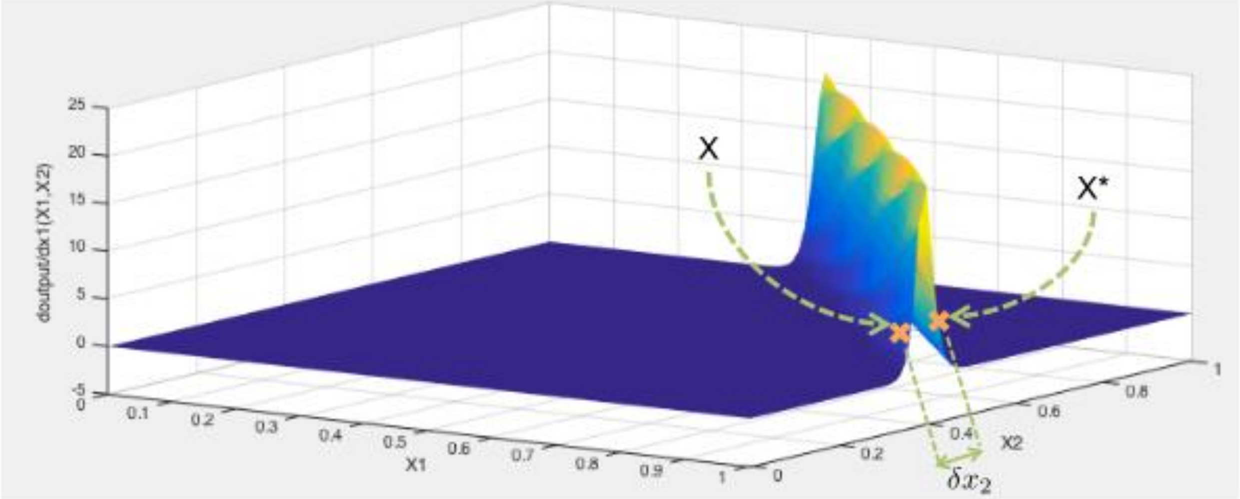


Fig. 5: Forward derivative of our simplified multi-layer perceptron according to input neuron x_2. Sample $\mathbf{X}$ is benign and $\mathbf{X}^*$ is adversarial, crafted by adding $\delta_{\mathbf{X}} = (0, \delta x_2)$.

$$\nabla \mathbf{F}(\mathbf{X}) = \left[\frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_1}, \frac{\partial \mathbf{F}(\mathbf{X})}{\partial x_2}\right]$$

We train this toy network to learn the AND function

Consider $\mathbf{X} = (1, 0.37)$ and $\mathbf{X}^* = (1, 0.43)$, which are both located near the spike in Figure 5. Although they only differ by a small amount ($\delta x_2 = 0.05$), they cause a significant change in the network's output, as $\mathbf{F}(\mathbf{X}) = 0.11$ and $\mathbf{F}(\mathbf{X}^*) = 0.95$.

Studying a Simple Neural Network

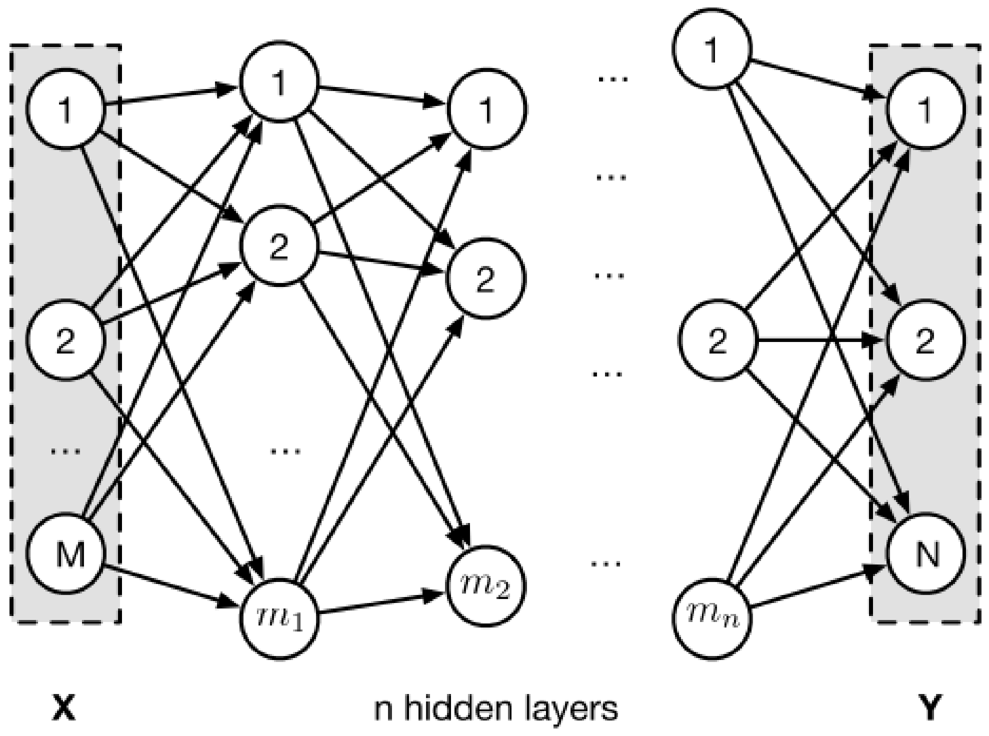
1. **Small input variations can lead to extreme variations of the output of the neural network,**
2. **Not all regions from the input domain are conducive to find adversarial samples, and**
3. **The forward derivative reduces the adversarial-sample search space**

