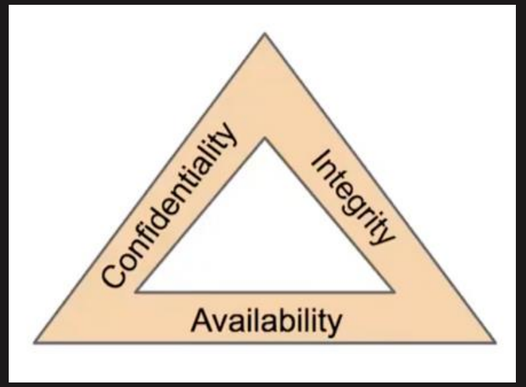
CRYPTOGRAPHY AND NETWORK SECURITY

OBFUSCATED GRADIENTS GIVE A FALSE SENSE OF SECURITY: CIRCUMVENTING DEFENSES TO ADVERSARIAL EXAMPLES

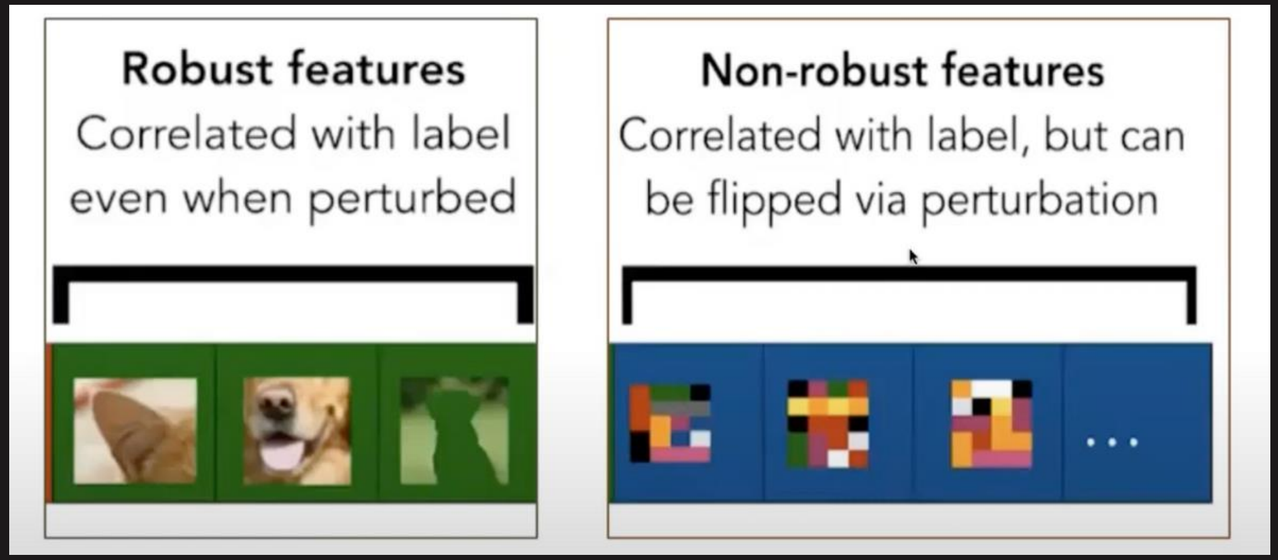
Suparshwa Patil (patis04)
Shruti Mandaokar (mands01)

ARE MACHINE LEARNING ALGORITHMS ROBUST ENOUGH?



Many modern ML algorithms.
- get the right answer for wrong reason on ML inputs.
- get very wrong answers on almost all inputs.

ML ALGORITHMS DO NOT LEARN LIKE HUMANS DO (at least not yet!)

