4.2D - Adversarial Attacks on Computer Vision

August 14, 2020

Modified from:

`https://www.tensorflow.org/beta/tutorials/generative/adversarial_fgsm`.

GitHub link!https://github.com/mwizard1010/Deakin-SIT799-Human_AlignedAI/blob/master/Week4/4.2D%20-%20Adversarial%20Attacks%20on%20Computer%20Vision/4.2D%20-%20Adversarial%20Attacks%20on%20Computer%20Vision.ipynb

Welcome to your assignment this week!

To better understand adverse attacks againsts AI and how it is possible to fool an AI system, in this assignment, we will look at a Computer Vision use case.

This assessment creates an *adversarial example* using the Fast Gradient Signed Method (FGSM) attack as described in Explaining and Harnessing Adversarial Examples by Goodfellow *et al.* This was one of the first and most popular attacks to fool a neural network.

1 What is an adversarial example?

Adversarial examples are specialised inputs created with the purpose of confusing a neural network, resulting in the misclassification of a given input. These notorious inputs are indistinguishable to the human eye, but cause the network to fail to identify the contents of the image. There are several types of such attacks, however, here the focus is on the fast gradient sign method attack, which is a *white box* attack whose goal is to ensure misclassification. A white box attack is where the attacker has complete access to the model being attacked.

2 Fast gradient sign method

The fast gradient sign method works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximises the loss. This new image is called the adversarial image. This can be summarised using the following expression:

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

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.

An intriguing property here, is the fact that the gradients are taken with respect to the input image. This is done because the objective is to create an image that maximises the loss. A method to accomplish this is to find how much each pixel in the image contributes to the loss value, and add a perturbation accordingly. This works pretty fast because it is easy find how each input pixel contributes to the loss, by using the chain rule, and finding the required gradients. Hence, the gradients are used with respect to the image. In addition, since the model is no longer being trained (thus the gradient is not taken with respect to the trainable variables, i.e., the model parameters), and so the model parameters remain constant. The only goal is to fool an already trained model.

3 Part 1

So let's try and fool a pretrained model. In this first part, the model is MobileNetV2 model, pretrained on ImageNet.

Run the following cell to install all the packages you will need.

[2]:
```
# ! pip3 install cython
# ! pip3 install tensornets
# ! pip3 install  numpy==1.16.1
# ! pip3 install tensorflow
# ! pip3 install matplotlib
```

Run the following cell to load the packages you will need.

[3]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import matplotlib as mpl
import matplotlib.pyplot as plt
import tensornets as nets
```

```
WARNING:tensorflow:From /Users/yama/opt/anaconda3/lib/python3.7/site-
packages/tensorflow/python/compat/v2_compat.py:96: disable_resource_variables
(from tensorflow.python.ops.variable_scope) is deprecated and will be removed in
a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

[4]:
```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.log_device_placement = True
config.allow_soft_placement = True
sess = tf.Session(config=config)
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
```

Let's define the computation graph.

[5]:
```
# Helper function to preprocess the image so that it can be inputted in␣
↪MobileNetV2
def preprocess(image):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (224, 224))
    image = image /  127.5
    image = image - 1.0
    image = image[None, ...]
    return image
def reverse_preprocess(image):
    image = image + 1.0
    image = image / 2.0
    return image

# Helper function to extract labels from probability vector
def get_imagenet_label(probs):
    return decode_predictions(probs, top=5)[0]


# Lets's import an image to process.
image_path = tf.keras.utils.get_file('YellowLabradorLooking_new.jpg', 'https://
↪storage.googleapis.com/download.tensorflow.org/example_images/
↪YellowLabradorLooking_new.jpg')
image_raw = tf.io.read_file(image_path)
image = tf.image.decode_png(image_raw)
input_image = preprocess(image)
reversed_image = reverse_preprocess(input_image)

input_image_placeholder = tf.placeholder(shape=[1, 224, 224, 3], dtype=tf.
↪float32)

pretrained_model = nets.MobileNet50v2(input_image_placeholder, reuse=tf.
↪AUTO_REUSE)

# node to load pretrained weights
pretrained_ops = pretrained_model.pretrained()

# decode predicted probabilities to ImageNet labels
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

```
WARNING:tensorflow:From /Users/yama/opt/anaconda3/lib/python3.7/site-
packages/tensornets/contrib_layers/layers.py:1057: Layer.apply (from
tensorflow.python.keras.engine.base_layer_v1) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `layer.__call__` method instead.
```

3.1 Original image

Let's use a sample image of a Labrador Retriever -by Mirko CC-BY-SA 3.0 from Wikimedia Common and create adversarial examples from it. The first step is to preprocess it so that it can be fed as an input to the MobileNetV2 model.

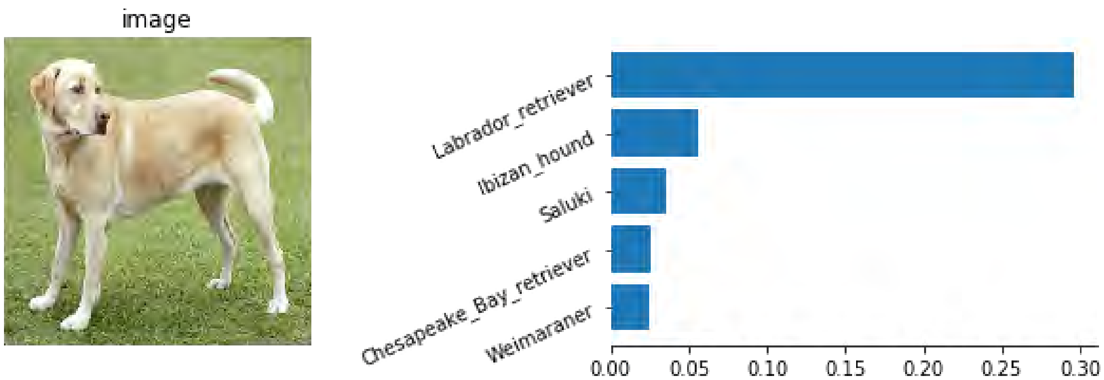
[6]:
```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.log_device_placement = True
sess = tf.Session(config=config)

sess.run(pretrained_ops)
preprocessed_img, reversed_img = sess.run([input_image, reversed_image])
image_probs = sess.run([pretrained_model], {input_image_placeholder:
 ↪preprocessed_img})
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
```

Let's have a look at the image.

[7]:
```
top5 = get_imagenet_label(image_probs[0])
tick_names = [x[1] for x in top5]
print(tick_names)
probs = [x[2] for x in top5]
plt.figure(figsize=(9, 3))
plt.subplot(121)
plt.imshow(reversed_img[0])
plt.title('image')
ax = plt.gca()
ax.axis('off')

plt.subplot(122)
tick_names = [x[1] for x in reversed(top5)]
probs = [x[2] for x in reversed(top5)]
plt.barh(tick_names, probs)
plt.yticks(rotation=25)
ax = plt.gca()
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
plt.tight_layout()
plt.show()
```

```
['Labrador_retriever', 'Ibizan_hound', 'Saluki', 'Chesapeake_Bay_retriever',
'Weimaraner']
```



4 Create the adversarial image

4.1 Implementing fast gradient sign method

The first step is to create perturbations which will be used to distort the original image resulting in an adversarial image. As mentioned, for this task, the gradients are taken with respect to the image.

TASK 1: Implement `create_adversarial_pattern()`. You will need to carry out 3 steps:

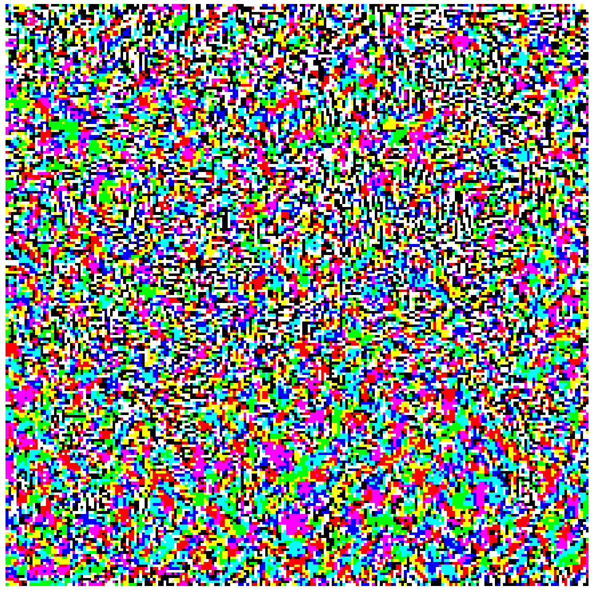
1. Create a loss object using `loss_object` using two argument `input_image` and `input_label`.
2. Get the gradients using `tf.gradients` of the `loss` w.r.t to the `input_image`.
3. Get the sign of the gradients to create the perturbation using `tf.sign`.

[8]:
```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy()

def create_adversarial_pattern(input_image, input_label):
    ## START YOU CODE HERE (3 lines)
    loss = loss_object(input_label, pretrained_model) # loss is differences based on real label and redicted label
    gradient = tf.gradients(loss, input_image) # add calculate gradient image
    signed_grad = tf.sign(gradient)[0] # get the signed of gradient image
    # END
    return signed_grad
```

The resulting perturbations can also be visualised.

[9]:
```
perturbations = create_adversarial_pattern(input_image_placeholder, tf.argmax(pretrained_model,1))
p_cliped = tf.clip_by_value(perturbations, 0, 1)
p_cliped_val = sess.run(p_cliped, {input_image_placeholder: preprocessed_img})
plt.figure()
plt.imshow(p_cliped_val[0])
plt.gca().axis('off')
```

