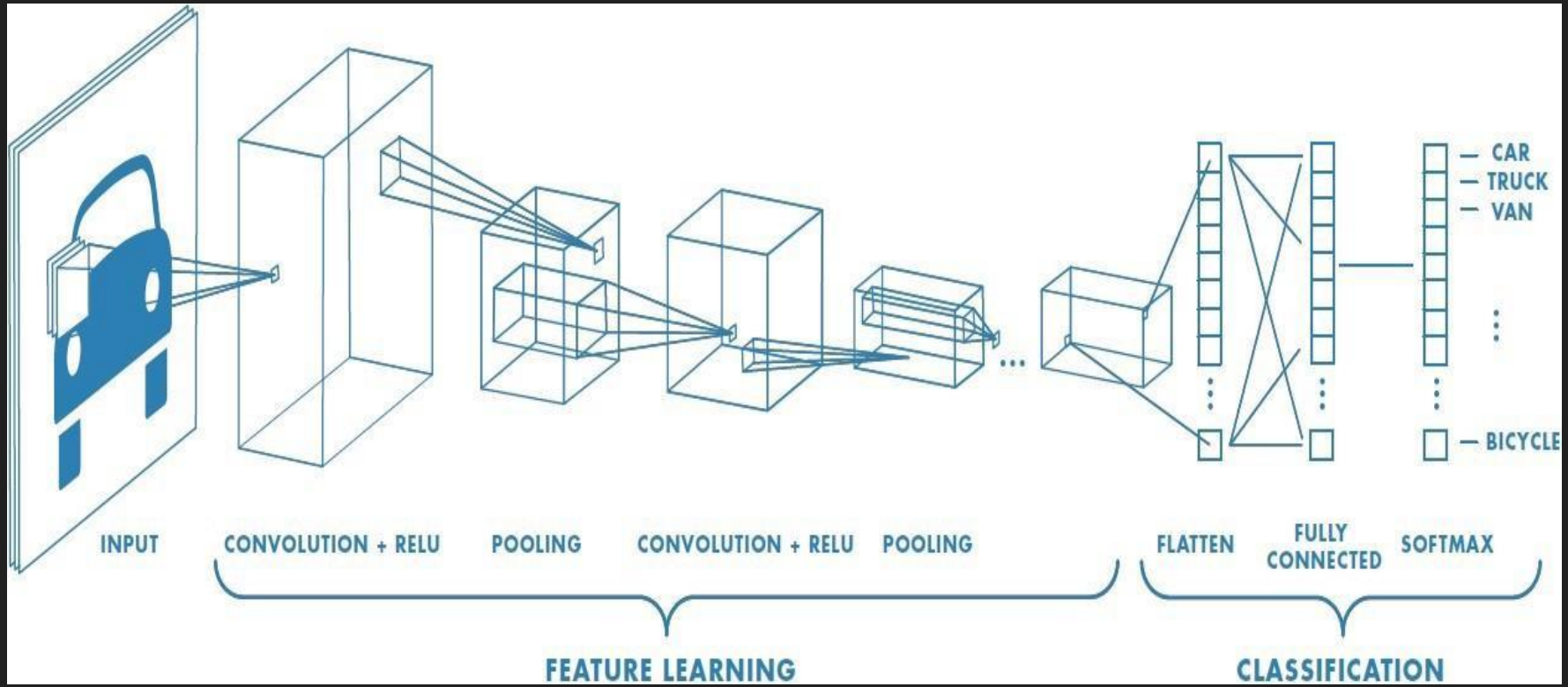
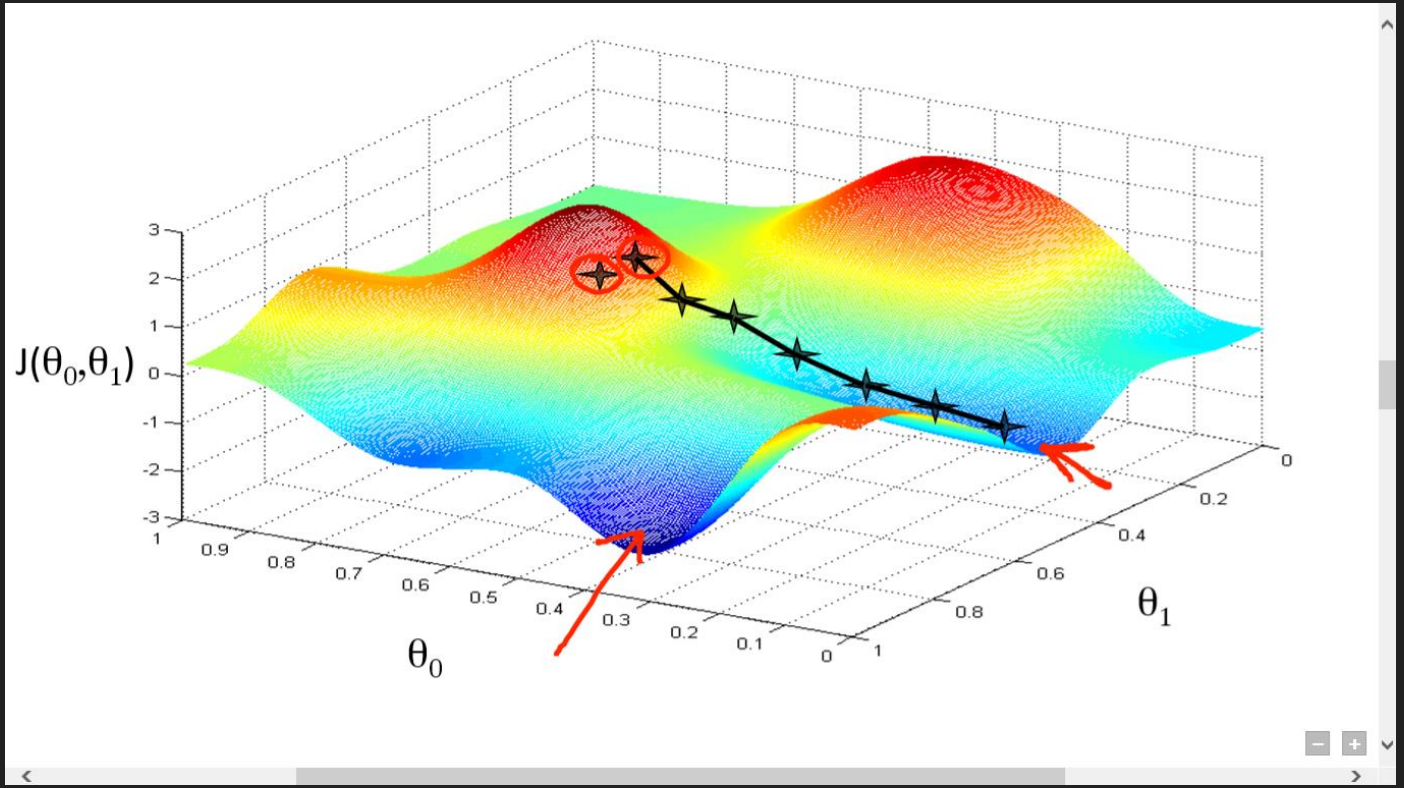
# Defense Against Adversarial Attacks