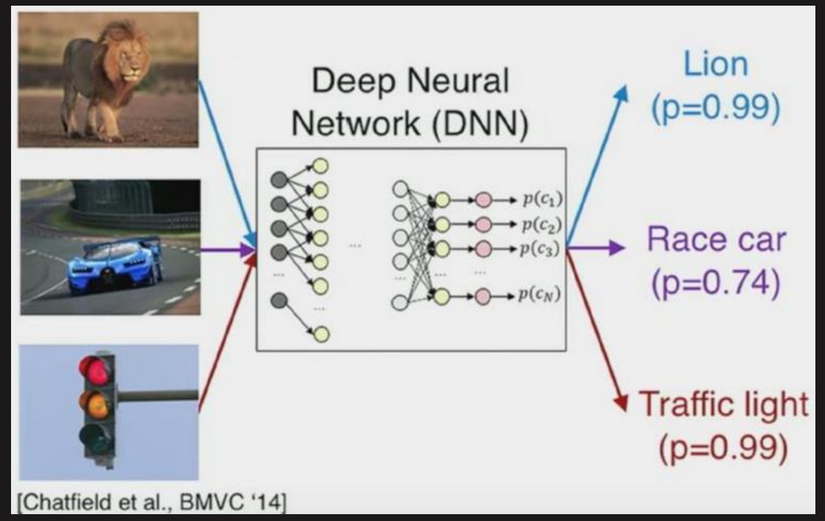


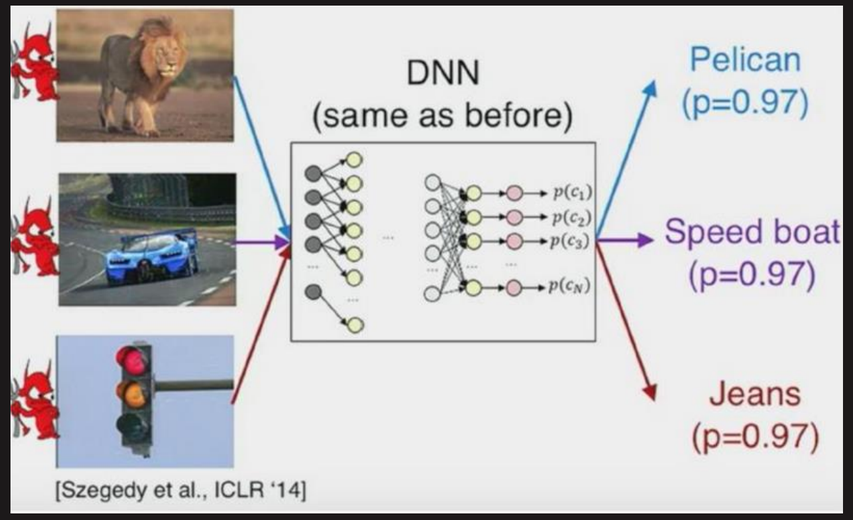
Two Questions :
- Privacy (Is the ML model protecting our privacy?)
- Trustworthiness (Can I trust the prediction of my ML model?)

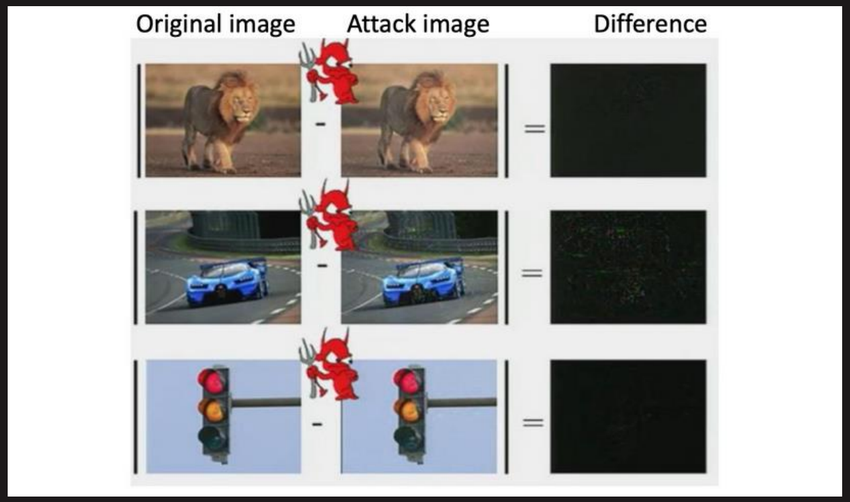
We don’t want our classifier to drastically change decisions with minor perbutations.

ADVERSARIAL MACHINE LEARNING :)

What do you see?







The differences between the original and manipulated images are very small (hardly noticeable to the human eye)

BUT, WHAT ABOUT MACHINES?