```
plt.show()
```



4.2 Fool the AI system

Let's try this out for different values of epsilon and observe the resultant image. You'll notice that as the value of epsilon is increased, it becomes easier to fool the network, however, this comes as a trade-off which results in the perturbations becoming more identifiable.

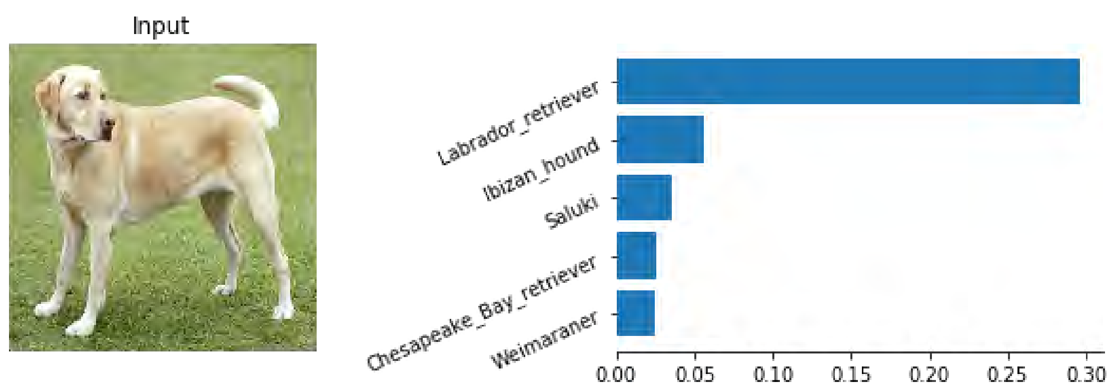
[10]:
```
def display_images(image, description):
    rev_image = reverse_preprocess(image)
    adv_img, raw_adv_img = sess.run([image, rev_image],
 ↪{input_image_placeholder: preprocessed_img})
    img_probs = sess.run(pretrained_model, {input_image_placeholder: adv_img})
    top5 = get_imagenet_label(img_probs)
    top5 = list(reversed(top5))
    plt.figure(figsize=(9, 3))
    plt.subplot(121)
    plt.imshow(raw_adv_img[0])
    plt.title(description)
    plt.gca().axis('off')
    plt.subplot(122)
    tick_names = [x[1] for x in top5]
    probs = [x[2] for x in top5]
    plt.barh(tick_names, probs)
    plt.yticks(rotation=25)
    ax = plt.gca()
    ax.spines['top'].set_visible(False)
```

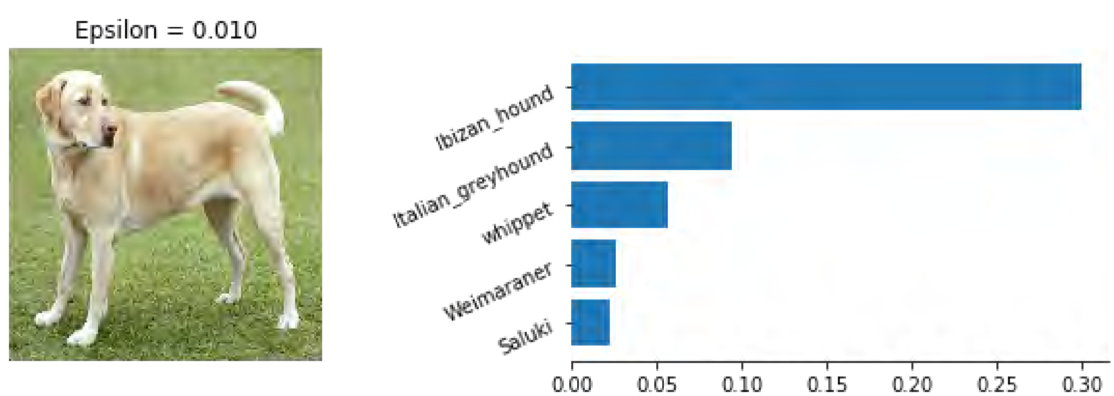
```
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
```

TASK 1: Generate adverse image using different values for ϵ:

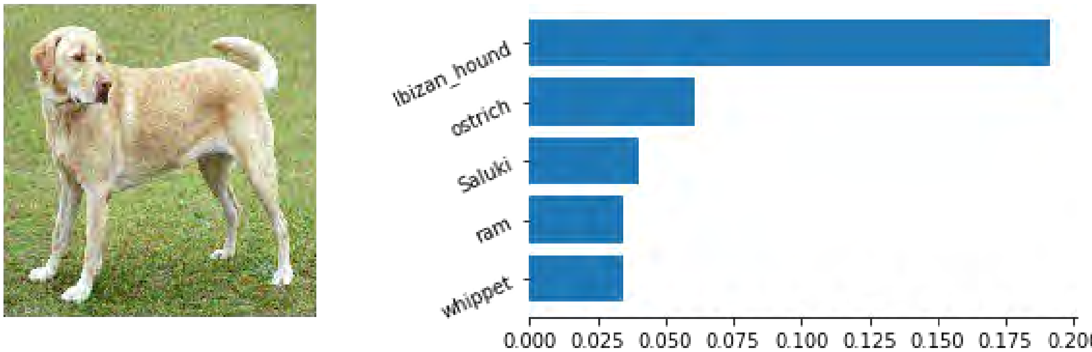
- adv_x = input_image + ϵ * perturbations

[11]:
```
epsilons = [0, 0.01, 0.1, 0.15, 0.3]
descriptions = [('Epsilon = {:0.3f}'.format(eps) if eps else 'Input')
                for eps in epsilons]

for i, eps in enumerate(epsilons):
    ## START YOU CODE HERE
    adv_x = input_image + eps * perturbations[0] # as function
    ## End
    adv_x = tf.clip_by_value(adv_x, -1, 1)
    display_images(adv_x, descriptions[i])
```

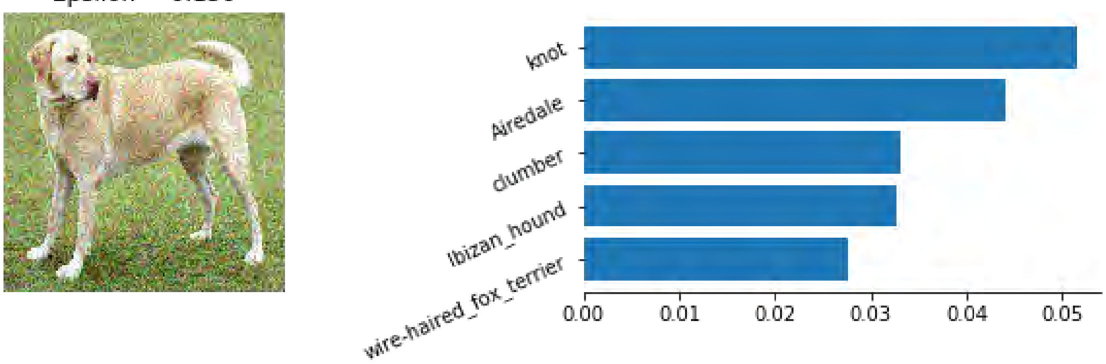




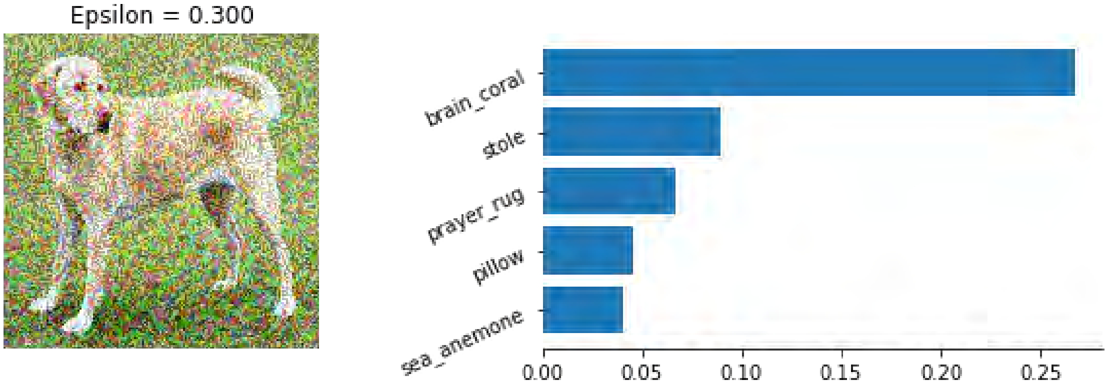
Epsilon = 0.100



Epsilon = 0.150



Epsilon = 0.300



is **TASK 2: What do you abserve?**

PLEASE ANSWER HERE!

. The more value put in epsilon the more fade image becames

. The more value put on epsilon the more false in identifier become, The more we fool the system.

At the value of epsilon = 0.01, 0.1, it can remain recognised th object as hound, greyhound but the confidence is reducing

At the value of epsilon = 0.15 it can not be recognised as hound, greyhound. It is a knot. However, abilty to be a hound still be in top 5 of object recognization

At the value of epsilon = 0.3, It is recognised as brain_coral with very high confidence, nex is stole, rug, or pillow. It can not indentify as a dog or hound in top 5 of objects

---

5 Part 2

Here, you are required to process adversarial attacks using FGSM for a small subset of ImageNet Dataset. We prepared 100 images from different categories (in `./input_dir/`), and the labels are encoded in `./input_dir/clean_image.list`.

For evaluation, each adversarial image generated by the attack model will be fed to an evaluation model, and we will calculate the successful rate of adversarial attacks. **The adversarial images that can fool the evaluation model with $\epsilon = 0.01$ will be considered as a success**.

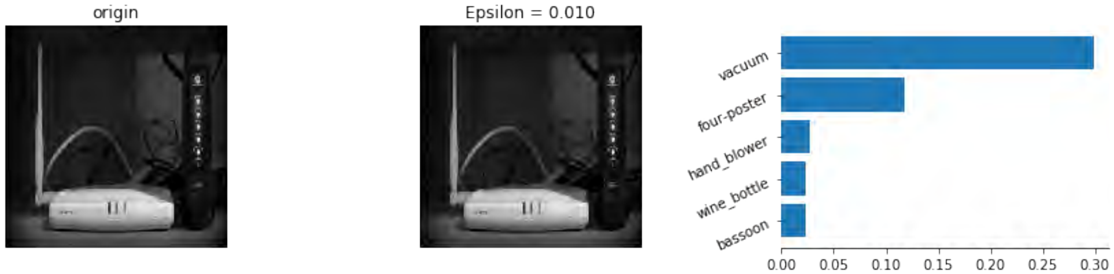
Task 3: Goal

With the previous FGSM example, you are required to implement an FGSM attack against all examples and calculate the success rate. Also, display the original image with the attacked image as well as the predicted class for each image.

[12]:
```
def display_images_custom(image_fooled, description, image, reverse_image):
    rev_image = reverse_preprocess(image_fooled)
    adv_img, raw_adv_img = sess.run([image_fooled, rev_image],␣
 ↪{input_image_placeholder: image})