Algorithm 1 **Crafting adversarial samples**

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

Input: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ

```
X* ← X
Γ = {1 ... |X|}
while F(X*) ≠ Y* and ||δ_X|| < Υ do
    Compute forward derivative ∇F(X*)
    S = saliency_map(∇F(X*), Γ, Y*)
    Modify X*_{i_max} by θ s.t. i_max = arg max_i S(X, Y*)[i]
    δ_X ← X* − X
end while
return X*
```



Notations

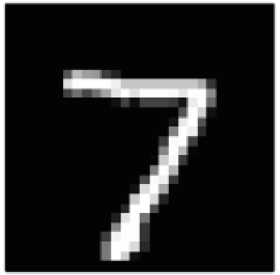
F: function learned by neural network during training
X: input of neural network
Y: output of neural network
M: input dimension (number of neurons on input layer)
N: output dimension (number of neurons on output layer)
n: number of hidden layers in neural network
f: activation function of a neuron
H_k: output vector of layer k neurons

Indices

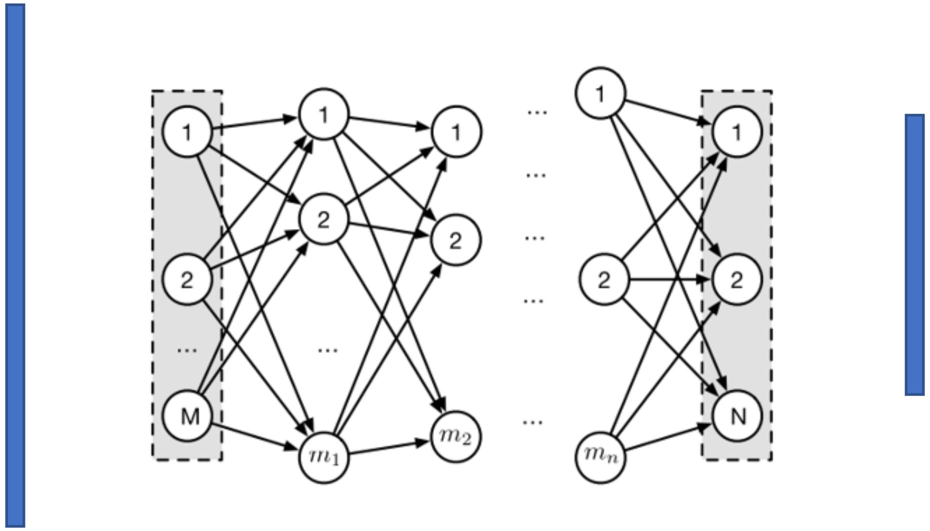
k: index for layers (between 0 and n+1)
i: index for input X component (between 0 and N)
j: index for output Y component (between 0 and M)
p: index for neurons (between 0 and m_k for any layer k)

Generalizing to Feedforward DNNs

1. The only assumption: the architecture's neurons form an acyclic DNN, and each use a differentiable activation function.



28*28



M=28*28=784 M*N Jacobian matrix N=10

2. Essentially the Jacobian of the function corresponding to what the neural network learned during training.

$$\nabla \mathbf{F}(\mathbf{X}) = \frac{\partial \mathbf{F}(\mathbf{X})}{\partial \mathbf{X}} = \left[\frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial x_i}\right]_{i\in 1..M,\, j\in 1..N} \qquad (3)$$

Forward Derivatives

1. Instead of propagating gradients backwards, we choose in our approach to propagate them forward, allowing to find input components that lead to significant changes in network outputs.
2. We now consider one element (i, j) ∈ [1..M] × [1..N] of the **M ×N forward derivative matrix** defined in Equation 3: that is the derivative of one output neuron Fj according to one input dimension xi.
3. Start at the first hidden layer of the neural network. We can differentiate the output of this first hidden layer in terms of the input components.

$$\frac{\partial \mathbf{H}_k(\mathbf{X})}{\partial x_i} = \left[\frac{\partial f_{k,p}(\mathbf{W}_{k,p} \cdot \mathbf{H}_{k-1} + b_{k,p})}{\partial x_i}\right]_{p \in 1..m_k}$$

