Adversarial examples

Recall the process by which we "invert" an image classifier:

Our optimization becomes

$$\begin{aligned}
&\mathbf{x} = \operatorname*{argmin}_{\mathbf{x}} \mathcal{L}^{(\mathbf{i})} \\
&\text{subject to} \\
&\hat{\mathbf{y}}^{(\mathbf{i})} = \mathbf{y}^{(0)}
\end{aligned}$$

which we solved via the derivative of the loss with respect to the inputs

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$$

That is, we are

- starting with $\mathbf{x} = \mathbf{x}^{(0)}$
- modifying $\mathbf{x}$
- to derive an input $\mathbf{x}$ that causes the NN to predict $\hat{y} = \mathbf{y}^{(0)}$ with highest probability

We are inverting the output.

We originally specified initializing the image with with $\mathbf{x} = \mathbf{x}^{(0)}$ where $\mathbf{x}^{(0)}$ was either random noise or all black image.

What would happen if

- we initialized $\mathbf{x} = \mathbf{x}^{(i')}$
- where $\mathbf{y}^{(0)} \neq \mathbf{y}^{(i')}$?

That is: our initial image is from class $\mathbf{y}^{(i')}$ but we give an objective target of $\mathbf{y}^{(0)} \neq \mathbf{y}^{(i')}$

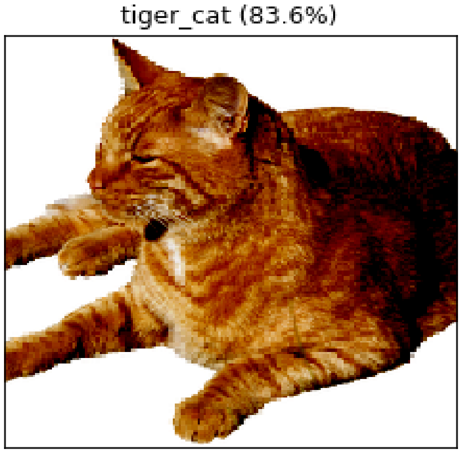
Gradient ascent would create an output

- classified as $\mathbf{y}^{(0)}$
- by modifying an image that is *not* from this class

The $\mathbf{x}$ created is called an *Adversarial Example* as it was intentionally created to cause misclassificatio.

Adversarial examples in action:

What class is this ?



What about this ?

What class is this ?

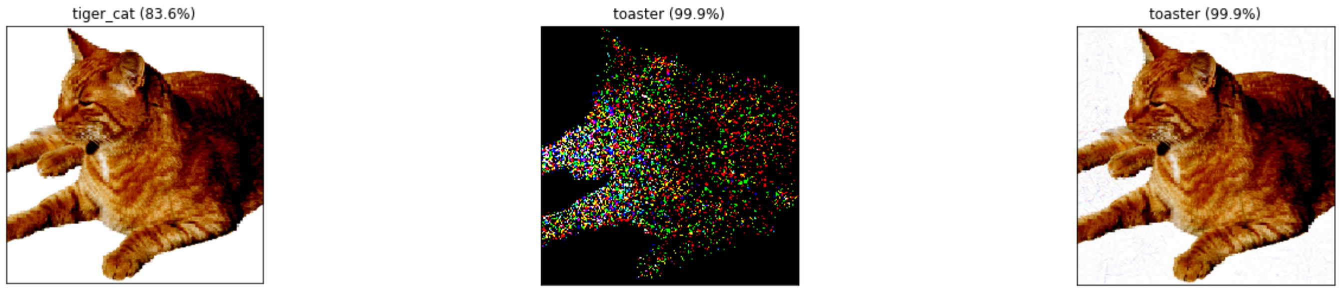
toaster (99.9%)

It's almost certainly a toaster !

Your eye can't pick up the difference: that's a real-world problem !

Here is the difference between the two images.

Adversarial Cat to Toaster



What harm can this do ?

