Modified from:

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

GitHub link!

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.

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.

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.

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.

```
! pip install cython
! pip install tensornets
! pip install numpy==1.16.1
! pip install tensorflow
! pip install matplotlib
```

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

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

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.log_device_placement = True
config.allow_soft_placement = True
sess = tf.Session(config=config)
```

Let's define the computation graph.

```
# Helper function to preprocess the image so that it can be inputted in MobileNetV
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://sto
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)
print(input_image.shape)
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
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/
90112/83281 [==============================] - 0s 0us/step
(1, 224, 224, ?)
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensornets/cor
Instructions for updating:
Please use `layer.__call__` method instead.
Downloading data from https://github.com/taehoonlee/deep-learning-models/rele
8019968/8012166 [==============================] - 1s 0us/step
```

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.

```
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_i
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:XLA_CPU:0 -> device: XLA_CPU device
/job:localhost/replica:0/task:0/device:XLA_GPU:0 -> device: XLA_GPU device
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: Tesla T4, pc
```

Let's have a look at the image.

```
top5 = get_imagenet_label(image_probs[0])
#print(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()
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/
40960/35363 [==============================] - 0s 0us/step
```

Create the adversarial image

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`.

```
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)
    gradient = tf.gradients(loss, input_image)
    signed_grad = tf.sign(gradient)[0]
    # END
    return signed_grad
```

The resulting perturbations can also be visualised.

```
perturbations = create_adversarial_pattern(input_image_placeholder, tf.argmax(pret
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()
```

Fool the AI system

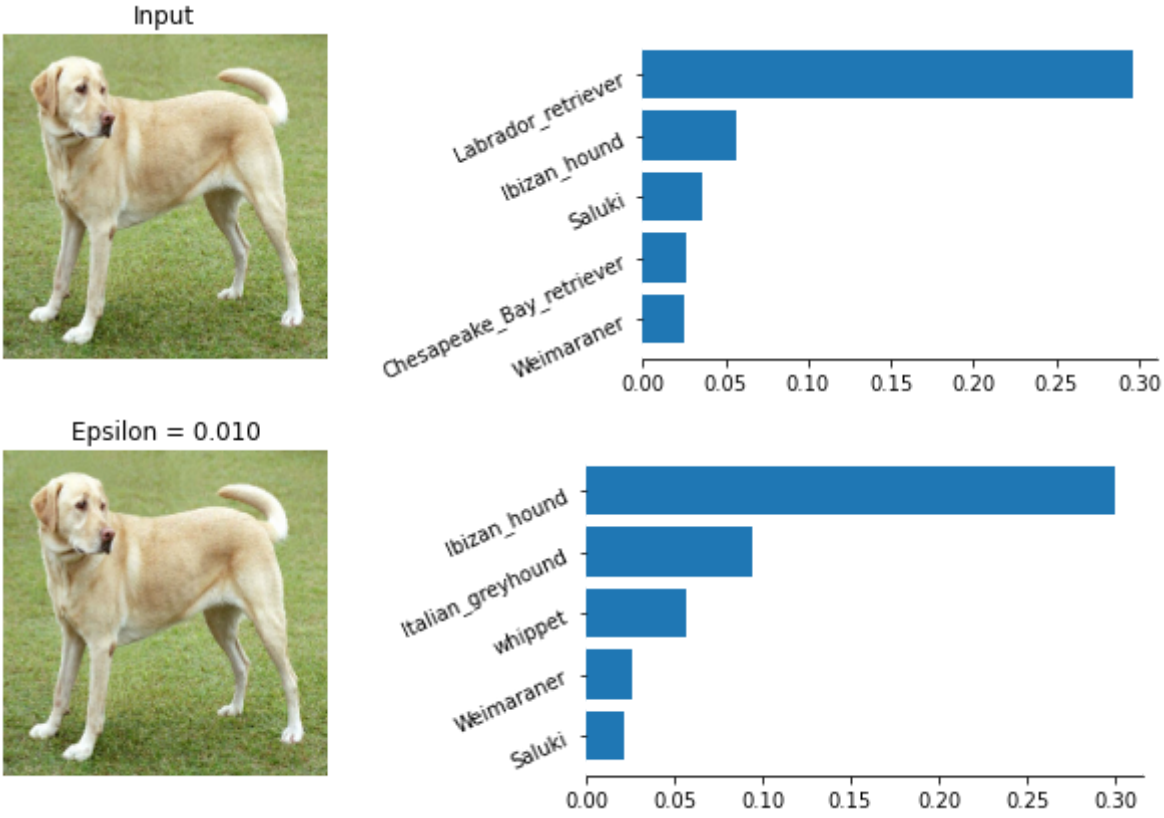
Let's try this out for different values of epsilon and observe the resultant image. You'll notice that as the value of epsilon is increased, it becomes easier to fool the network, however, this comes as a trade-off which results in the perturbations becoming more identifiable.

```
def display_images(image, description):
    #print(1)
    rev_image = reverse_preprocess(image)
    #print(2)
    adv_img, raw_adv_img = sess.run([image, rev_image], {input_image_placeholder:
    #print(3)
    img_probs = sess.run(pretrained_model, {input_image_placeholder: adv_img})
    #print(4)
    top5 = get_imagenet_label(img_probs)
    #print(top5)
    top5 = list(reversed(top5))
    #print(6)
    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()
    #print(top5)
    return top5[4][1]


def display_images111(image, description):

   # _, label, confidence = get_imagenet_label(pretrained_model)
    plt.figure()
    plt.imshow(image[0]*0.5+0.5)
    plt.title('{} \n {} : {:.2f}% Confidence'.format(description,
                                                     label, confidence*100))
    plt.show()
```

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

- adv_x = input_image + ϵ * perturbations

```
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)
    ## End
    adv_x = tf.clip_by_value(adv_x, -1, 1)
    display_images(adv_x, descriptions[i])
```



TASK 2: What do you observe?

As the value of epsilon is increased, it becomes easier to fool the network. The training data or even being able to correctly label the test data does not imply that our models truly understand the classification. At points which do not occur in the classification, the results might go incorrect.

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 ϵ = 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.

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving clean_image.list to clean_image.list
Saving n01443537.JPEG to n01443537.JPEG
Saving n01530575.JPEG to n01530575.JPEG
Saving n01537544.JPEG to n01537544.JPEG
Saving n01664065.JPEG to n01664065.JPEG
Saving n01688243.JPEG to n01688243.JPEG
Saving n01773157.JPEG to n01773157.JPEG
Saving n01817953.JPEG to n01817953.JPEG
Saving n01829413.JPEG to n01829413.JPEG
Saving n01877812.JPEG to n01877812.JPEG
Saving n01955084.JPEG to n01955084.JPEG
Saving n02012849.JPEG to n02012849.JPEG
Saving n02013706.JPEG to n02013706.JPEG
Saving n02018207.JPEG to n02018207.JPEG
Saving n02033041.JPEG to n02033041.JPEG
Saving n02071294.JPEG to n02071294.JPEG
Saving n02086240.JPEG to n02086240.JPEG
Saving n02093256.JPEG to n02093256.JPEG
Saving n02093647.JPEG to n02093647.JPEG
Saving n02095889.JPEG to n02095889.JPEG
Saving n02099267.JPEG to n02099267.JPEG
Saving n02107908.JPEG to n02107908.JPEG
Saving n02109525.JPEG to n02109525.JPEG
Saving n02110185.JPEG to n02110185.JPEG
Saving n02110341.JPEG to n02110341.JPEG
Saving n02129604.JPEG to n02129604.JPEG
Saving n02177972.JPEG to n02177972.JPEG
Saving n02229544.JPEG to n02229544.JPEG
Saving n02259212.JPEG to n02259212.JPEG
Saving n02326432.JPEG to n02326432.JPEG
Saving n02422106.JPEG to n02422106.JPEG
Saving n02443114.JPEG to n02443114.JPEG
Saving n02444819.JPEG to n02444819.JPEG
Saving n02480495.JPEG to n02480495.JPEG
Saving n02489166.JPEG to n02489166.JPEG
Saving n02655020.JPEG to n02655020.JPEG
Saving n02667093.JPEG to n02667093.JPEG
Saving n02704792.JPEG to n02704792.JPEG
Saving n02708093.JPEG to n02708093.JPEG
Saving n02769748.JPEG to n02769748.JPEG
Saving n02777292.JPEG to n02777292.JPEG
Saving n02790996.JPEG to n02790996.JPEG
Saving n02791270.JPEG to n02791270.JPEG
Saving n02794156.JPEG to n02794156.JPEG
```

```
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(name=
```

```
User uploaded file "clean_image.list" with length 1890 bytes
User uploaded file "n01443537.JPEG" with length 11803 bytes
User uploaded file "n01530575.JPEG" with length 8979 bytes
User uploaded file "n01537544.JPEG" with length 9147 bytes
User uploaded file "n01664065.JPEG" with length 11630 bytes
User uploaded file "n01688243.JPEG" with length 12127 bytes
User uploaded file "n01773157.JPEG" with length 7237 bytes
User uploaded file "n01817953.JPEG" with length 8553 bytes
User uploaded file "n01829413.JPEG" with length 12141 bytes
User uploaded file "n01877812.JPEG" with length 13162 bytes
User uploaded file "n01955084.JPEG" with length 12162 bytes
User uploaded file "n02012849.JPEG" with length 13722 bytes
User uploaded file "n02013706.JPEG" with length 16647 bytes
User uploaded file "n02018207.JPEG" with length 12948 bytes
User uploaded file "n02033041.JPEG" with length 8621 bytes
User uploaded file "n02071294.JPEG" with length 12583 bytes
User uploaded file "n02086240.JPEG" with length 13827 bytes
User uploaded file "n02093256.JPEG" with length 8458 bytes
User uploaded file "n02093647.JPEG" with length 7960 bytes
User uploaded file "n02095889.JPEG" with length 15013 bytes
User uploaded file "n02099267.JPEG" with length 11302 bytes
User uploaded file "n02107908.JPEG" with length 8581 bytes
User uploaded file "n02109525.JPEG" with length 11079 bytes
User uploaded file "n02110185.JPEG" with length 9150 bytes
User uploaded file "n02110341.JPEG" with length 12708 bytes
User uploaded file "n02129604.JPEG" with length 13727 bytes
User uploaded file "n02177972.JPEG" with length 7142 bytes
User uploaded file "n02229544.JPEG" with length 8126 bytes
User uploaded file "n02259212.JPEG" with length 8659 bytes
User uploaded file "n02326432.JPEG" with length 8704 bytes
User uploaded file "n02422106.JPEG" with length 9287 bytes
User uploaded file "n02443114.JPEG" with length 11499 bytes
User uploaded file "n02444819.JPEG" with length 16011 bytes
User uploaded file "n02480495.JPEG" with length 12063 bytes
User uploaded file "n02489166.JPEG" with length 14106 bytes
User uploaded file "n02655020.JPEG" with length 12928 bytes
User uploaded file "n02667093.JPEG" with length 13895 bytes
User uploaded file "n02704792.JPEG" with length 11838 bytes
User uploaded file "n02708093.JPEG" with length 5991 bytes
User uploaded file "n02769748.JPEG" with length 9555 bytes
User uploaded file "n02777292.JPEG" with length 8160 bytes
User uploaded file "n02790996.JPEG" with length 10585 bytes
User uploaded file "n02791270.JPEG" with length 10137 bytes
User uploaded file "n02794156.JPEG" with length 9956 bytes
User uploaded file "n02814860.JPEG" with length 2990 bytes
User uploaded file "n02815834.JPEG" with length 6076 bytes
User uploaded file "n02966687.JPEG" with length 15389 bytes
User uploaded file "n03000134.JPEG" with length 15990 bytes
User uploaded file "n03016953.JPEG" with length 11089 bytes
User uploaded file "n03042490.JPEG" with length 11620 bytes
User uploaded file "n03063599.JPEG" with length 10660 bytes
User uploaded file "n03109150.JPEG" with length 7110 bytes
User uploaded file "n03131574.JPEG" with length 11386 bytes
User uploaded file "n03160309.JPEG" with length 9021 bytes
User uploaded file "n03197337.JPEG" with length 7300 bytes
User uploaded file "n03201208.JPEG" with length 11125 bytes
User uploaded file "n03207941.JPEG" with length 13699 bytes
User uploaded file "n03216828.JPEG" with length 13787 bytes
User uploaded file "n03379051.JPEG" with length 15406 bytes
User uploaded file "n03384352.JPEG" with length 10485 bytes
User uploaded file "n03393912.JPEG" with length 10013 bytes
```

```
User uploaded file "n03394916.JPEG" with length 14570 bytes
User uploaded file "n03424325.JPEG" with length 11876 bytes
User uploaded file "n03467068.JPEG" with length 14327 bytes
User uploaded file "n03485407.JPEG" with length 8270 bytes
User uploaded file "n03584254.JPEG" with length 10013 bytes
User uploaded file "n03673027.JPEG" with length 8697 bytes
User uploaded file "n03721384.JPEG" with length 10918 bytes
User uploaded file "n03777754.JPEG" with length 6444 bytes
User uploaded file "n03794056.JPEG" with length 4505 bytes
User uploaded file "n03873416.JPEG" with length 15225 bytes
User uploaded file "n03956157.JPEG" with length 6300 bytes
User uploaded file "n03977966.JPEG" with length 12463 bytes
User uploaded file "n04023962.JPEG" with length 8842 bytes
User uploaded file "n04026417.JPEG" with length 11988 bytes
User uploaded file "n04049303.JPEG" with length 10569 bytes
User uploaded file "n04065272.JPEG" with length 12266 bytes
User uploaded file "n04131690.JPEG" with length 8220 bytes
User uploaded file "n04153751.JPEG" with length 8178 bytes
User uploaded file "n04251144.JPEG" with length 9642 bytes
User uploaded file "n04311174.JPEG" with length 9277 bytes
User uploaded file "n04376876.JPEG" with length 4310 bytes
User uploaded file "n04398044.JPEG" with length 9881 bytes
User uploaded file "n04493381.JPEG" with length 12143 bytes
User uploaded file "n04525038.JPEG" with length 9326 bytes
User uploaded file "n04536866.JPEG" with length 7002 bytes
User uploaded file "n04548280.JPEG" with length 8099 bytes
User uploaded file "n04590129.JPEG" with length 10390 bytes
User uploaded file "n04597913.JPEG" with length 16554 bytes
User uploaded file "n04606251.JPEG" with length 10186 bytes
User uploaded file "n07583066.JPEG" with length 14864 bytes
```

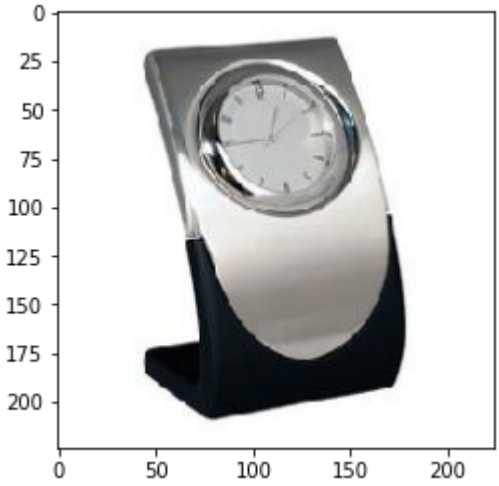
```
#f = open('clean_image.list', 'rb')
with open('clean_image.list', encoding="utf8", errors='ignore') as f:
  labels = [line.rstrip("\n") for line in f.readlines()]
f.close()
for i,x in enumerate(labels):
    if ' ' in x:
        labels[i] = x[:x.index(' ')]
labels
```

```
['n02708093.JPEG',
 'n03000134.JPEG',
 'n03384352.JPEG',
 'n03777754.JPEG',
 'n03721384.JPEG',
 'n03424325.JPEG',
 'n03673027.JPEG',
 'n02229544.JPEG',
 'n07695742.JPEG',
 'n02018207.JPEG',
 'n02107908.JPEG',
 'n04026417.JPEG',
 'n02444819.JPEG',
 'n02259212.JPEG',
 'n02480495.JPEG',
 'n02095889.JPEG',
 'n03216828.JPEG',
 'n01817953.JPEG',
 'n02422106.JPEG',
 'n04376876.JPEG',
 'n02099267.JPEG',
 'n04398044.JPEG',
 'n04023962.JPEG',
 'n02093256.JPEG',
 'n03109150.JPEG',
 'n12267677.JPEG',
 'n02667093.JPEG',
 'n03956157.JPEG',
 'n02033041.JPEG',
 'n02071294.JPEG',
 'n02704792.JPEG',
 'n02013706.JPEG',
 'n02177972.JPEG',
 'n03042490.JPEG',
 'n01443537.JPEG',
 'n02093647.JPEG',
 'n04536866.JPEG',
 'n07717556.JPEG',
 'n03016953.JPEG',
 'n04493381.JPEG',
 'n04590129.JPEG',
 'n02769748.JPEG',
 'n03063599.JPEG',
 'n04065272.JPEG',
 'n02012849.JPEG',
 'n02655020.JPEG',
 'n04548280.JPEG',
 'n02129604.JPEG',
 'n03393912.JPEG',
 'n02110341.JPEG',
 'n02791270.JPEG',
 'n04251144.JPEG',
 'n02443114.JPEG',
 'n12144580.JPEG',
 'n02966687.JPEG',
 'n03584254.JPEG',
 'n02110185.JPEG',
 'n02790996.JPEG',
 'n02815834.JPEG',
 'n07831146.JPEG',
 'n04311174.JPEG',
```

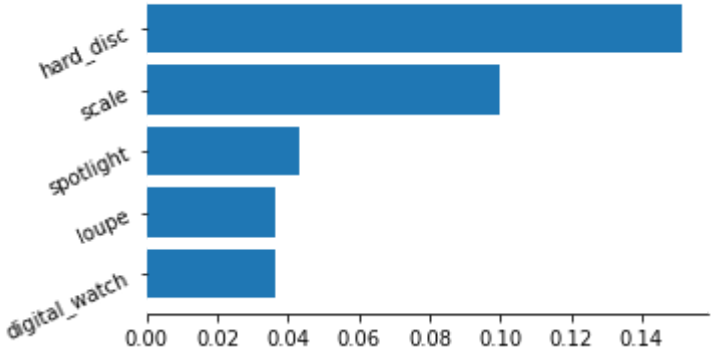
```
        'n01955084.JPEG',
        'n09332890.JPEG',
        'n03197337.JPEG',
        'n03467068.JPEG',
        'n03201208.JPEG',
        'n03485407.JPEG',
        'n03873416.JPEG',
        'n03131574.JPEG',
        'n07754684.JPEG',
        'n02086240.JPEG',
        'n07836838.JPEG',
        'n03160309.JPEG',
        'n09399592.JPEG',
        'n02777292.JPEG',
        'n02326432.JPEG',
        'n01664065.JPEG',
        'n01537544.JPEG',
        'n07583066.JPEG',
        'n02814860.JPEG',
        'n04153751.JPEG',
        'n03379051.JPEG',
```

```
from PIL import Image
from io import BytesIO
pred = []
true = []
for label in labels:
  for fn in uploaded.keys():
    if fn in label:
      im = Image.open(BytesIO(uploaded[fn]))
      plt.imshow(im)

      input_image = preprocess(im)
      reversed_image = reverse_preprocess(input_image)

      preprocessed_img, reversed_img = sess.run([input_image, reversed_image])
      image_probs = sess.run([pretrained_model], {input_image_placeholder:preproce
      tp = get_imagenet_label(image_probs[0])
      print("True label: ", tp[0][1])
      true.append(tp[0][1])
      adv_x = input_image + (0.01*perturbations)
      adv_x = tf.clip_by_value(adv_x, -1, 1)
      p = display_images(adv_x, label+' Epsilon = 0.01')
      print("Predicted label: ", p, "\n")
      pred.append(p)
accuracy = len([true[i] for i in range(0, len(true)) if true[i] == pred[i]]) / len
print("Rate of misclassification: ", 100 - accuracy)
```

```
True label:  analog_clock
```

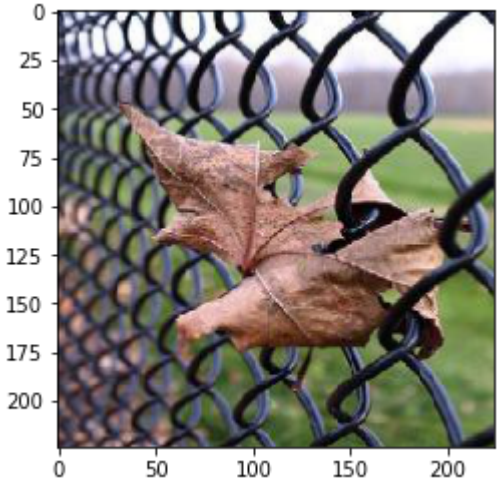


n02708093.JPEG Epsilon = 0.01

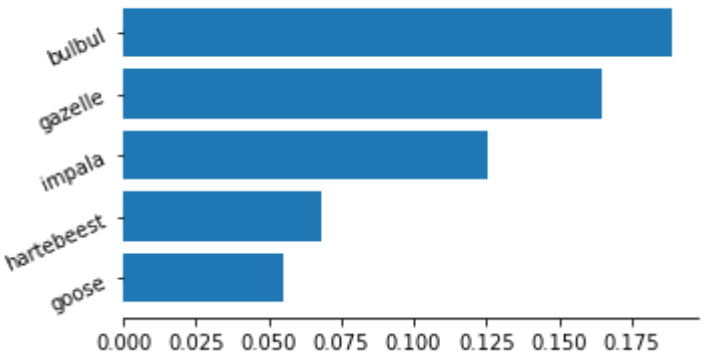



```
Predicted label:  hard_disc
```

```
True label:  chainlink_fence
```

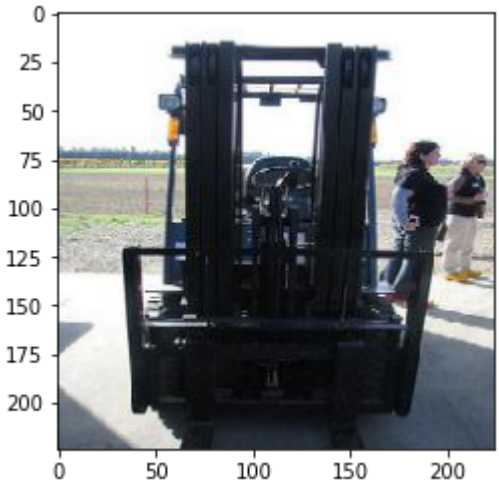


n03000134.JPEG Epsilon = 0.01

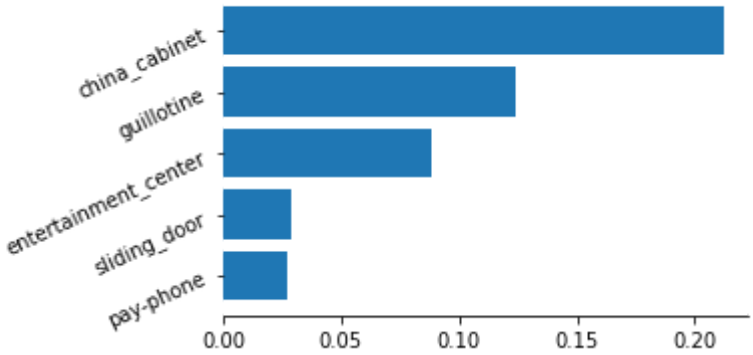



```
Predicted label:  bulbul
```

```
True label:  forklift
```