4. By applying the chain rule:

$$\left.\frac{\partial \mathbf{H}_k(\mathbf{X})}{\partial x_i}\right|_{p \in 1..m_k} = \left(\mathbf{W}_{k,p} \cdot \frac{\partial \mathbf{H}_{k-1}}{\partial x_i}\right) \times \frac{\partial f_{k,p}}{\partial x_i}(\mathbf{W}_{k,p} \cdot \mathbf{H}_{k-1} + b_{k,p}) \quad (5)$$

$$\frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial x_i} = \left(\mathbf{W}_{n+1,j} \cdot \frac{\partial \mathbf{H}_n}{\partial x_i}\right) \times \frac{\partial f_{n+1,j}}{\partial x_i}(\mathbf{W}_{n+1,j} \cdot \mathbf{H}_n + b_{n+1,j}) \quad (6)$$

Adversarial Saliency Map

1. These maps indicate which input features an adversary should perturb in order to effect the desired changes in network output most efficiently.
2. Adversarial saliency maps are defined to suit problem-specific adversarial goals.

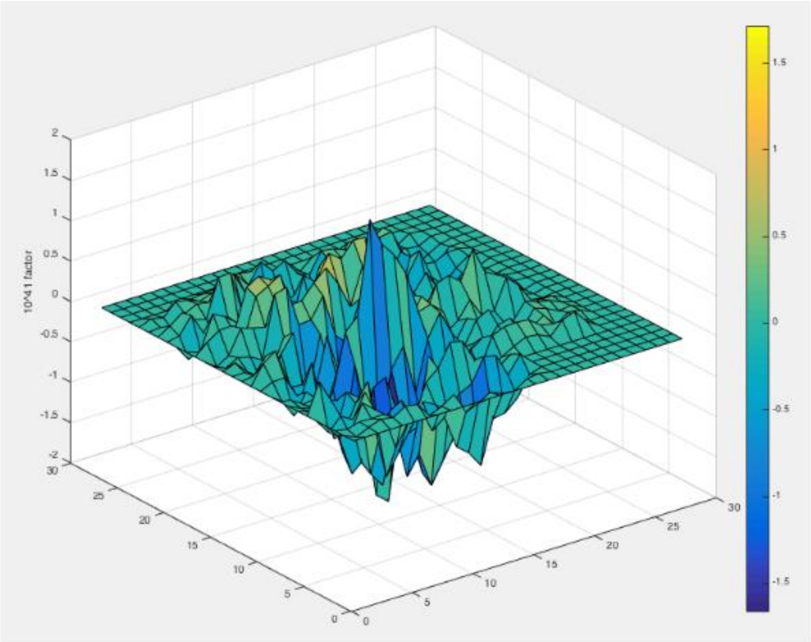


Fig. 7: Saliency map of a 784-dimensional input to the LeNet architecture (cf. validation section). The 784 input dimensions are arranged to correspond to the 28x28 image pixel alignment. Large absolute values correspond to features with a significant impact on the output when perturbed.

$$S(\mathbf{X},t)[i] = \begin{cases} 0 \text{ if } \frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i} < 0 \text{ or } \sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i} > 0 \\ \left(\frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}\right) \left|\sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}\right| \text{ otherwise} \end{cases} \tag{8}$$

where i is an input feature. The condition specified on the first line rejects input components with a negative target derivative or an overall positive derivative on other classes. Indeed, $\frac{\partial \mathbf{F}_t(\mathbf{X})}{\partial \mathbf{X}_i}$ should be positive in order for $\mathbf{F}_t(\mathbf{X})$ to increase when feature $\mathbf{X}_i$ increases. Similarly, $\sum_{j \neq t} \frac{\partial \mathbf{F}_j(\mathbf{X})}{\partial \mathbf{X}_i}$ needs to be negative to decrease or stay constant when feature $\mathbf{X}_i$ is increased. The product on the second line allows us to consider

Crafting in Digits

1. We investigate a DNN based on the well-studied LeNet architecture, which has proven to be an excellent classifier for handwritten digits.
2. We train our network using the MNIST training dataset of 60,000 samples, normalized.
3. Algorithm 2 iteratively modifies a sample X by perturbing two input features (i.e., pixel intensities) p1 and p2 selected by saliency_map.
4. The crafting algorithm is fine-tuned by three parameters:
   a) **Maximum distortion Y**: expressed as a percentage, corresponds to the maximum number of pixels to be modified when crafting the adversarial sample
   b) **Saliency map**
   c) **Feature variation per iteration θ**

Algorithm 2 **Crafting adversarial samples for LeNet-5**

$\mathbf{X}$ is the benign image, $\mathbf{Y}^*$ is the target network output, $\mathbf{F}$ is the function learned by the network during training, Υ is the maximum distortion, and θ is the change made to pixels.