Adversarial Stop Sign

So what ? Adversarial Examples 2

“Speed Limit 45”

Eykholt et. all, https://arxiv.org/pdf/1707.08945.pdf

[Robust Physical-World Attacks on Deep Learning Models (https://arxiv.org/abs/1707.08945)](https://arxiv.org/abs/1707.08945)

Why does this occur ?

Recall the fundamental assumption of Machine Learning:

- an example from the test set is drawn from the same distribution as the training set

In the case of Adversarial Examples, this condition is not satisfied.

How do we remedy this before I get into a self-driving car ?

- Adversarial training
    - augment training set with adversarial images
    - but attacks are very robust
        - if this is some artifact that can signal fakery
        - adjust your Cost function to penalize for creating the artifact !

- Can create adversarial example
    - without manipulating training set
    - without manipulating the trained classifier's weights
    - without access to the classifier !
        - black box versus white box attacks
- It turns out that an adversarial example that can fool *several* classifiers
    - is also good at fooling a (time-limited) human !

Adversarial Reprogramming

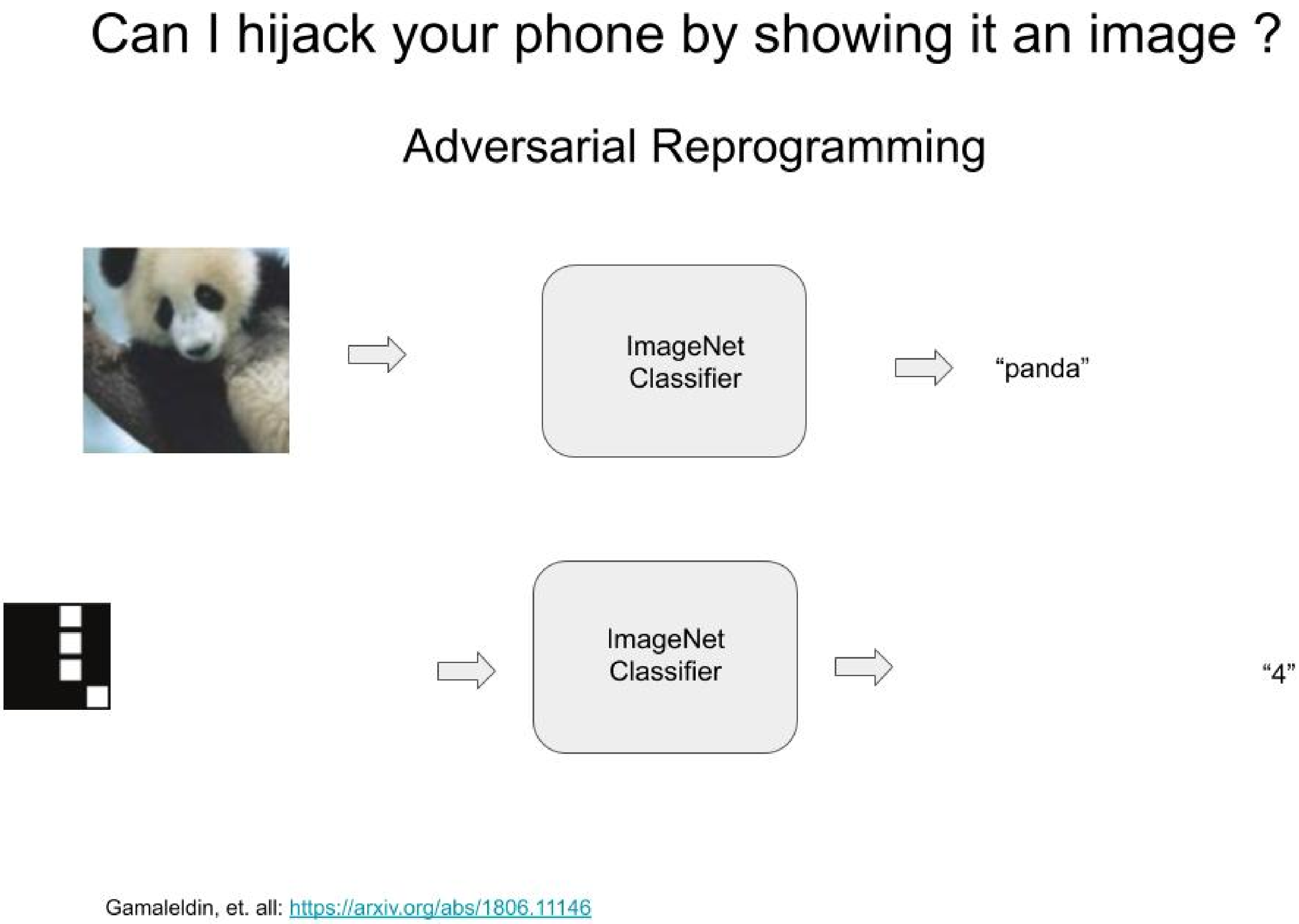
We can extend the Gradient Ascent method to perform even bigger tricks:

- Getting a Classifier for Task 1 to do something completely different !

Can we get an ImageNet Classifier to count squares ? Imagenet

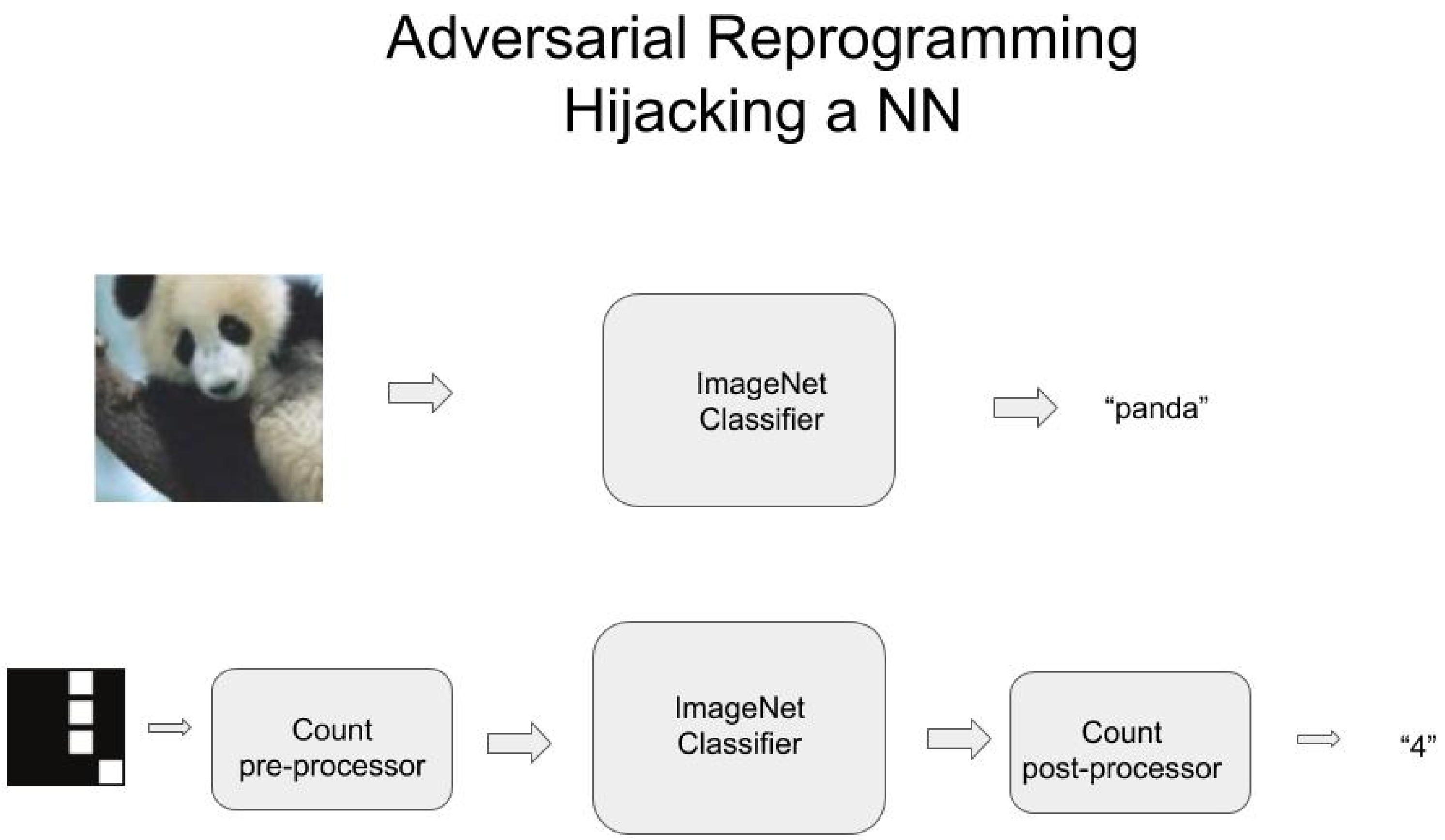
- does not have squares as an input image
- or numbers as an output class