By,
N Kausik (COE17B010)
Bharathvaj (COE17B047)
Venkat (CED17I046)

# Brief Intro to ConvNet . . .



# Gradient Descent



# Gradient Descent

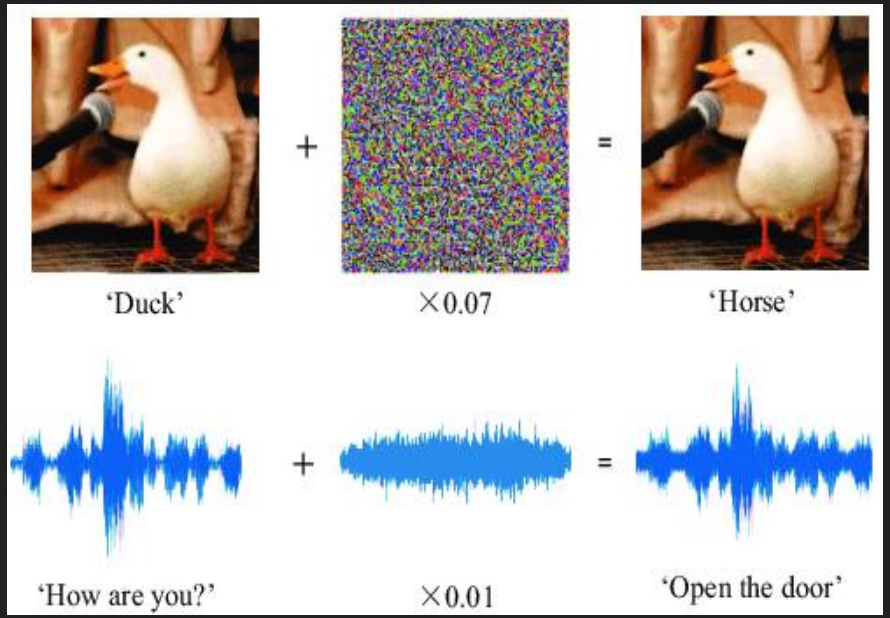
Used for finding local minima of a function iteratively

For a function ‘F’,

- Start a random value a(n)
- we find gradient at the current value a(n)
- Multiply gradient by a coefficient
- Subtract this value from the current value to get the next value
- Repeat above steps for the new value

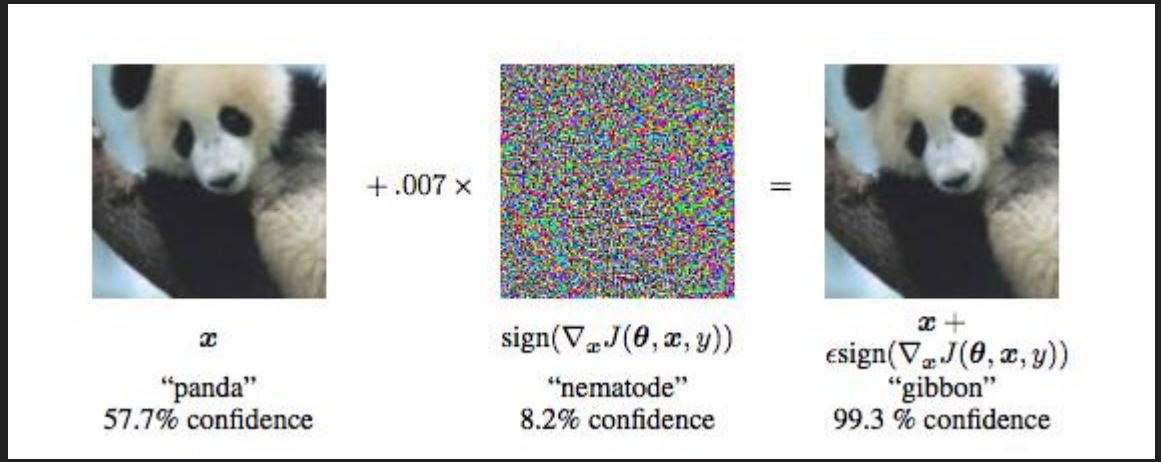
$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

# Attacking Machine Learning with Adversarial Examples




- **Adversarial Examples** are inputs to ML models that an attacker has intentionally designed to cause the model to make a mistake. They are like optical illusions for machines
- Similar to True Data but can fool a classifier
- Difficult to defend against them

# Classical Example: Panda image




Starting with an image of a Panda, the attacker adds a small perturbation that has been carefully calculated (It is not a random noise) to make the image be recognized as a Gibbon with much higher confidence than that of the original Panda itself.

- Robust
- Adversarial examples can even be printed out on standard paper, then photographed with a standard smartphone, and can still fool systems

# Some real time Threats of Adversarial Attacks . . .

1. Faulty autonomous vehicle driving - Using stickers or paints to create an adversarial stop sign, not detecting pedestrian etc.