Input: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ

$\mathbf{X}^* \leftarrow \mathbf{X}$

$\Gamma = \{1 \dots |\mathbf{X}|\}$ ▷ search domain is all pixels

$\texttt{max\_iter} = \left\lfloor \frac{784 \cdot \Upsilon}{2 \cdot 100} \right\rfloor$

$s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$ ▷ source class

$t = \arg\max_j \mathbf{Y}^*_j$ ▷ target class

while $s \neq t$ & $\texttt{iter} < \texttt{max\_iter}$ & $\Gamma \neq \emptyset$ **do**

  Compute forward derivative $\nabla \mathbf{F}(\mathbf{X}^*)$

  $p_1, p_2 = \texttt{saliency\_map}(\nabla \mathbf{F}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$

  Modify p_1 and p_2 in $\mathbf{X}^*$ by θ

  Remove p_1 from Γ if $p_1 == 0$ or $p_1 == 1$

  Remove p_2 from Γ if $p_2 == 0$ or $p_2 == 1$

  $s = \arg\max_j \mathbf{F}(\mathbf{X}^*)_j$

  $\texttt{iter}++$

end while

return $\mathbf{X}^*$

described in Equation 8. Searching for pairs of pixels is more likely to match the condition because one of the pixels can compensate a minor flaw of the other pixel. Let's consider a simple example: p_1 has a target derivative of 5 but a sum of other classes derivatives equal to 0.1, while p_2 as a target derivative equal to -0.5 and a sum of other classes derivatives equal to -6. Individually, these pixels do not match the

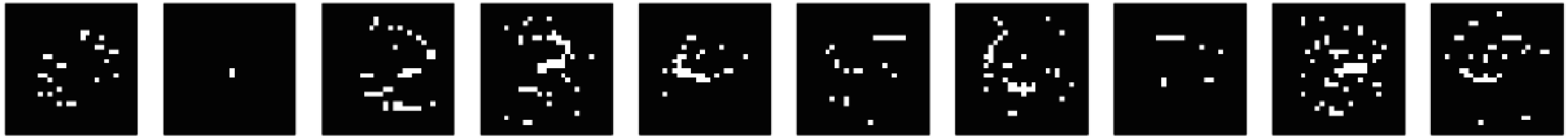


Fig. 9: Adversarial samples generated by feeding the crafting algorithm an empty input. Each sample produced corresponds to one target class from 0 to 9. Interestingly, for classes $0, 2, 3$ and 5 one can clearly recognize the target digit.

Experiments

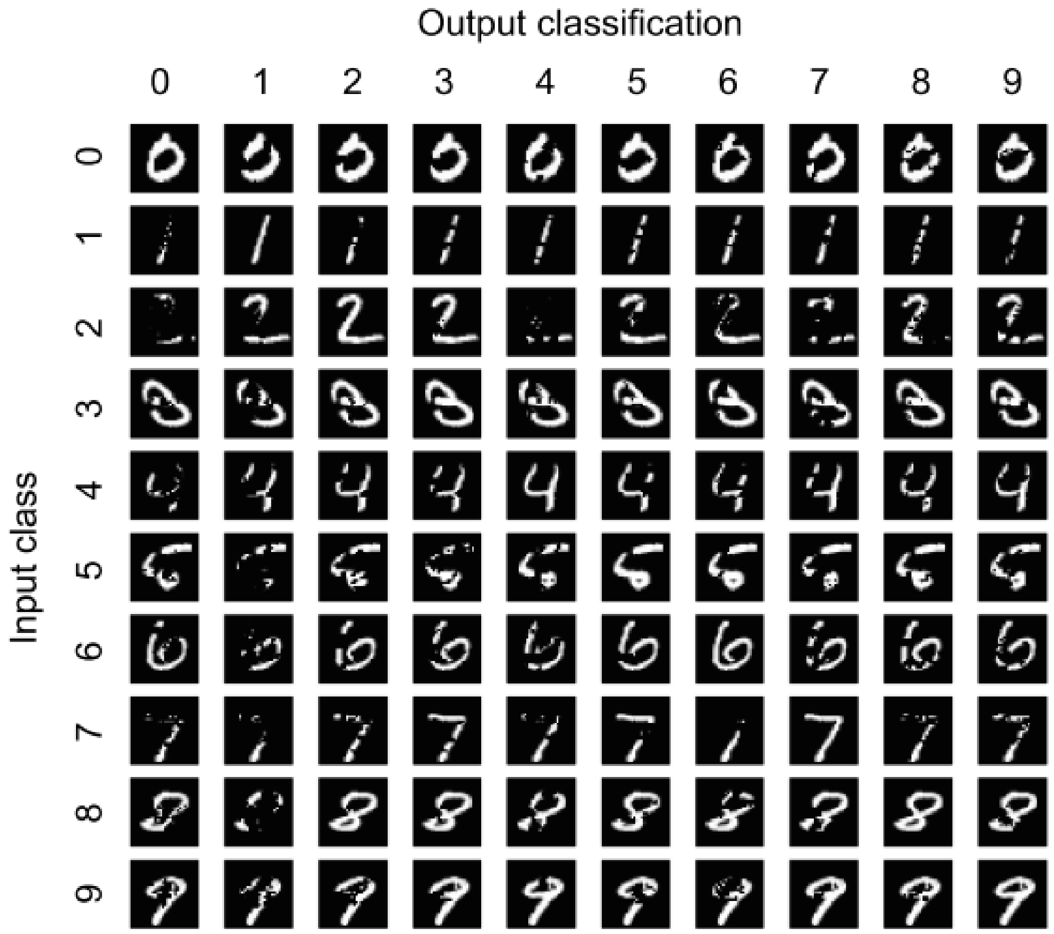


Fig. 10: Adversarial samples obtained by decreasing pixel intensities. Original samples from the MNIST dataset are found on the diagonal, whereas adversarial samples are all non-diagonal elements. Samples are organized by columns each corresponding to a class from 0 to 9.