    img_probs_origin = sess.run(pretrained_model, {input_image_placeholder:␣
 ↪image})
    img_probs_ep = sess.run(pretrained_model, {input_image_placeholder:␣
 ↪adv_img})
    top5 = get_imagenet_label(img_probs_ep)
    top5 = list(reversed(top5))
    plt.figure(figsize=(12, 3))
    plt.subplot(131)
    plt.imshow(reverse_image[0])
    plt.title('origin')
    plt.gca().axis('off')

    plt.subplot(132)
```

```
    plt.imshow(raw_adv_img[0])
    plt.title(description)
    plt.gca().axis('off')
    plt.subplot(133)
    tick_names = [x[1] for x in top5]
    probs = [x[2] for x in top5]
    plt.barh(tick_names, probs)
    plt.yticks(rotation=25)
    ax = plt.gca()
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
    return top5[4][0] # return the most approriate classname
```

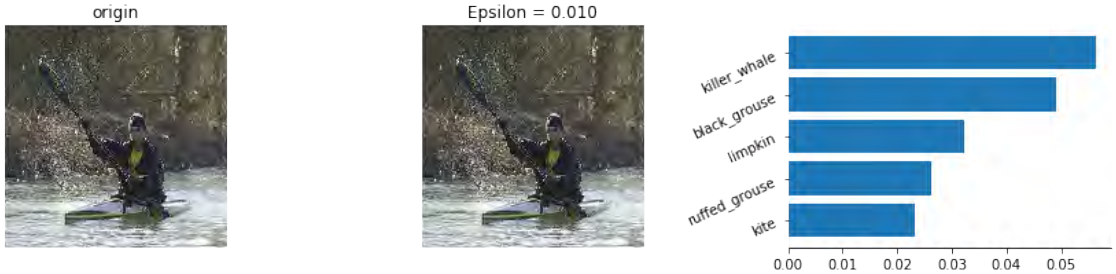
[14]:
```
import numpy as np

# %run 'utils.py'

labels = []
predict_label = []
path = 'input_dir'

epsilon = 0.01

for img in os.listdir(path):  # iterate over each image
    if img.lower().endswith(('.jpg', '.jpeg')):
        ## Add image
        image_path = os.path.join(path,img)
        image_raw = tf.io.read_file(image_path)
        image = tf.image.decode_png(image_raw)
        labels.append(os.path.splitext(img)[0])

        ## Preprocessing
        in_image = preprocess(image)
        re_image = reverse_preprocess(in_image)
        preprocessed_img, reversed_img = sess.run([in_image, re_image])

