

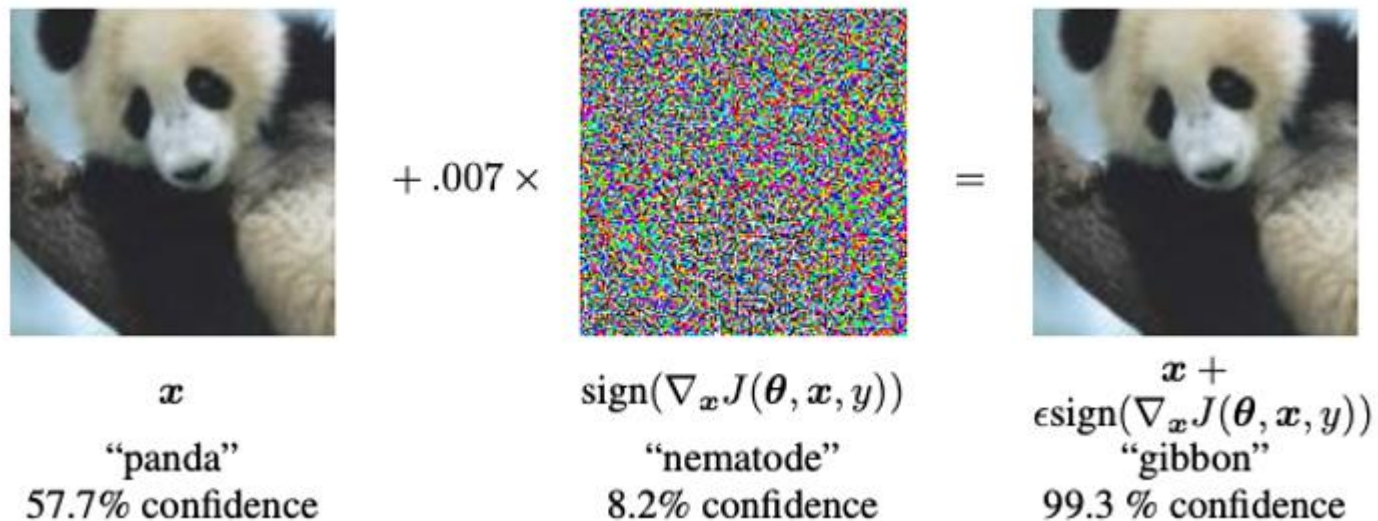
Adversarial example using FGSM

Outline

- Introduction
- Implementing
- Result

Introduction

- Adversarial examples
 - Special inputs which are confusing a neural network
 - Indistinguishable to the human eye
 - Cause the network to fail to identify
- Example



The diagram illustrates the process of creating an adversarial example. It shows three images in a row, separated by mathematical operators. The first image is a clear photo of a panda, labeled x with a confidence of 57.7%. The second image is a square of random noise, labeled $\text{sign}(\nabla_x J(\theta, x, y))$ with a confidence of 8.2%. The third image is the result of adding the noise to the panda image, labeled $x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ with a confidence of 99.3%.

$$\begin{array}{ccc} \text{Image 1} & + .007 \times & \text{Image 2} & = & \text{Image 3} \\ \begin{array}{c} x \\ \text{"panda"} \\ 57.7\% \text{ confidence} \end{array} & & \begin{array}{c} \text{sign}(\nabla_x J(\theta, x, y)) \\ \text{"nematode"} \\ 8.2\% \text{ confidence} \end{array} & & \begin{array}{c} x + \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \\ \text{"gibbon"} \\ 99.3\% \text{ confidence} \end{array} \end{array}$$

Introduction

- Fast gradient sign method
 - which is a *white box* attack whose goal is to ensure misclassification
 - Using the gradients of the loss to maximises the loss
 - Adversarial image can be summarised using the following

$$adv_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

where

- adv_x : Adversarial image.
- x : Original input image.
- y : Original input label.
- ϵ : Multiplier to ensure the perturbations are small.
- θ : Model parameters.
- J : Loss.

Implementing

$$\text{sign}(\nabla_x J(\theta, x, y))$$

```
# 使用Crossentropy當loss function
loss_object = tf.keras.losses.CategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label):
    with tf.GradientTape() as tape:
        tape.watch(input_image)
        prediction = pretrained_model(input_image)
        loss = loss_object(input_label, prediction)

    # Get the gradients of the loss w.r.t to the input image.
    gradient = tape.gradient(loss, input_image)
    # Get the sign of the gradients to create the perturbation
    signed_grad = tf.sign(gradient)
    return signed_grad
```

Implementing

$$adv_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

```
# Get the input label of the image.
# 這邊如果你的圖是dog則 tf.one_hot input 吃dog_index, 這樣他才會產生變成貓的noise
# cat則使用cat_index
dog_index = 1
cat_index = 0
label = tf.one_hot(dog_index, image_probs.shape[-1])
label = tf.reshape(label, (1, image_probs.shape[-1]))
# label = image_probs

perturbations = create_adversarial_pattern(image, label)

# plt.imshow(perturbations[0]*0.5+0.5); # To change [-1, 1] to [0,1]

epsilons = [0, 0.01, 0.1, 0.15]
descriptions = [('Epsilon = {:.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]

for i, eps in enumerate(epsilons):
    # 生產出adversaria的圖, 使用epsilons來控制noise的影響
    adv_x = image + eps*perturbations
    adv_x = tf.cast(adv_x, tf.uint8)
    tf.keras.preprocessing.image.save_img('adversarial_dog1_'+str(i)+'.png',adv_x[0])
```

Result



$+ \varepsilon \times$



$=$



Result



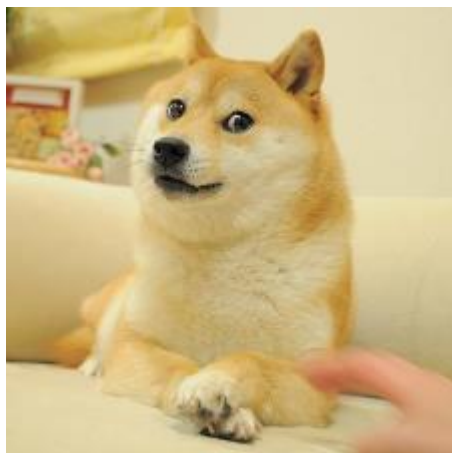
$+ \varepsilon \times$



$=$



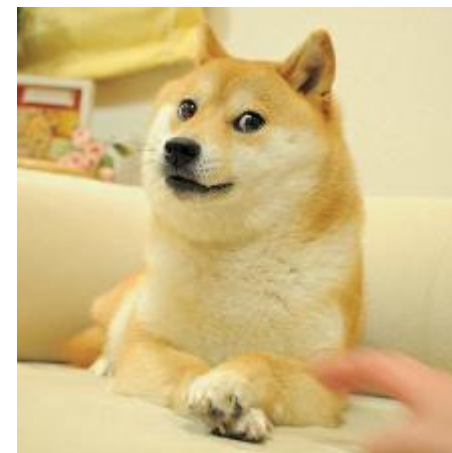
Result



$+$ $\varepsilon \times$



$=$



Result

```
adversaria_dog1_2.png is cat
  1.000 cat
  0.000 dog

adversaria_cat1_2.png is dog
  0.998 dog
  0.002 cat

adversaria_cat2_2.png is dog
  0.999 dog
  0.001 cat
```