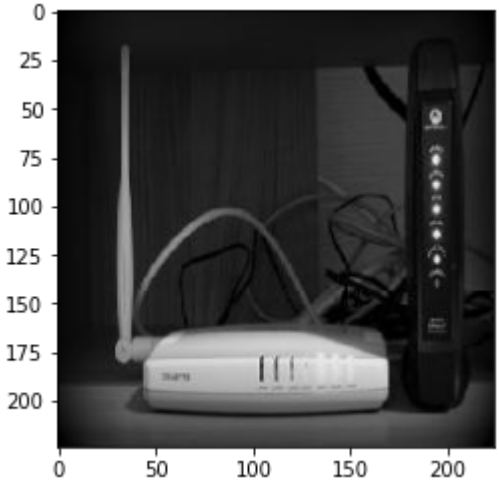


n03384352.JPEG Epsilon = 0.01

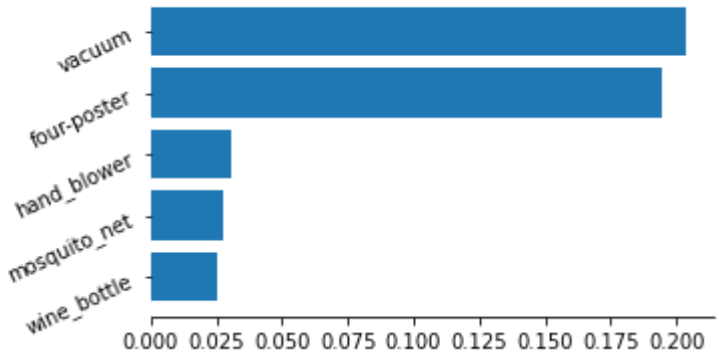



```
Predicted label:  china_cabinet
```

```
True label:  modem
```

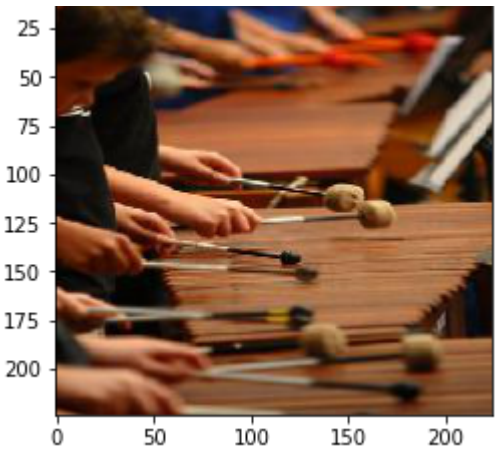


n03777754.JPEG Epsilon = 0.01

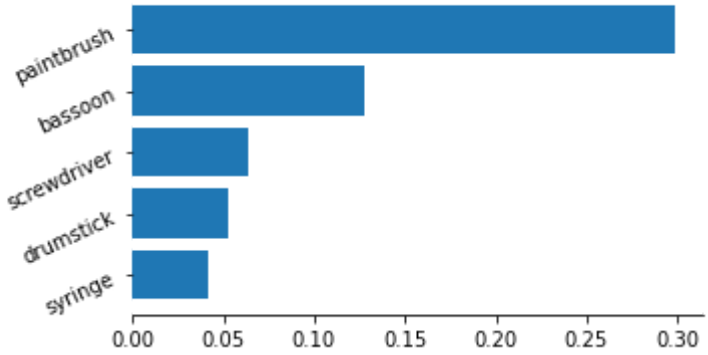



```
Predicted label:  vacuum
```

```
True label:  marimba
```

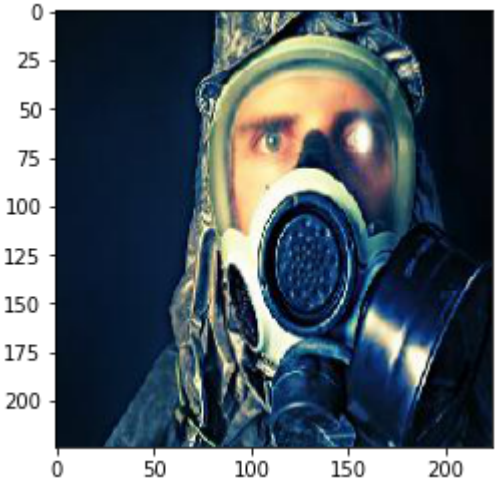


n03721384.JPEG Epsilon = 0.01

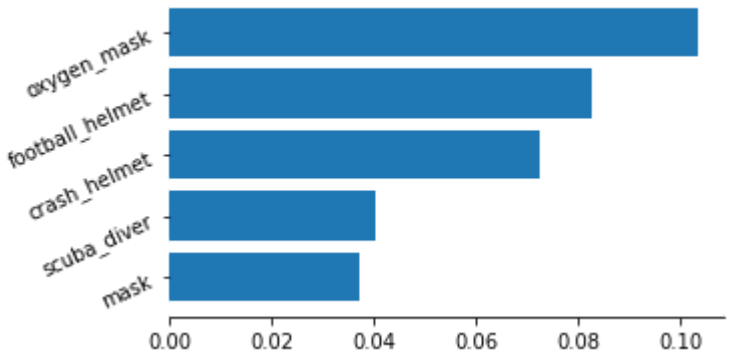



```
Predicted label:  paintbrush
```

```
True label:  gasmask
```

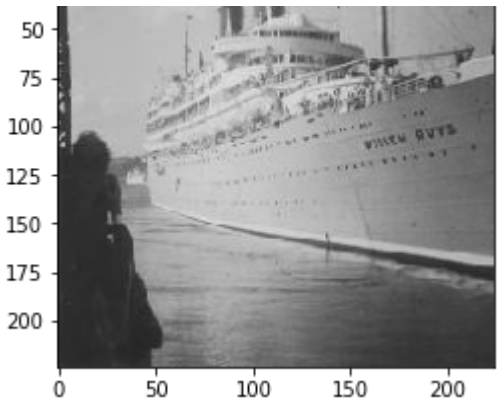


n03424325.JPEG Epsilon = 0.01

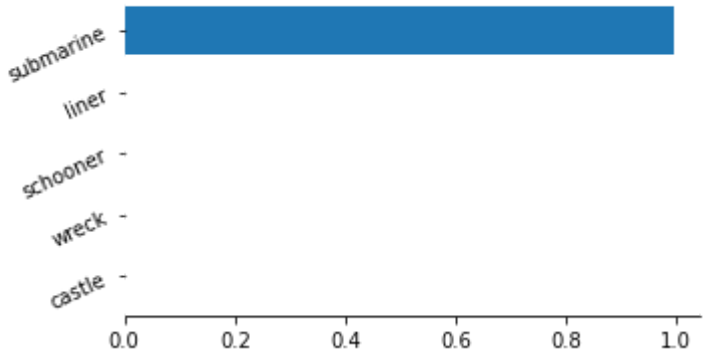



```
Predicted label:  oxygen_mask
```

```
True label:  liner
```

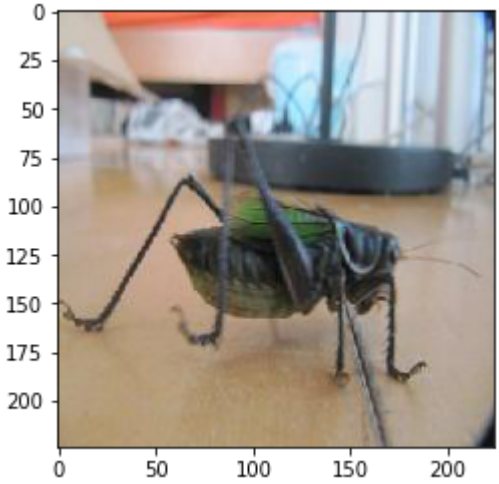


n03673027.JPEG Epsilon = 0.01

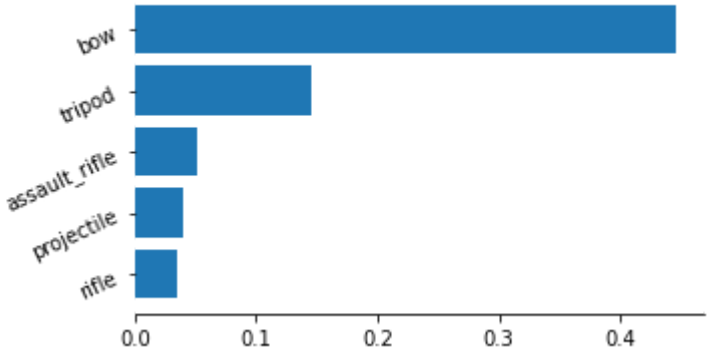



```
Predicted label:  submarine
```

```
True label:  cricket
```

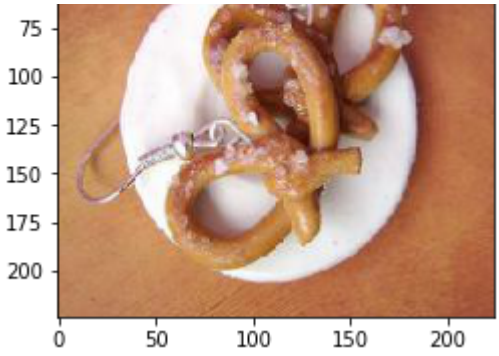


n02229544.JPEG Epsilon = 0.01

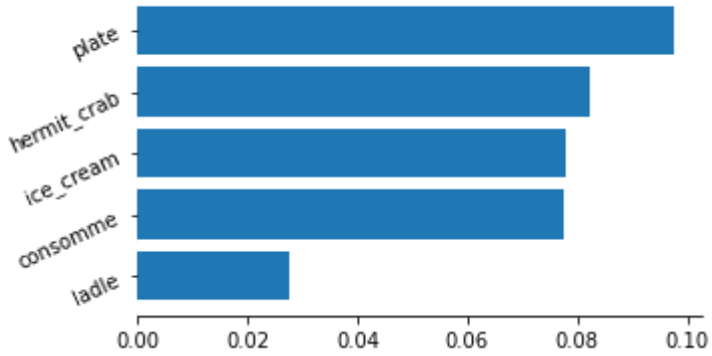



```
Predicted label:  bow
```

```
True label:  pretzel
```

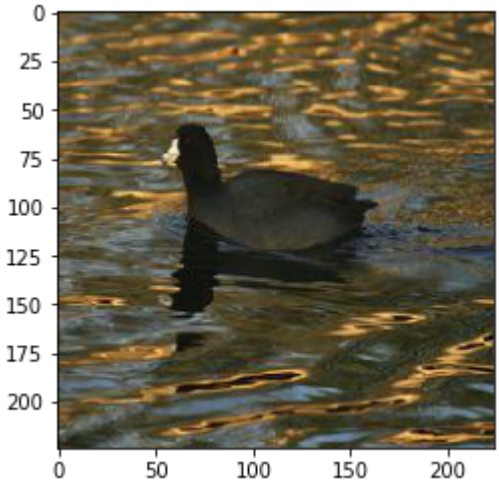


n07695742.JPEG Epsilon = 0.01

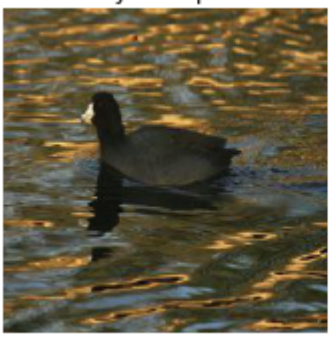


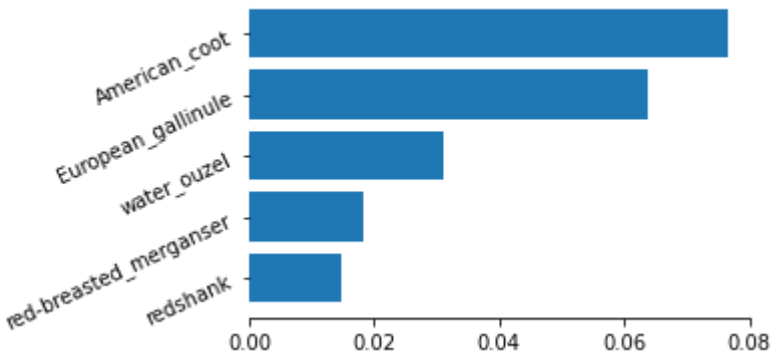

```
Predicted label:  plate
```

```
True label:  American_coot
```



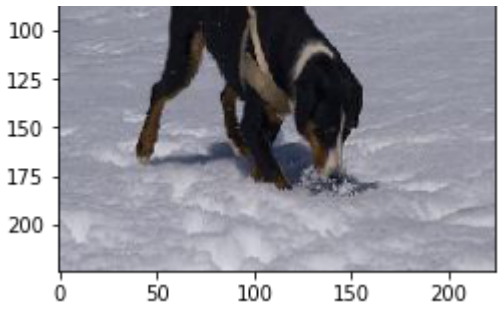
n02018207.JPEG Epsilon = 0.01



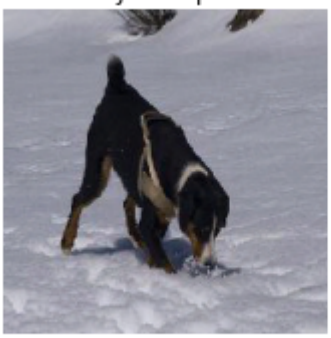


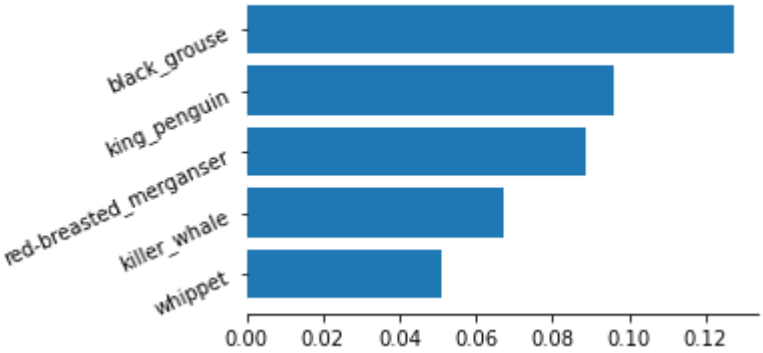

```
Predicted label:  American_coot
```

```
True label:  Rottweiler
```



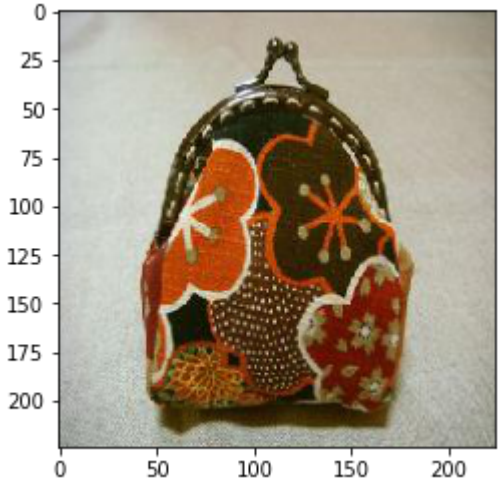
n02107908.JPEG Epsilon = 0.01



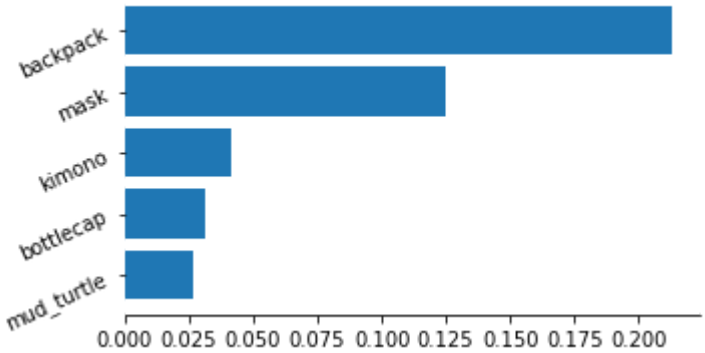



```
Predicted label:  black_grouse
```

```
True label:  purse
```



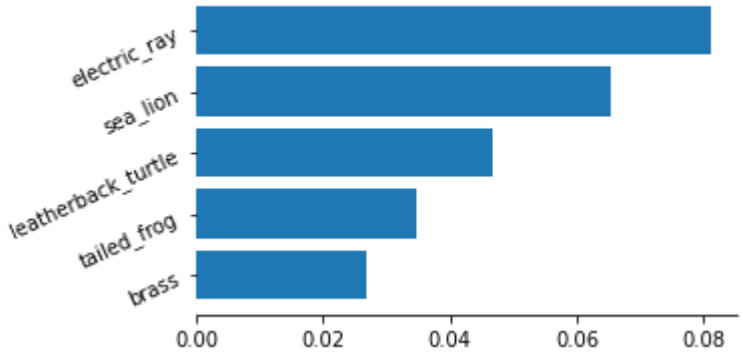
n04026417.JPEG Epsilon = 0.01




```
Predicted label:  backpack
```

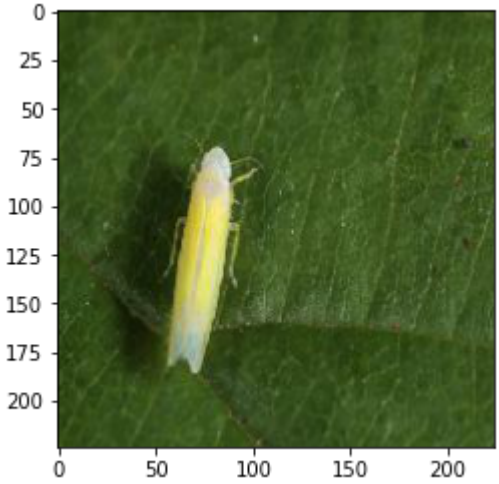
```
True label:  otter
```

n02444819.JPEG Epsilon = 0.01

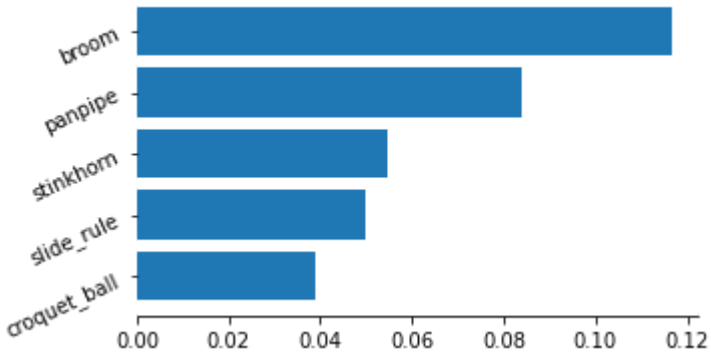


Predicted label: electric_ray

True label: leafhopper

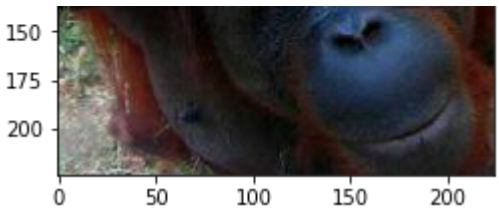


n02259212.JPEG Epsilon = 0.01

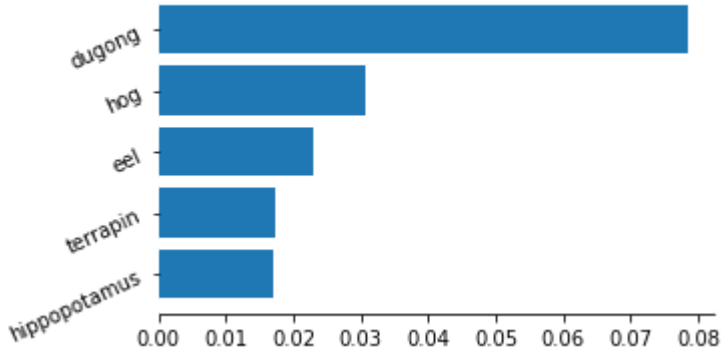


Predicted label: broom

True label: orangutan

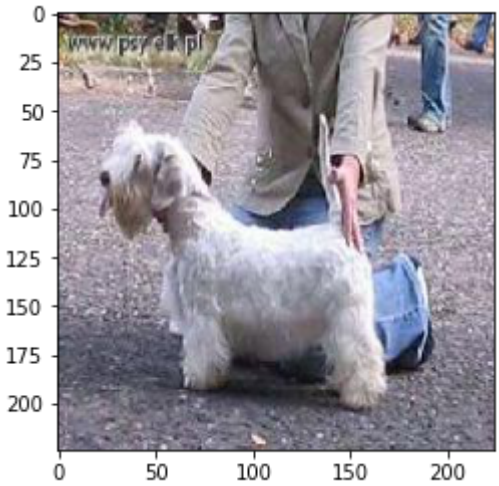


n02480495.JPEG Epsilon = 0.01

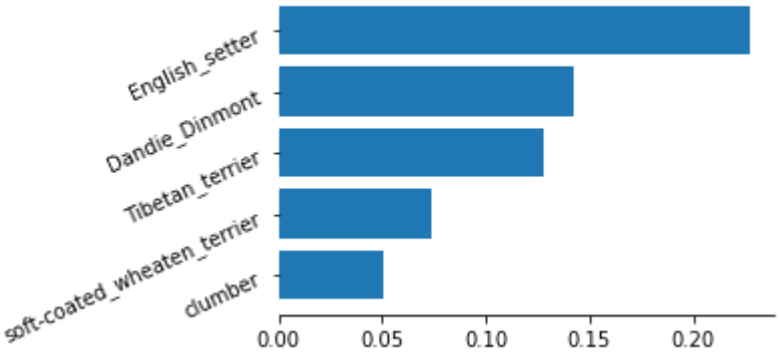



```
Predicted label:  dugong
```

```
True label:  Sealyham_terrier
```

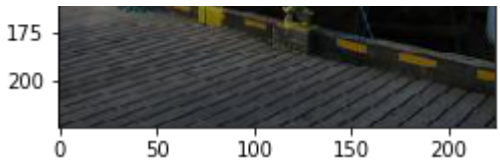


n02095889.JPEG Epsilon = 0.01

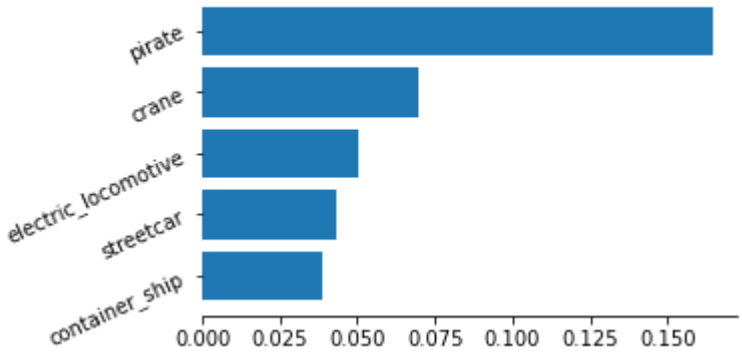



```
Predicted label:  English_setter
```

```
True label:  dock
```

