Object Detection Demo

Welcome to the object detection inference walkthrough! This notebook will walk you step by step through the process of using a pre-trained model to detect objects in an image. Make sure to follow the installation instructions (https://github.com/tensorflow/models/blob/master/object_detection/g3doc/installation.md) before you start.

Imports

In [2]:
```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
```

Env setup

In [3]:
```
# This is needed to display the images.
%matplotlib inline

# This is needed since the notebook is stored in the object_detection
 folder.
sys.path.append("..")
```

Object detection imports

Here are the imports from the object detection module.

In [4]:
```
from utils import label_map_util

from utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_CKPT` to point to a new .pb file.

By default we use an "SSD with Mobilenet" model here. See the detection model zoo (https://github.com/tensorflow/models/blob/master/object_detection/g3doc/detection_model_zoo.md) for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

In [24]:
```
# What model to download.
MODEL_NAME = 'faster_rcnn_inception_resnet_v2_atrous_coco_11_06_2017'
#MODEL_NAME = 'faster_rcnn_resnet101_coco_11_06_2017'
#MODEL_NAME = 'rfcn_resnet101_coco_11_06_2017'

# These models are unaffected
#MODEL_NAME = 'ssd_inception_v2_coco_11_06_2017'
#MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'

MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detecti
on/'

# Path to frozen detection graph. This is the actual model that is us
ed for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90
```

Download Model

In [ ]:
```
opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
```

Load a (frozen) Tensorflow model into memory.

In [25]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts 5, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [26]:
```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories =
label_map_util.convert_label_map_to_categories(label_map, max_num_cla
sses=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

Helper code

In [27]:
```
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)
```

Detection

In [28]:
```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the
 images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.j
pg'.format(i)) for i in range(1, 3) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

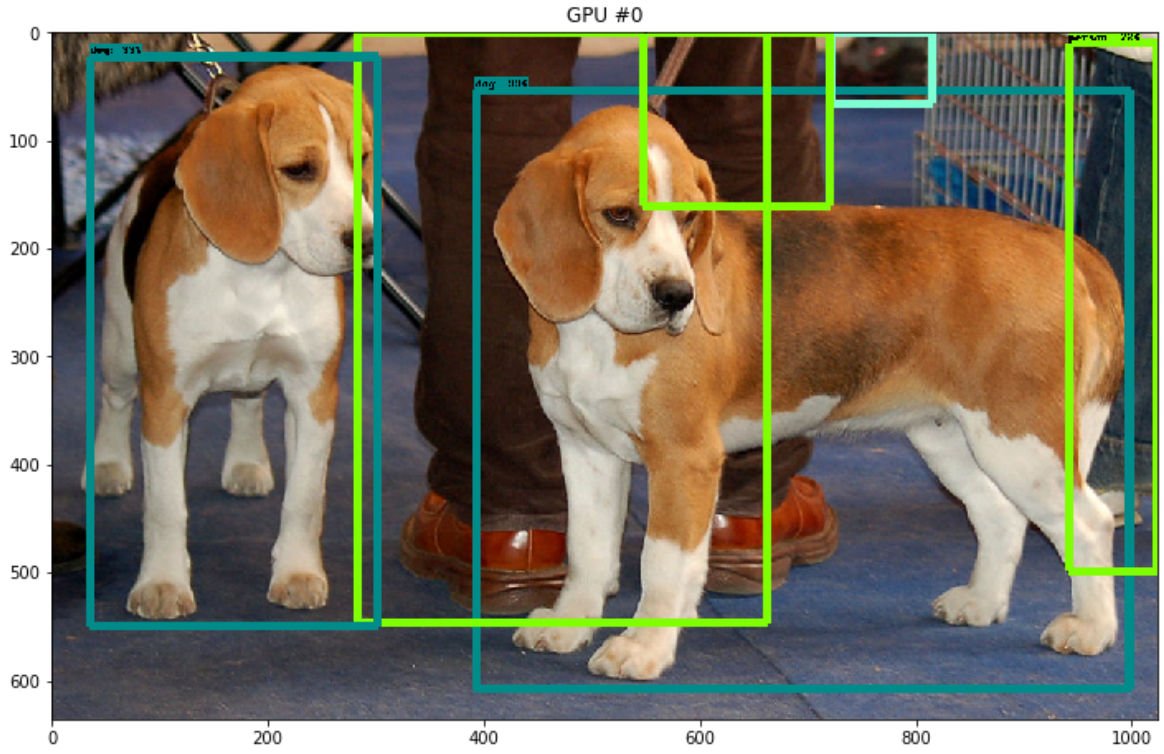
Set up separate GPU configs and evaluate on one image

In [29]:
```
config0 = tf.ConfigProto()
config0.gpu_options.visible_device_list = '0'