1. Face recognition using Neural Networks may be fooled leading to wrong classifications

1. Reinforcement learning agents can also be manipulated by adversarial examples

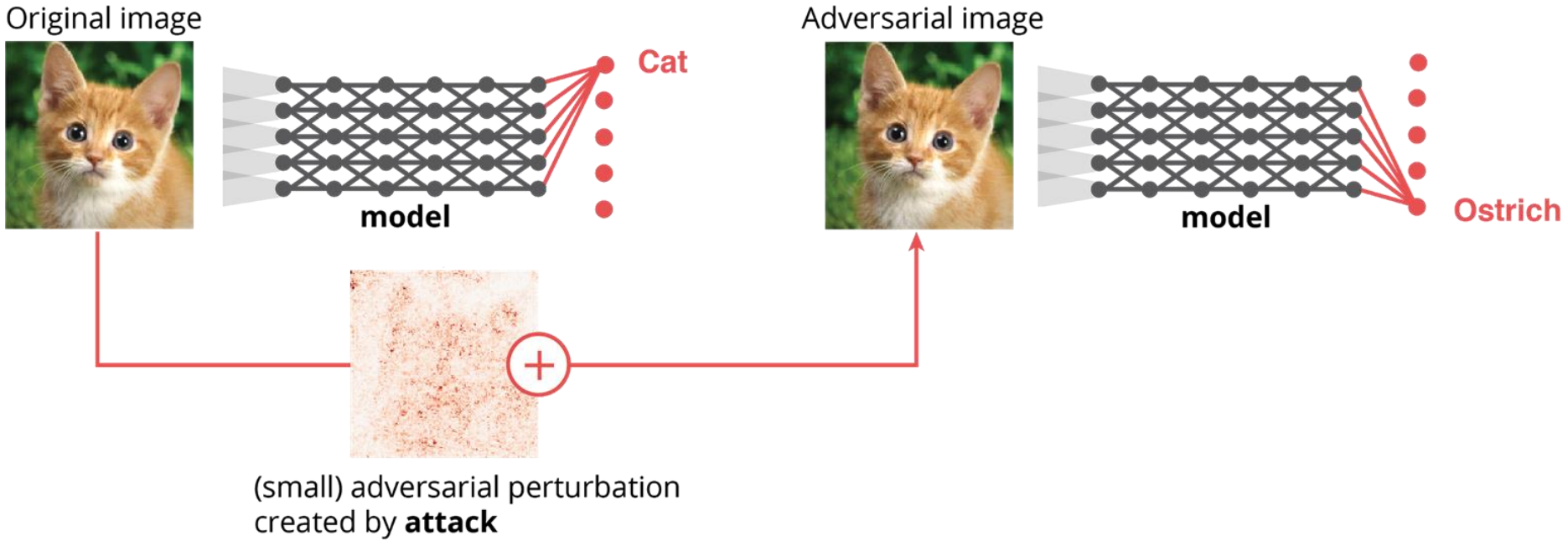
1. False Predictions in medical field like false positives in cancer detection, etc

Why does this matter?

- Computer vision is used everywhere!
- Facial Recognition
- Self Driving Cars (change speed limit sign) https://arxiv.org/pdf/1511.07528.pdf
- Biometric Recognition
- Text Applications too! (Think CNNs for Sentiment Analysis or Text classification)
- Ad-blockers can incorrectly classify ads https://arxiv.org/pdf/1811.03194.pdf
- Spam Classifiers can incorrectly identify malicious mails https://www.covert.io/research-papers/deep-learning-security/Large-scale%20Malware%20Classification%20using%20Random%20Projections%20and%20Neural%20Networks.pdf