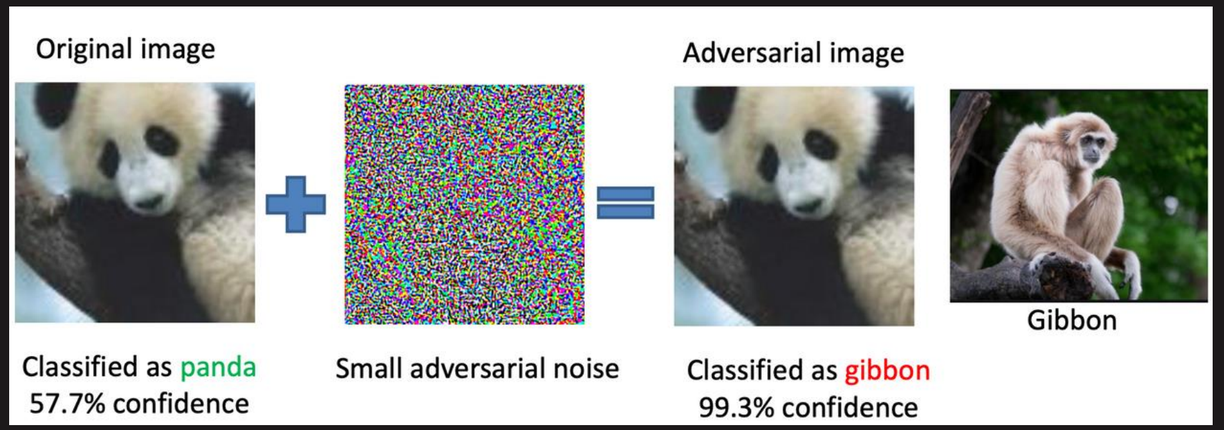
ADVERSARIAL NETWORKS: ILLUSIONS FOR MACHINES

Inputs to Machine Learning Models that are intentionally crafted to cause misclassification.

Just like Optical Illusions for machines.

Small changes to inputs, sometimes imperceptible to the human eye, can result in misclassification.

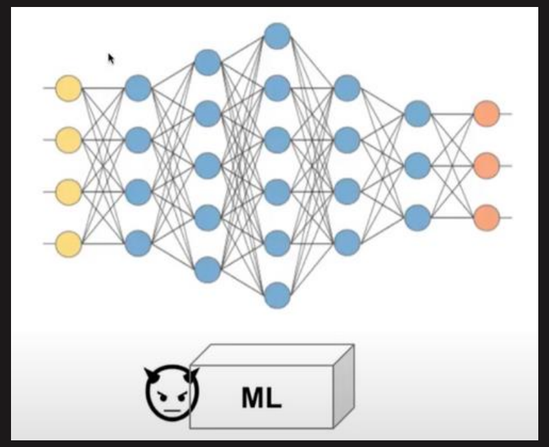
Adversarial ML:
ML in adversarial settings.
– Attack is a major component of AML
– Bad actors do bad things
▪ Their main objective is not to get detected (change behavior to avoid detection)



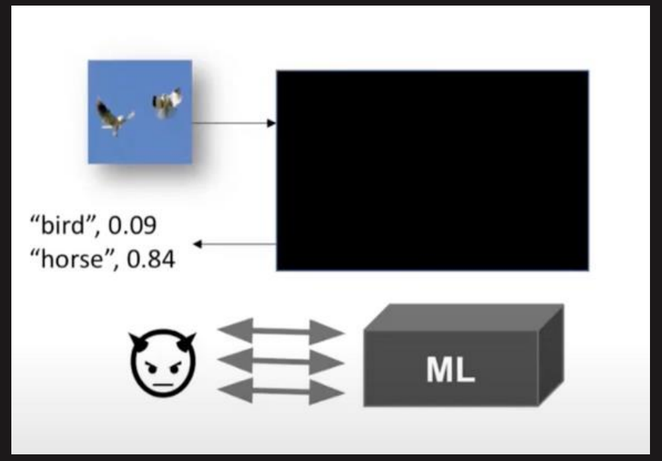
Small Perbutation : The difference between original image x and adversarial image x’ is minimal according to a Distance Metric D.