Source set of $10,000$ original samples	Adversarial samples successfully misclassified	Average distortion	
		All adversarial samples	Successful adversarial samples
Training	97.05%	4.45%	4.03%
Validation	97.19%	4.41%	4.01%
Test	97.05%	4.45%	4.03%

Fig. 11: Results on larger sets of $10,000$ samples

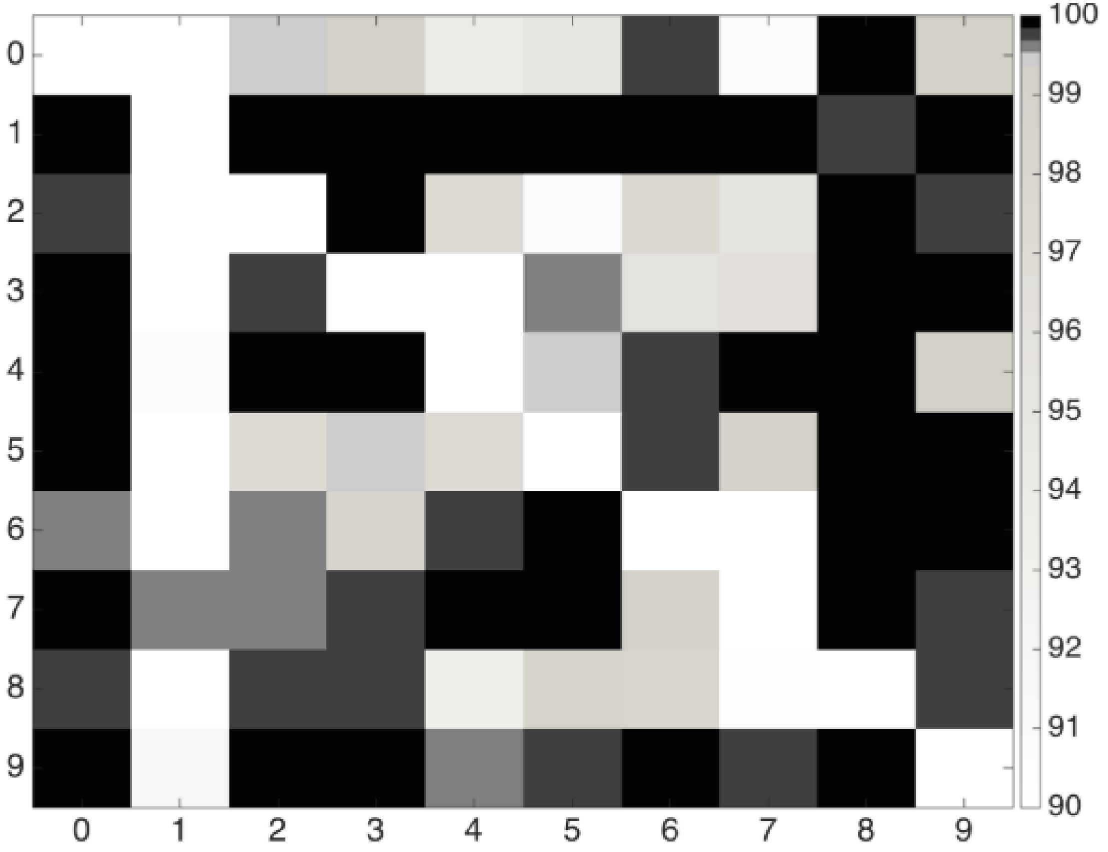


Fig. 12: Success rate per source-target class pair.