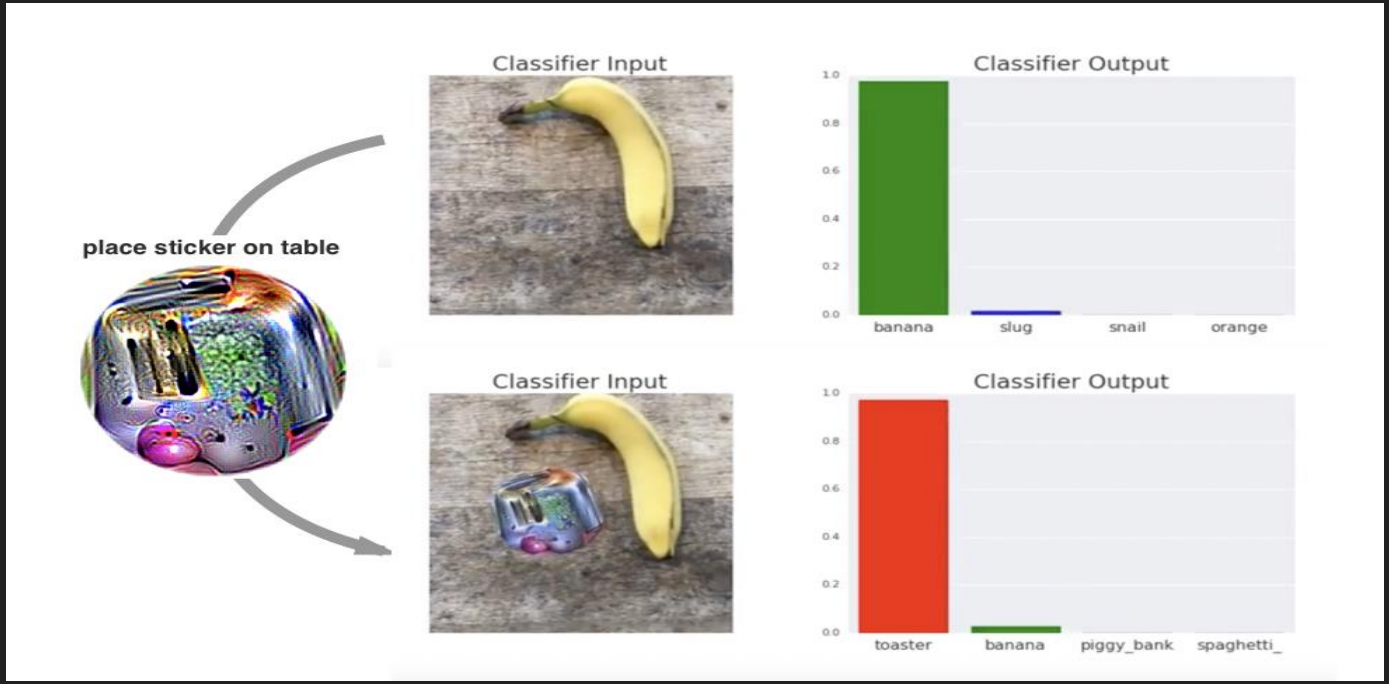
# Attack and Defence Methods

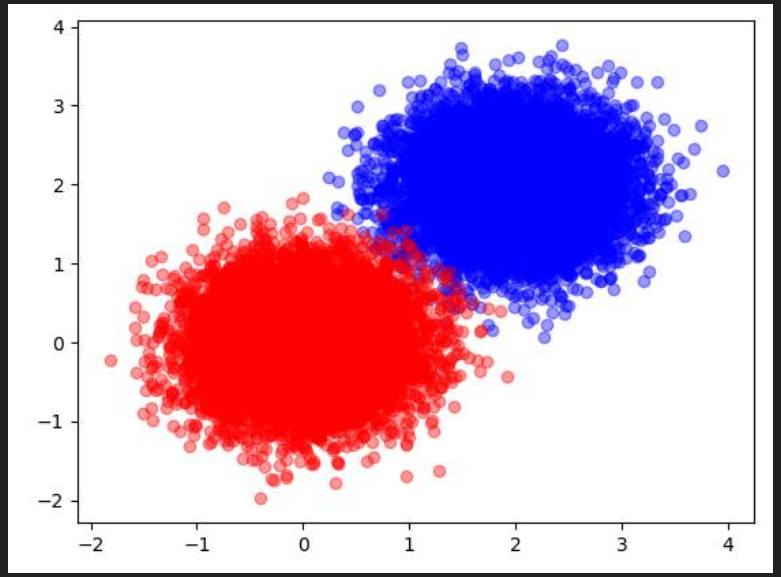
# Live demo: AdVis.js

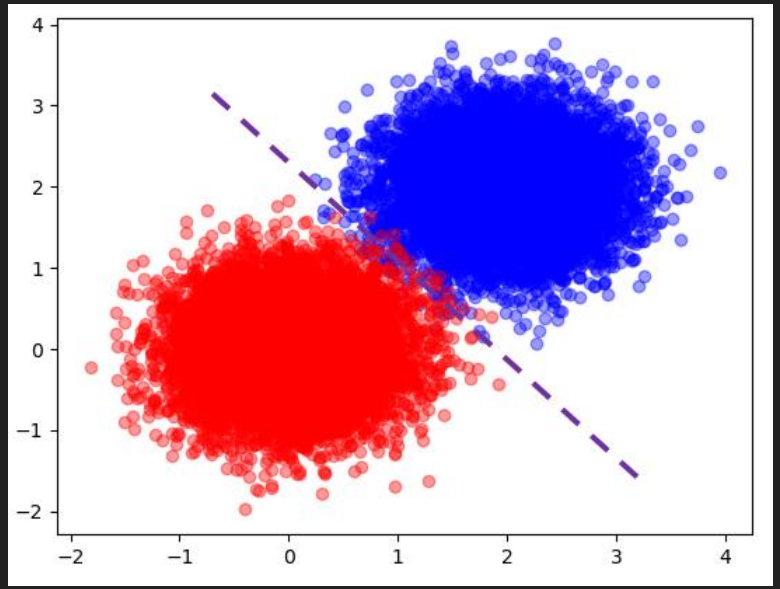


# Other Examples

https://adversarial-attacks.net/

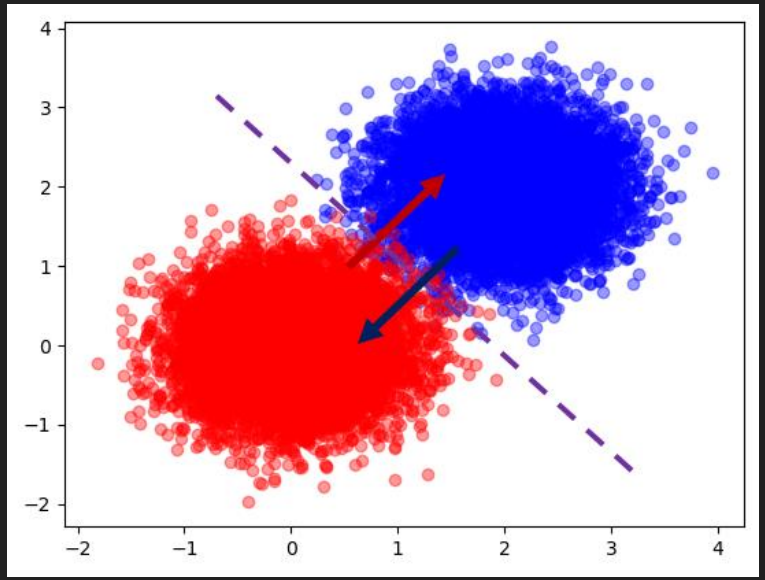





- Let's say we want to build a classifier for these 2 classes (red and blue)
- We can see that the decision boundary should be right in the middle of the two clusters.
- With that in place, any new data falling in the top right region would then be classified as blue and those falling in the left bottom region would then be classified as red.

- Adversarial Examples are the well designed samples with an intention to cause machine learning to make mistake.
- How can we cause the purple line to make error if we cannot move the purple line?
- It turns out that it is not difficult at all, we just need to move the sample to the opposite region so that the blue point falls into the red region, and the red point falls in the blue region.
- Not a surprise at all



# Fast Gradient Sign Method (FGSM)

- Uses Gradients of Loss wrt input image which when combined with original image maximises the loss (can be viewed as gradient ascent) thus causing the wrong prediction
- Where
  - adv_x = Adversarial image
  - x = Original input image
  - y = Original input label
  - $\epsilon$ = Multiplier to make perturbations small
  - $$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