        ## Fool system
        adv_x = in_image + epsilon * perturbations[0] # as function
        adv_x = tf.clip_by_value(adv_x, -1, 1)
        classname = display_images_custom(adv_x, 'Epsilon = {:0.3f}'.
↪format(epsilon), preprocessed_img, reversed_img)
        predict_label.append(classname)
```


origin
Epsilon = 0.010
vacuum
four-poster
hand_blower
wine_bottle
bassoon
0.00 0.05 0.10 0.15 0.20 0.25 0.30


origin
Epsilon = 0.010
Lhasa
Pekinese
Shih-Tzu
pug
Maltese_dog
0.00 0.05 0.10 0.15 0.20 0.25


origin
Epsilon = 0.010
dumbbell
hook
barbell
chain
neck_brace
0.00 0.05 0.10 0.15 0.20 0.25


origin
Epsilon = 0.010
consomme
plate
hot_pot
carbonara
spaghetti_squash
0.00 0.05 0.10 0.15 0.20


origin
Epsilon = 0.010
killer_whale
black_grouse
limpkin
ruffed_grouse
kite
0.00
0.01
0.02
0.03
0.04
0.05

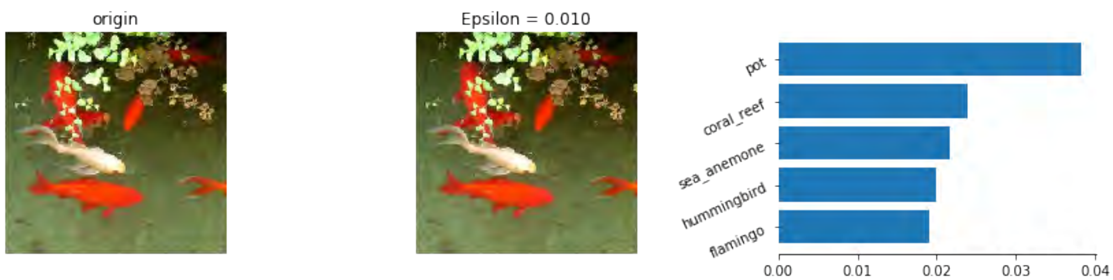

origin
Epsilon = 0.010
china_cabinet
guillotine
entertainment_center
sliding_door
pay-phone
0.0
0.1
0.2
0.3


origin
Epsilon = 0.010
wood_rabbit
grey_fox
coyote
hare
fox_squirrel
0.000
0.025
0.050
0.075
0.100
0.125
0.150


origin
Epsilon = 0.010
Indian_cobra
garter_snake
thunder_snake
vine_snake
night_snake
0.00
0.02
0.04
0.06


origin
Epsilon = 0.010
English_setter
miniature_schnauzer
standard_schnauzer
bluetick
cocker_spaniel
0.000
0.005
0.010
0.015
0.020

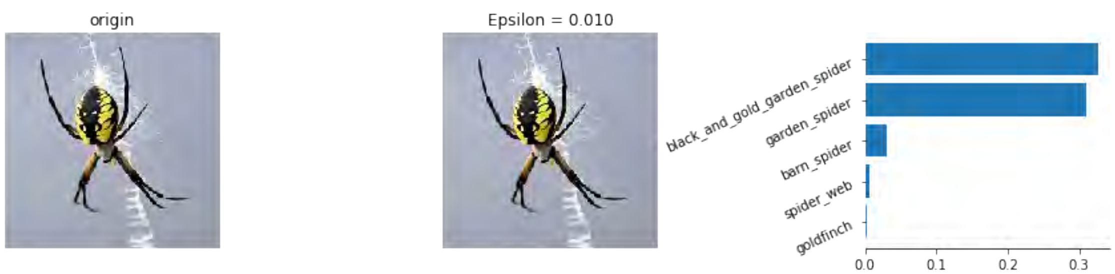

origin
Epsilon = 0.010
seashore
lakeside
dam
radio_telescope
planetarium
0.00
0.05
0.10
0.15


origin
Epsilon = 0.010
pot
coral_reef
sea_anemone
hummingbird
flamingo
0.00
0.01
0.02
0.03
0.04

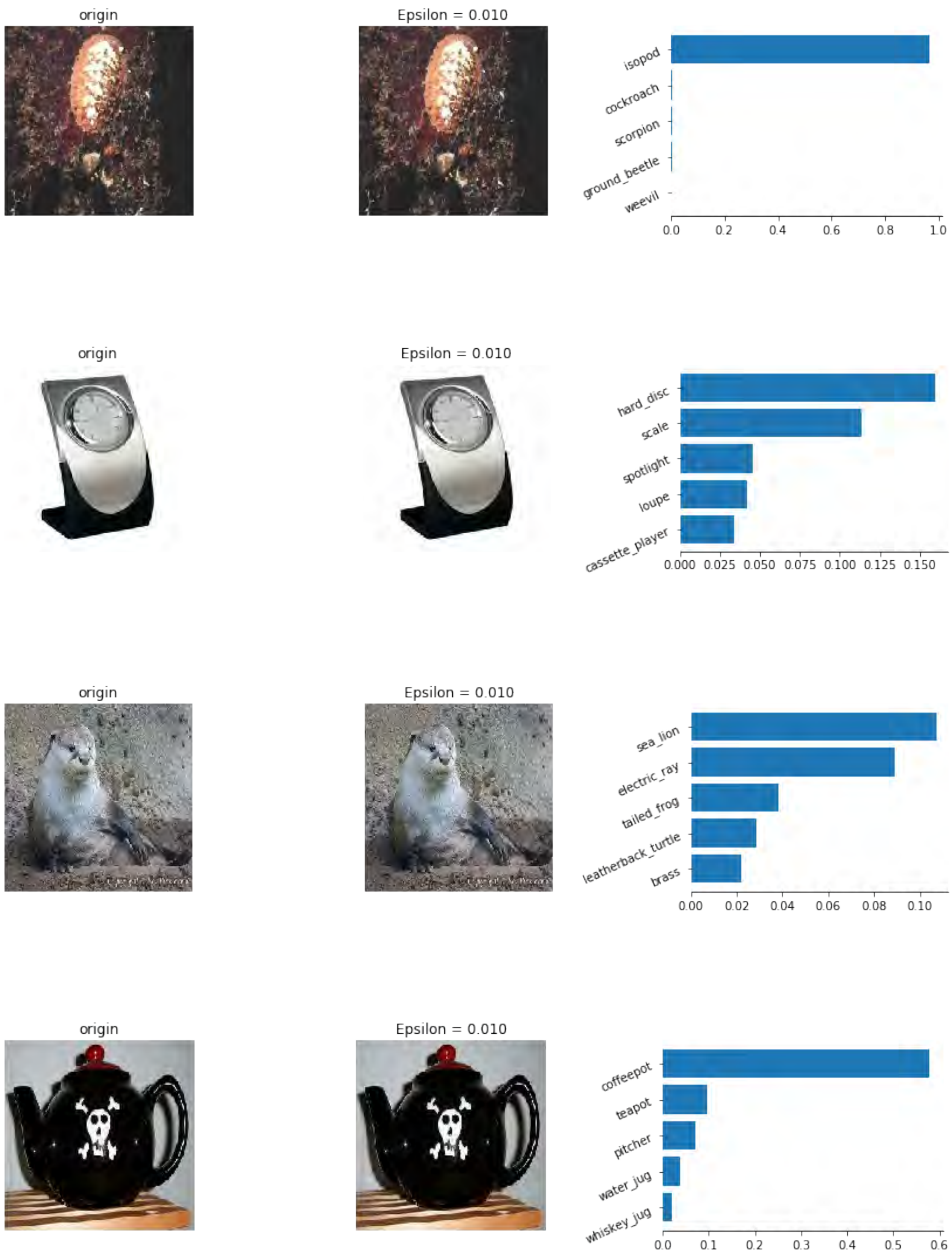

origin
Epsilon = 0.010
stretcher
groom
sleeping_bag
toilet_tissue
prison
0.00
0.02
0.04
0.06
0.08
0.10


origin
Epsilon = 0.010
fur_coat
Irish_wolfhound
komondor
Norwegian_elkhound
borzoi
0.0 0.1 0.2 0.3 0.4

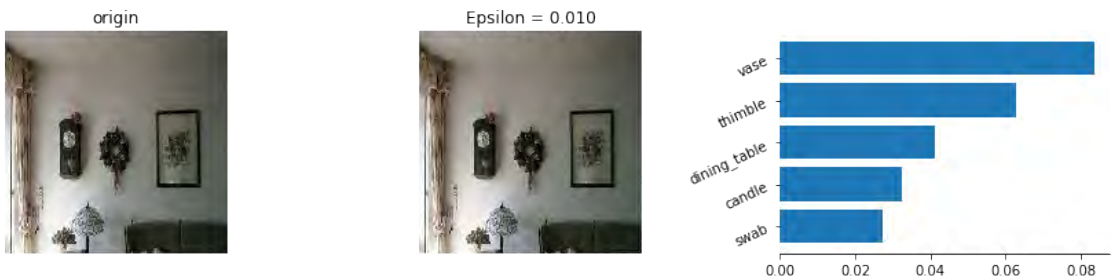

origin
Epsilon = 0.010
spaghetti_squash
plate
hot_pot
potpie
broccoli
0.000 0.025 0.050 0.075 0.100 0.125 0.150


origin
Epsilon = 0.010
black_and_gold_garden_spider
garden_spider
barn_spider
spider_web
goldfinch
0.0 0.1 0.2 0.3


origin
Epsilon = 0.010
Norfolk_terrier
Norwich_terrier
go-kart
steam_locomotive
suspension_bridge
0.00 0.01 0.02 0.03 0.04 0.05 0.06

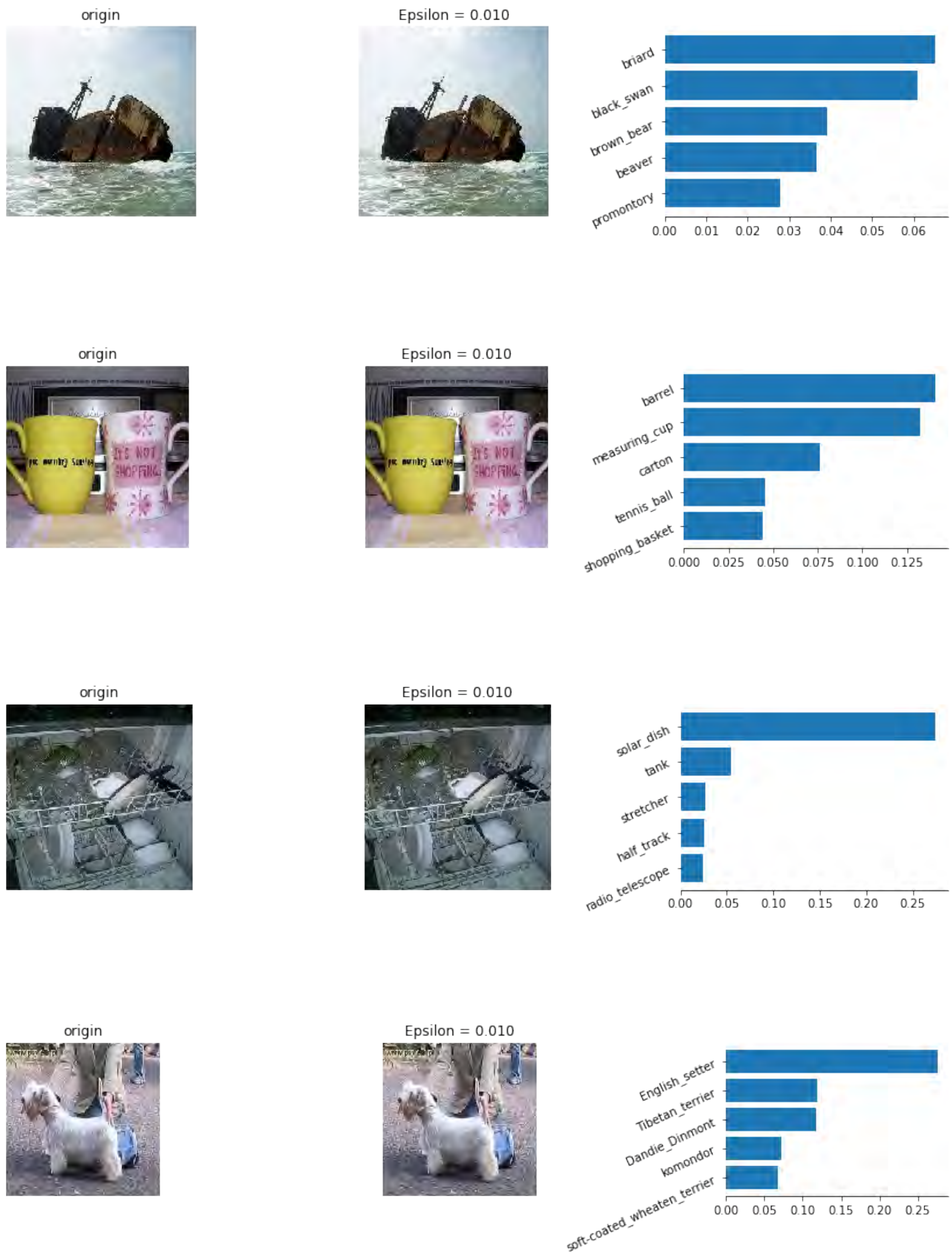

origin
Epsilon = 0.010
isopod
cockroach
scorpion
ground_beetle
weevil
0.0 0.2 0.4 0.6 0.8 1.0
origin
Epsilon = 0.010
hard_disc
scale
spotlight
loupe
cassette_player
0.000 0.025 0.050 0.075 0.100 0.125 0.150
origin
Epsilon = 0.010
sea_lion
electric_ray
tailed_frog
leatherback_turtle
brass
0.00 0.02 0.04 0.06 0.08 0.10
origin
Epsilon = 0.010
coffeepot
teapot
pitcher
water_jug
whiskey_jug
0.0 0.1 0.2 0.3 0.4 0.5 0.6


origin
Epsilon = 0.010
revolver
beach_wagon
pickup
minibus
motor_scooter
0.00
0.05
0.10
0.15


origin
Epsilon = 0.010
vase
thimble
dining_table
candle
swab
0.00