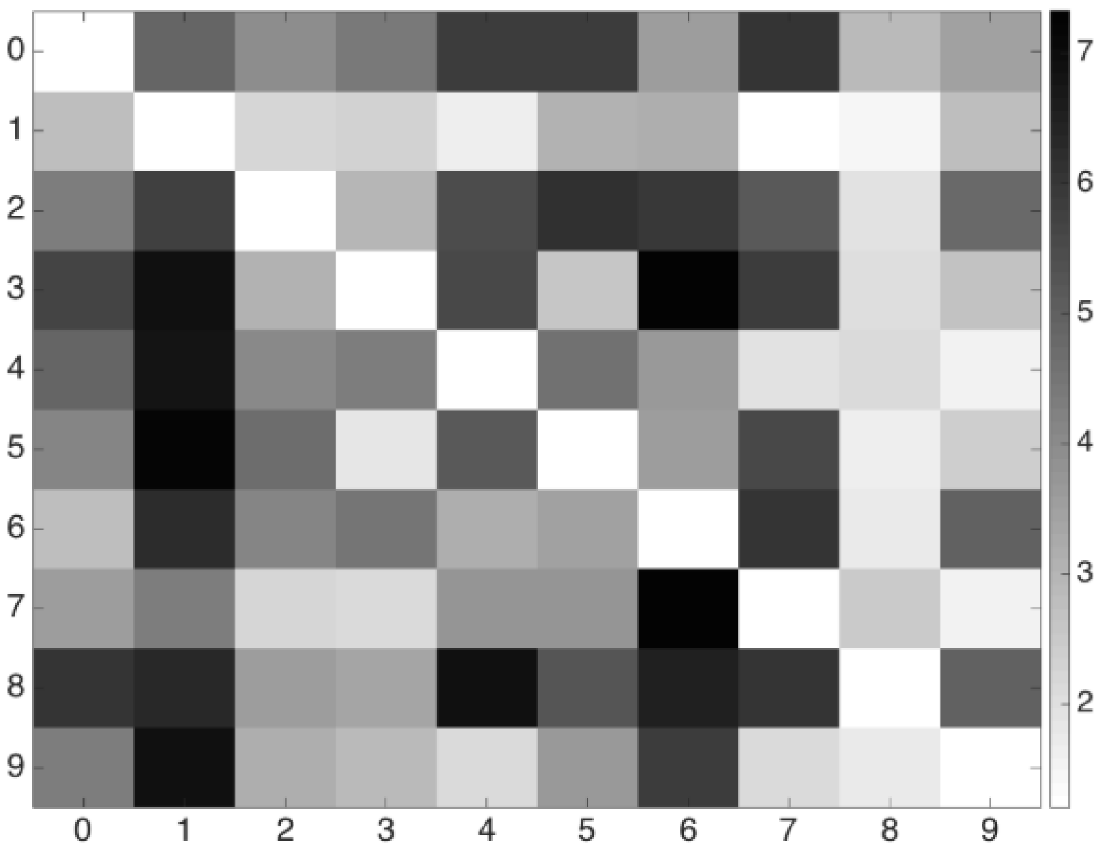


Fig. 13: Average distortion ε of successful samples per source-target class pair. The scale is a percentage of pixels.

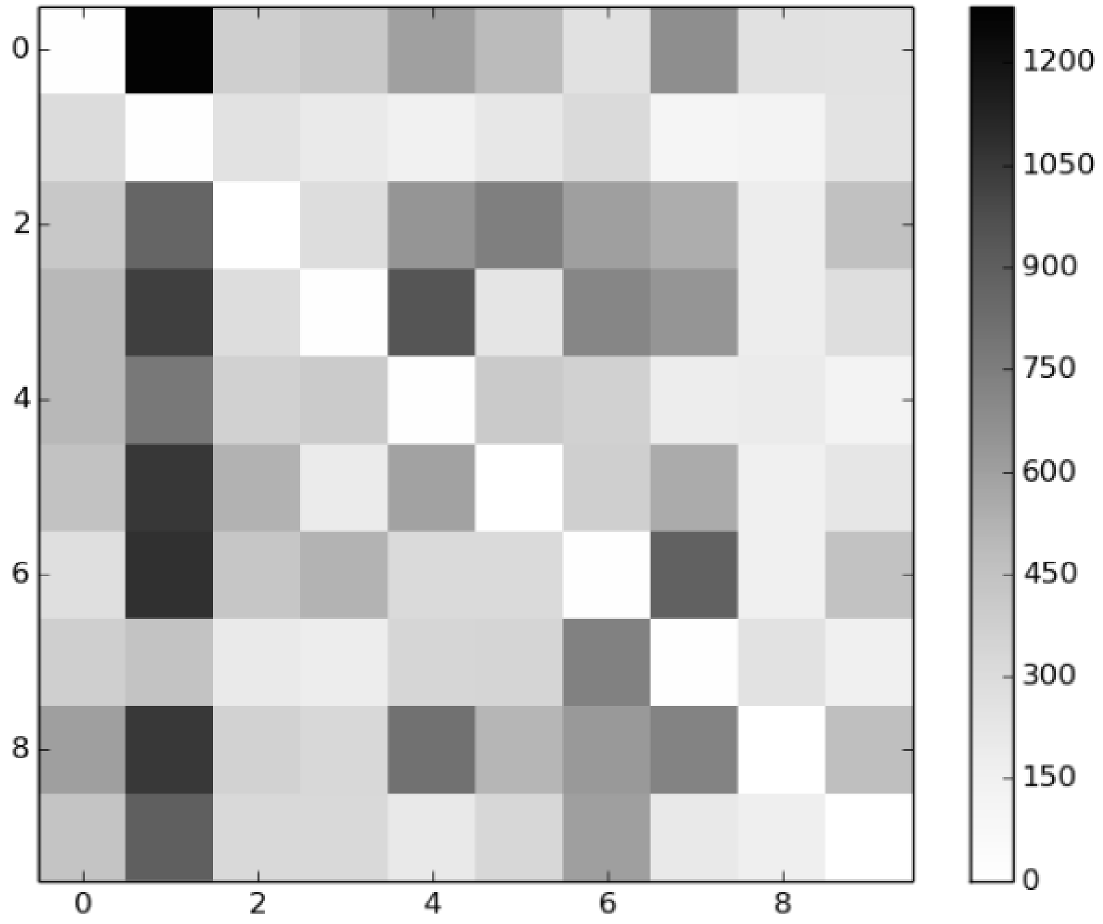


Fig. 14: Hardness matrix of source-target class pairs. Darker shades correspond to harder to achieve misclassifications.