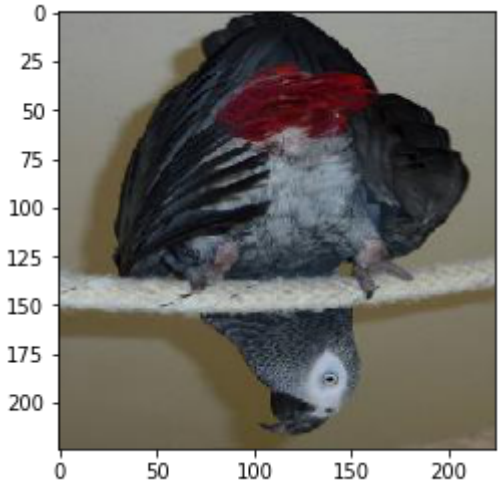


n03216828.JPEG Epsilon = 0.01

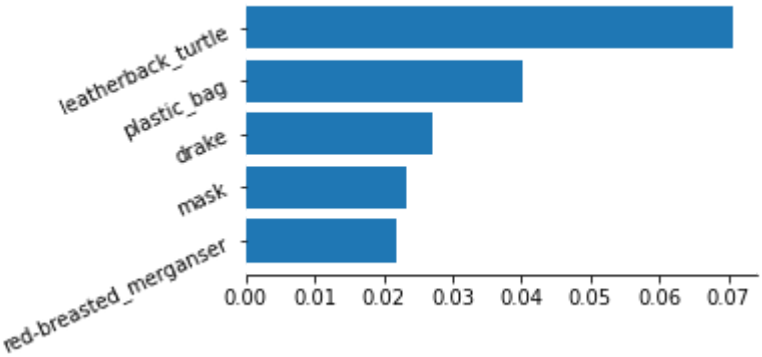



```
Predicted label:  pirate
```

```
True label:  African_grey
```



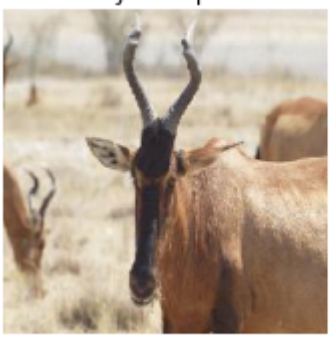
n01817953.JPEG Epsilon = 0.01

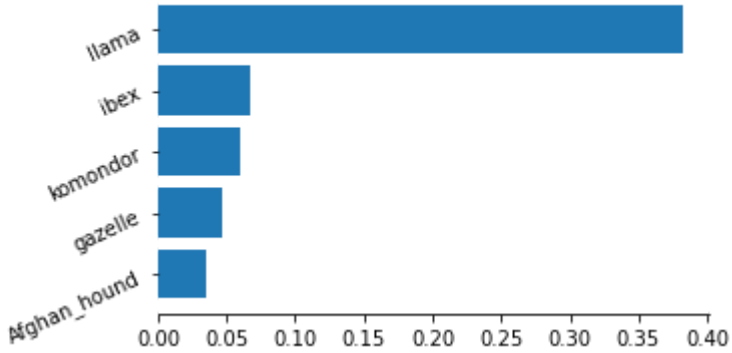



```
Predicted label:  leatherback_turtle
```

```
True label:  hartebeest
```

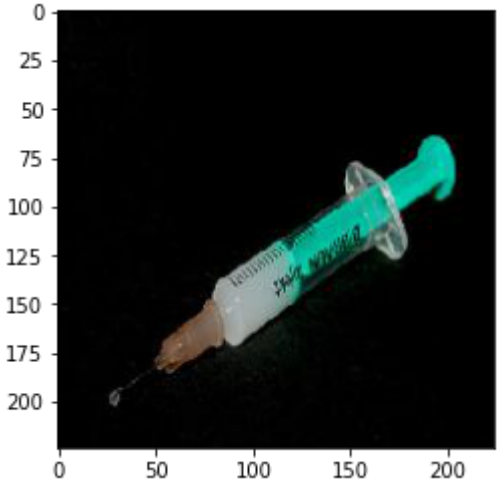
n02422106.JPEG Epsilon = 0.01



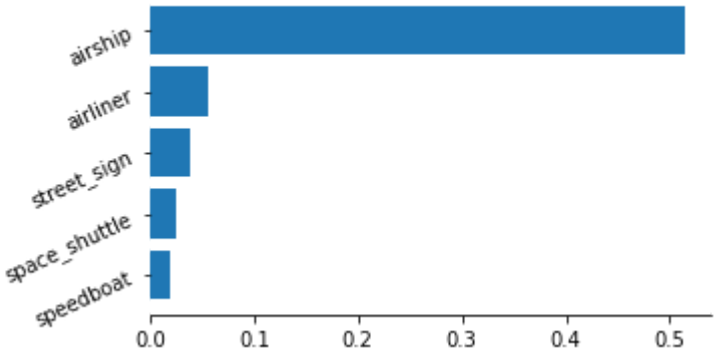



```
Predicted label:  llama
```

```
True label:  syringe
```

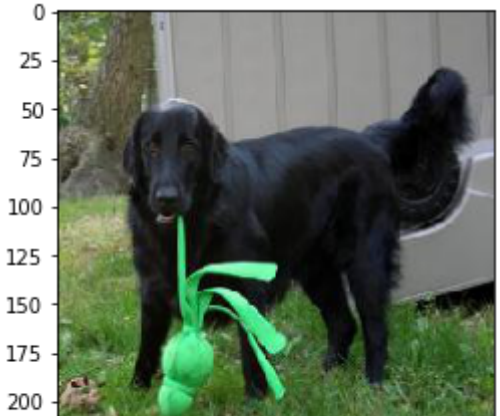


n04376876.JPEG Epsilon = 0.01

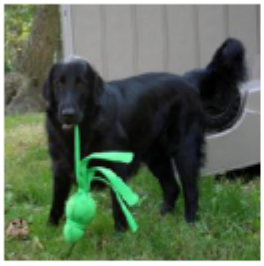


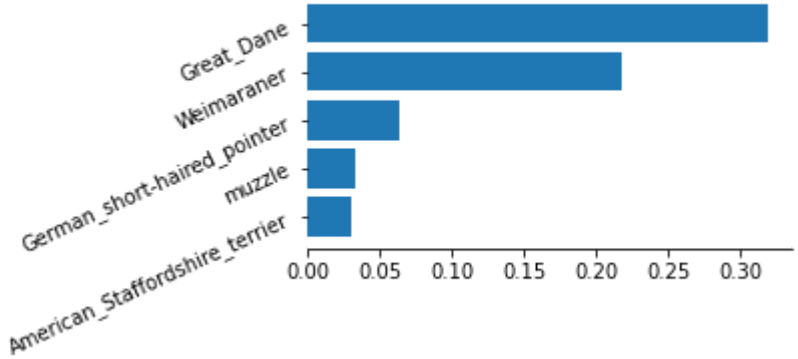

```
Predicted label:  airship
```

```
True label:  flat-coated_retriever
```



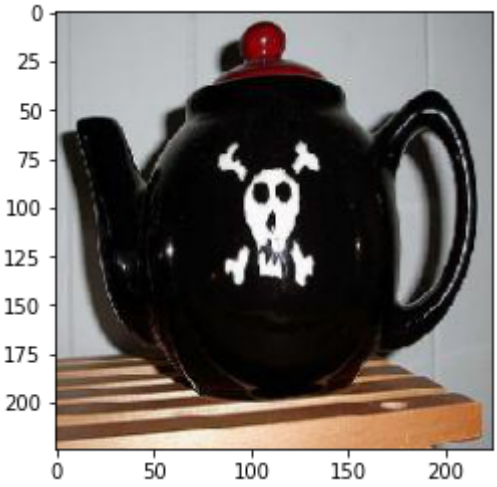
n02099267.JPEG Epsilon = 0.01



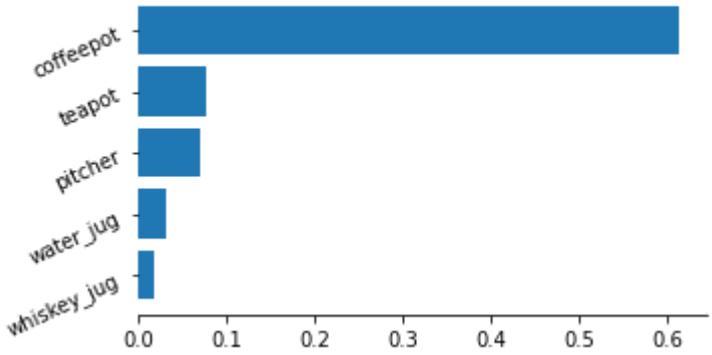



```
Predicted label:  Great_Dane
```

```
True label:  teapot
```

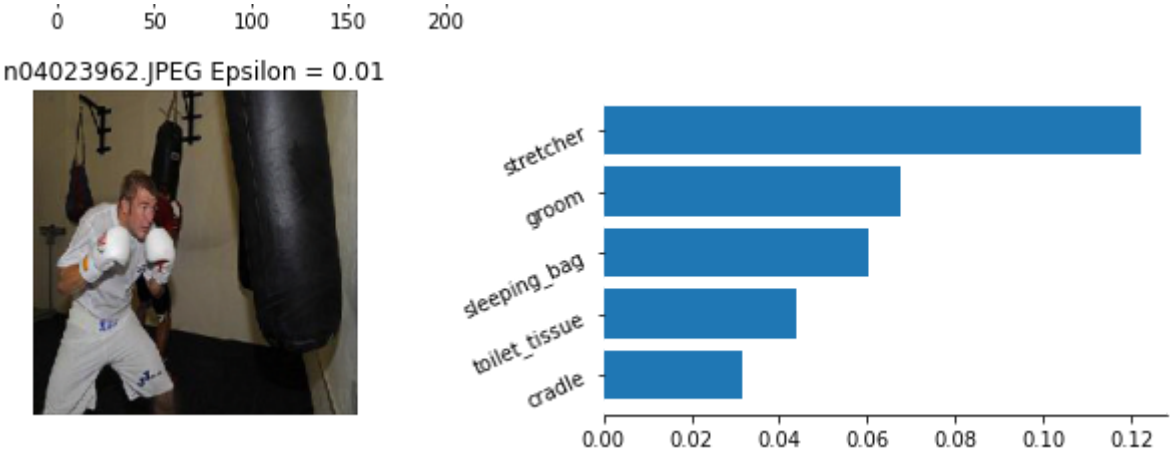


n04398044.JPEG Epsilon = 0.01



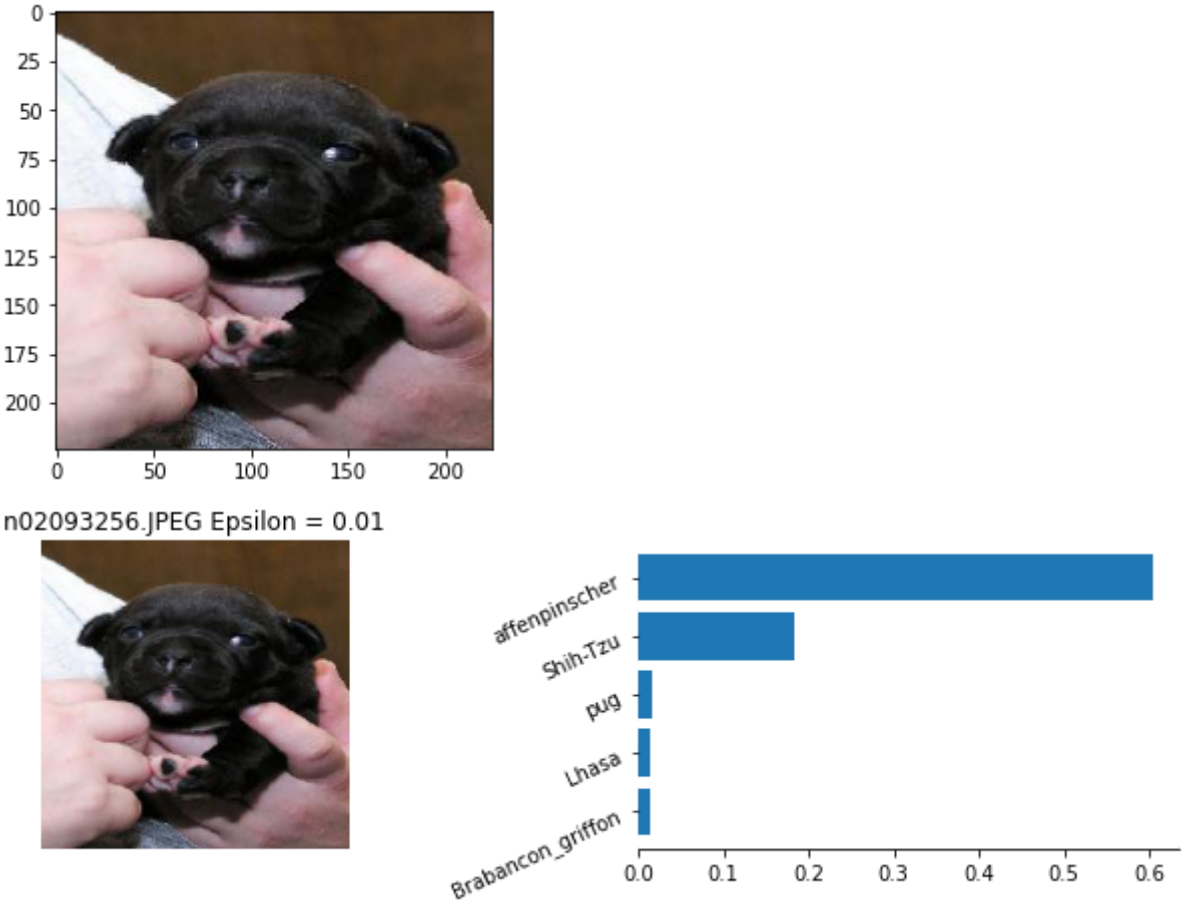

```
Predicted label:  coffeepot
```

```
True label:  punching_bag
```



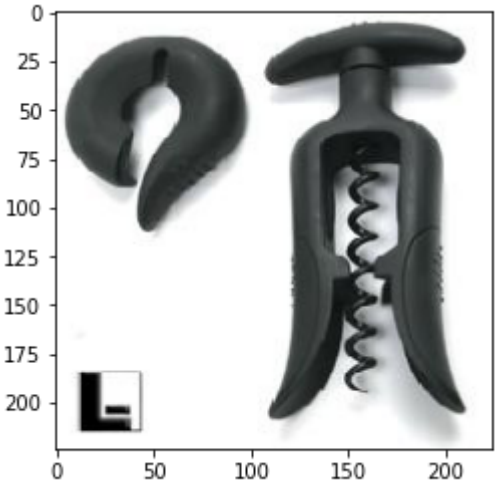

```
Predicted label:  stretcher
```

```
True label:  Staffordshire_bullterrier
```

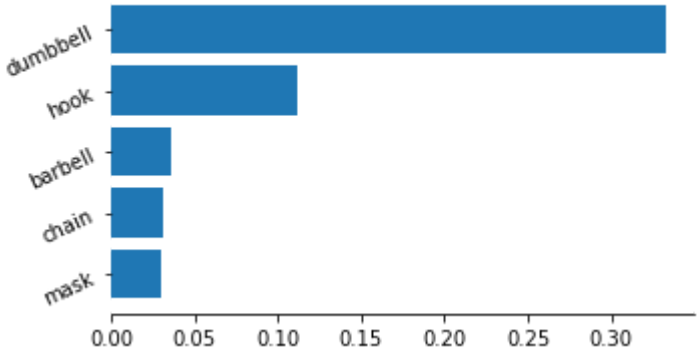



```
Predicted label:  affenpinscher
```

```
True label:  corkscrew
```

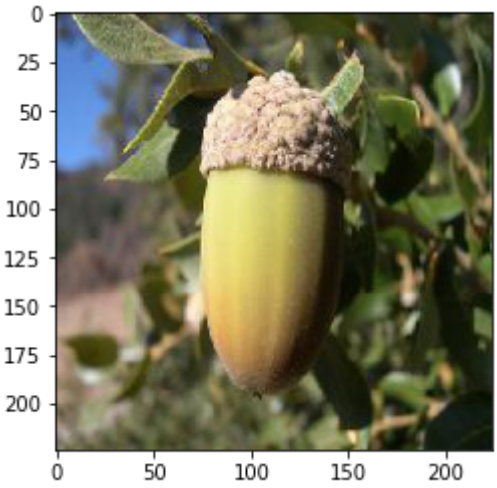


n03109150.JPEG Epsilon = 0.01

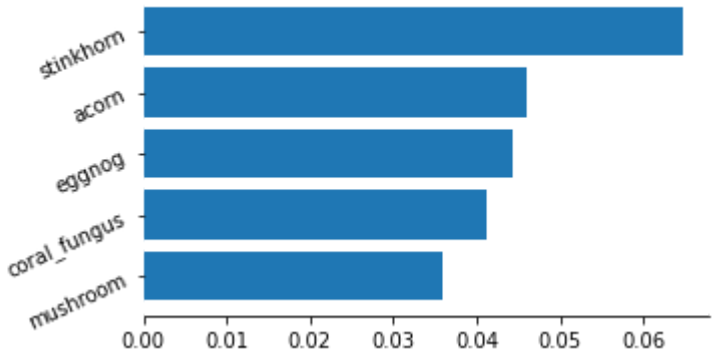



```
Predicted label:  dumbbell
```

```
True label:  acorn
```



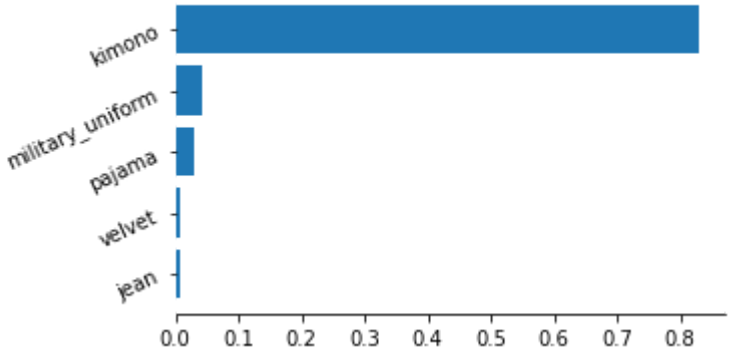
n12267677.JPEG Epsilon = 0.01



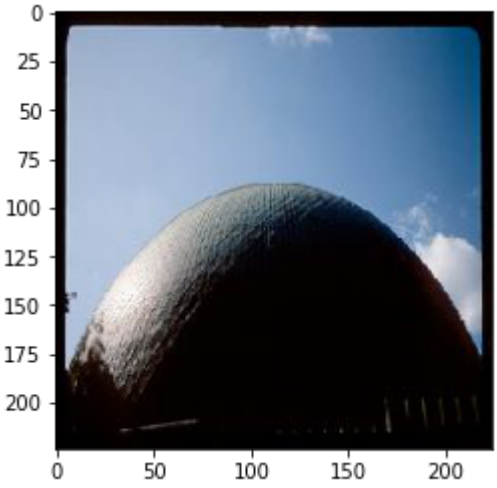

```
Predicted label:  stinkhorn
```

```
True label:  abaya
```

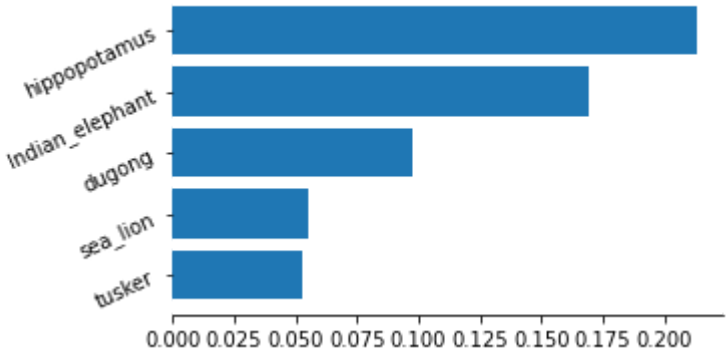
n02667093.JPEG Epsilon = 0.01



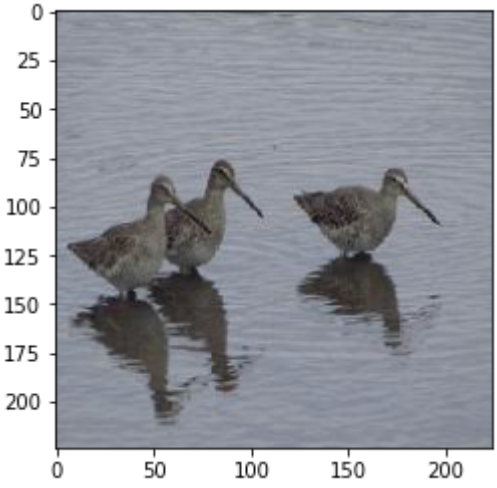

```
Predicted label:  kimono

True label:  planetarium
```



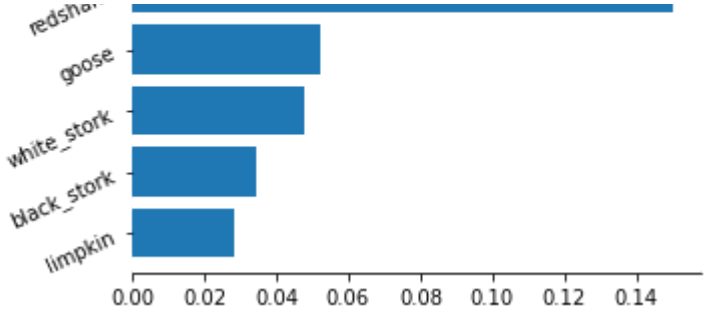
n03956157.JPEG Epsilon = 0.01




```
Predicted label:  hippopotamus

True label:  dowitcher
```

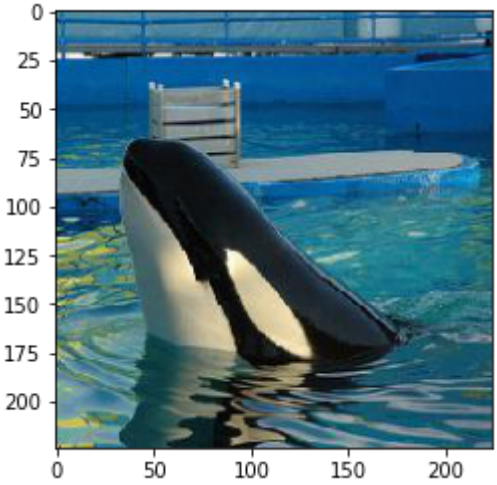


n02033041.JPEG Epsilon = 0.01

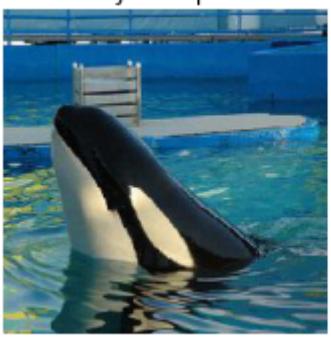


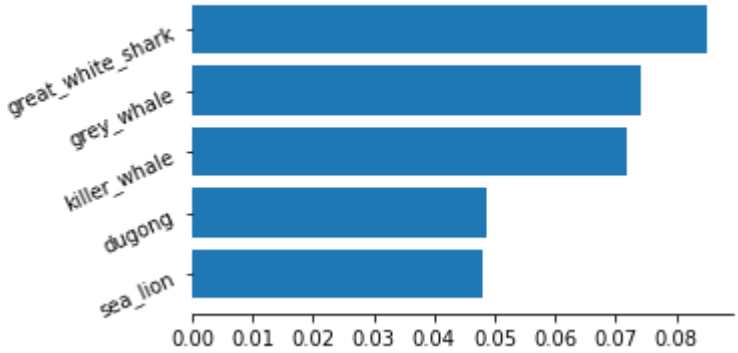

```
Predicted label:  redshank
```

```
True label:  killer_whale
```