Misclassification : Despite appearing similar to the original image, the model misclassifies x’ (c(x) is not equal to c(x’)

TYPES OF ATTACKS (AS PER ATTACKER'S KNOWLEDGE)



"White Box Attacker"
Knows model architecture and all parameters.



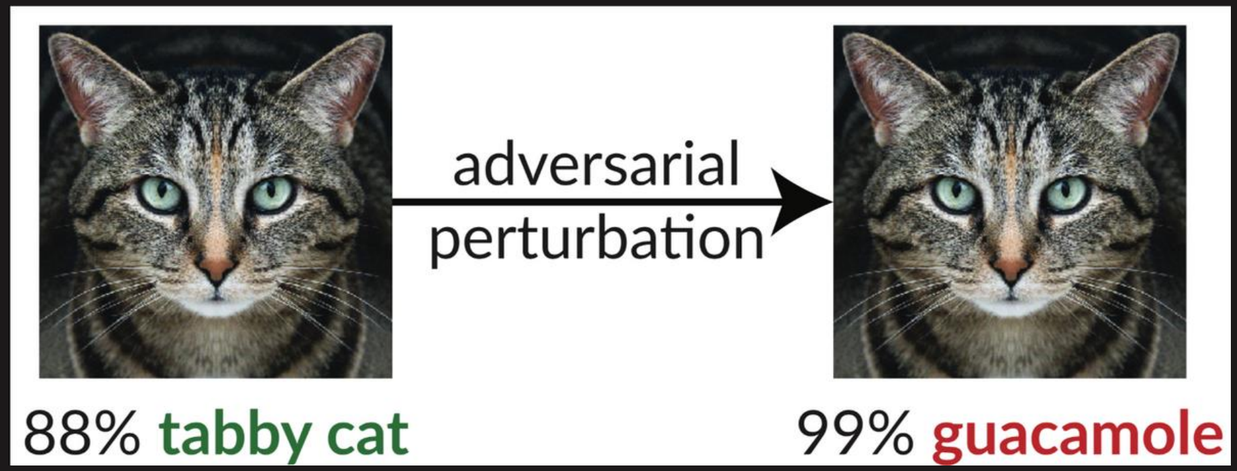
"Black Box Attacker"
Interacts with Model through API

PROBLEM STATEMENT

Construct adversarial examples with iterative optimization based methods.
Goal : Circumvent Common Defense Method

Results :

- Address flaws in past adversarial defense techniques that were totally reliant on gradient masking.
- Provide experimental proof of their successful 3 Attacks + 7 Defenses on past techniques.
- Enhancement on existing work.



How do attackers attack on the ML model?

They break the gradients or reconstruct the gradients of model to attack.

GRADIENT MASKING

GRADIENTS

In Backpropogation, gradients are slope of loss function with respect to model parameters.

GRADIENT MASKING

It is a failed defense method that work by trying to deny the attacker the access to a useful gradient.

Adversarial Attack -

Add a little noise to the input so that classifier classifies incorrectly.

OBFUSCATED GRADIENTS

It is a special case of gradient masking. It tries to break the gradient -

- Shattered : caused when a defense is non-differentiable, introduces numeric instability. (non-existing or incorrect)
- Stochastic : caused by random defenses (using random network or random input)
- Vanishing/Exploding Gradient : caused by defenses that cause multiple iteration (output of one to next)

-

3 ATTACKS

1. State Through Estimator
2. Backward Pass Differentiable Approximation (BPDA)
3. Expectation Over Transformation (EOT)

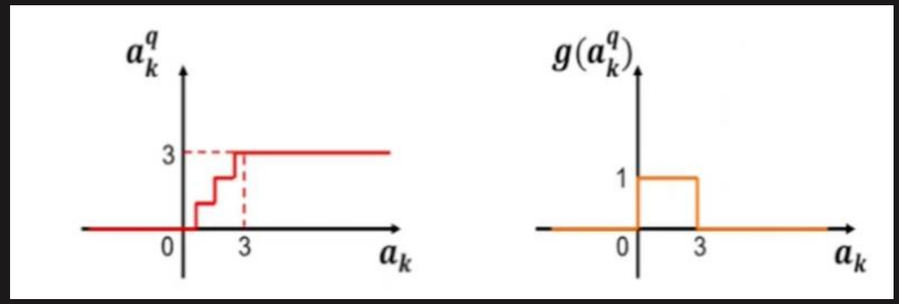
7 DEFENSES

1. Thermometer Encoding (Gradient Shattering)
2. Input Transformation (Gradient Shattering)
3. Stochastic Activation Parsing (Stochastic Gradient)
4. Mitigating through Randomization (Stochastic Gradient)
5. PixelDefend (Vanishing and Exploding Gradient)
6. Defense GAN (Vanishing and Exploding Gradient)
7. Local Intrinsic Dimensionality