# Method to generate Adversarial Example

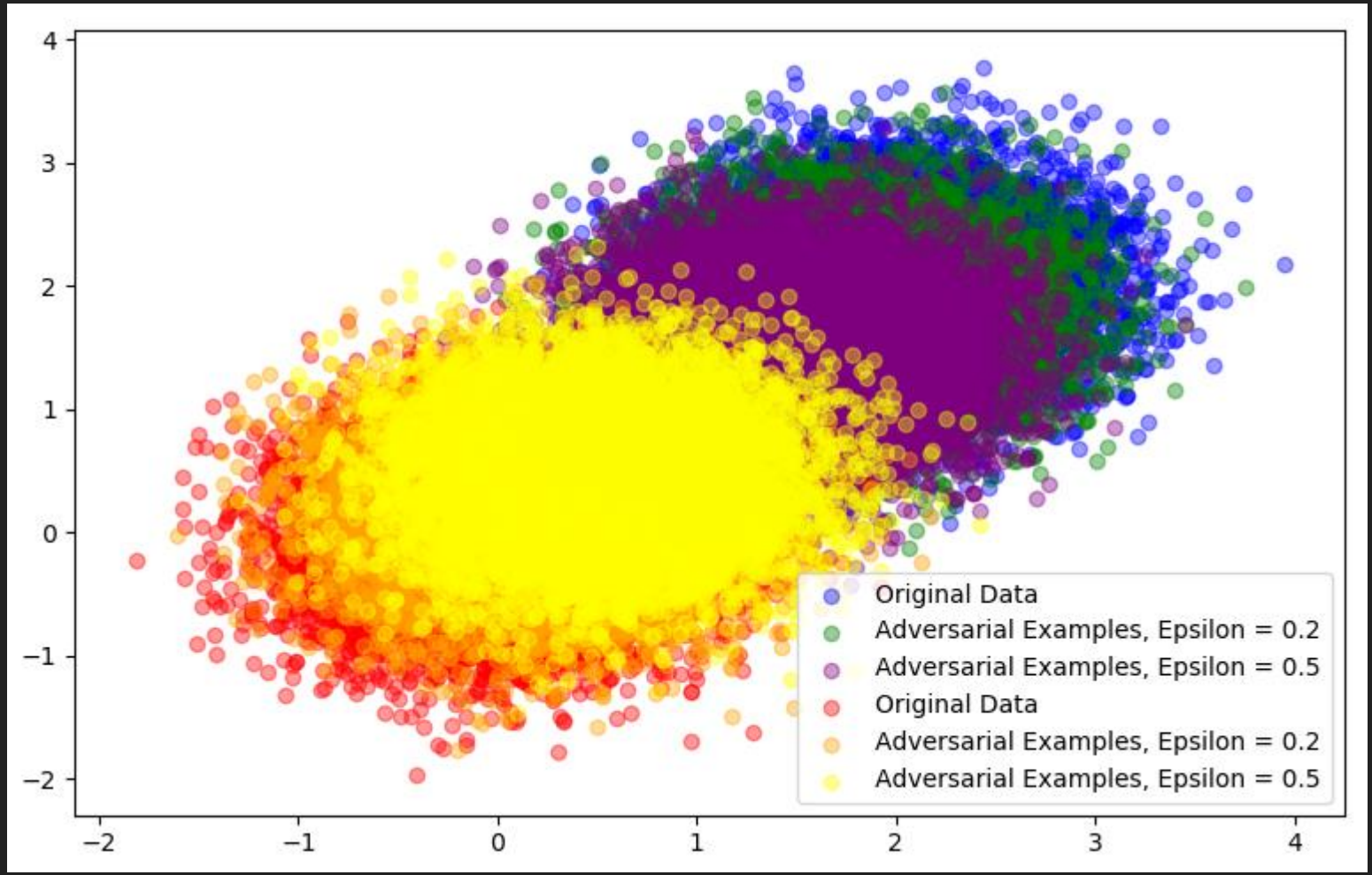
- Take an input image
- Using the trained model, generate a prediction for the input image
- Using the predicted value and true value, find the loss value
- Find the gradient of the loss value wrt the input image
- Find the signs of the values in the gradient
- Multiply the gradient by a coefficient($\epsilon$)
- Add with the original image
- The final output is the adversarial example

- Epsilon is a small number controlling the size of adversarial attack, which needs to be chosen in order to be effective but not to be too obvious.

- With the increase of epsilon, we can foresee the decision boundary cannot work out. The logistic regression model error grows from 2.9% to 7.6% when epsilon becomes 0.2, and to 22.6% when epsilon becomes 0.5.

- A trained CNN model acts as a linear separator for high dimensional data for different classes, where every point(image) is associated with its class.

- Of course, the boundary of separation is not perfect. This provides an opportunity to push one image from one class to another (cross the boundary) i.e. perturbing the input data in the direction of another class.


Original Data
Adversarial Examples, Epsilon = 0.2
Adversarial Examples, Epsilon = 0.5
Original Data
Adversarial Examples, Epsilon = 0.2
Adversarial Examples, Epsilon = 0.5
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4

# Variants of FGSM - Targeted FGSM

$$x^{adv} = x - \varepsilon \cdot \mathtt{sign}(\nabla_x J(x, y_{target})),$$

where

$y_{target}$ is the target label for the adversarial attack.

Here Label Leaking is prevented as we require only target label and dont need the TRUE label of the input data