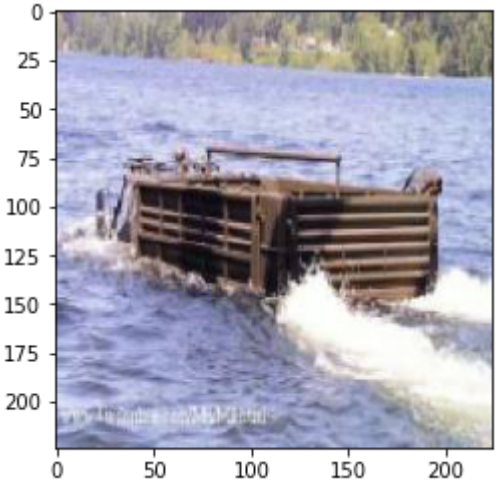
n02071294.JPEG Epsilon = 0.01



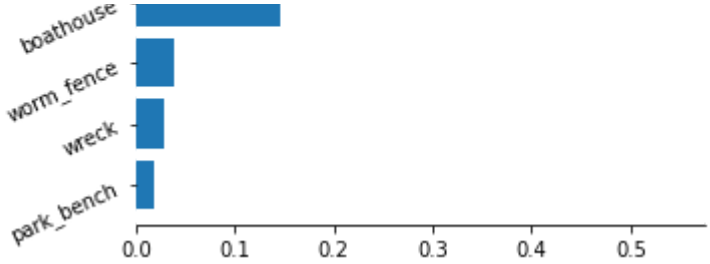



```
Predicted label:  great_white_shark
```

```
True label:  amphibian
```

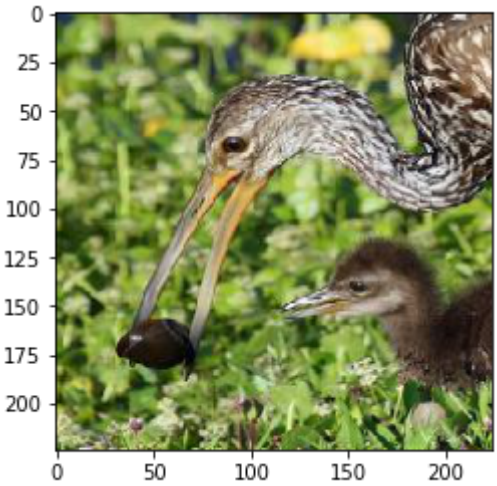


n02704792.JPEG Epsilon = 0.01

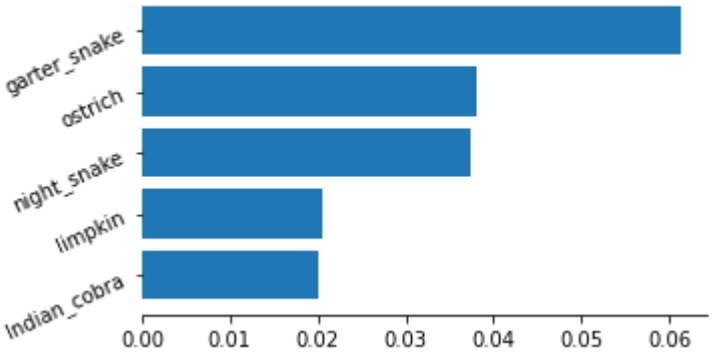



```
Predicted label:  lakeside
```

```
True label:  limpkin
```

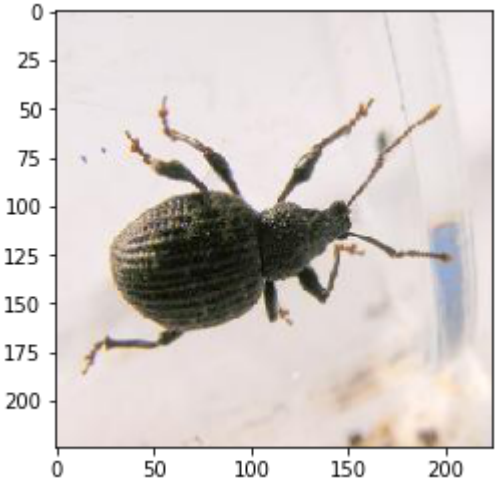


n02013706.JPEG Epsilon = 0.01

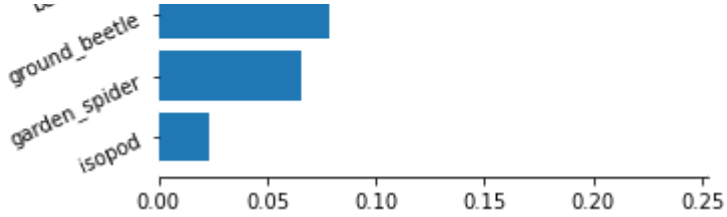



```
Predicted label:  garter_snake
```

```
True label:  weevil
```

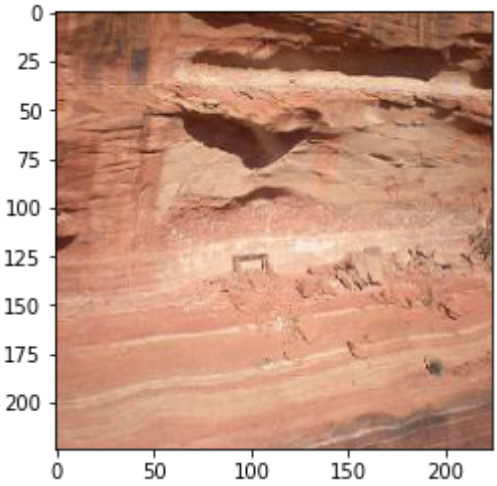


n02177972.JPEG Epsilon = 0.01

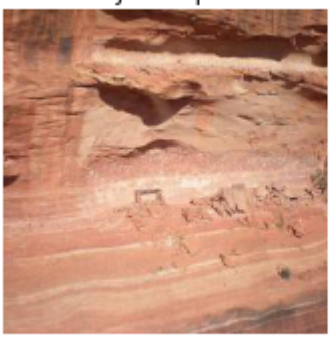


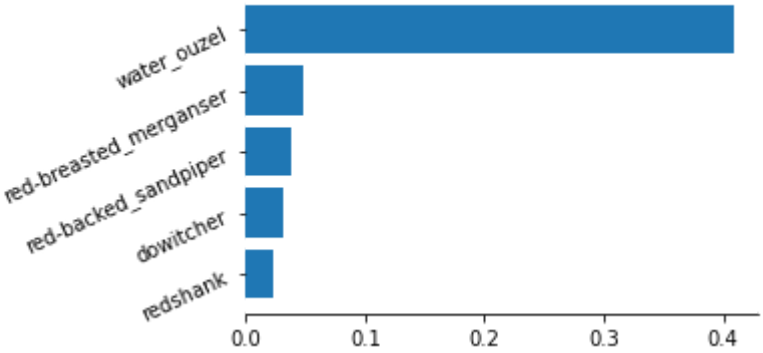

```
Predicted label:  tick
```

```
True label:  cliff_dwelling
```



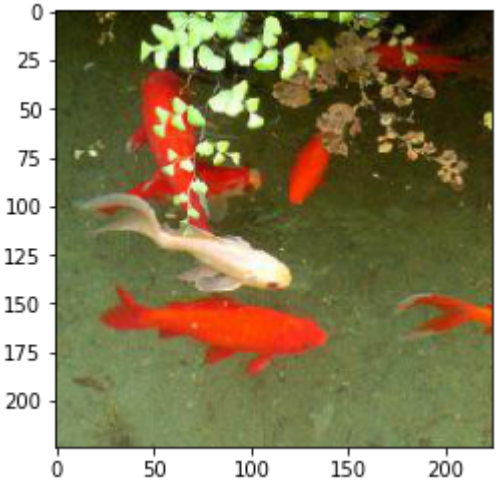
n03042490.JPEG Epsilon = 0.01



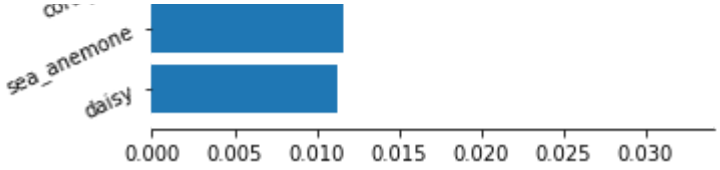



```
Predicted label:  water_ouzel
```

```
True label:  goldfish
```

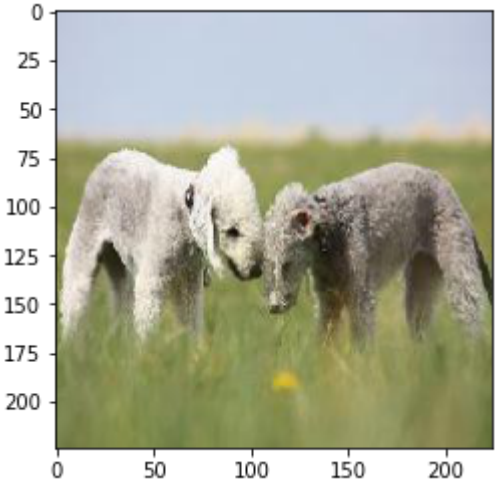


n01443537.JPEG Epsilon = 0.01

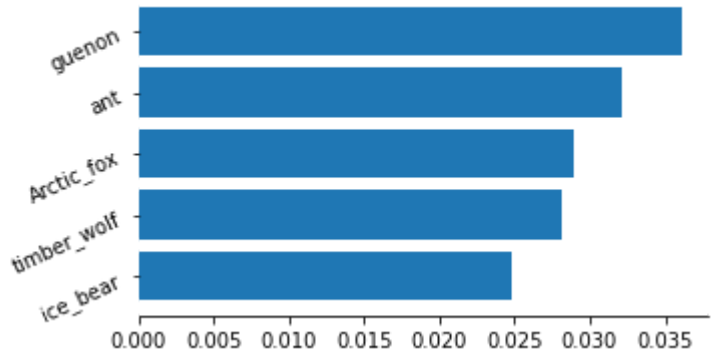



```
Predicted label:  hummingbird
```

```
True label:  Bedlington_terrier
```

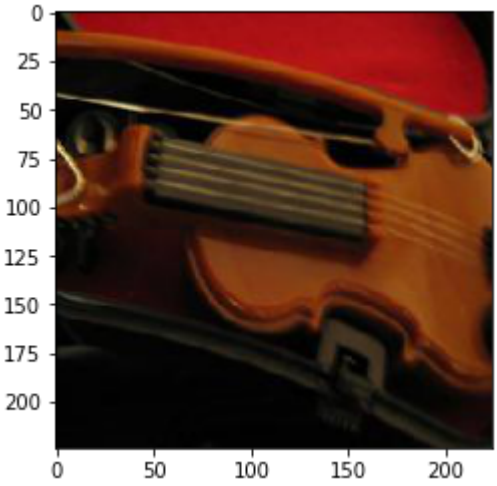


n02093647.JPEG Epsilon = 0.01

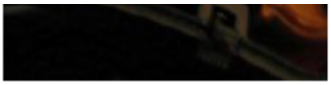


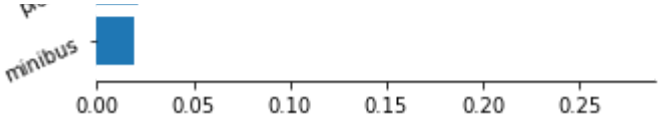

```
Predicted label:  guenon
```

```
True label:  violin
```



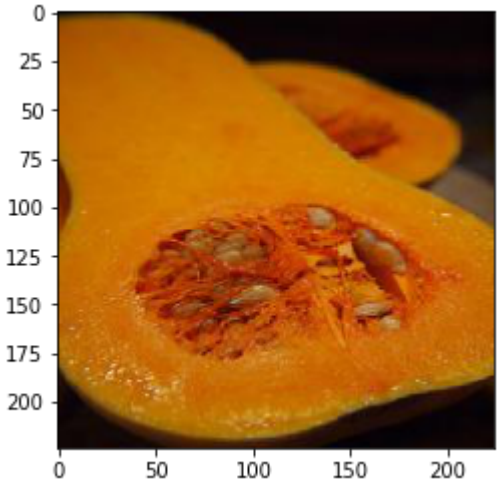
n04536866.JPEG Epsilon = 0.01



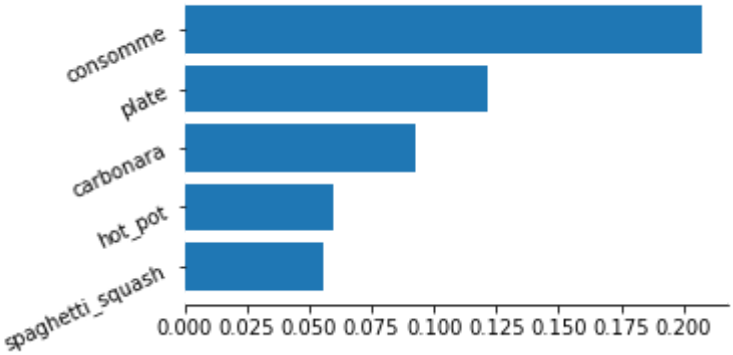



```
Predicted label:  revolver
```

```
True label:  butternut_squash
```



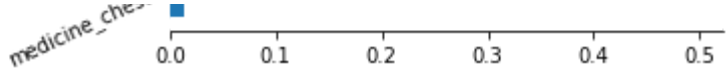
n07717556.JPEG Epsilon = 0.01




```
Predicted label:  consomme
```

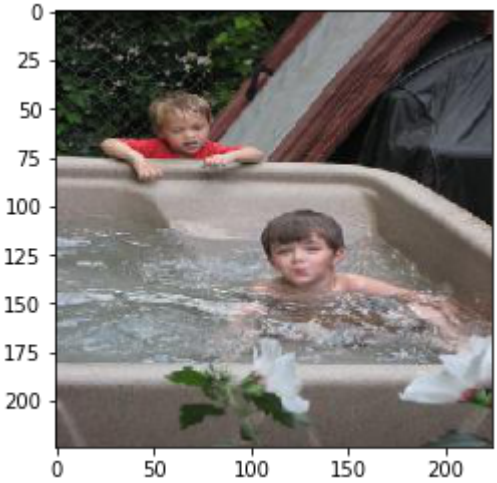
```
True label:  chiffonier
```

n03016953.JPEG Epsilon = 0.01

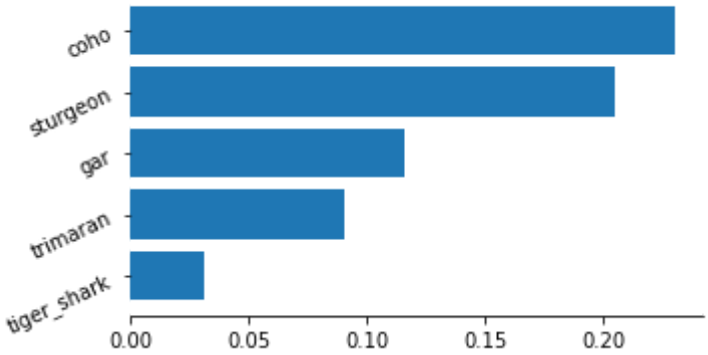



```
Predicted label:  chiffonier
```

```
True label:  tub
```

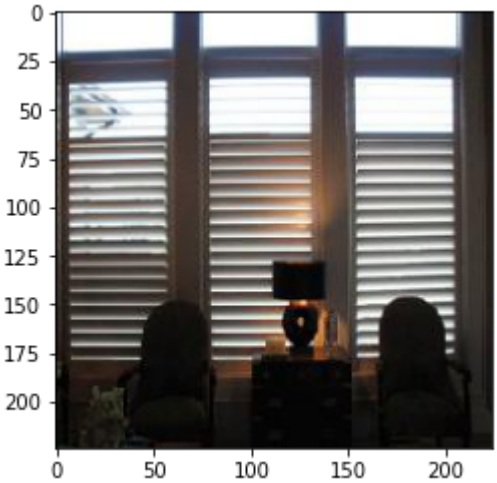


n04493381.JPEG Epsilon = 0.01

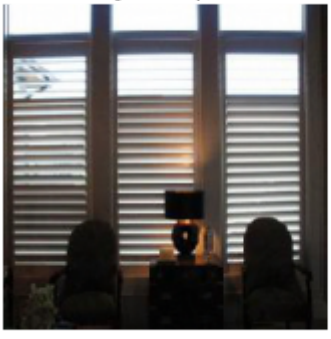


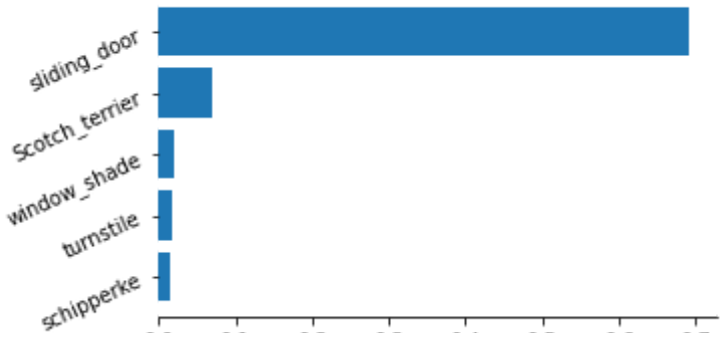

```
Predicted label:  coho
```

```
True label:  window_shade
```



n04590129.JPEG Epsilon = 0.01

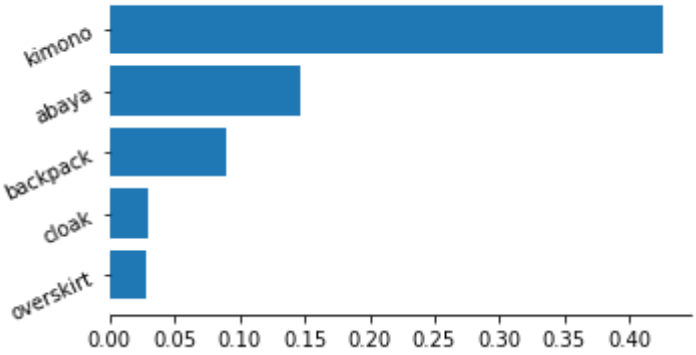





```
Predicted label:  sliding_door
```

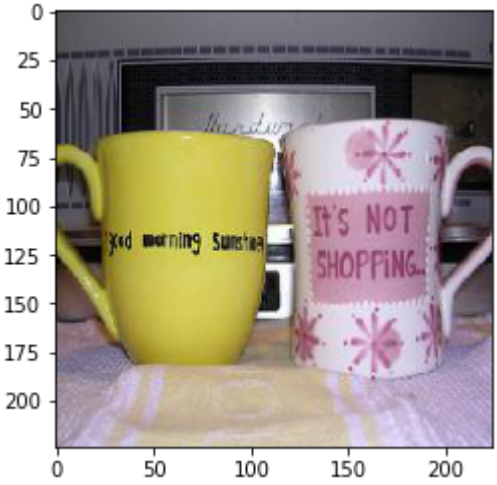
```
True label:  backpack
```

n02769748.JPEG Epsilon = 0.01

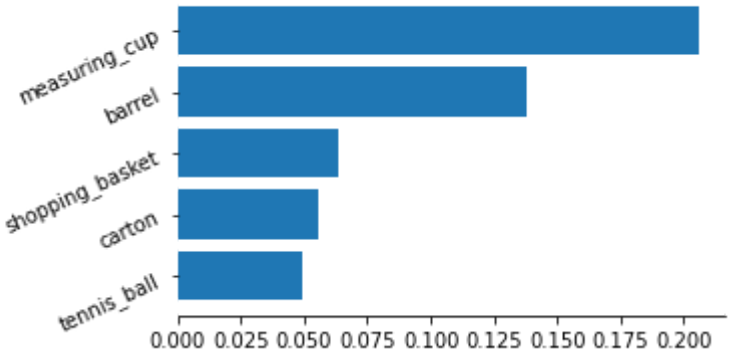



```
Predicted label:  kimono
```

```
True label:  cup
```



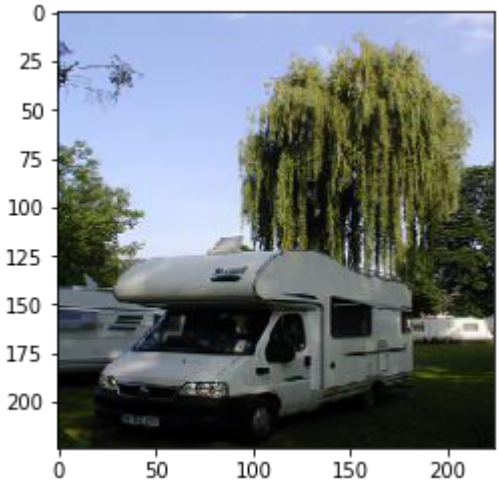
n03063599.JPEG Epsilon = 0.01



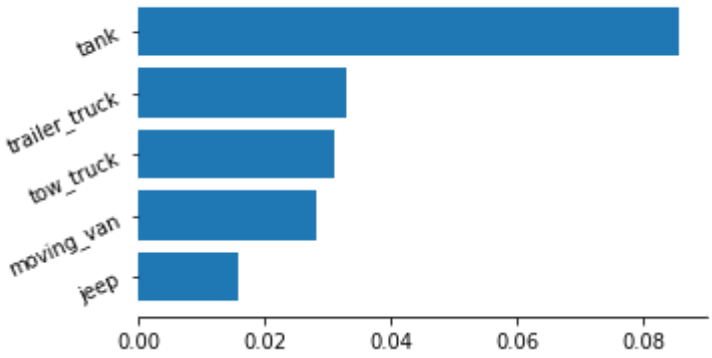

```
Predicted label:  measuring_cup
```