0.02
0.04
0.06
0.08


origin
Epsilon = 0.010
lakeside
boathouse
worm_fence
wreck
park_bench
0.0
0.1
0.2
0.3
0.4
0.5

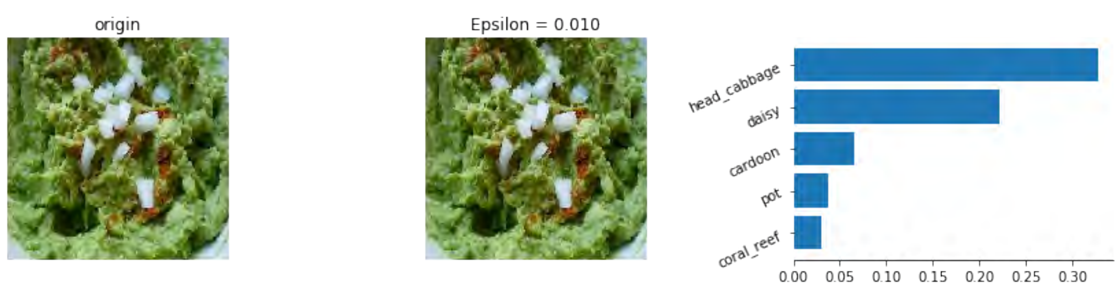

origin
Epsilon = 0.010
Australian_terrier
kit_fox
red_fox
Norwich_terrier
otterhound
0.00
0.01
0.02
0.03
0.04
0.05
0.06


origin
Epsilon = 0.010
briard
black_swan
brown_bear
beaver
promontory
0.00 0.01 0.02 0.03 0.04 0.05 0.06
origin
Epsilon = 0.010
barrel
measuring_cup
carton
tennis_ball
shopping_basket
0.000 0.025 0.050 0.075 0.100 0.125
origin
Epsilon = 0.010
solar_dish
tank
stretcher
half_track
radio_telescope
0.00 0.05 0.10 0.15 0.20 0.25
origin
Epsilon = 0.010
English_setter
Tibetan_terrier
Dandie_Dinmont
komondor
soft-coated_wheaten_terrier
0.00 0.05 0.10 0.15 0.20 0.25


origin
Epsilon = 0.010
dam
greenhouse
rain_barrel
boathouse
barrel
0.00
0.01
0.02
0.03
0.04
0.05
0.06


origin
Epsilon = 0.010
hen-of-the-woods
gyromitra
mushroom
earthstar
bolete
0.0
0.1
0.2
0.3
0.4


origin
Epsilon = 0.010
bearskin
miniskirt
steel_drum
panpipe
shower_cap
0.00
0.02
0.04
0.06
0.08
0.10


origin
Epsilon = 0.010
head_cabbage
daisy
cardoon
pot
coral_reef
0.00
0.05
0.10
0.15
0.20
0.25
0.30

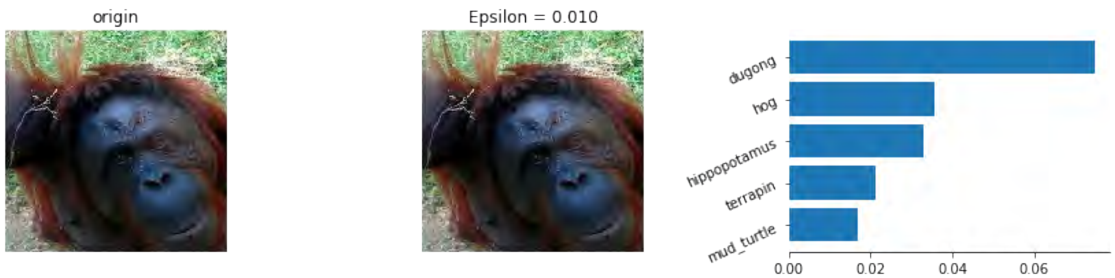

origin
Epsilon = 0.010
mortar
candle
bucket
goblet
mixing_bowl
0.00 0.01 0.02 0.03 0.04 0.05 0.06


origin
Epsilon = 0.010
carpenter's_kit
screw
quill
maraca
screwdriver
0.00 0.02 0.04 0.06 0.08 0.10


origin
Epsilon = 0.010
buckle
sunglasses
hair_slide
coffee_mug
brassiere
0.0 0.2 0.4 0.6 0.8


origin
Epsilon = 0.010
refrigerator
dishwasher
microwave
apron
apiary
0.000 0.025 0.050 0.075 0.100 0.125 0.150


origin
Epsilon = 0.010
space_bar
notebook
hand-held_computer
typewriter_keyboard
computer_keyboard
0.00 0.02 0.04 0.06 0.08 0.10 0.12


origin
Epsilon = 0.010
dugong
hog
hippopotamus
terrapin
mud_turtle
0.00 0.02 0.04 0.06


origin
Epsilon = 0.010
chiffonier
file
wardrobe
chest
medicine_chest
0.0 0.1 0.2 0.3 0.4 0.5

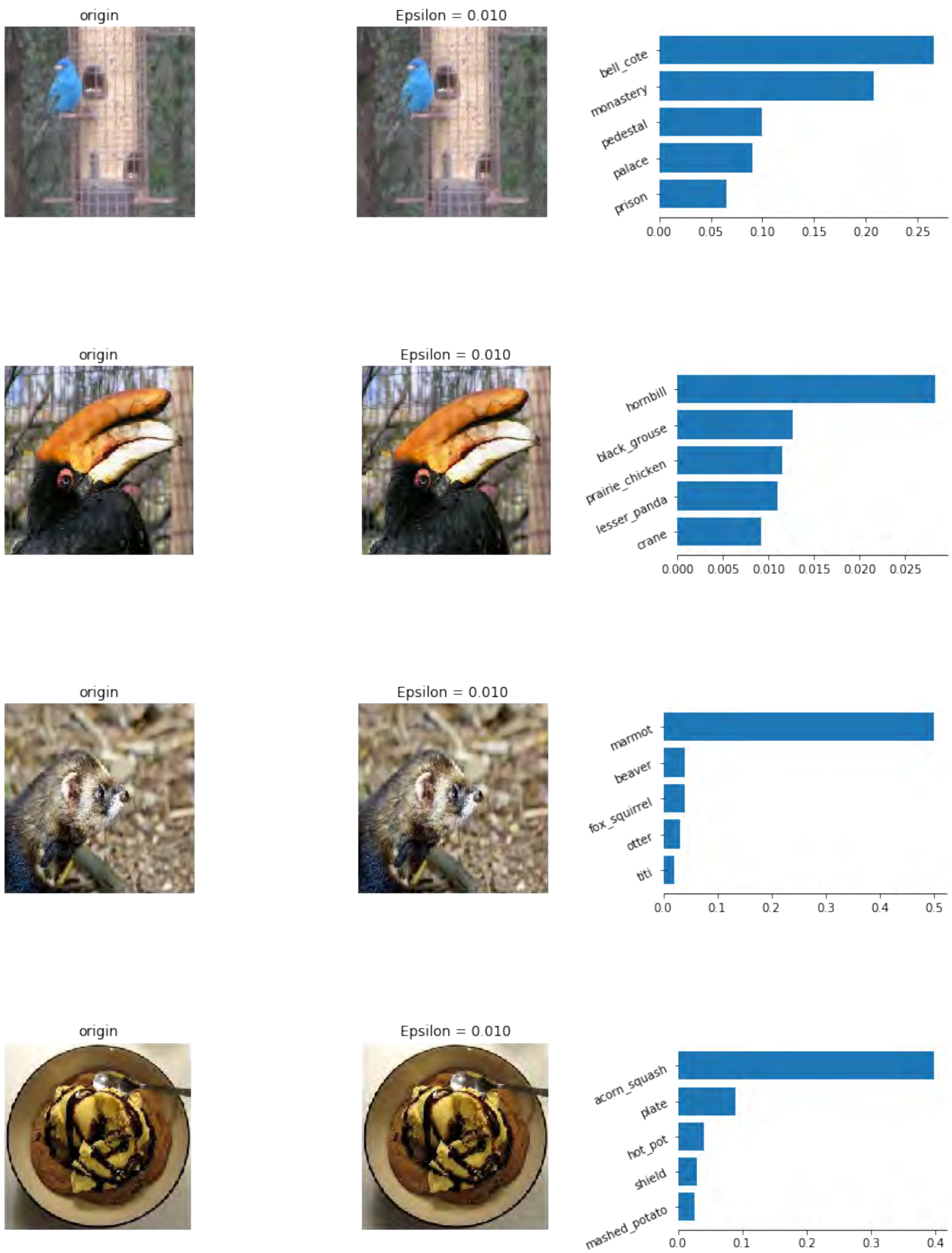

origin
Epsilon = 0.010
racer
cab
garbage_truck
snowplow
pickup
0.00 0.01 0.02 0.03 0.04 0.05 0.06

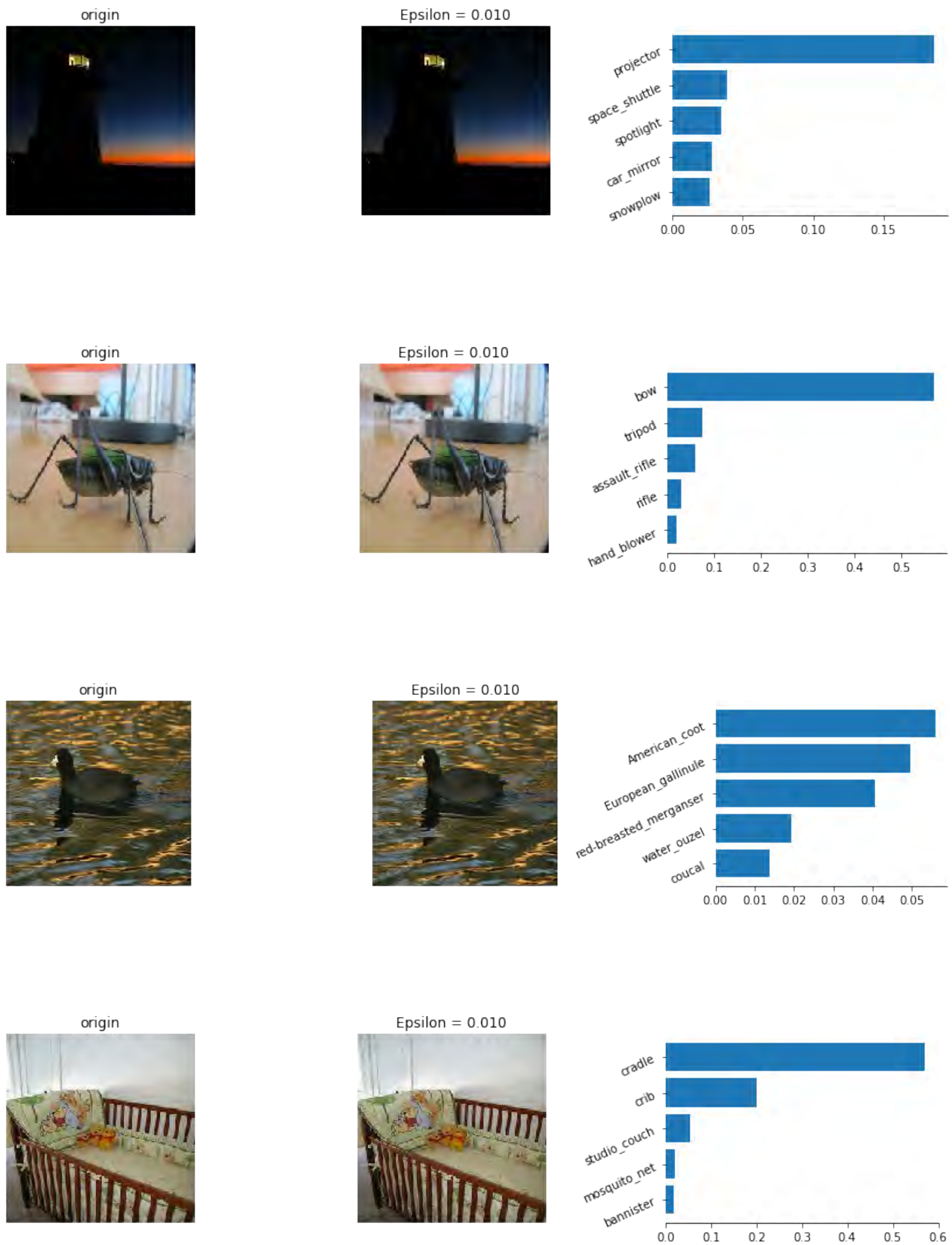

origin
Epsilon = 0.010
kimono
abaya
backpack
overskirt
cloak
0.00 0.05 0.10 0.15 0.20 0.25 0.30


origin
Epsilon = 0.010
oxygen_mask
crash_helmet
football_helmet
mask
gasmask
0.00 0.02 0.04 0.06 0.08 0.10


origin
Epsilon = 0.010
great_white_shark
sea_lion
grey_whale
killer_whale
dugong
0.00 0.02 0.04 0.06 0.08


origin
Epsilon = 0.010
consomme
plate
ice_cream
hermit_crab
buckeye
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175


origin
Epsilon = 0.010
bell_cote
monastery
pedestal
palace
prison
0.00
0.05
0.10
0.15
0.20
0.25
origin
Epsilon = 0.010
hornbill
black_grouse
prairie_chicken
lesser_panda
crane
0.000
0.005
0.010
0.015
0.020
0.025
origin
Epsilon = 0.010
marmot
beaver
fox_squirrel
otter
titi
0.0