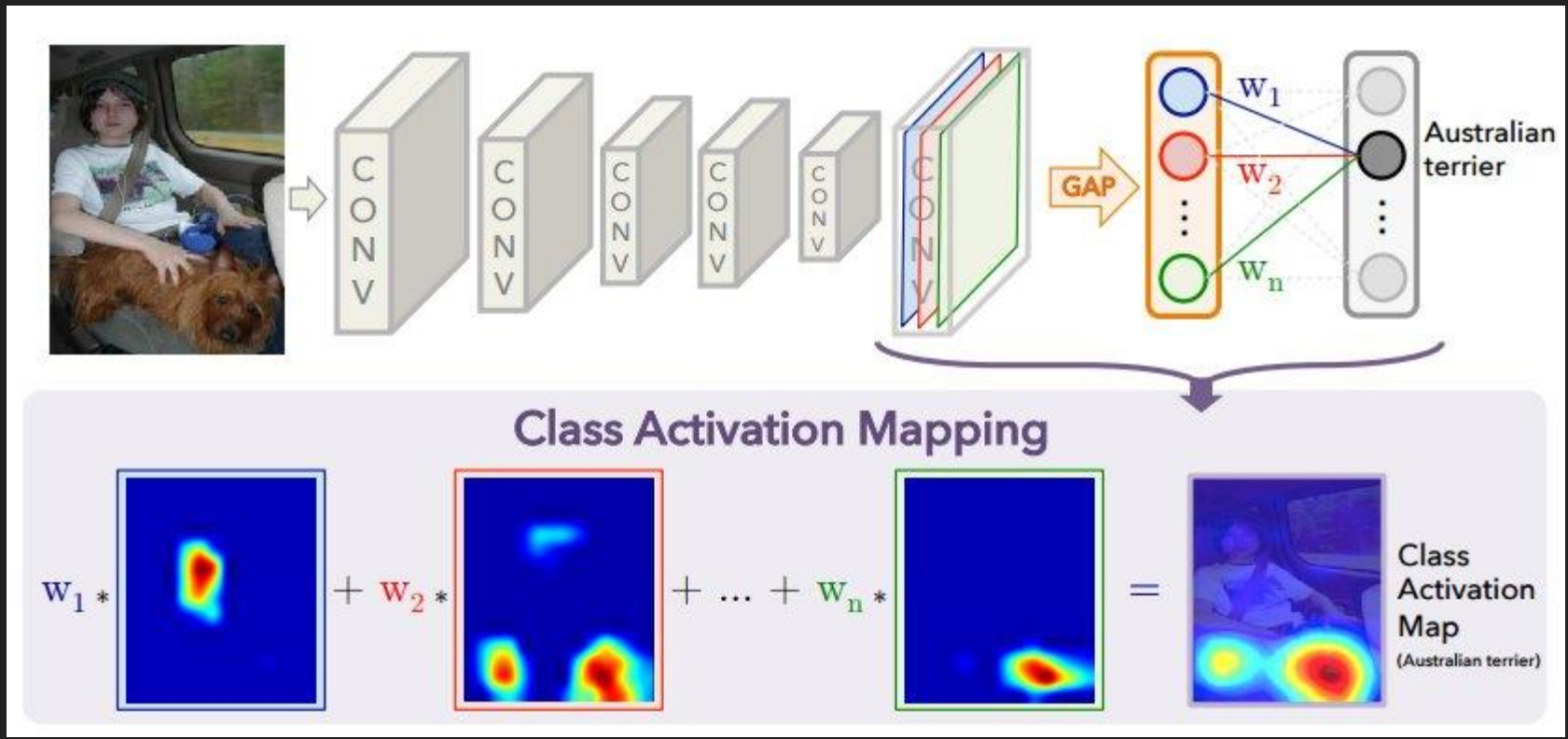
# Variants of FGSM - Iterative FGSM

- Instead of finding gradients in a single step, we do in T steps
- $\alpha = \varepsilon / T$

$$x_0^{adv} = x, \quad x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot \mathtt{sign}\left(\nabla_x J(x_t^{adv}, y)\right).$$

Perturbed_image = image + epsilon∗sign(data gradient)

- This technique adds noise (not random noise) whose direction is the same as the gradient of the cost function with respect to the data.
- The noise is scaled by epsilon, which is usually constrained to be a small number via max norm.
- The magnitude of gradient does not matter in this formula, but the direction (+/-).
- FGSM can be implemented in Python in a few lines.
- In gradient descent, the gradient of cost with respect to weight is (Y_Prediction — Y_True)X.
- In adversarial examples, the gradient of cost with respect to data is (Y_Prediction — Y_True)W.
- little change means we can recycle the code of gradient calculation by just replacing data with weight.


CONV
CONV
CONV
CONV
CONV
CONV
GAP
$w_1$
$w_2$
$w_n$
Australian terrier
Class Activation Mapping
$w_1 *$
+ $w_2 *$
+ ... + $w_n *$
=
Class Activation Map
(Australian terrier)

example: https://colab.research.google.com/github/dennis-sell/pytorch-fun/blob/master/Adversarial_Example_Tutorial.ipynb

## CleverHans

Also if you want a defense library - look no further than CleverHans https://github.com/tensorflow/cleverhans/blob/master/cleverhans_tutorials/mnist_tutorial_tf.py https://cleverhans.readthedocs.io/en/latest/source/attacks.html

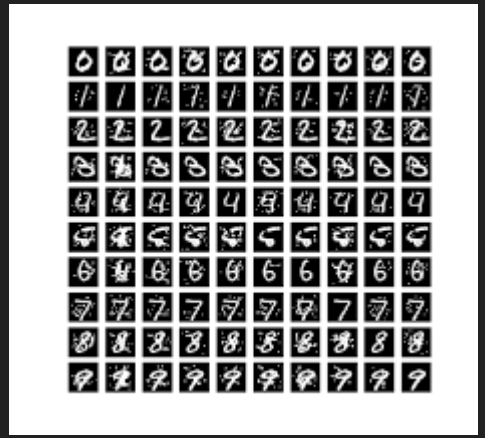
# Why is it hard to defend against adversarial examples?

- Difficult to construct a theoretical model of the adversarial example crafting process
- Requires ML models to produce good outputs for every possible input
- The defense like adversarial training and gradient masking are not adaptive
- It may block one kind of attack, but it leaves another vulnerability open to an attacker who knows about the defense being used

# Attempted defenses against adversarial examples

Traditional techniques for making ML models more robust (such as weight decay, dropout etc.) doesn't provide a practical defense against adversarial examples

1. Adversarial Training - Brute force solution
    - Simply **generate** a lot of adversarial examples and explicitly train the model not to be fooled by each of them



- One way for Adversarial training is to proactively generate adversarial examples as part of the training procedure.
- We have already seen how we can leverage FGSM to generate adversarial examples inexpensively in large batches.
- The model is then trained to assign the same label to the adversarial example as to the original example (i.e generate a cat, perturb that image to be misclassified, label it a cat regardless)
- Adversarial training is a standard brute force approach where the defender simply generates a lot of adversarial examples and augments these perturbed data while training the targeted model.
- Adversarial training of a model is useful only on adversarial examples which are crafted on the original model.

# Sad story of a failed defense . . . (Gradient Masking)

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

- To explain it, the attackers look at a picture of an airplane, they test which direction in picture space makes the probability of the ‘cat’ class increase, and then they give a little push (i.e., they perturb the input) in that direction. The new, modified image is  mis-recognized as a cat

- But what if there were no gradient???