Predicted label: measuring_cup

True label: recreational_vehicle

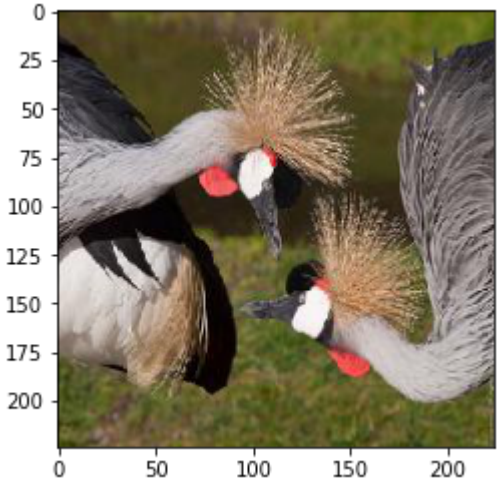


n04065272.JPEG Epsilon = 0.01

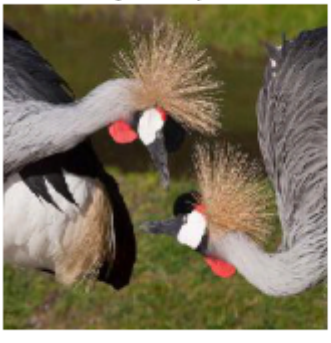


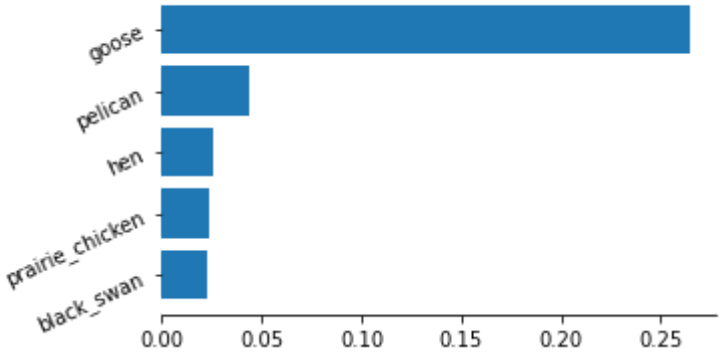
Predicted label: tank

True label: crane



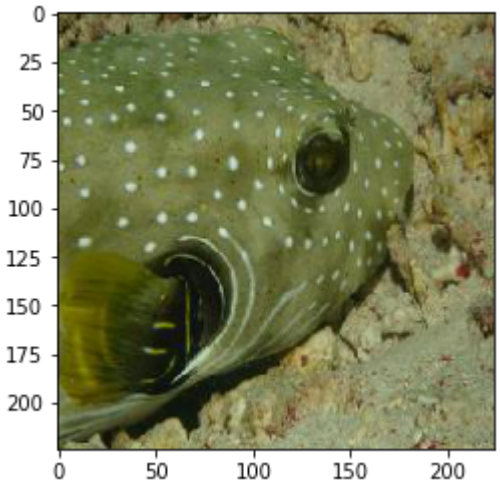
n02012849.JPEG Epsilon = 0.01



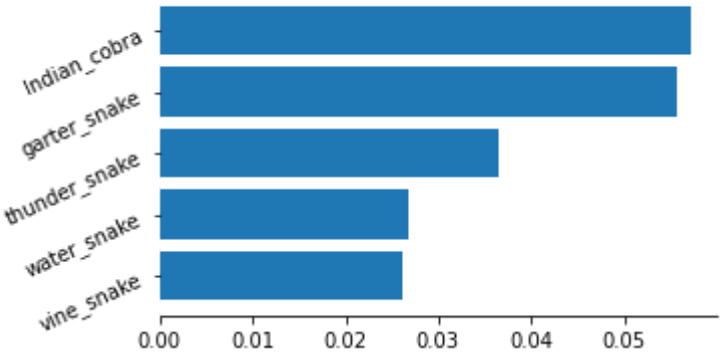


Predicted label: goose

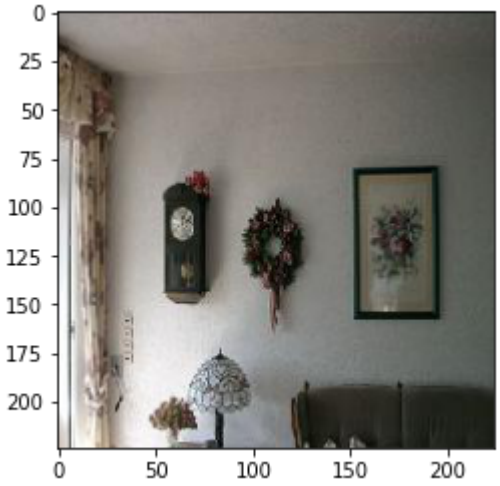
```
True label:  puffer
```



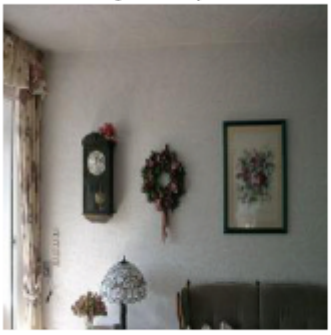
n02655020.JPEG Epsilon = 0.01

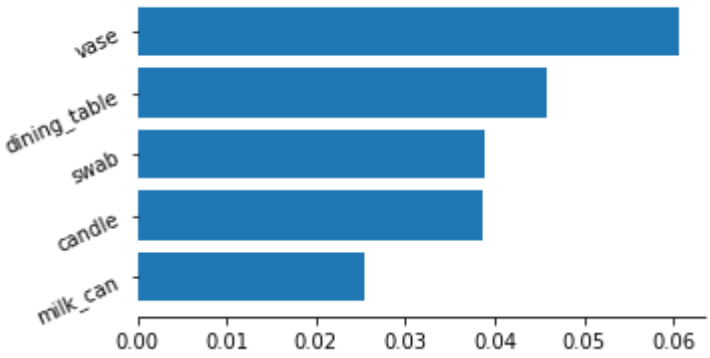



```
Predicted label:  Indian_cobra

True label:  wall_clock
```

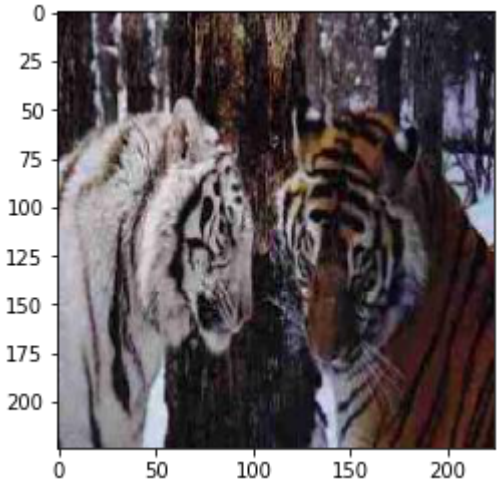


n04548280.JPEG Epsilon = 0.01

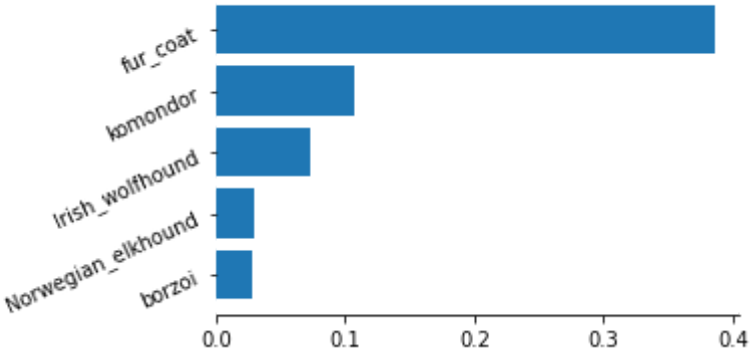





```
Predicted label:  vase

True label:  tiger
```

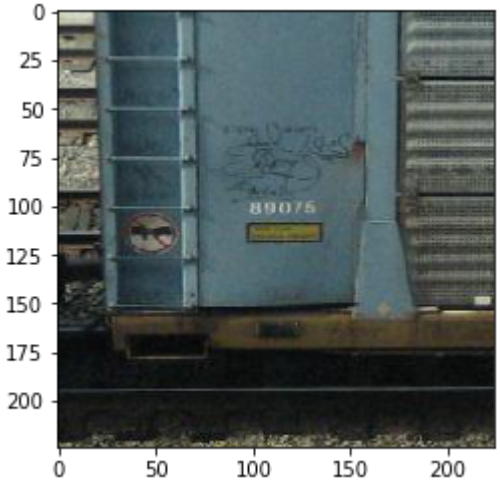


n02129604.JPEG Epsilon = 0.01

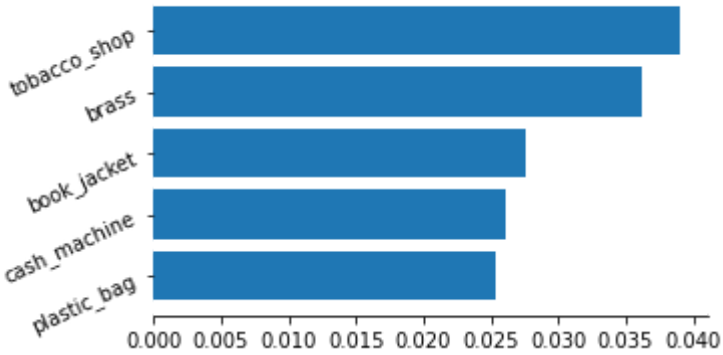



```
Predicted label:  fur_coat
```

```
True label:  freight_car
```

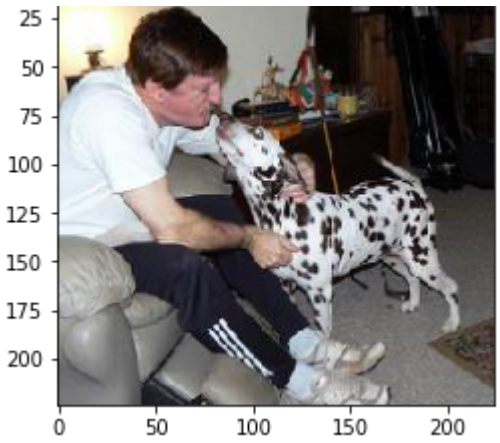


n03393912.JPEG Epsilon = 0.01

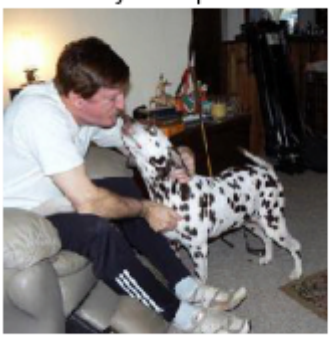


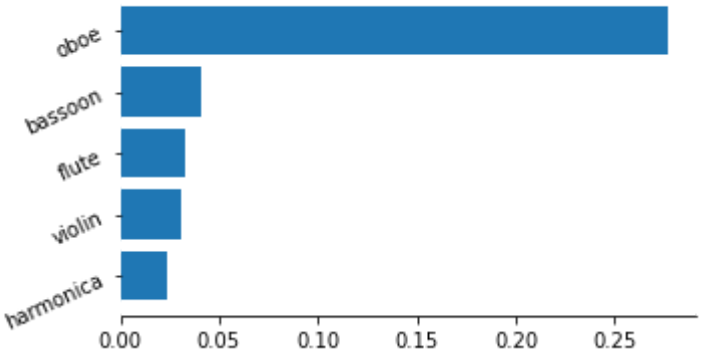

```
Predicted label:  tobacco_shop
```

```
True label:  dalmatian
```



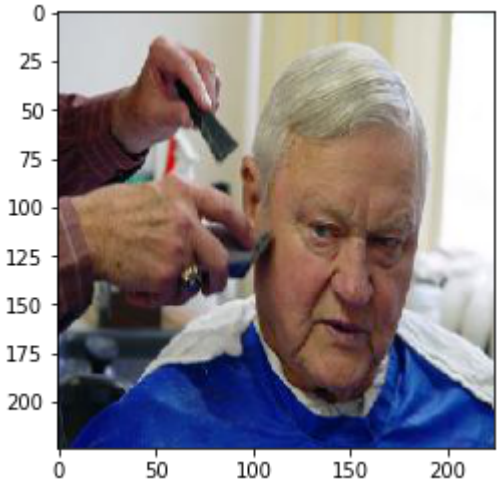
n02110341.JPEG Epsilon = 0.01



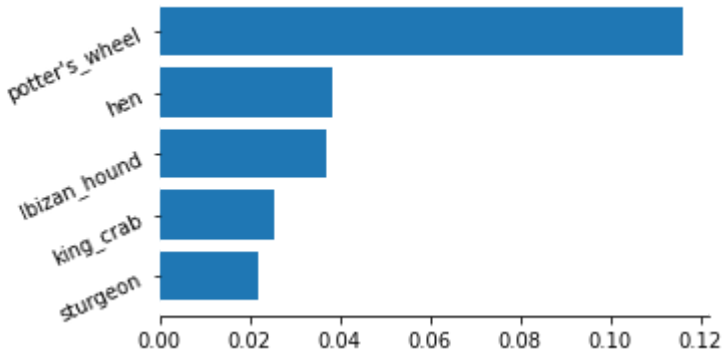



```
Predicted label:  oboe
```

```
True label:  barbershop
```

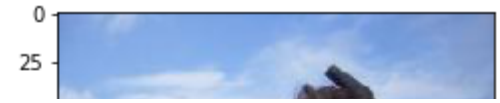


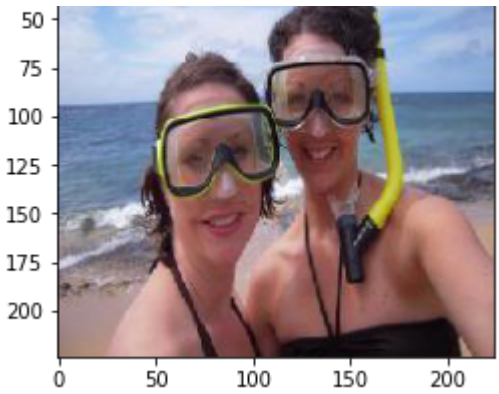
n02791270.JPEG Epsilon = 0.01



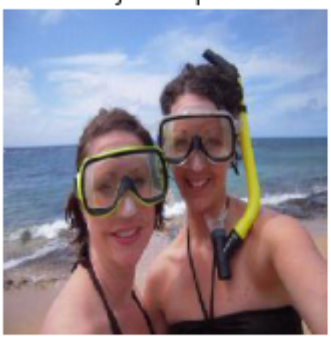

```
Predicted label:  potter's_wheel
```

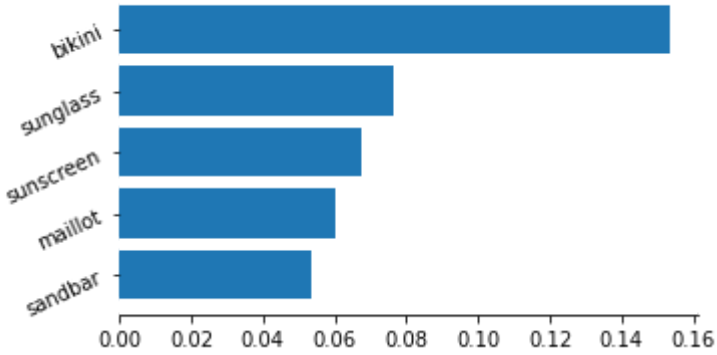
```
True label:  snorkel
```





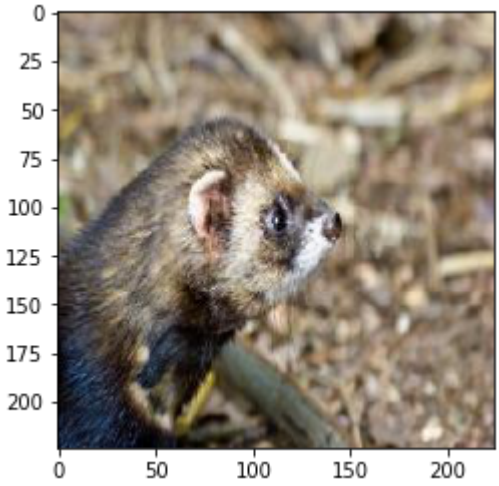
n04251144.JPEG Epsilon = 0.01



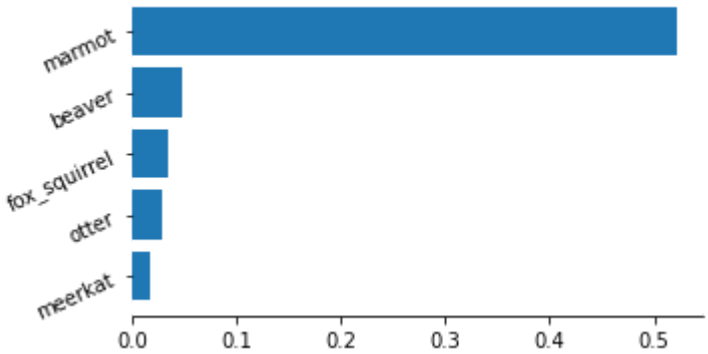



```
Predicted label:  bikini
```

```
True label:  polecat
```

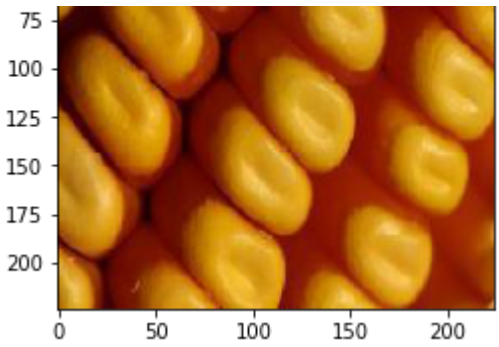


n02443114.JPEG Epsilon = 0.01

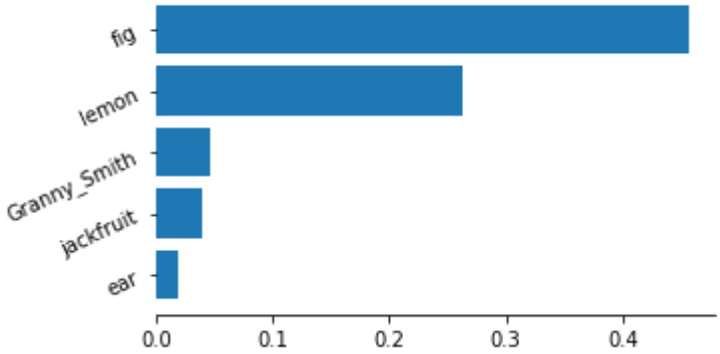



```
Predicted label:  marmot
```

```
True label:  corn
```

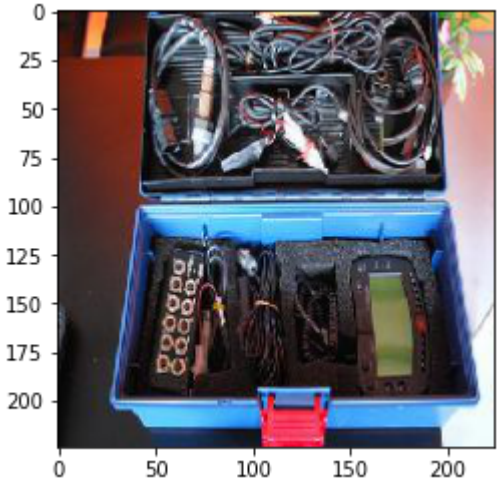


n12144580.JPEG Epsilon = 0.01

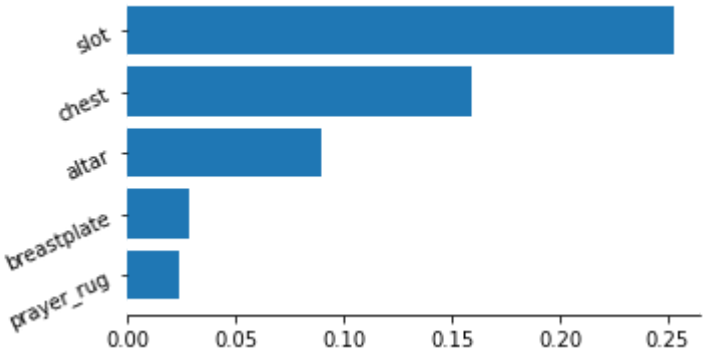



```
Predicted label:  fig
```

```
True label:  carpenter's_kit
```

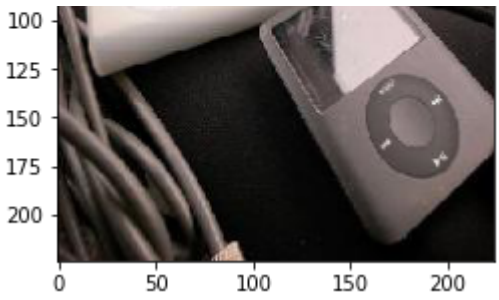


n02966687.JPEG Epsilon = 0.01

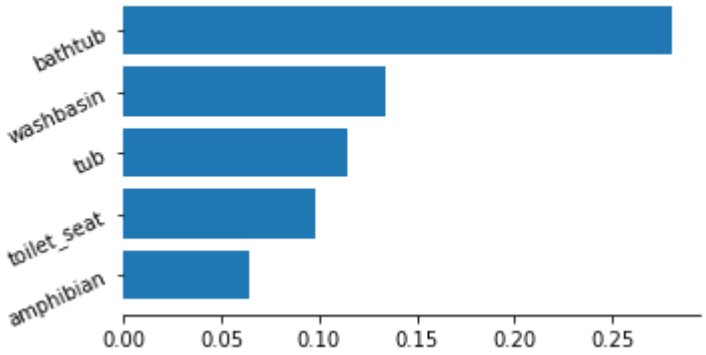



```
Predicted label:  slot
```

```
True label:  iPod
```

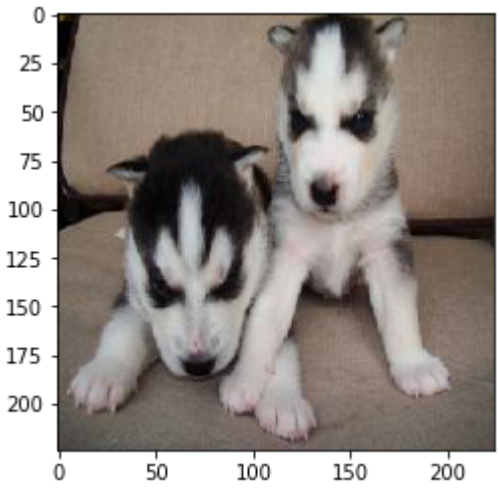


n03584254.JPEG Epsilon = 0.01

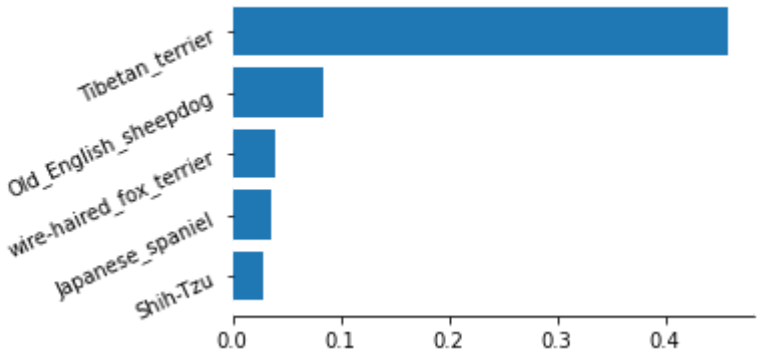



```
Predicted label:  bathtub
```

```
True label:  Siberian_husky
```

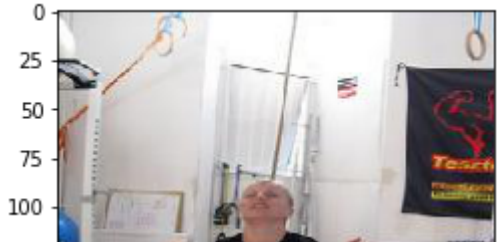


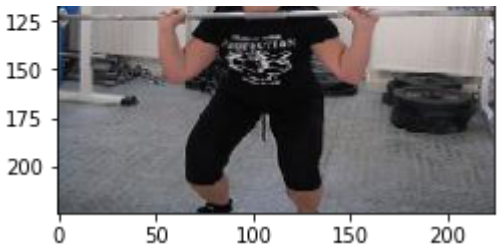
n02110185.JPEG Epsilon = 0.01



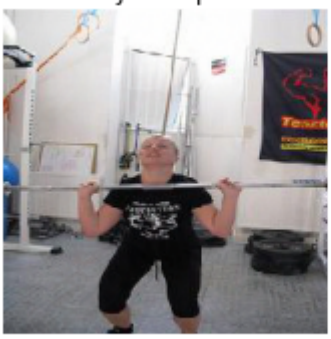

```
Predicted label:  Tibetan_terrier
```