config1 = tf.ConfigProto()
config1.gpu_options.visible_device_list = '1'

config2 = tf.ConfigProto()
config2.gpu_options.visible_device_list = '2'

configs = [config0, config1, config2]
```

Run on GPUs

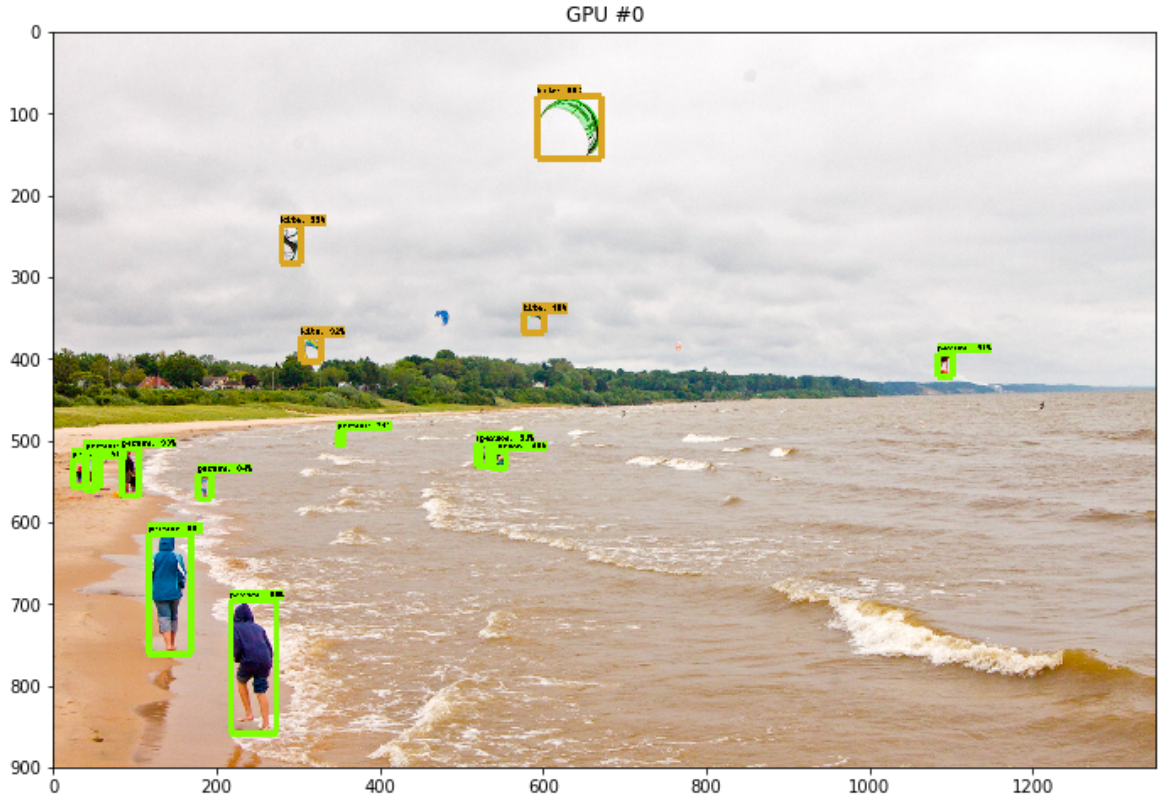
In [30]:
```
with detection_graph.as_default():
  for idx_image, image_path in enumerate(TEST_IMAGE_PATHS):
      print('\n\nEvaluating image %d'%idx_image)
      for idx_gpu, config in enumerate(configs):
        print('\n\tRunning on GPU #%d'%idx_gpu)
        print('\t\tTop 4 box scores: ')
        for idx_repeat in range(3):
            with tf.Session(graph=detection_graph, config=config) as
sess:
              image = Image.open(image_path)
              # the array based representation of the image will be u
sed later in order to prepare the
              # result image with boxes and labels on it.
              image_np = load_image_into_numpy_array(image)
              # Expand dimensions since the model expects images to h
ave shape: [1, None, None, 3]
              image_np_expanded = np.expand_dims(image_np, axis=0)
              image_tensor = detection_graph.get_tensor_by_name('imag
e_tensor:0')
              # Each box represents a part of the image where a parti
cular object was detected.
              boxes = detection_graph.get_tensor_by_name('detection_b
oxes:0')
              # Each score represent how level of confidence for each
 of the objects.
              # Score is shown on the result image, together with the
 class label.
              scores = detection_graph.get_tensor_by_name('detection_
scores:0')
              classes = detection_graph.get_tensor_by_name('detection
_classes:0')
              num_detections = detection_graph.get_tensor_by_name('nu
m_detections:0')
              # Actual detection.
              (boxes, scores, classes, num_detections) = sess.run(
                  [boxes, scores, classes, num_detections],
                  feed_dict={image_tensor: image_np_expanded})