This is called *Adversarial Reprogramming*.



[Adversarial Reprogramming of Neural Networks (https://arxiv.org/abs/1806.11146)](https://arxiv.org/abs/1806.11146)

Here's a pictorial to describe the process:



We refer to our original classifier as solving the Source task.

Our goal is to get the classifier to solve the Target task.

The first issue to address:

- the $(\mathbf{x^{(i)}}, \mathbf{y^{(i)}})$ pairs of the Source task come from a different domain than that of the Target task

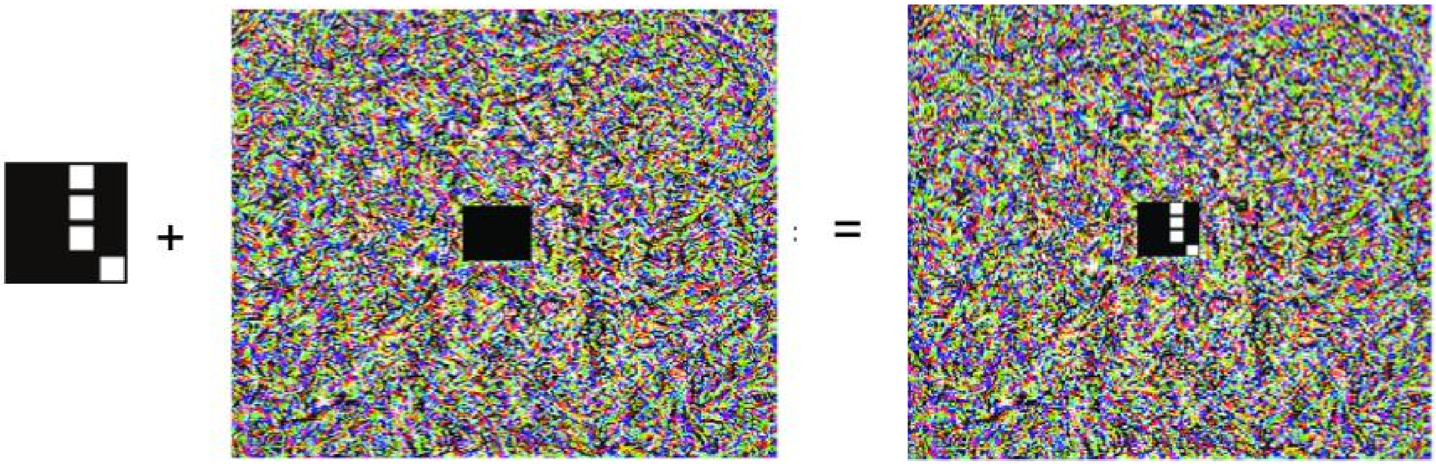
$$
\begin{array}{lll}
\mathbf{X}_\text{source}, \mathbf{y}_\text{source} & : & \text{examples for Source task} \\
\mathbf{X}_\text{target}, \mathbf{y}_\text{target} & : & \text{examples for Target task}
\end{array}
$$