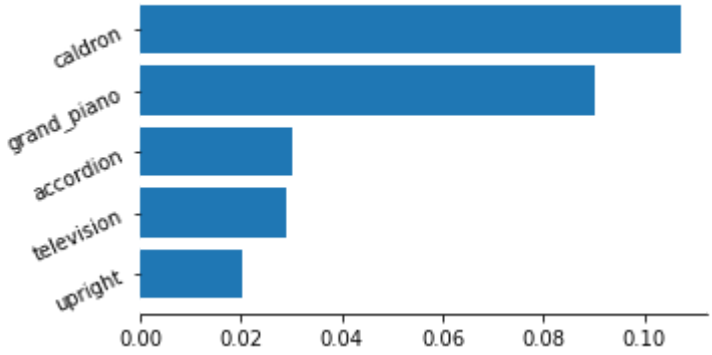
```
True label:  barbell
```





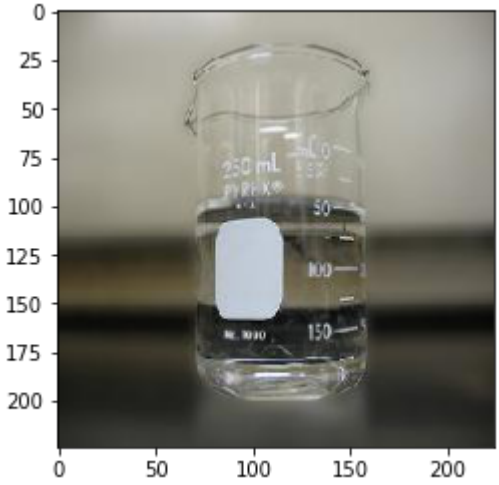
n02790996.JPEG Epsilon = 0.01



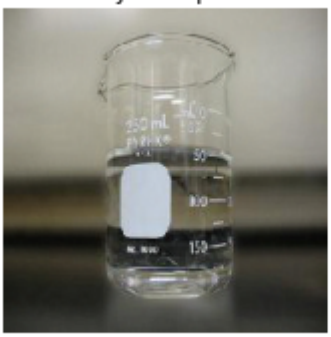


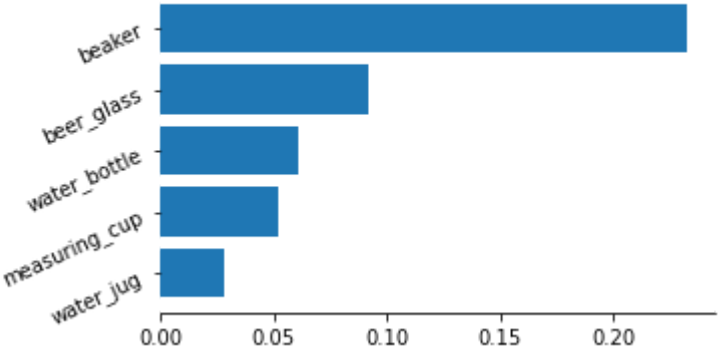
Predicted label: caldron

True label: beaker



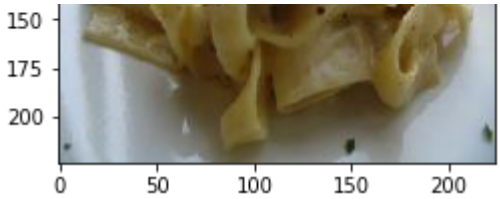
n02815834.JPEG Epsilon = 0.01



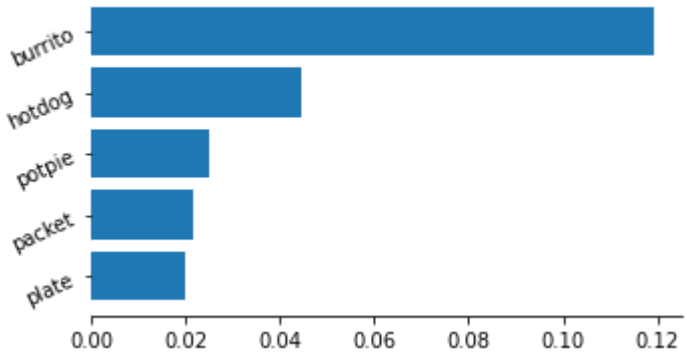


Predicted label: beaker

True label: carbonara

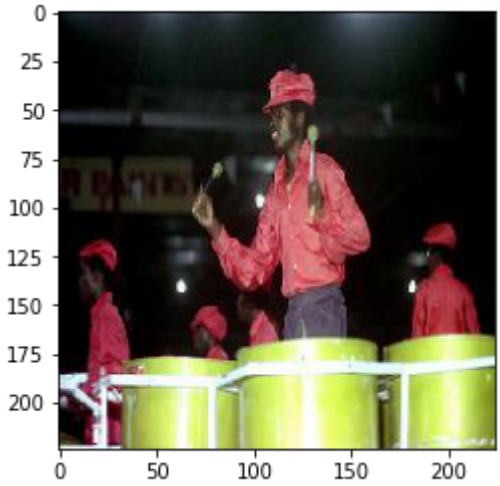


n07831146.JPEG Epsilon = 0.01

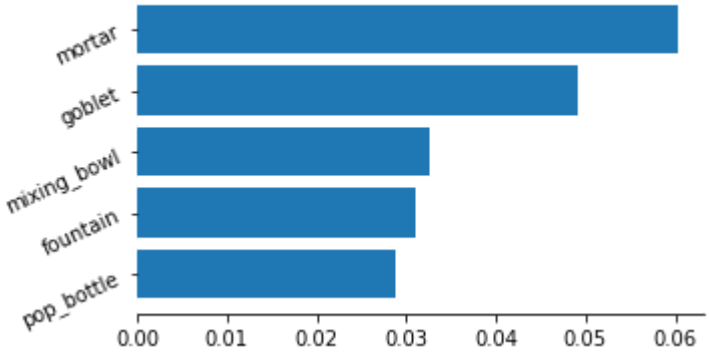



```
Predicted label:  burrito
```

```
True label:  steel_drum
```

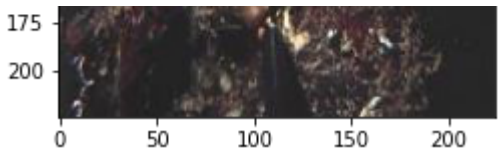


n04311174.JPEG Epsilon = 0.01

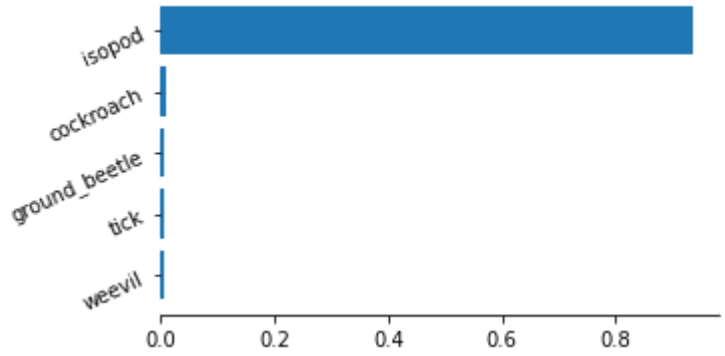



```
Predicted label:  mortar
```

```
True label:  chiton
```

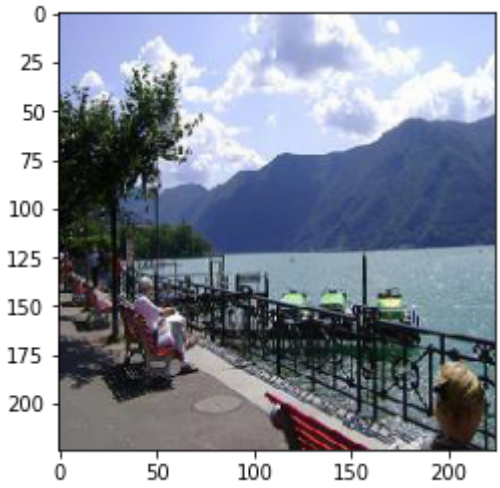


n01955084.JPEG Epsilon = 0.01

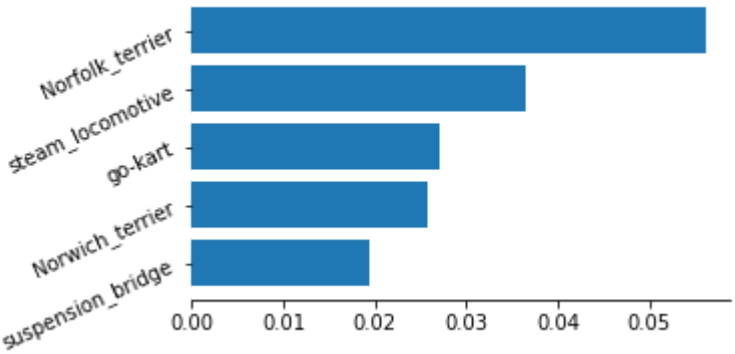



```
Predicted label:  isopod
```

```
True label:  lakeside
```

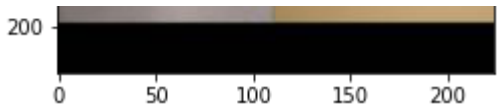


n09332890.JPEG Epsilon = 0.01

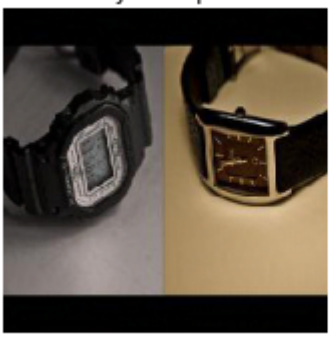


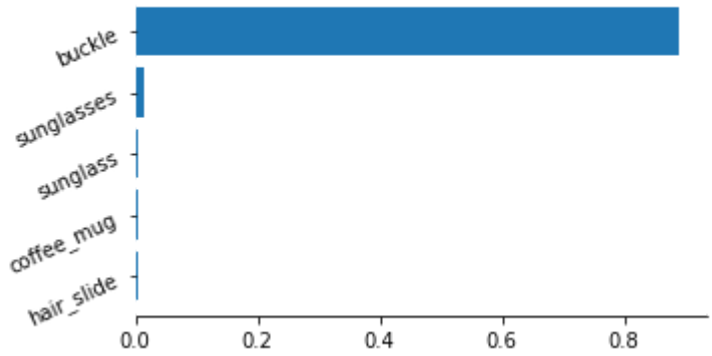

```
Predicted label:  Norfolk_terrier
```

```
True label:  digital_watch
```



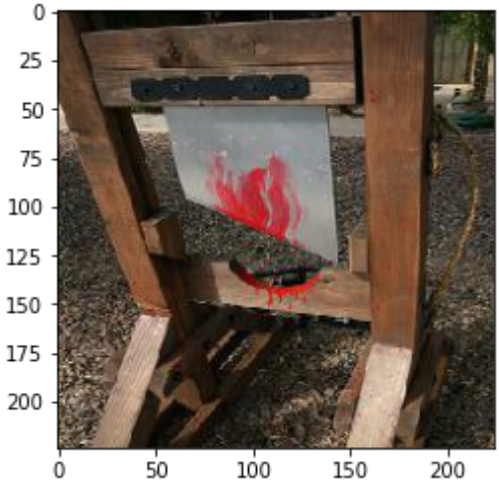
n03197337.JPEG Epsilon = 0.01



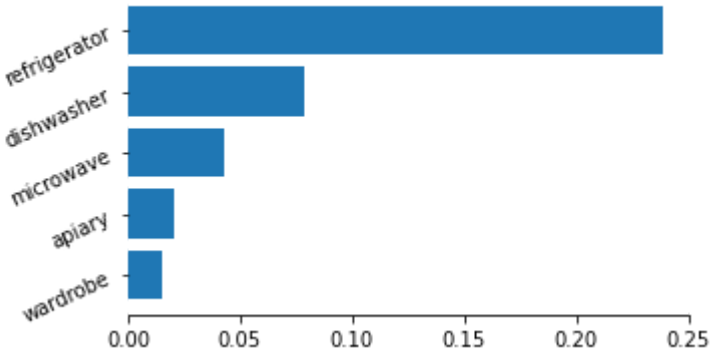



```
Predicted label:  buckle
```

```
True label:  guillotine
```



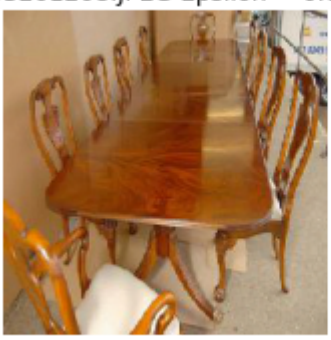
n03467068.JPEG Epsilon = 0.01

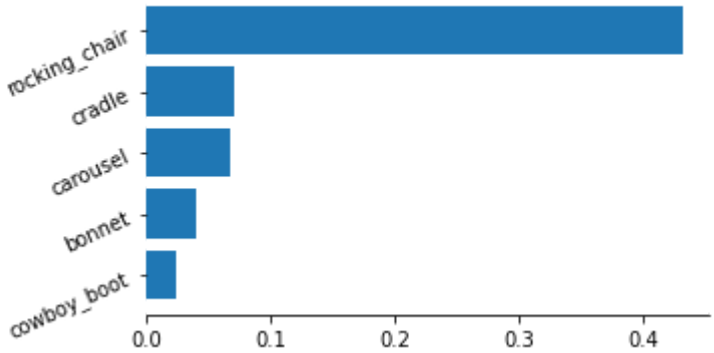



```
Predicted label:  refrigerator
```

```
True label:  dining_table
```

n03201208.JPEG Epsilon = 0.01

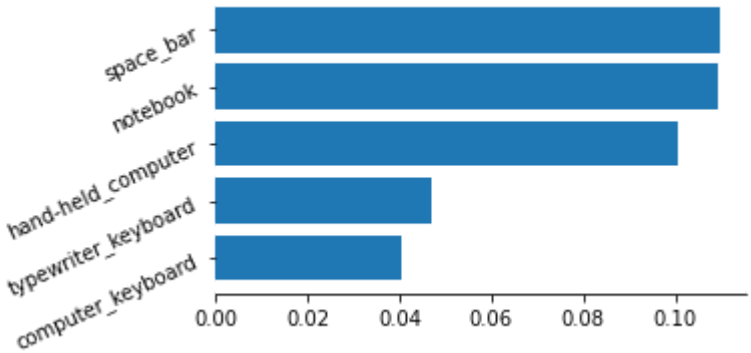





```
Predicted label:  rocking_chair
```

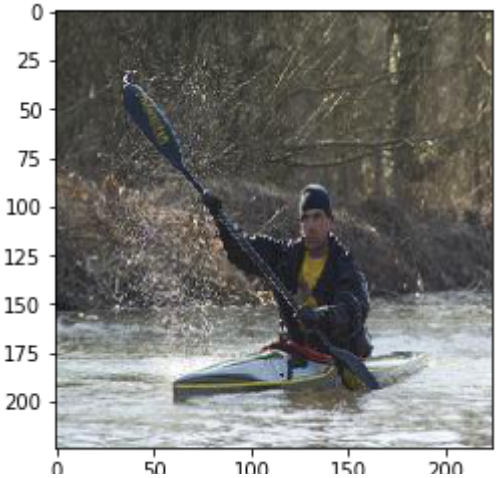
```
True label:  hand-held_computer
```

n03485407.JPEG Epsilon = 0.01

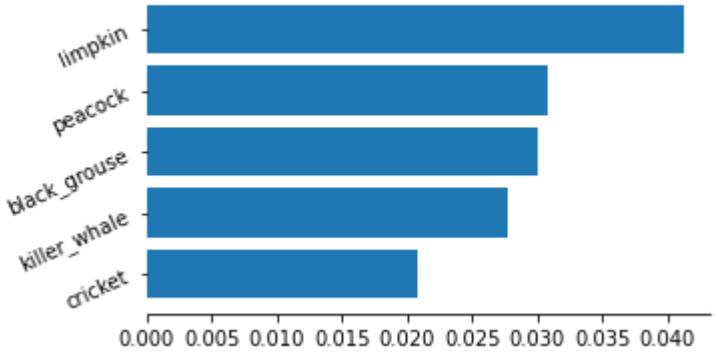



```
Predicted label:  space_bar
```

```
True label:  paddle
```

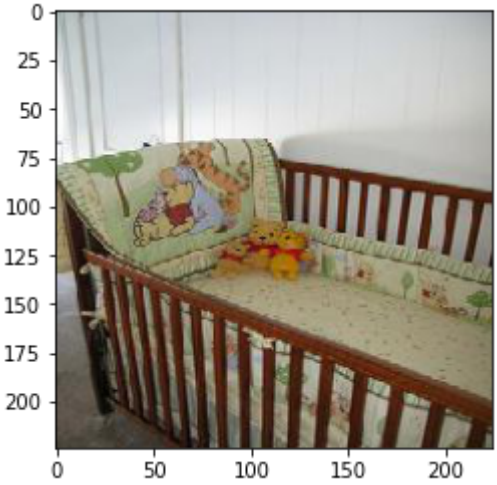


n03873416.JPEG Epsilon = 0.01

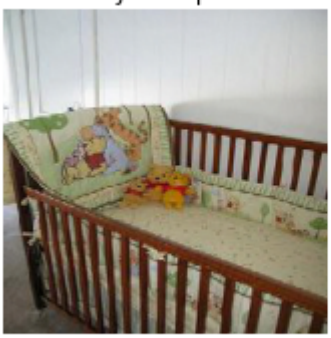


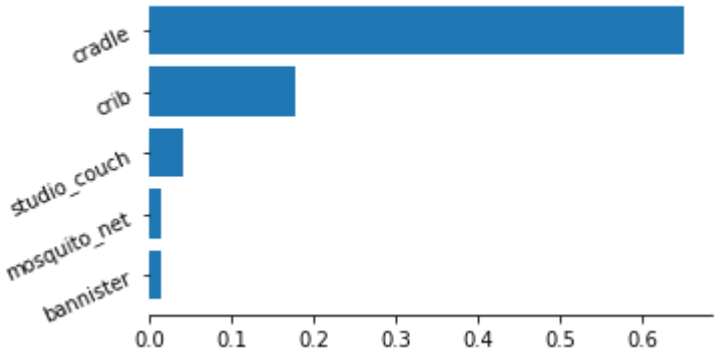
Predicted label: limpkin

True label: crib



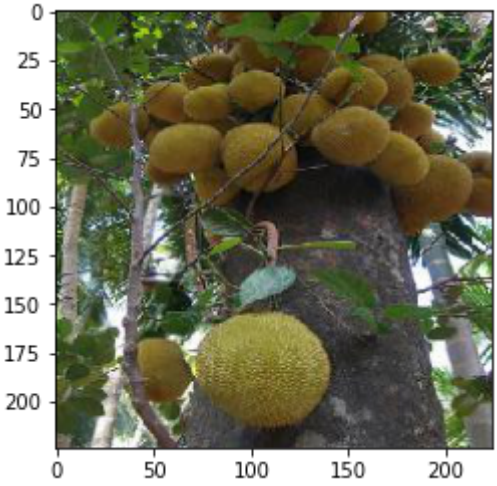
n03131574.JPEG Epsilon = 0.01



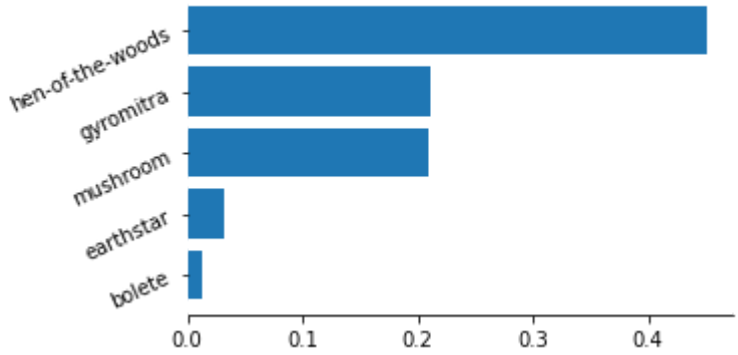


Predicted label: cradle

True label: jackfruit

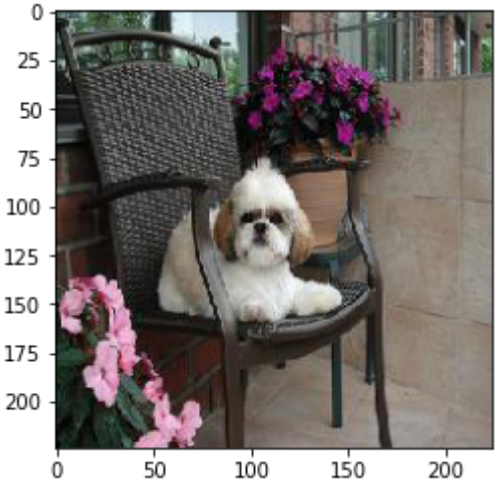


n07754684.JPEG Epsilon = 0.01

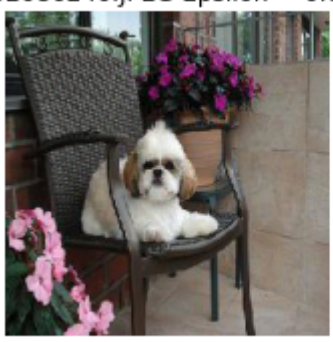


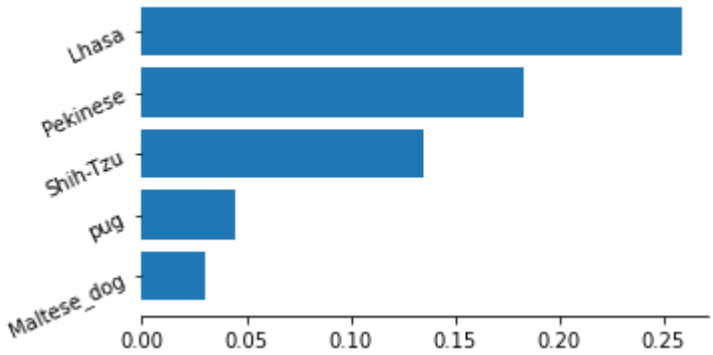

```
Predicted label:  hen-of-the-woods
```

```
True label:  Shih-Tzu
```



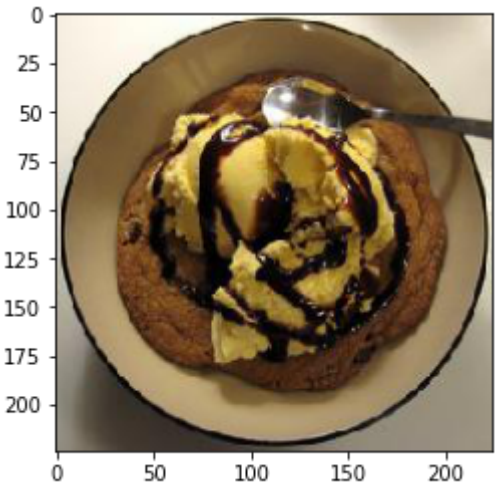
n02086240.JPEG Epsilon = 0.01



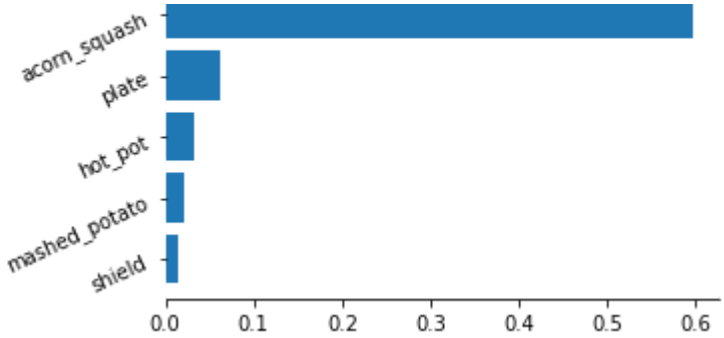



```
Predicted label:  Lhasa
```

```
True label:  chocolate_sauce
```