ATTACKS

- to overcome obfuscated gradients

01 STATE THROUGH ESTIMATOR



Non-Differentiable Defenses: Defense transformations that are not easy to differentiate mathematically.

Given a pretrained classifier - $f(\cdot)$ construct a preprocessor $g(\cdot)$ and let secured classifier

$$f'(x) = f(g(x))$$

When the preprocessor $g(\cdot)$ is non-smooth, backpropagation for gradient-based attacks becomes challenging.

<u>The Straight-Through Estimator:</u>

Concept: Approximate non-smooth preprocessor $g(\cdot)$ with the identity function during gradient computation.

We perform forward propagation through neural networks as usual, but on backward pass we replace $g(\cdot)$ for gradient calculation.

02 BACKWARD PASS DIFFERENTIABLE APPROXIMATION

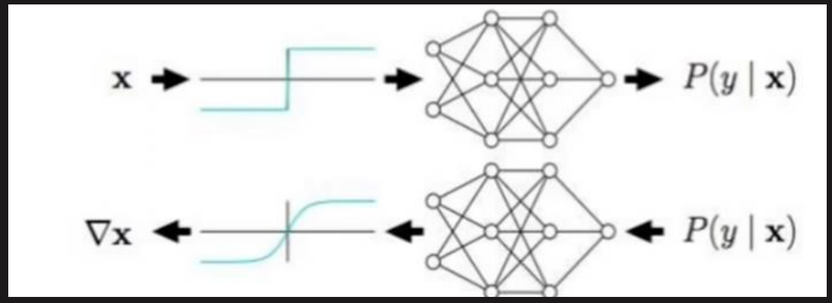
BPDA is a method to approximate gradients for non-differentiable layers $f_i(.)$ within a neural network $f(\cdot)$

Let $f(\cdot) = f_1 \ldots . f_j(.)$ denote the neural network, where $f_i(.)$ represent non-differentiable layers

Find a differentiable approximation $g(x)$ such that $g(x)$ closely resembles $f_i(x)$

BPDA facilitates the derivation of approximated gradients by substituting non-differentiable layers $f_i(x)$ with differentiable approximations $g(x)$

As long as the two functions are similar, we find that the slightly inaccurate gradients still prove useful in constructing an adversarial example.



03 EXPECTATION OVER TRANSFORMATION

EOT: Technique for attacking defenses with randomized input transformations.
Objective: Optimize expectation over random transformations
Problem: Defenders apply random transformations before classification.

Gradient Computation: Chain rule applied to $f(\cdot)$ and $t(\cdot)$

Approximation: Estimate expectation via sampling from T.

Iterative Process: Gradient descent updates input based on sampled transformations.

EOT finds examples that are adversarial over a distribution of transformations T, allowing for attacking defenses that apply randomized input transformations for robustness.

EOT solves optimization problem using gradient descent, by approximating with samples at each gradient step.

$$\arg\max_{\mathbf{x}'} \quad \mathop{\mathbb{E}}_{t\sim T}\left[-\log P\left(y \mid t(\mathbf{x}')\right)\right]$$
$$\text{subject to } d(\mathbf{x}, \mathbf{x}') < \epsilon$$

$$_T - \log P\left(y \mid t(\mathbf{x})\right) = \mathbb{E}_{t\sim T} \nabla - \log P\left(y \mid t(\mathbf{x})\right)$$

DEFENSE

1. Thermometer Encoding

The purpose of thermometer encoding is to break linearity.