0.1
0.2
0.3
0.4
0.5
origin
Epsilon = 0.010
acorn_squash
plate
hot_pot
shield
mashed_potato
0.0
0.1
0.2
0.3
0.4


origin
Epsilon = 0.010
projector
space_shuttle
spotlight
car_mirror
snowplow
0.00
0.05
0.10
0.15
origin
Epsilon = 0.010
bow
tripod
assault_rifle
rifle
hand_blower
0.0
0.1
0.2
0.3
0.4
0.5
origin
Epsilon = 0.010
American_coot
European_gallinule
red-breasted_merganser
water_ouzel
coucal
0.00
0.01
0.02
0.03
0.04
0.05
origin
Epsilon = 0.010
cradle
crib
studio_couch
mosquito_net
bannister
0.0
0.1
0.2
0.3
0.4
0.5
0.6


origin
Epsilon = 0.010
pirate
crane
electric_locomotive
container_ship
dock
0.000 0.025 0.050 0.075 0.100 0.125


origin
Epsilon = 0.010
goose
pelican
black_swan
prairie_chicken
hen
0.00 0.05 0.10 0.15 0.20 0.25 0.30

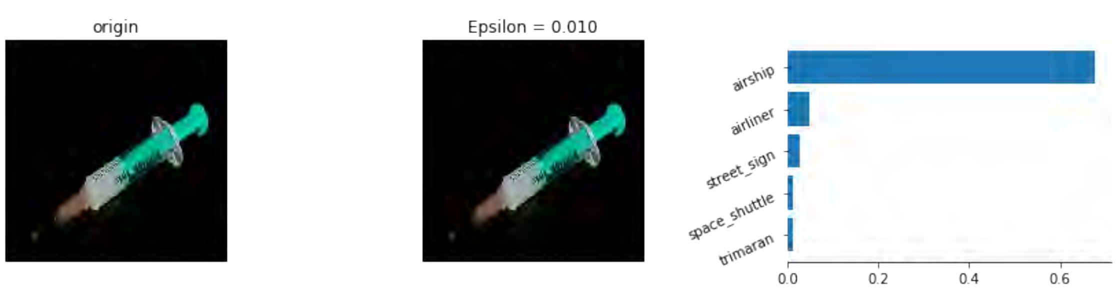

origin
Epsilon = 0.010
bikini
sunglass
maillot
sunscreen
sandbar
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175


origin
Epsilon = 0.010
chest
slot
altar
breastplate
grille
0.000 0.025 0.050 0.075 0.100 0.125 0.150 0.175


origin
Epsilon = 0.010
oil_filter
screw
accordion
cocktail_shaker
nail
0.00
0.05
0.10
0.15
0.20


origin
Epsilon = 0.010
stinkhorn
eggnog
acorn
mushroom
birdhouse
0.00
0.02
0.04
0.06
0.08
0.10


origin
Epsilon = 0.010
backpack
mask
kimono
mud_turtle
pickelhaube
0.000
0.025
0.050
0.075
0.100
0.125
0.150


origin
Epsilon = 0.010
airship
airliner
street_sign
space_shuttle
trimaran
0.0
0.2
0.4
0.6

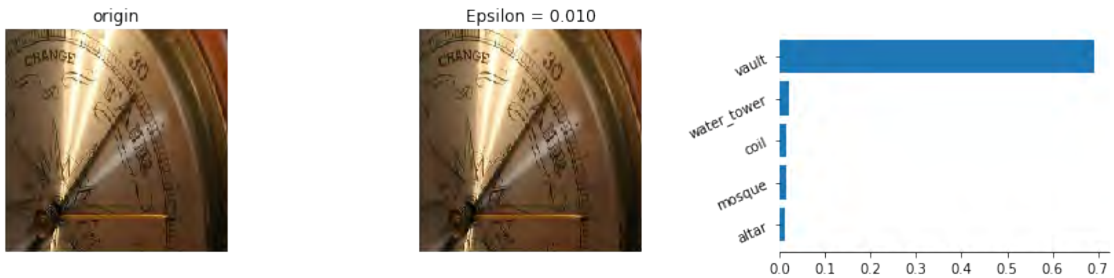

origin
Epsilon = 0.010
mountain_tent
sarong
yurt
sleeping_bag
volcano
0.0
0.1
0.2
0.3


origin
Epsilon = 0.010
junco
hen
English_springer
ruffed_grouse
hamster
0.0
0.1
0.2
0.3
0.4
0.5


origin
Epsilon = 0.010
corkscrew
scabbard
fountain_pen
paintbrush
letter_opener
0.00
0.02
0.04
0.06
0.08


origin
Epsilon = 0.010
whippet
black_grouse
red-breasted_merganser
Saluki
king_penguin
0.000
0.025
0.050
0.075
0.100
0.125

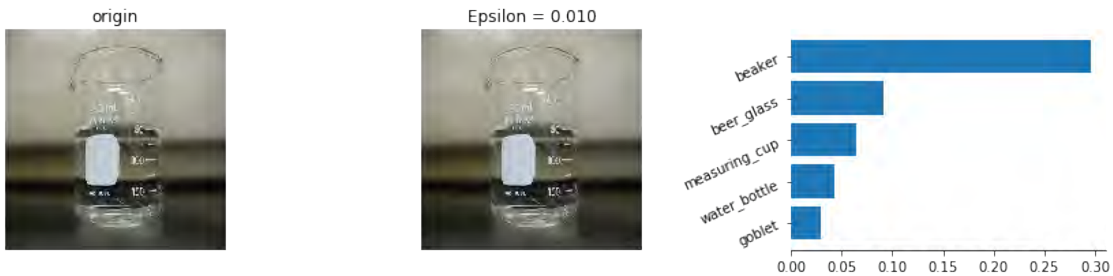

origin
Epsilon = 0.010
rocking_chair
cradle
carousel
plate_rack
cowboy_boot
0.0 0.1 0.2 0.3 0.4 0.5 0.6


origin
Epsilon = 0.010
vault
water_tower
coil
mosque
altar
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7


origin
Epsilon = 0.010
sliding_door
Scotch_terrier
briard
window_shade
Bouvier_des_Flandres
0.0 0.1 0.2 0.3 0.4 0.5 0.6


origin
Epsilon = 0.010
kimono
military_uniform
pajama
velvet
fur_coat
0.0 0.1 0.2 0.3 0.4 0.5 0.6

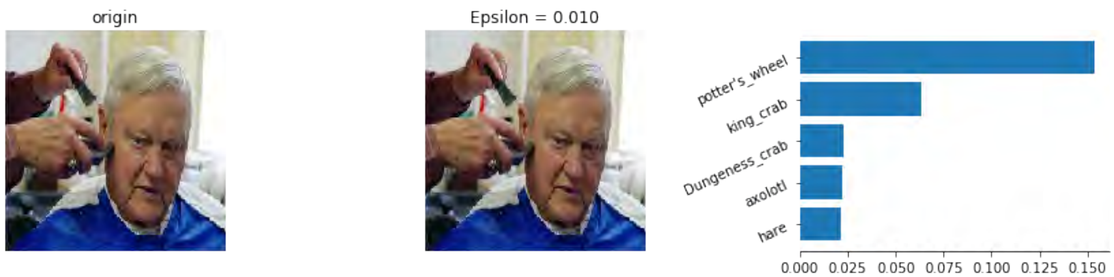

origin
Epsilon = 0.010
pomegranate
chocolate_sauce
spindle
ladle
strainer
0.00
0.05
0.10
0.15
0.20
0.25

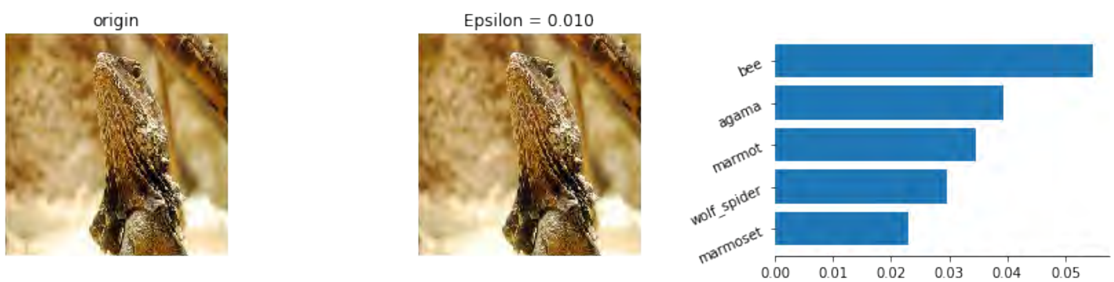

origin
Epsilon = 0.010
beaker
beer_glass
measuring_cup
water_bottle
goblet
0.00
0.05
0.10
0.15
0.20
0.25
0.30


origin
Epsilon = 0.010
burrito
hotdog
conch
potpie
packet
0.00
0.02
0.04
0.06
0.08
0.10


origin
Epsilon = 0.010
paintbrush
bassoon
drumstick
screwdriver
oboe
0.000