- This seems to provide some defense because the attacker doesn’t know which way to “push” the image

- Most image classification models can be run in 2 modes:
    1. Output the most likely class
    2. Output probabilities of each class

- Gradients tell us which changes will increase the probability of other wrong class

- So if we HIDE the output probabilities (probability mode) and use just the most likely class as an output, the attacker no longer knows where to go to find inputs that will be classified as wrong class.

- This is some sort of defense (just less clues). But if the attacker can guess which points are adversarial examples, those points will still be misclassified

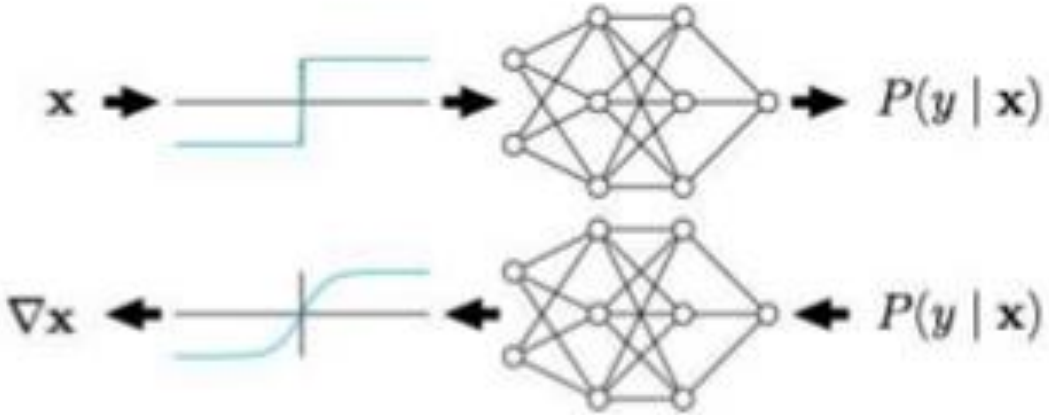
- The attacker can train their own model, make adversarial examples for their model and they can still fool our model

- Defended model (non-smooth) vs Substituted model (smooth)

# Backward Pass Differentiable Approximation

**Backward Pass Differentiable Approximation (BPDA)**

BPDA allows for attacking non-differentiable networks by approximating the gradient of the non-differentiable layers. The gradient is estimated by computing the forward pass normally but replacing a non-differentiable layer $f(\cdot)$ with a differentiable approximation $h(\cdot) \approx f(\cdot)$ on the backward pass.



- If a gradient from f(x) is unattainable (like if a layer is non differentiable), you can define a new network g(x), and train a neural network to approximate f(x).

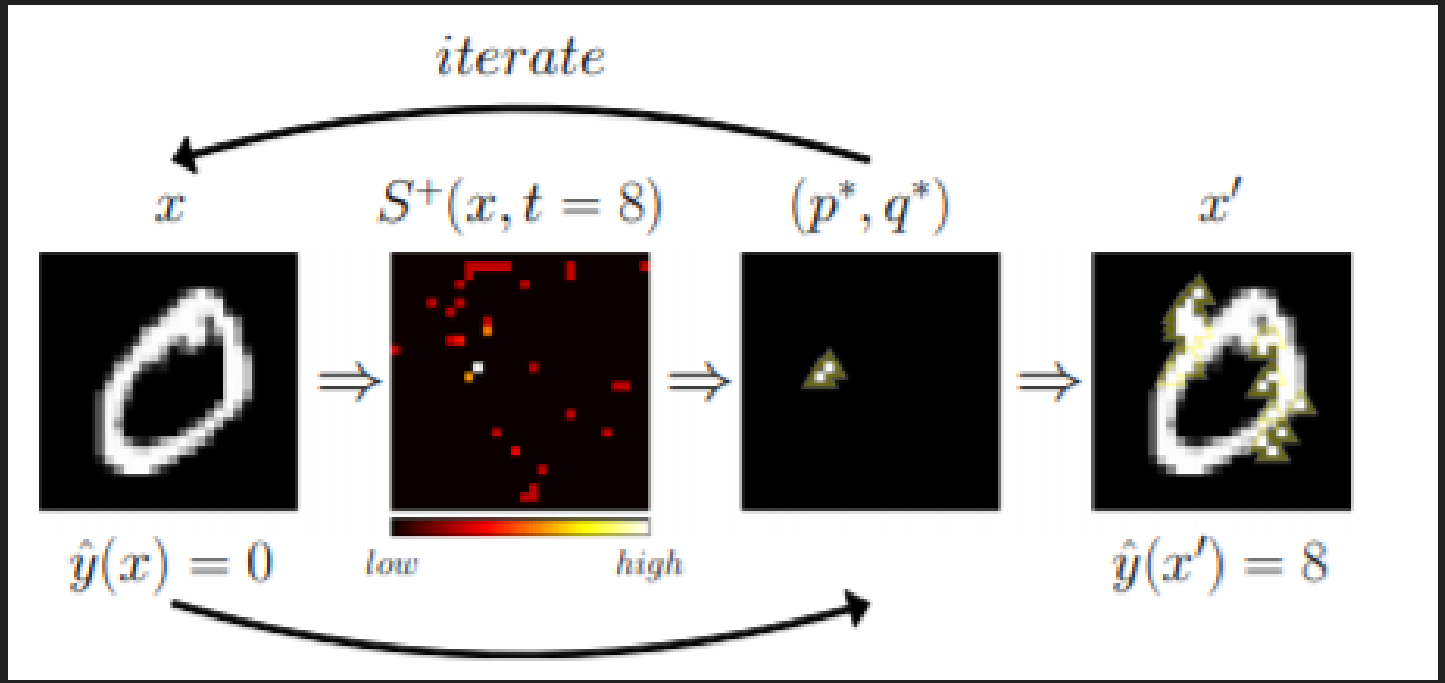
- If g(x) approximates f(x), we have no problem getting it's gradient and using it to replace the one that would come from f(x) when running their optimizer.