We create a simple function h_f to map an $\mathbf{x} \in \mathbf{X}_{\text{target}}$ to an $\mathbf{x} \in \mathbf{X}_{\text{source}}$.

This ensures that the input to the Source task is of the right "type".

Adversarial Reprogramming

Adversarial Program



X_t + W = $\hat{X}_s$

h_f Counting pre-processor

h_f simply embeds the Target input into an image, which is the domain of the Source task.

Similarly, we create a function h_g to map the Target label to a Source Label.

This will ensure that the output of the Source task is of the right type.

Adversarial Reprogramming

(a)

counting y_{adv}	**ImageNet** y
1 square	tench
2 squares	goldfish
3 squares	white shark
4 squares	tiger shark
5 squares	hammerhead
6 squares	electric ray
7 squares	stingray
8 squares	cock
9 squares	hen
10 squares	ostrich

h_g Counting post-processor

Finally, the Cost function to optimize

$$\mathbf{W} = \underset{\mathbf{W}}{\operatorname{argmin}} - \log(p(h_g(\mathbf{y}_t) \mid \tilde{\mathbf{X}}_{\text{source}})) + \lambda||\mathbf{W}||^2$$

where

$$\begin{aligned}
\tilde{\mathbf{X}}_{\text{source}} &= h_f(\mathbf{W}, \mathbf{X}_{\text{target}}) \\
h_f : & \quad \mathbf{y}_{\text{target}} \mapsto \mathbf{y}_{\text{source}} \quad \text{map source X to target X} \\
h_g : & \quad \mathbf{y}_{\text{target}} \mapsto \mathbf{y}_{\text{source}} \quad \text{map source label y to target label}
\end{aligned}$$

- Given an input in the Target domain $\mathbf{X}_{\text{target}}$
- Transform it into an input $\tilde{\mathbf{X}}_{\text{source}}$ in the Source domain.
- Use the Source Classifier to predict $\tilde{\mathbf{X}}_{\text{source}}$ a label in the Source domain
    - The correct label in the Target domain is $\mathbf{y}_t$
    - This maps to label $h_g(\y_t) in the Source domain

So we are trying to

- maximize the likelihood that the Source classifier creates the encoding for the correct Target label
- subject to constraining the weights $\mathbf{W}$ (the "frame" into which the Target input is placed)

How do we find the frame $\mathbf{W}$ that "reprograms" the Source Classifier ?

By training it of course ! Just plain old ML.

Misaligned objectives

We have framed the problem of Deep Learning as one of defining a Cost function that meets your objectives.

This is not as easy as it sound.

Consider the difference between

- "Maximize profit"
- "Maximize profit subject to legal and ethical constraints"

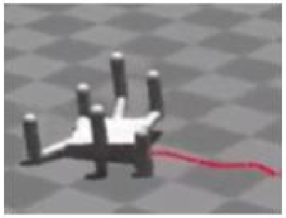
We (hopefully) don't have to state the additional constraints to a human -- we take it for granted.

Not so with a machine that has not been trained with additional objectives.

AI Safety

- AI Safety = Harmed caused **by** AI
- Some causes:
    - Biased training data
        - Polar bears
    - Objective functions not fully aligned with human goals
        - Consider
            - Maximize reward
            - Maximize reward subject to legal and moral norms
        - Reward Hacking in Reinforcement Learning





In [ ]:
```
print("Done")
```