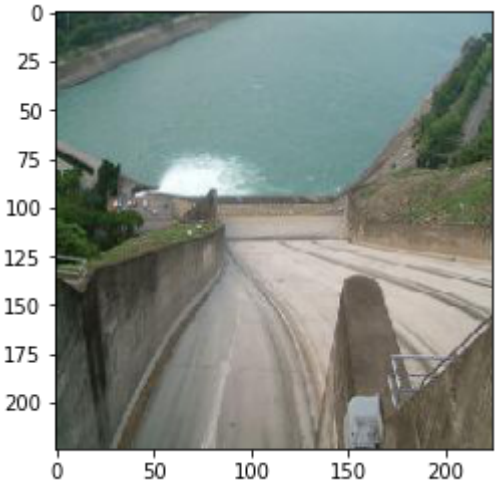


n07836838.JPEG Epsilon = 0.01

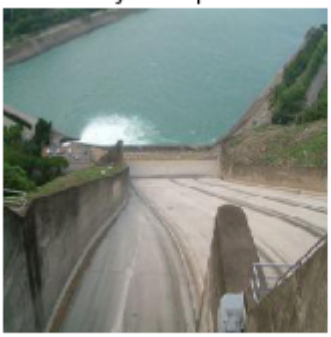


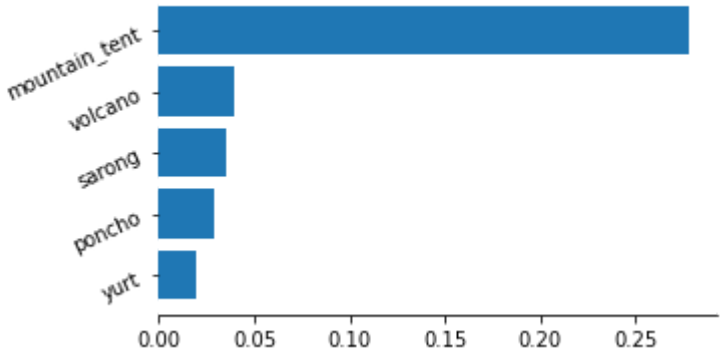

```
Predicted label:  acorn_squash
```

```
True label:  dam
```



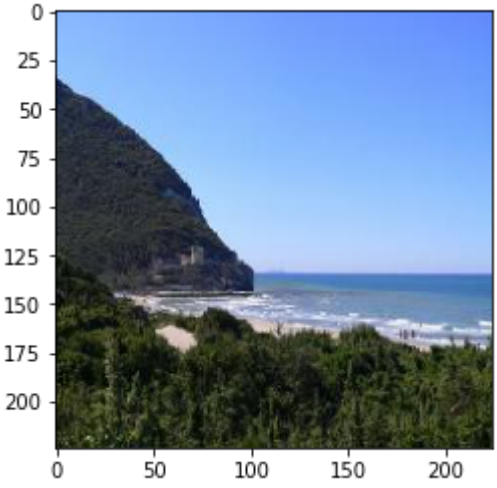
n03160309.JPEG Epsilon = 0.01



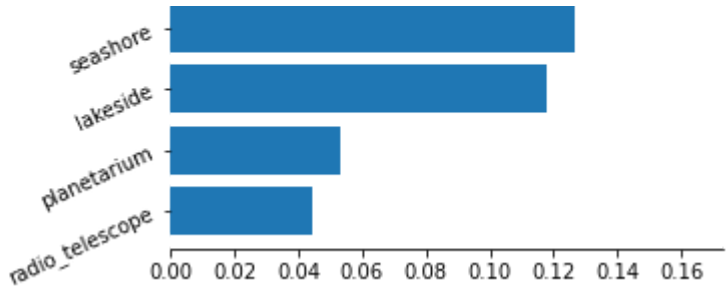



```
Predicted label:  mountain_tent
```

```
True label:  promontory
```

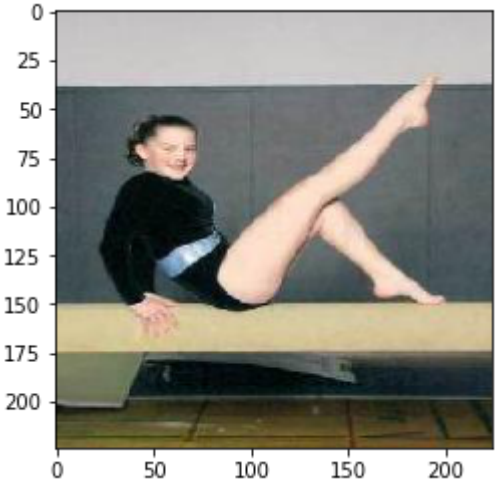


n09399592.JPEG Epsilon = 0.01

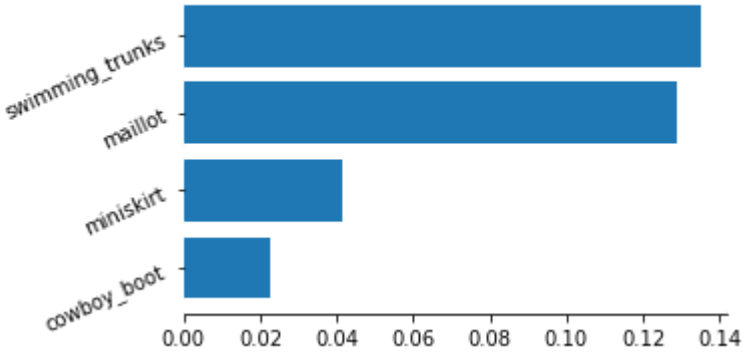



```
Predicted label:  dam
```

```
True label:  balance_beam
```

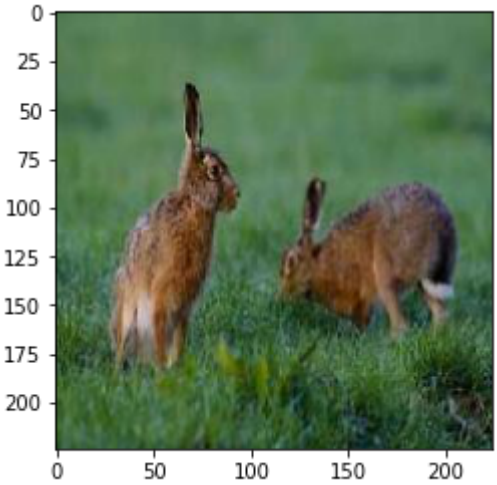


n02777292.JPEG Epsilon = 0.01

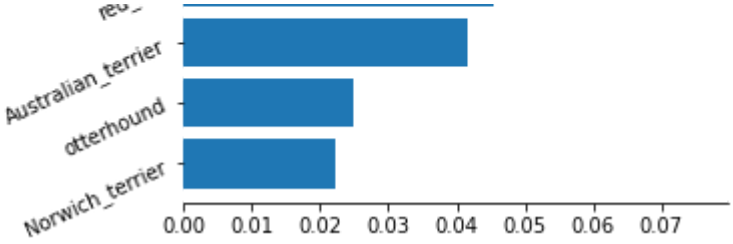



```
Predicted label:  swimming_trunks
```

```
True label:  hare
```

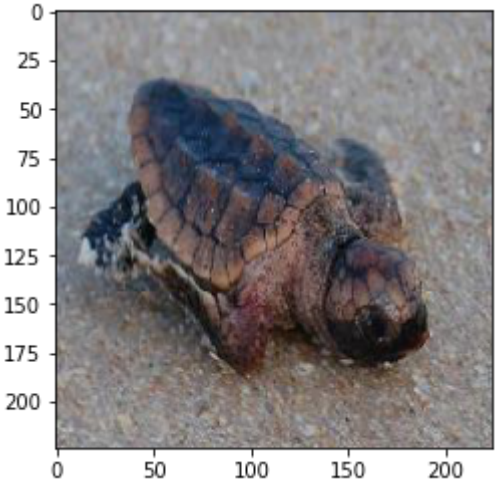


n02326432.JPEG Epsilon = 0.01

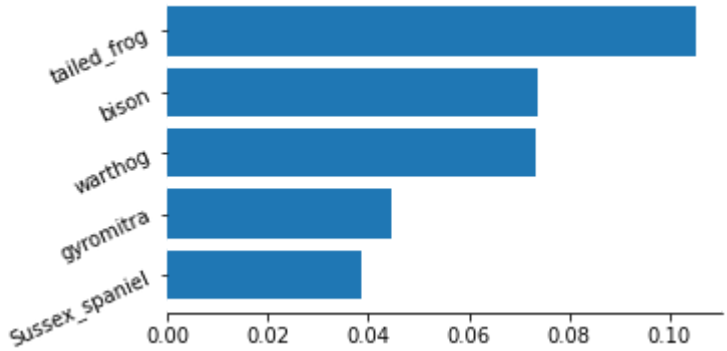



```
Predicted label:  kit_fox
```

```
True label:  loggerhead
```

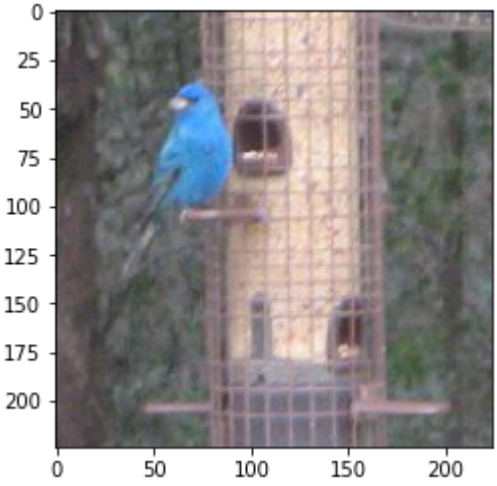


n01664065.JPEG Epsilon = 0.01

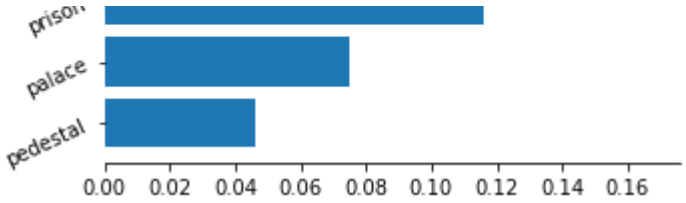



```
Predicted label:  tailed_frog
```

```
True label:  indigo_bunting
```

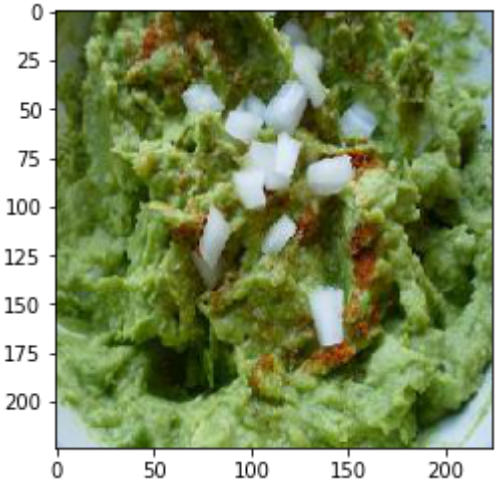


n01537544.JPEG Epsilon = 0.01

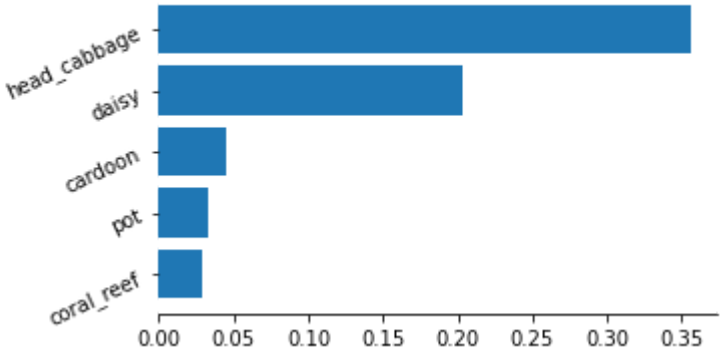



```
Predicted label:  bell_cote
```

```
True label:  guacamole
```

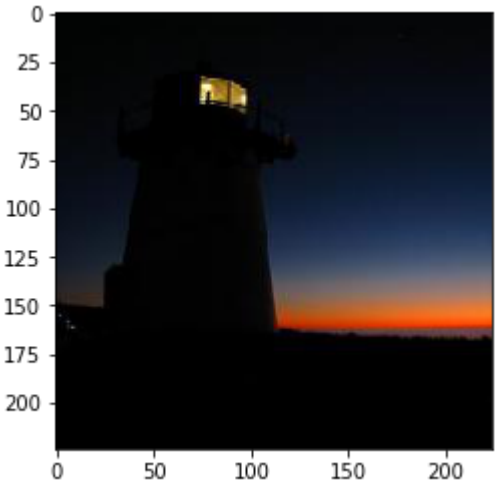


n07583066.JPEG Epsilon = 0.01

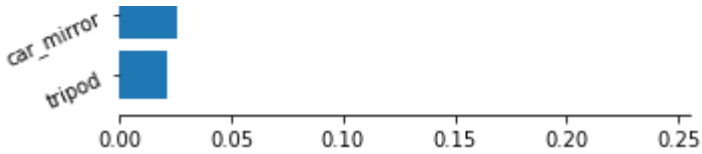



```
Predicted label:  head_cabbage
```

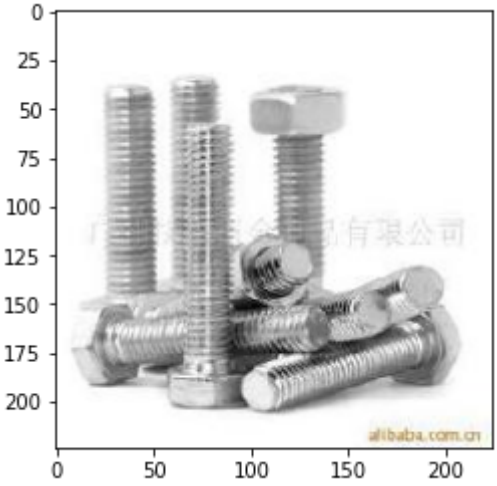
```
True label:  beacon
```



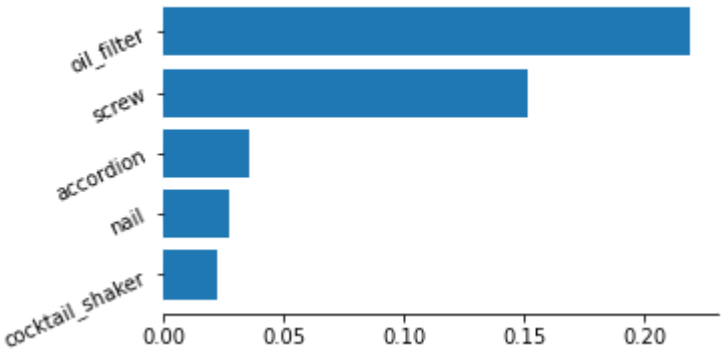
n02814860.JPEG Epsilon = 0.01



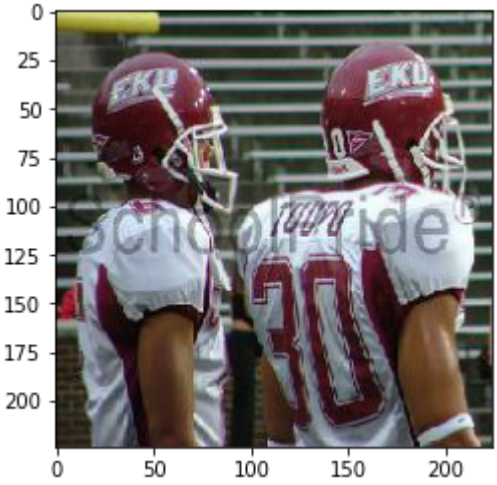

```
Predicted label:  projector

True label:  screw
```



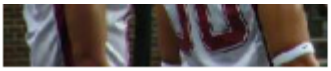
n04153751.JPEG Epsilon = 0.01

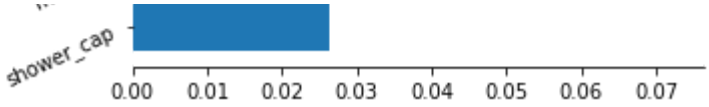



```
Predicted label:  oil_filter

True label:  football_helmet
```



n03379051.JPEG Epsilon = 0.01

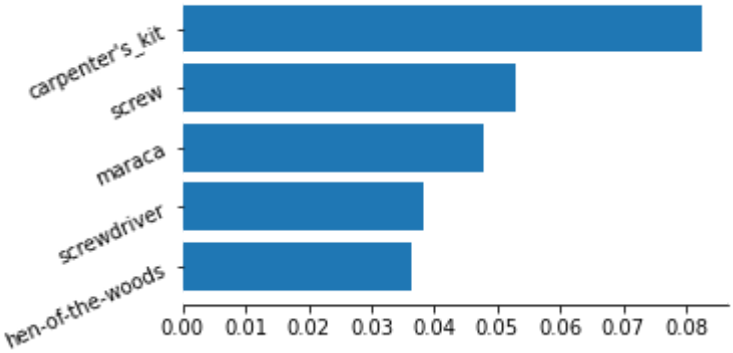





```
Predicted label:  bearskin
```

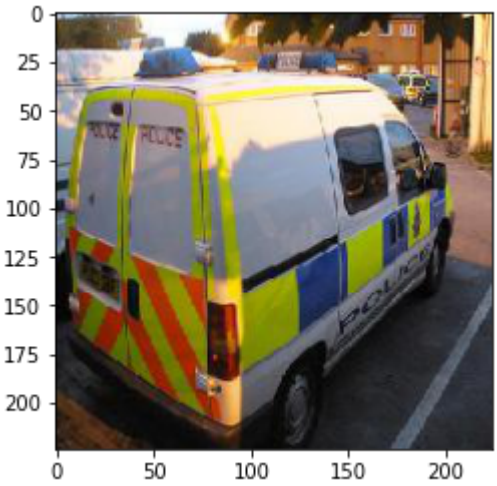
```
True label:  wooden_spoon
```

n04597913.JPEG Epsilon = 0.01

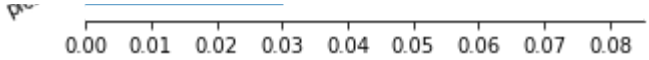



```
Predicted label:  carpenter's_kit
```

```
True label:  police_van
```

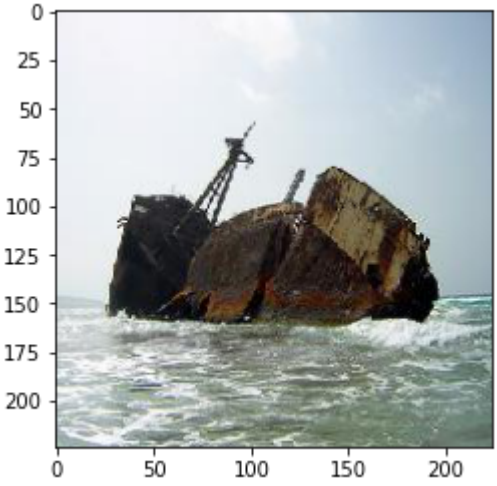


n03977966.JPEG Epsilon = 0.01

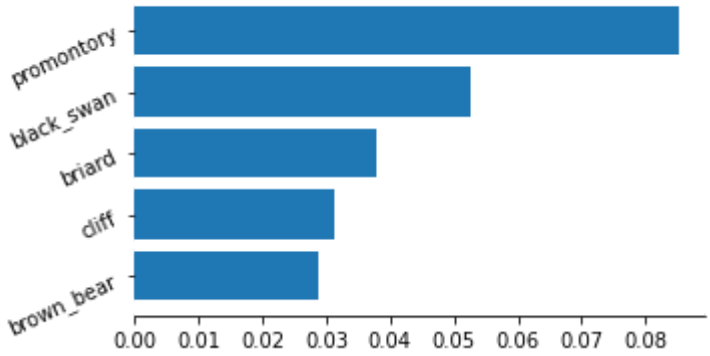



```
Predicted label:  racer
```

```
True label:  wreck
```

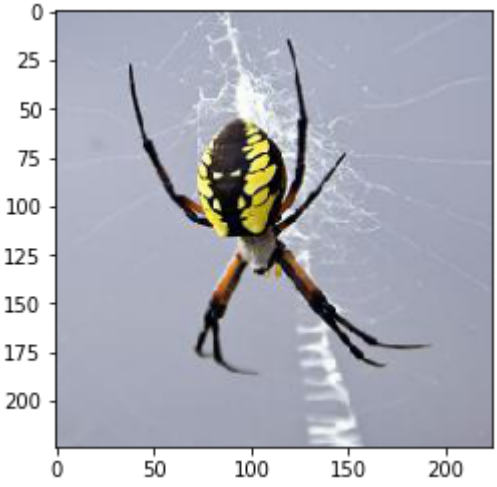


n04606251.JPEG Epsilon = 0.01

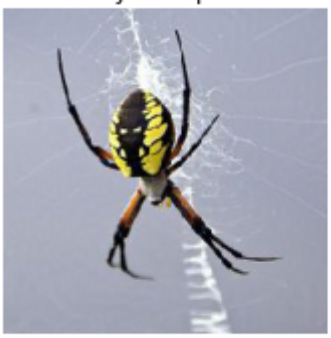


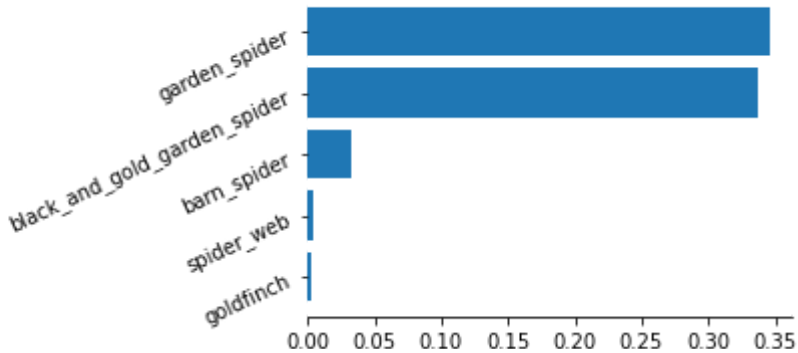

```
Predicted label:  promontory
```

```
True label:  black_and_gold_garden_spider
```



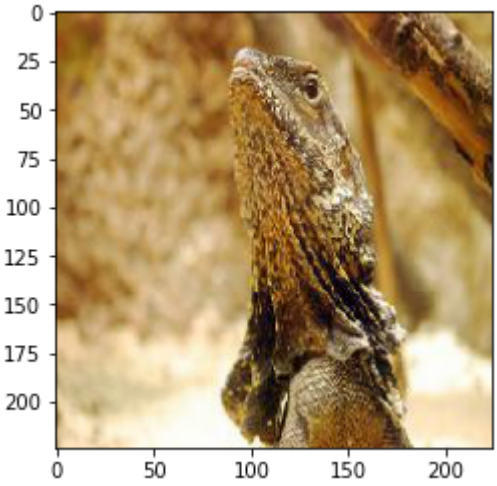
n01773157.JPEG Epsilon = 0.01



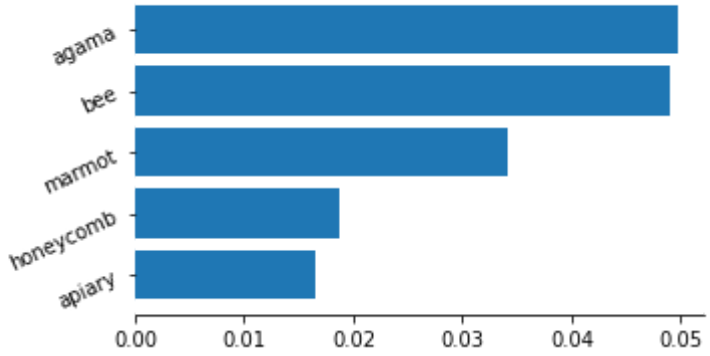


```
Predicted label:  garden_spider
```

```
True label:  frilled_lizard
```