# Jacobian-based Saliency Map Attack (JSMA)

- Perturbs one feature (xi) by a constant offset in each iteration step that maximizes the Saliency Map

$$S^{+}(x_{(i)}, c) = \begin{cases} 0 & \text{if } \frac{\partial f(x)_{(c)}}{\partial x_{(i)}} < 0 \text{ or } \sum_{c' \neq c} \frac{\partial f(x)_{(c')}}{\partial x_{(i)}} > 0 \\ -\frac{\partial f(x)_{(c)}}{\partial x_{(i)}} \cdot \sum_{c' \neq c} \frac{\partial f(x)_{(c')}}{\partial x_{(i)}} & \text{otherwise} \end{cases}$$

S +(·) measures how much x(i) positively correlates with c, while also negatively correlates with all other classes c’




Clearly lesser perturbs - Only some pixels are added to noise

# Comparing JSMA vs FGSM

- FGSM computations are easier and can be done faster than JSMA
- JSMA perturbs fewer pixels than FGSM - Image after JSMA is more closer to original image than after FGSM

# Method of Defense (BaRT):
# The Barrage of Random Transforms

- We apply a barrage of transforms on the input image with random parameters and in random order.

- So our model gets trained in a large set of variations on the input image thereby becomes robust to adversarial attacks

# Transforms used

- Color Precision Reduction:

  Reducing bit-resolution of color
  (Eg. from 16bit to 8bit)

- JPEG Noise

  Artifacts due to jpeg compression

# Transforms used

- Swirl

  Rotating pixels around a randomly selected point

- Noise Injection

  Randomly selecting from a variety of noise like Gaussian, Salt & Pepper, etc

# Transforms used

- Other transforms like

  Zooming

  Color Space Changing

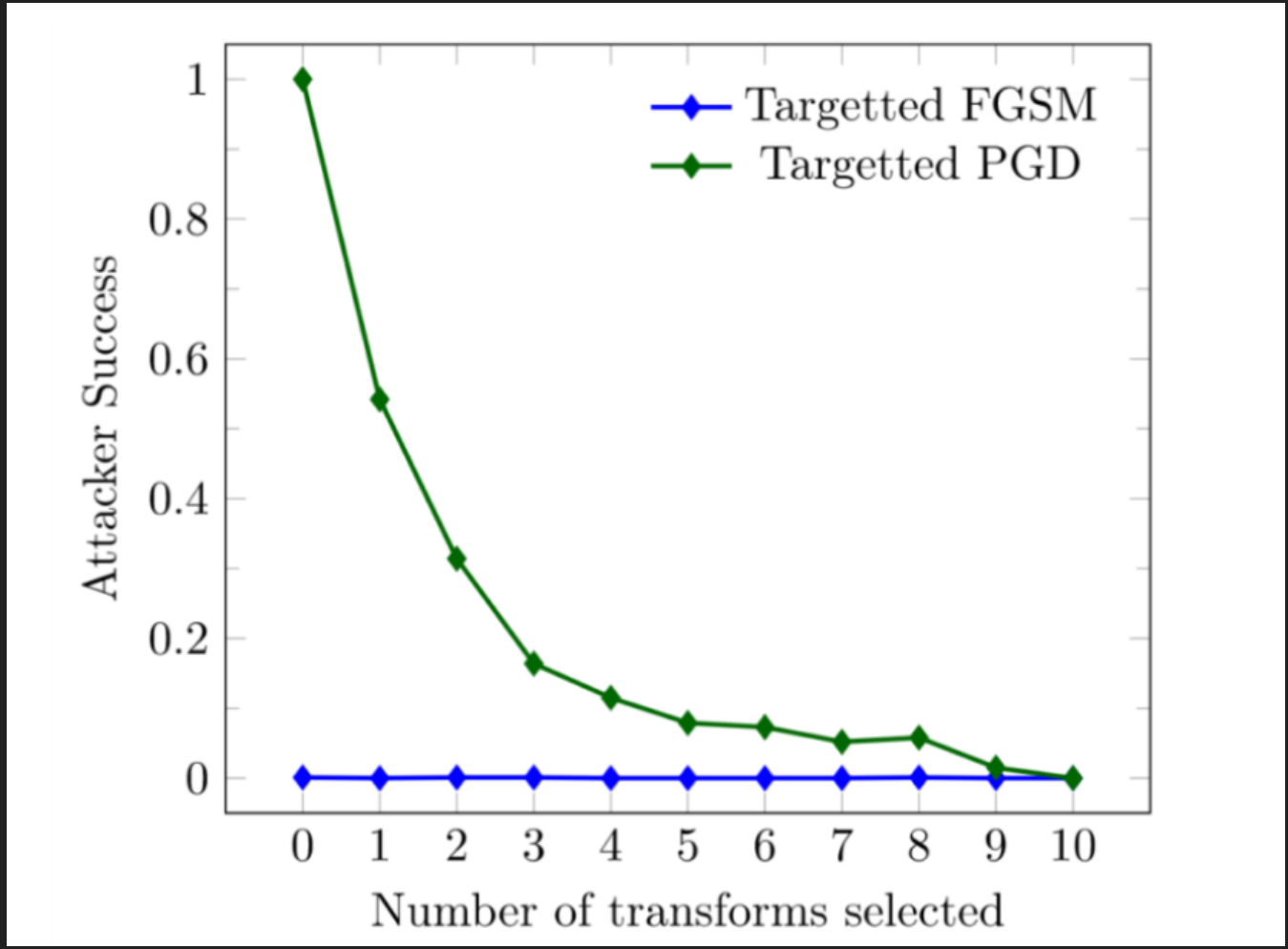
  Contrast Changing

  Greyscale

  Denoising Changing

# Method to do BaRT

- Take input image
- Apply any number of these transforms with random parameters in a random order
- Repeat above step to get a variety of variants of the same image
- Train the model on all of these variants
- The model thus becomes robust against these transforms and hence cant be fooled that easily


Targetted FGSM
Targetted PGD
Attacker Success
1
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9 10
Number of transforms selected

- So, after 10 transforms, Targeted attacks like projected gradient descent and FGSM can't fool the network

# Conclusion

- AI and Neural Networks are becoming more and more important in our daily lives and for solving important problems like medical detections, self driving cars, etc
- So, it is important that the predictions made are as close to perfect as possible in critical problems (Medical Detections)
- So, while building neural networks we should also focus on the possible attacks and we should employ proper defence mechanisms to protect our model

Thank You