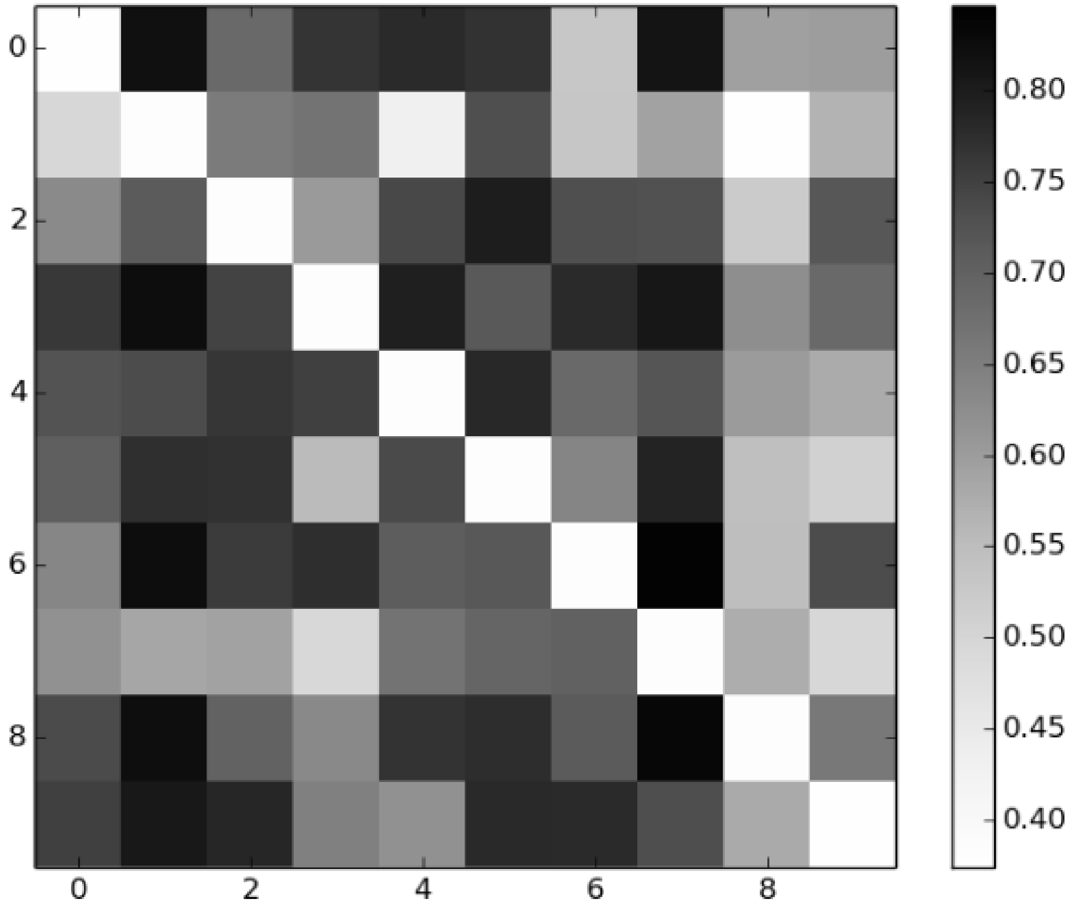


Fig. 15: Adversarial distance averaged per source-destination class pairs computed with 1000 samples.