0.025
0.050
0.075
0.100
0.125
0.150


origin
Epsilon = 0.010
llama
ibex
Afghan_hound
gazelle
komondor
0.0
0.1
0.2
0.3


origin
Epsilon = 0.010
potter's_wheel
king_crab
Dungeness_crab
axolotl
hare
0.000
0.025
0.050
0.075
0.100
0.125
0.150

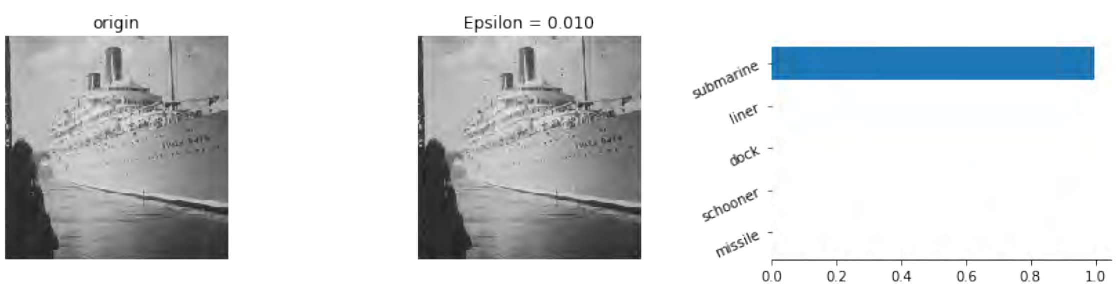

origin
Epsilon = 0.010
bee
agama
marmot
wolf_spider
marmoset
0.00
0.01
0.02
0.03
0.04
0.05

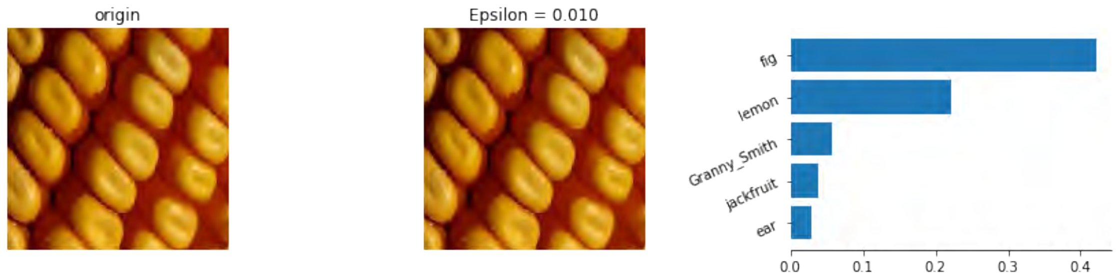

origin
Epsilon = 0.010
Tibetan_terrier
Old_English_sheepdog
Japanese_spaniel
wire-haired_fox_terrier
Shih-Tzu
0.0
0.1
0.2
0.3
0.4
0.5


origin
Epsilon = 0.010
ostrich
garter_snake
night_snake
limpkin
Indian_cobra
0.00
0.01
0.02
0.03
0.04
0.05


origin
Epsilon = 0.010
Great_Dane
Weimaraner
German_short-haired_pointer
muzzle
American_Staffordshire_terrier
0.0
0.1
0.2
0.3

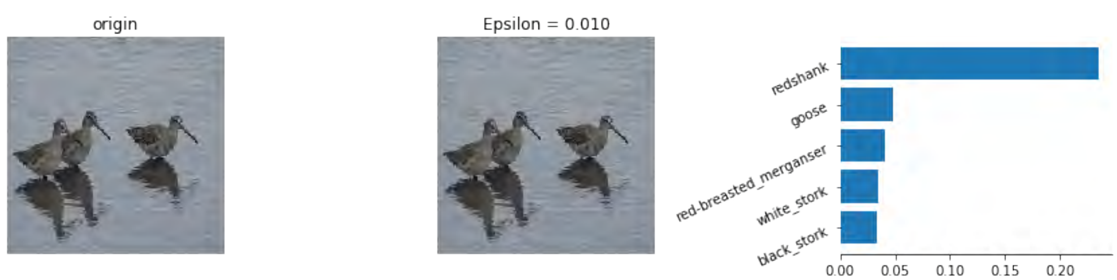

origin
Epsilon = 0.010
tank
trailer_truck
moving_van
tow_truck
limousine
0.00
0.01
0.02
0.03
0.04
0.05


origin
Epsilon = 0.010
submarine
liner
dock
schooner
missile
0.0
0.2
0.4
0.6
0.8
1.0


origin
Epsilon = 0.010
fig
lemon
Granny_Smith
jackfruit
ear
0.0
0.1
0.2
0.3
0.4


origin
Epsilon = 0.010
hourglass
cocktail_shaker
perfume
soap_dispenser
corkscrew
0.000
0.025
0.050
0.075
0.100
0.125


origin
Epsilon = 0.010
coho
sturgeon
gar
trimaran
tiger_shark
0.00
0.05
0.10
0.15
0.20
0.25


origin
Epsilon = 0.010
redshank
goose
red-breasted_merganser
white_stork
black_stork
0.00
0.05
0.10
0.15
0.20

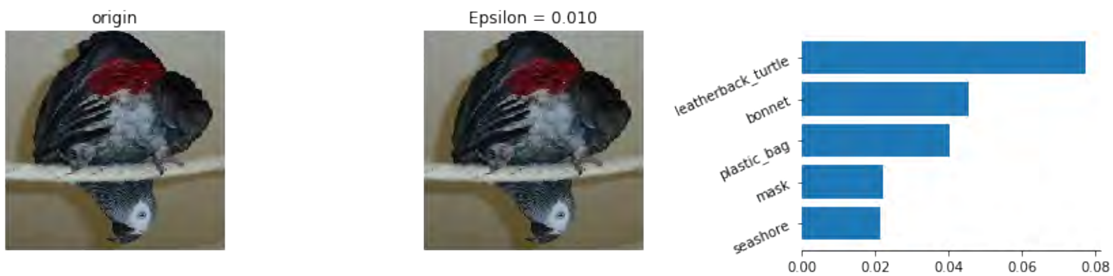

origin
Epsilon = 0.010
broom
panpipe
stinkhorn
slide_rule
ear
0.00 0.02 0.04 0.06 0.08 0.10

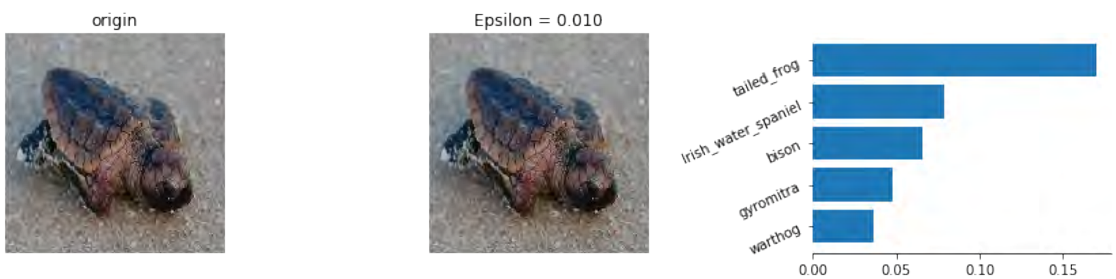

origin
Epsilon = 0.010
water_ouzel
red-breasted_merganser
dowitcher
red-backed_sandpiper
redshank
0.0 0.1 0.2 0.3 0.4


origin
Epsilon = 0.010
oboe
bassoon
flute
violin
harmonica
0.00 0.05 0.10 0.15 0.20 0.25 0.30

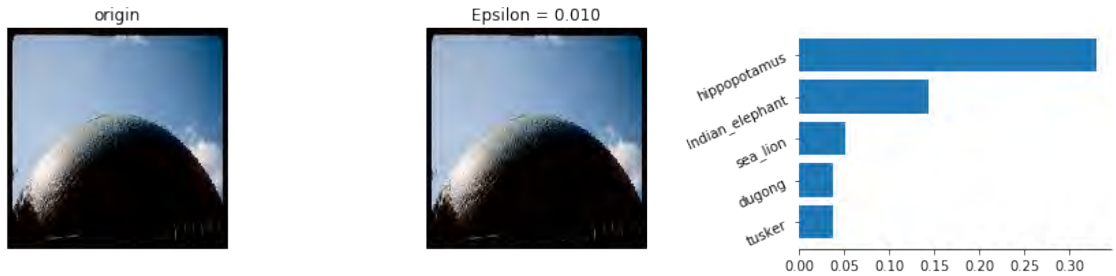

origin
Epsilon = 0.010
bathtub
amphibian
tub
washbasin
toilet_seat
0.00 0.05 0.10 0.15 0.20


origin
Epsilon = 0.010
gazelle
bulbul
impala
hartebeest
goose
0.00
0.05
0.10
0.15
0.20

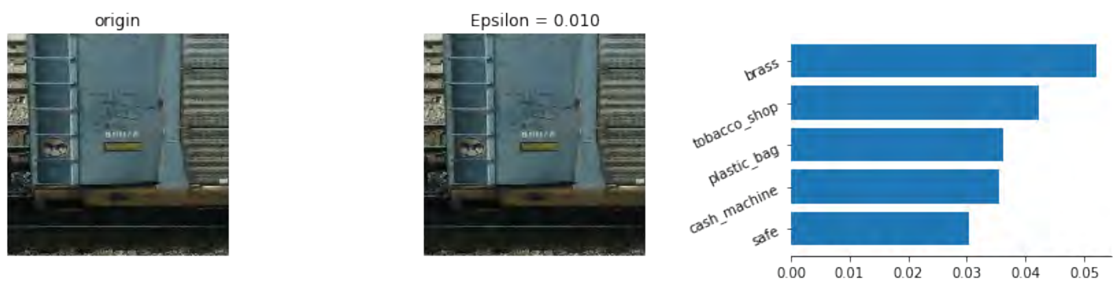

origin
Epsilon = 0.010
leatherback_turtle
bonnet
plastic_bag
mask
seashore
0.00
0.02
0.04
0.06
0.08


origin
Epsilon = 0.010
tailed_frog
Irish_water_spaniel
bison
gyromitra
warthog
0.00
0.05
0.10
0.15

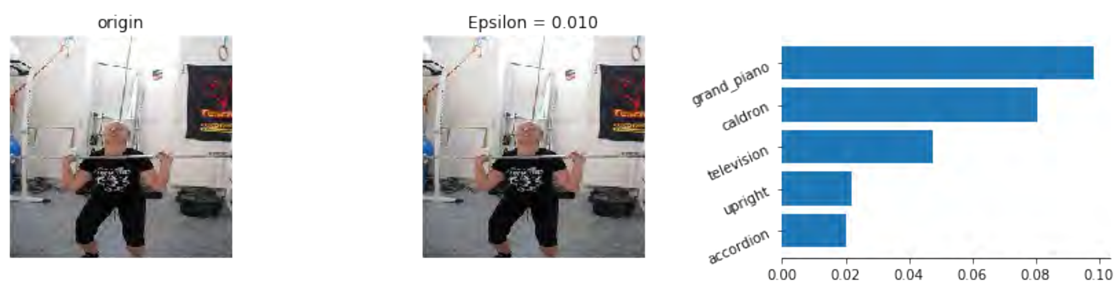

origin
Epsilon = 0.010
tick
barn_spider
ground_beetle
garden_spider
isopod
0.0
0.1
0.2
0.3


origin
Epsilon = 0.010
hippopotamus
Indian_elephant
sea_lion
dugong
tusker
0.00 0.05 0.10 0.15 0.20 0.25 0.30


origin
Epsilon = 0.010
bolo_tie
padlock
knot
purse
safety_pin
0.00 0.05 0.10 0.15 0.20 0.25


origin
Epsilon = 0.010
brass
tobacco_shop
plastic_bag
cash_machine
safe
0.00 0.01 0.02 0.03 0.04 0.05


origin
Epsilon = 0.010
ice_bear
guenon
Arctic_fox
timber_wolf
ant
0.00 0.01 0.02 0.03 0.04