$x \to$ image

$x_{i,j,c} \to$ pixel color

$$\hat{\tau}(x_{i,j,c})_k = \min(\max(x_{i,j,c} - k/l, 0), 1)$$

$\tau(x_{i,j,c}) \in \mathbb{R}^l \to l$-level thermometer encoding

$\tau(x_{i,j,c})_k = 1$ if $x_{i,j,c} > \frac{k}{l}$

$= 0$ otherwise

$$\tau(x_{i,j,c})_k = \text{floor}(\hat{\tau}(x_{i,j,c})_k)$$

let $g(x) = \hat{\tau}(x)$

$\tau(0.66) = 1111110000 \to 10$-level thermometer encoding

2. Input Transformations

(a) randomly drop pixels and restore them
by performing total variance minimization; and
(b) image quilting: reconstruct images by replacing
small patches with patches from "clean" images.

3. Local Intrinsic Dimensionality (LID)

$\overrightarrow{\text{LID}}(x) = \{\text{LID}_{d_j}(x)\}_{j=1}^n$ $\quad f^{1..j} \to$ composition of layers 1 through j

$d_j(x, y) = \left\| f^{1..j}(x) - f^{1..j}(y) \right\|_2$

$$\text{LID}_d(x) = -\left(\frac{1}{k} \sum_{i=1}^{k} \log \frac{r_i(x)}{r_k(x)} \right)^{-1}$$

$r_i(x) \to$ distance between sample x and its i-th nearest neighbor

Train a logistic regression classifier to detect adversarial examples!

4. Stochastic Activation Pruning (SAP)

SAP randomly drops some neurons of each layer with probability proportional to their absolute value. Values which are retained are scaled up (as is done in dropout) to retain accuracy.

5. Mitigating through Randomization

For a classifier that takes a 299×299 input, the defense first randomly rescales the image to a $r \times r$ image, with $r \in [299, 331)$, and then randomly zero-pads the image so that the result is 331×331.

6. PixelDefend

PixelDefend proposes using a PixelCNN generative model to project a potential adversarial example back onto the data manifold before feeding it into a classifier.

7. Defense-GAN

Defense-GAN uses a Generative Adversarial Network to project samples onto the manifold of the generator before classifying them.

Obfuscated Gradients

– shattered gradients: non-existent or incorrect
– stochastic gradients: randomized network or input
– vanishing/exploding gradients

Backward Pass Differentiable Approximation (BPDA)

$f^i \to$ non-differentiable (or not usefully-differentiable) layer

Find a differentiable approximation g to f^i, and perform backward pass by replacing f^i by g.

Expectation over Transformation (EoT)

$$\nabla \mathbb{E}_{t \sim T} f(t(x)) = \mathbb{E}_{t \sim T} \nabla f(t(x))$$

Reparameterization

vanishing/exploding gradients

DEMO

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\Test_randomization.py
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\Test_randomization.py:7: The name tf.Session is deprecated. Please use tf.compat
.v1.Session instead.

WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\Test_randomization.py:16: The name tf.placeholder is deprecated. Please use tf.c
ompat.v1.placeholder instead.

WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\inceptionv3.py:30: The name tf.image.resize_bilinear is deprecated. Please use t
f.compat.v1.image.resize_bilinear instead.

WARNING:tensorflow:From C:\Users\ashut\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow_core\contrib\layers\python\layers\layers.py:1057: Layer.apply (from tensorflow.py
thon.keras.engine.base_layer) is deprecated and will be removed in a future version.
Instructions for updating:
Please use `layer.__call__` method instead.
WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\utils.py:14: The name tf.train.NewCheckpointReader is deprecated. Please use tf.
compat.v1.train.NewCheckpointReader instead.

WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\utils.py:16: The name tf.global_variables is deprecated. Please use tf.compat.v1
.global_variables instead.