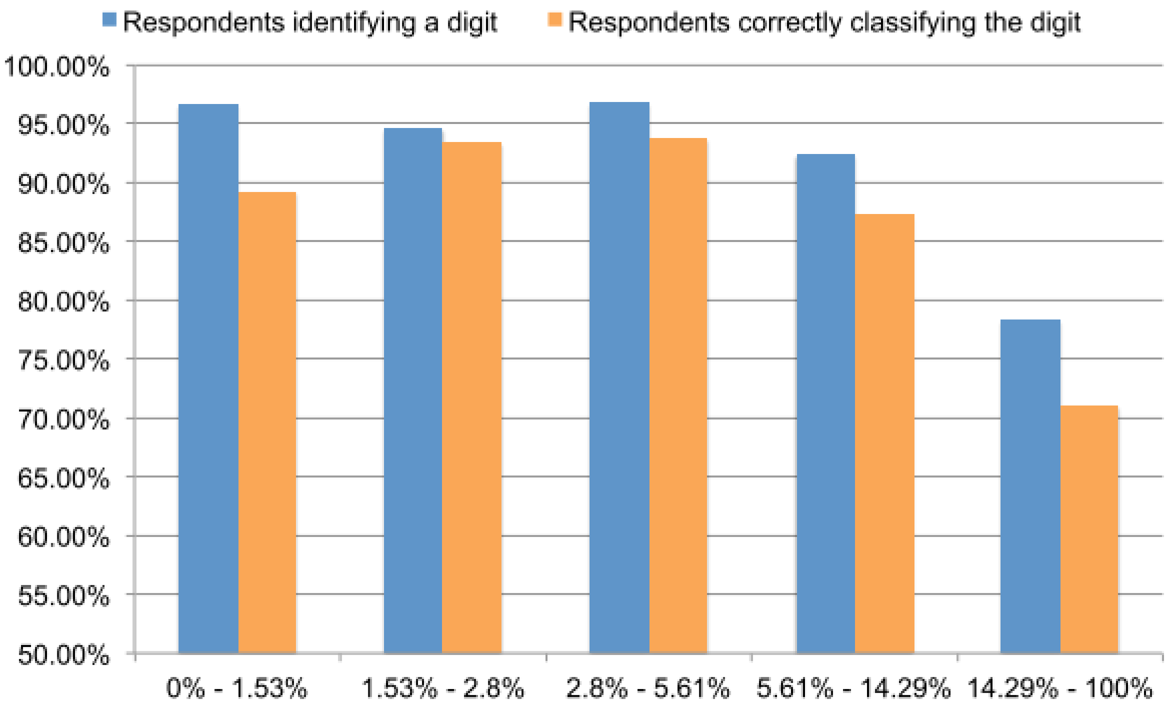


Fig. 16: Human perception of different distortions ε.

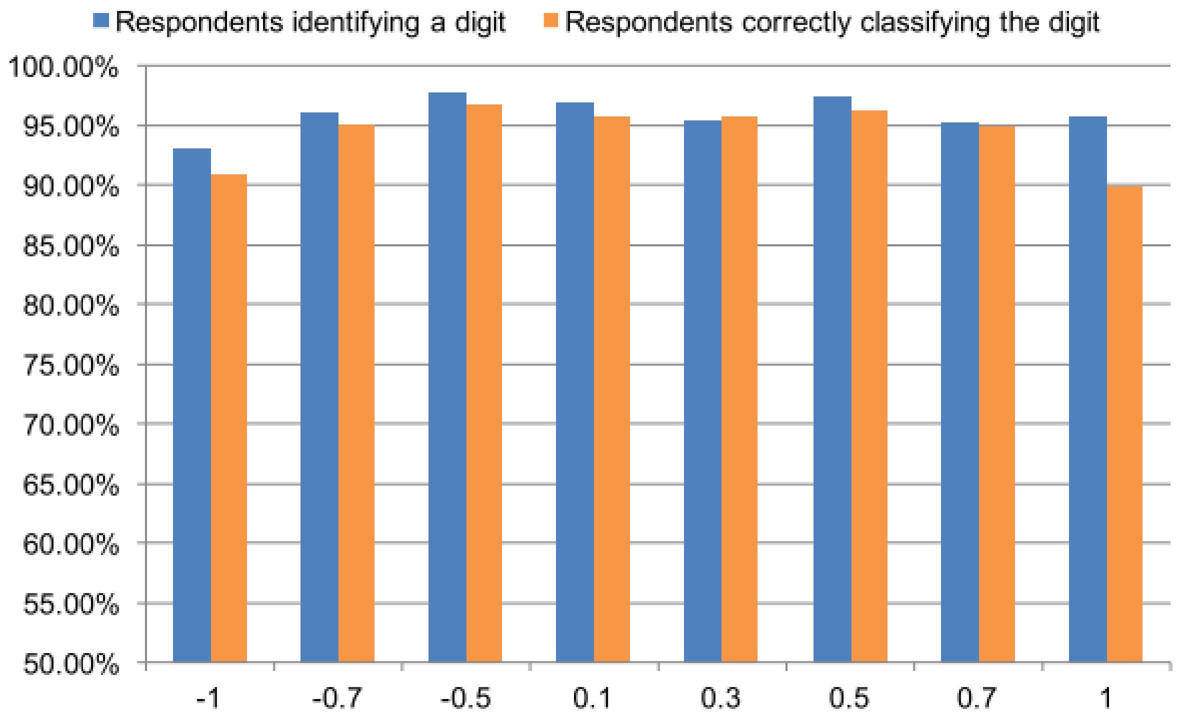


Fig. 17: Human perception of different intensity variations θ.

participants were asked for each sample: (a) 'is this sample a numeric digit?', and (b) 'if yes to (a) what digit is it?'. These

Conclusions

1. In addition to exploring the goals and capabilities of DNN adversaries, we introduced a new class of algorithms to craft adversarial samples based on computing forward derivatives.
2. Given the network's weights and input image's intensities, we can compute a forward derivative Jacobian Matrix.
3. Such a Jacobian Matrix can tell where and by how much we can change the original image into any target class.