```
['n03777754', 'n02086240', 'n03109150', 'n07717556', 'n03873416', 'n03384352',
'n01877812', 'n02655020', 'n02109525', 'n09399592', 'n01443537', 'n04023962',
'n02129604', 'n07697313', 'n01773157', 'n09332890', 'n01955084', 'n02708093',
'n02444819', 'n04398044', 'n04536866', 'n04548280', 'n02704792', 'n02326432',
```

```
'n04606251', 'n03063599', 'n03207941', 'n02095889', 'n04049303', 'n07754684',
'n03379051', 'n07583066', 'n04311174', 'n04597913', 'n03197337', 'n03467068',
'n03485407', 'n02480495', 'n03016953', 'n03977966', 'n02769748', 'n03424325',
'n02071294', 'n07695742', 'n01537544', 'n01829413', 'n02443114', 'n07836838',
'n02814860', 'n02229544', 'n02018207', 'n03131574', 'n03216828', 'n02012849',
'n04251144', 'n02966687', 'n04153751', 'n12267677', 'n04026417', 'n04376876',
'n03160309', 'n01530575', 'n03794056', 'n02107908', 'n03201208', 'n02794156',
'n04590129', 'n02667093', 'n04525038', 'n02815834', 'n07831146', 'n03721384',
'n02422106', 'n02791270', 'n01688243', 'n02110185', 'n02013706', 'n02099267',
'n04065272', 'n03673027', 'n12144580', 'n04131690', 'n04493381', 'n02033041',
'n02259212', 'n03042490', 'n02110341', 'n03584254', 'n03000134', 'n01817953',
'n01664065', 'n02177972', 'n03956157', 'n03394916', 'n03393912', 'n02093647',
'n02489166', 'n02093256', 'n02777292', 'n02790996']
['n04517823', 'n02098413', 'n03255030', 'n07584110', 'n02071294', 'n03018349',
'n02325366', 'n01748264', 'n02100735', 'n09428293', 'n03991062', 'n04336792',
'n03404251', 'n07716906', 'n01773157', 'n02094114', 'n01990800', 'n03492542',
'n02077923', 'n03063689', 'n04086273', 'n04522168', 'n09332890', 'n02096294',
'n02105251', 'n02795169', 'n04258138', 'n02100735', 'n03160309', 'n13052670',
'n02817516', 'n07714571', 'n03786901', 'n02966687', 'n02910353', 'n04070727',
'n04264628', 'n02074367', 'n03016953', 'n04037443', 'n03617480', 'n03868863',
'n01484850', 'n07584110', 'n02825657', 'n01829413', 'n02361337', 'n07717410',
'n04009552', 'n02879718', 'n02018207', 'n03125729', 'n03947888', 'n01855672',
'n02837789', 'n03014705', 'n03843555', 'n13040303', 'n02769748', 'n02692877',
'n03792972', 'n01534433', 'n03109150', 'n02091134', 'n04099969', 'n04523525',
'n04239074', 'n03617480', 'n07768694', 'n02815834', 'n07880968', 'n03876231',
'n02437616', 'n03992509', 'n02206856', 'n02097474', 'n01518878', 'n02109047',
'n04389033', 'n04347754', 'n07753113', 'n03544143', 'n02536864', 'n02028035',
'n02906734', 'n01601694', 'n03838899', 'n02808440', 'n02423022', 'n01665541',
'n01644900', 'n01776313', 'n02398521', 'n02865351', 'n02892201', 'n02134084',
'n02493793', 'n02110627', 'n03710721', 'n03452741']
```

[22]:
```python
# print(labels)
# print(predict_label)
from sklearn.metrics import accuracy_score
acc = accuracy_score(labels, predict_label)
print('Current accuracy of model: ', acc)
```

```
Current accuracy of model:  0.05
```

We see that the model is nolonger predict correctly, the accuracy only 5%. We polluted the model completely with epsilon = 0.01 .

6 Congratulations!

You've come to the end of this assignment, and have seen a lot of the ways attack and fool an AI system. Here are the main points you should remember:

- It is very easy to fool a computer vision system if you know the model and its parameters.

- When designing an AI system, you need to think of adverse attacks againsts your system.

Congratulations on finishing this notebook!