WARNING:tensorflow:From E:\Crypto_Try\obfuscated-gradients-master\obfuscated-gradients-master\randomization\utils.py:19: The name tf.variable_scope is deprecated. Please use tf.compat.v1.v
ariable_scope instead.
```

Ln: 5 Col: 0

ENHANCEMENT

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
 12%|#2        | 6/50 [00:05<00:38,  1.15it/s] step 6, pred=975
 14%|#4        | 7/50 [00:06<00:36,  1.17it/s] step 7, pred=975
 16%|#6        | 8/50 [00:07<00:35,  1.20it/s] step 8, pred=975
 18%|#8        | 9/50 [00:08<00:35,  1.17it/s] step 9, pred=287
 20%|##        | 10/50 [00:09<00:33,  1.19it/s] step 10, pred=975
 22%|##2       | 11/50 [00:09<00:32,  1.20it/s] step 11, pred=287
 24%|##4       | 12/50 [00:10<00:31,  1.21it/s] step 12, pred=272
 26%|##6       | 13/50 [00:11<00:30,  1.22it/s] step 13, pred=277
 28%|##8       | 14/50 [00:12<00:29,  1.21it/s] step 14, pred=537
 30%|###       | 15/50 [00:13<00:29,  1.19it/s] step 15, pred=975
 32%|###2      | 16/50 [00:14<00:28,  1.20it/s] step 16, pred=287
 34%|###4      | 17/50 [00:14<00:27,  1.20it/s] step 17, pred=975
 36%|###6      | 18/50 [00:15<00:26,  1.21it/s] step 18, pred=272
 38%|###8      | 19/50 [00:16<00:25,  1.22it/s] step 19, pred=287
 40%|####      | 20/50 [00:17<00:24,  1.22it/s] step 20, pred=287
 42%|####2     | 21/50 [00:18<00:23,  1.22it/s] step 21, pred=293
 44%|####4     | 22/50 [00:19<00:22,  1.23it/s] step 22, pred=287
 46%|####6     | 23/50 [00:19<00:22,  1.22it/s] step 23, pred=975
 48%|####8     | 24/50 [00:20<00:21,  1.22it/s] step 24, pred=975
 50%|#####     | 25/50 [00:21<00:20,  1.23it/s] step 25, pred=975
 52%|#####2    | 26/50 [00:22<00:19,  1.23it/s] step 26, pred=287
 54%|#####4    | 27/50 [00:23<00:19,  1.19it/s] step 27, pred=287
 56%|#####6    | 28/50 [00:23<00:18,  1.20it/s] step 28, pred=287
 58%|#####8    | 29/50 [00:24<00:18,  1.16it/s] step 29, pred=272
 60%|######    | 30/50 [00:25<00:18,  1.09it/s] step 30, pred=280
 62%|######2   | 31/50 [00:26<00:17,  1.06it/s] step 31, pred=287
 64%|######4   | 32/50 [00:27<00:16,  1.10it/s] step 32, pred=975
 66%|######6   | 33/50 [00:28<00:15,  1.11it/s] step 33, pred=287
 68%|######8   | 34/50 [00:29<00:14,  1.11it/s] step 34, pred=287
 70%|#######   | 35/50 [00:30<00:13,  1.11it/s] step 35, pred=277
 72%|#######2  | 36/50 [00:31<00:12,  1.14it/s] step 36, pred=287
 74%|#######4  | 37/50 [00:32<00:11,  1.14it/s] step 37, pred=277
 76%|#######6  | 38/50 [00:33<00:10,  1.12it/s] step 38, pred=277
 78%|#######8  | 39/50 [00:33<00:09,  1.12it/s] step 39, pred=287
 80%|########  | 40/50 [00:34<00:08,  1.13it/s] step 40, pred=272
 82%|########2 | 41/50 [00:35<00:08,  1.12it/s] step 41, pred=287
 84%|########4 | 42/50 [00:36<00:07,  1.12it/s] step 42, pred=272
 86%|########6 | 43/50 [00:37<00:06,  1.11it/s] step 43, pred=975
 88%|########8 | 44/50 [00:38<00:05,  1.12it/s] step 44, pred=272
 90%|######### | 45/50 [00:39<00:04,  1.11it/s] step 45, pred=280
 92%|#########2| 46/50 [00:40<00:03,  1.12it/s] step 46, pred=280
 94%|#########3| 47/50 [00:41<00:02,  1.12it/s] step 47, pred=287
 96%|#########6| 48/50 [00:42<00:01,  1.11it/s] step 48, pred=287
 98%|#########8| 49/50 [00:42<00:00,  1.15it/s] step 49, pred=280
100%|##########| 50/50 [00:43<00:00,  1.14it/s]100%|##########| 50/50 [00:43<00:00,  1.14it/s]
total time: 43.83798837661743
>>>
```

Ln: 933 Col: 4

THANK YOU :)