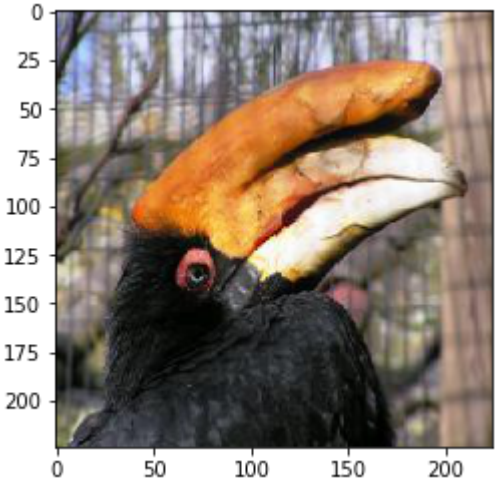


n01688243.JPEG Epsilon = 0.01

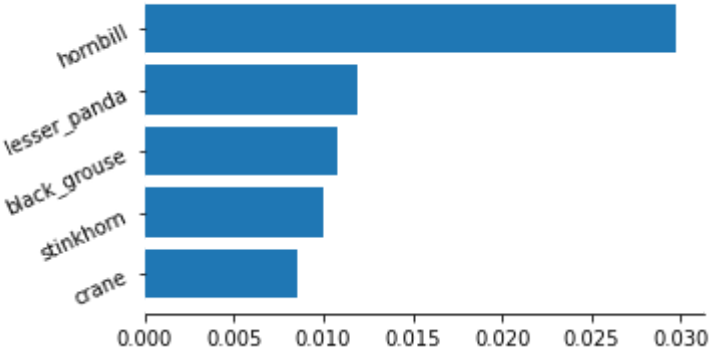



```
Predicted label:  agama
```

```
True label:  hornbill
```

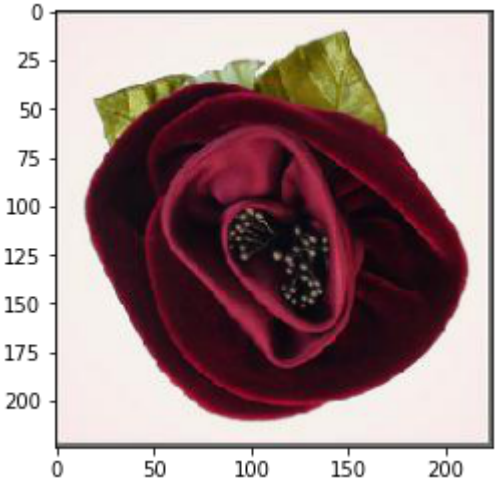


n01829413.JPEG Epsilon = 0.01

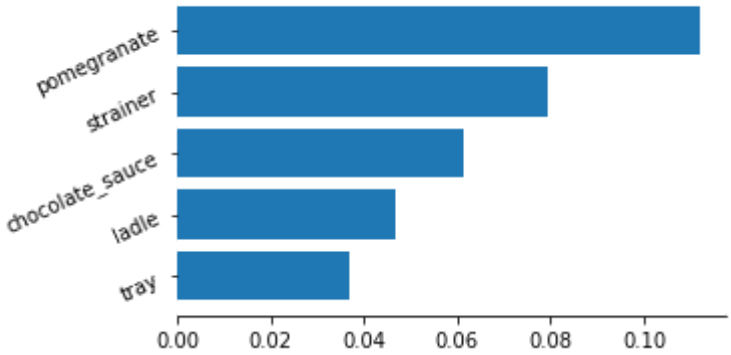



```
Predicted label:  hornbill
```

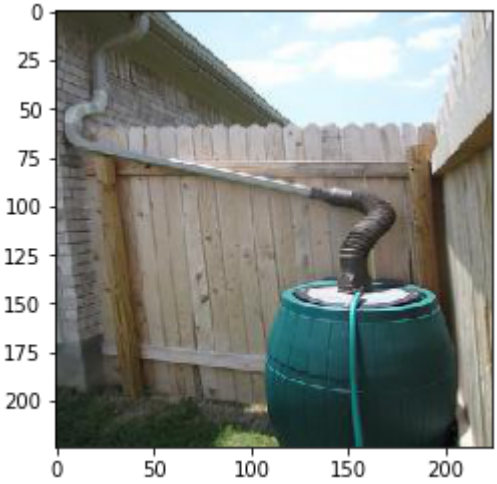
```
True label:  velvet
```



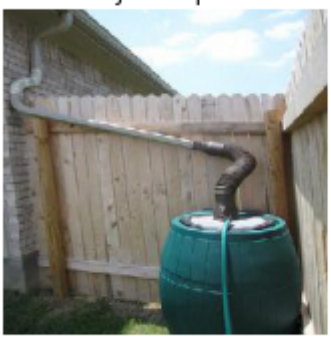
n04525038.JPEG Epsilon = 0.01

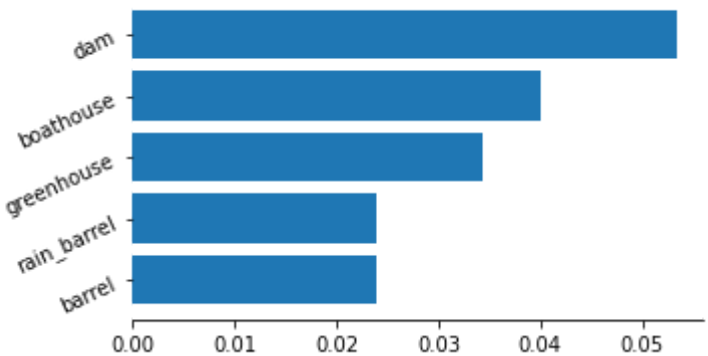



```
Predicted label:  pomegranate

True label:  rain_barrel
```



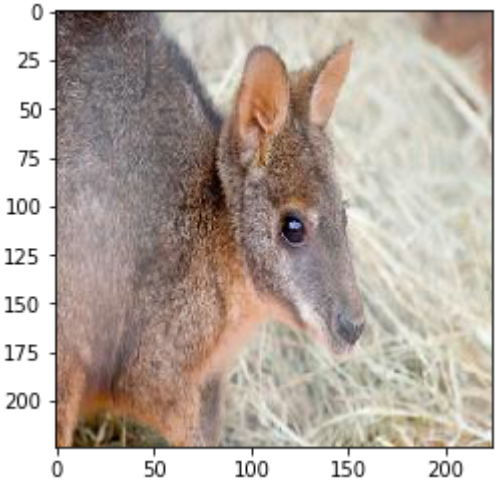
n04049303.JPEG Epsilon = 0.01



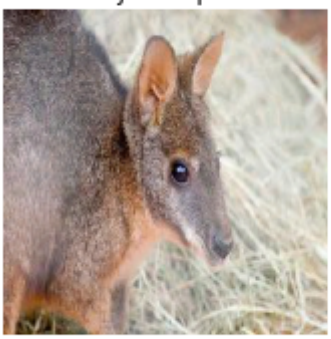


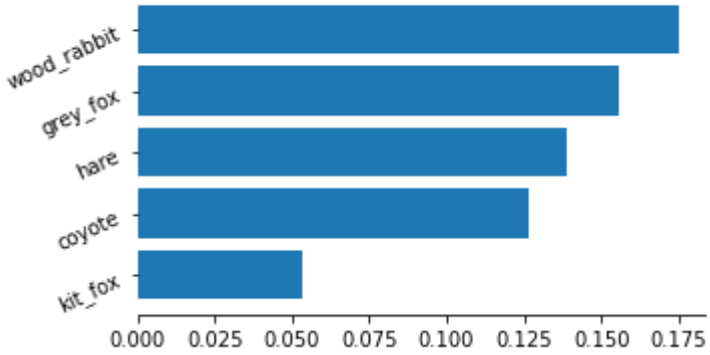

```
Predicted label:  dam

True label:  wallaby
```

True label: wallaby



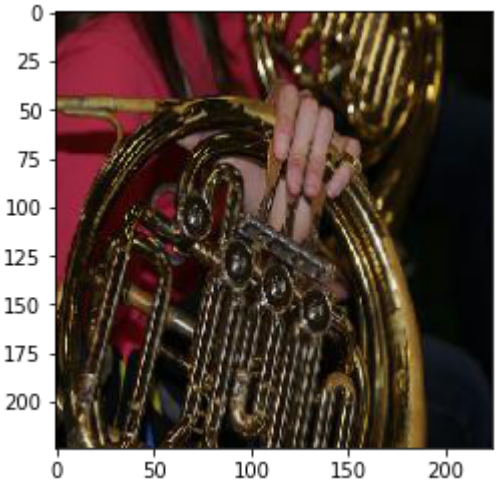
n01877812.JPEG Epsilon = 0.01



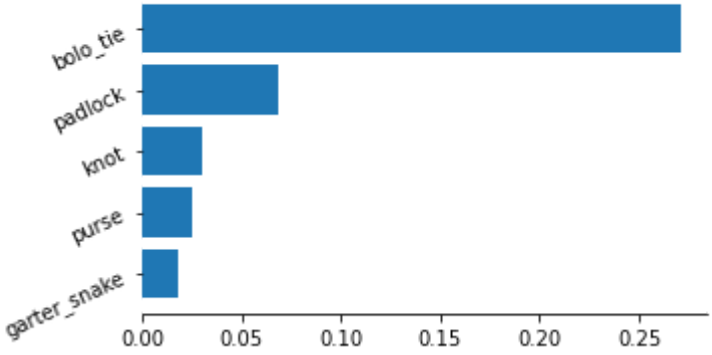


Predicted label: wood_rabbit

True label: French_horn

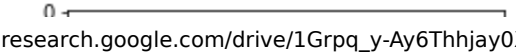


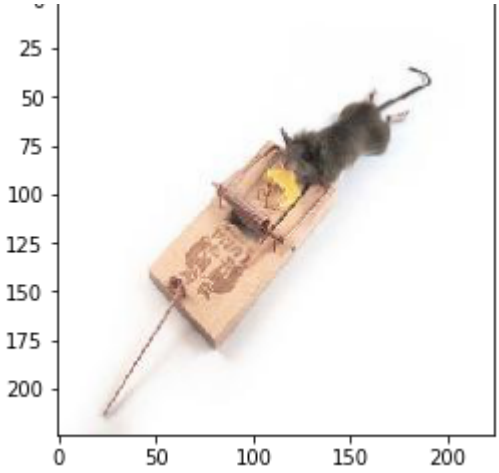
n03394916.JPEG Epsilon = 0.01



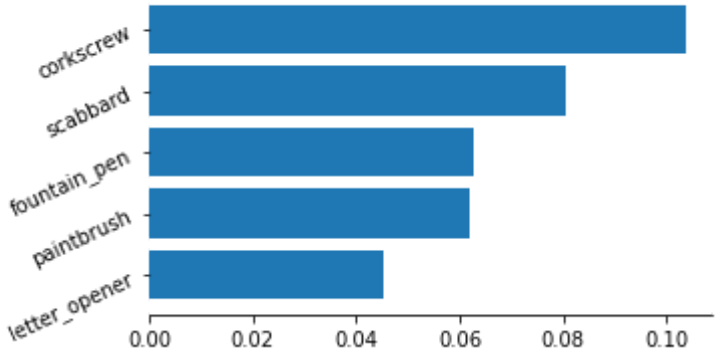
Predicted label: bolo_tie

True label: mousetrap



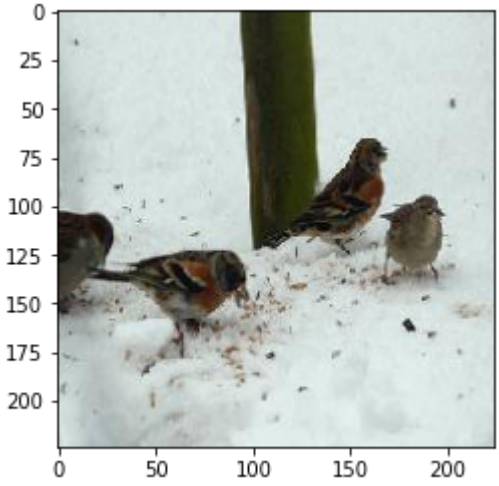


n03794056.JPEG Epsilon = 0.01

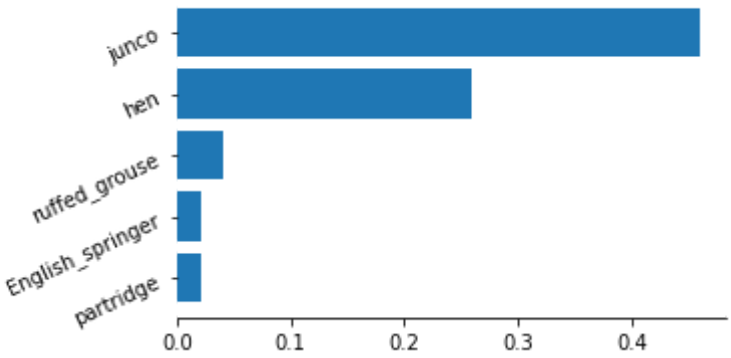


Predicted label: corkscrew

True label: brambling

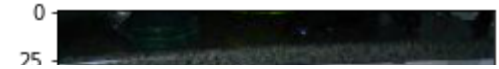


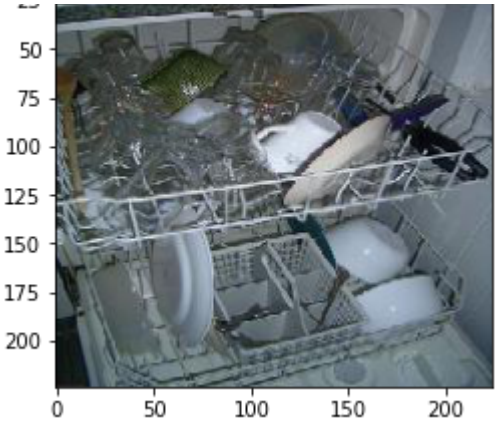
n01530575.JPEG Epsilon = 0.01



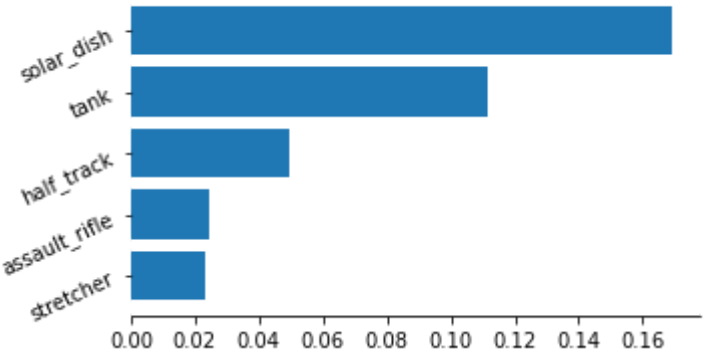
Predicted label: junco

True label: dishwasher



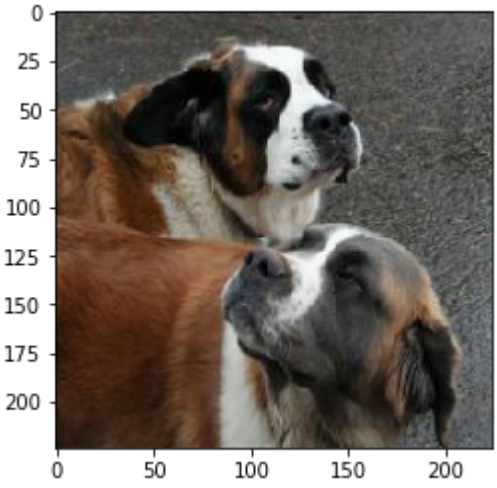


n03207941.JPEG Epsilon = 0.01

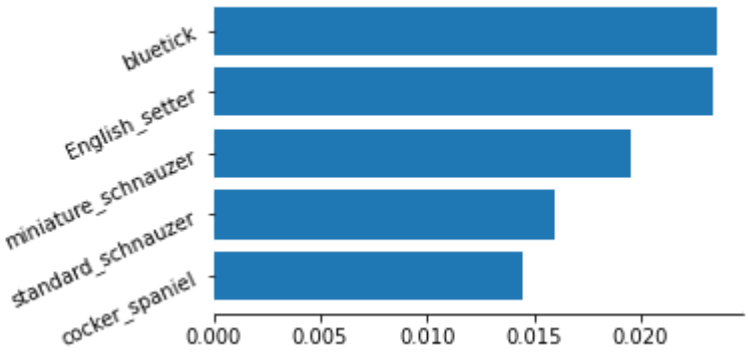


Predicted label: solar_dish

True label: Saint_Bernard



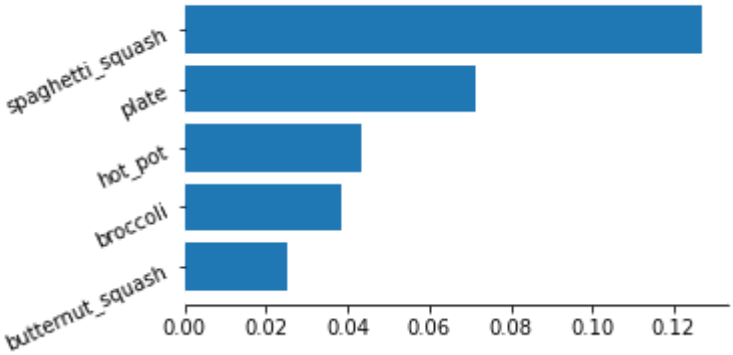
n02109525.JPEG Epsilon = 0.01



Predicted label: bluetick

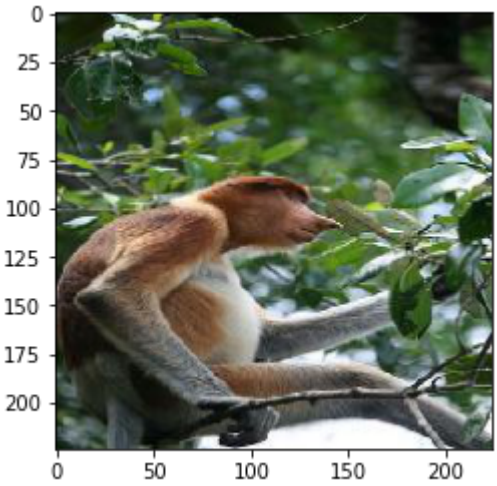
True label: cheeseburger

n07697313.JPEG Epsilon = 0.01

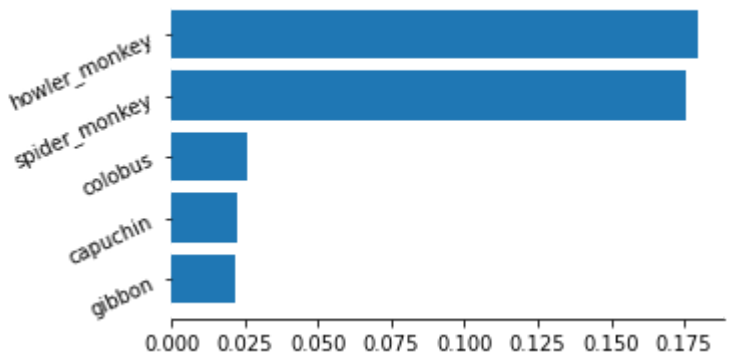



```
Predicted label:  spaghetti_squash
```

```
True label:  proboscis_monkey
```

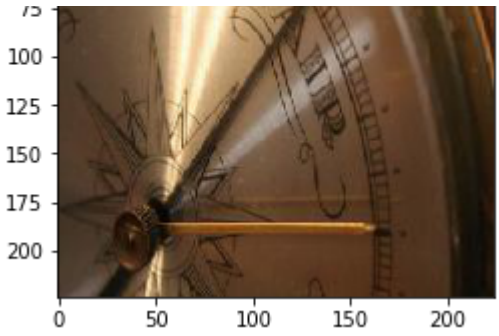


n02489166.JPEG Epsilon = 0.01

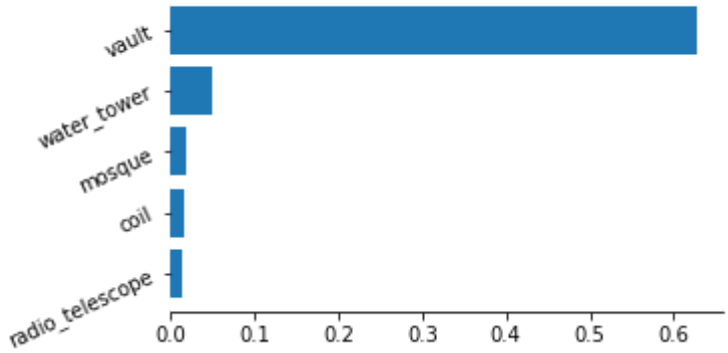



```
Predicted label:  howler_monkey
```

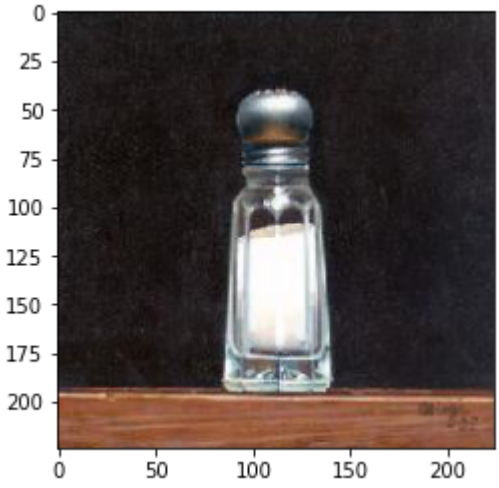
```
True label:  barometer
```



n02794156.JPEG Epsilon = 0.01




```
Predicted label:  vault

True label:  saltshaker
```



n04131690.JPEG Epsilon = 0.01


```
Rate of misclassification:  99.96
```

```
pred = []
true = []
for label in labels:
  image_raw = tf.io.read_file(label)
  print("image name: ", label)
  image = tf.image.decode_png(image_raw)
  input_image = preprocess(image)
  reversed_image = reverse_preprocess(input_image)
  preprocessed_img, reversed_img = sess.run([input_image, reversed_image])
  image_probs = sess.run([pretrained_model], {input_image_placeholder:preprocessed
```

```
    tp = get_imagenet_label(image_probs[0])
    print("True label: ", tp[0][1])
    true.append(tp[0][1])
    adv_x = input_image + (0.01*perturbations)
    adv_x = tf.clip_by_value(adv_x, -1, 1)
    p = display_images(adv_x, label)
    print("Predicted label: ", p, "\n")
    pred.append(p)

accuracy = len([true[i] for i in range(0, len(true)) if true[i] == pred[i]]) / len
print("Rate of misclassification: ", 100 - accuracy)
```

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!