              print('\t\t' + str(scores[0,:4]))
              # Visualization of the results of a detection.
              if idx_repeat == 0:
                  vis_util.visualize_boxes_and_labels_on_image_array(
                      image_np,
                      np.squeeze(boxes),
                      np.squeeze(classes).astype(np.int32),
                      np.squeeze(scores),
                      category_index,
                      use_normalized_coordinates=True,
                      line_thickness=8)
                  plt.figure(figsize=IMAGE_SIZE)
                  plt.imshow(image_np)
                  plt.title('GPU #%d'%idx_gpu)
```

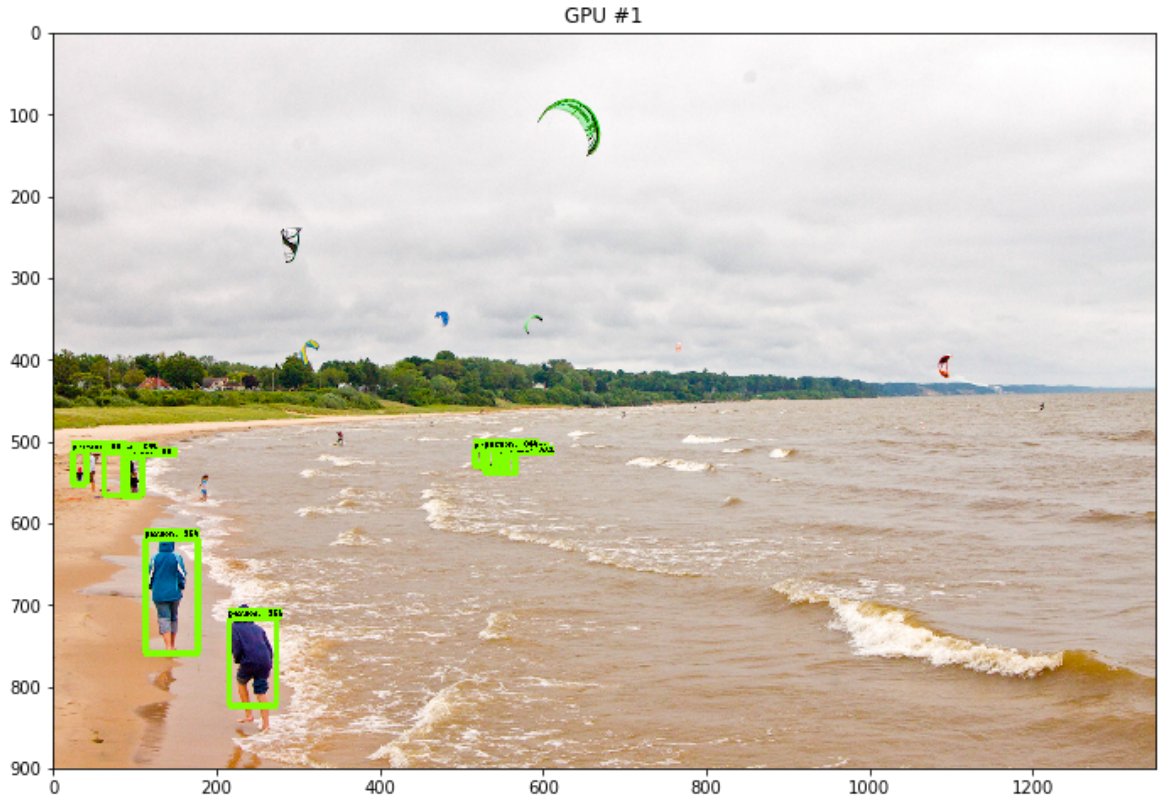
```
Evaluating image 0

        Running on GPU #0
                Top 4 box scores:
                [ 0.99978215  0.99857557  0.95300484  0.91580492]
                [ 0.99978215  0.99857557  0.95300484  0.91580492]
                [ 0.99978215  0.99857557  0.95300484  0.91580492]

        Running on GPU #1
                Top 4 box scores:
                [ 0.68702352  0.16781448  0.13143283  0.12993629]
                [ 0.18502565  0.16854601  0.08074528  0.07859289]
                [ 0.18502565  0.16854601  0.05546702  0.05111229]

        Running on GPU #2
                Top 4 box scores:
                [ 0.68702352  0.16781448  0.13143283  0.12993629]
                [ 0.18941374  0.18502565  0.16854601  0.16230994]
                [ 0.18502565  0.16854601  0.05546702  0.05482833]


Evaluating image 1

        Running on GPU #0
                Top 4 box scores:
                [ 0.99755412  0.99750346  0.99380219  0.99067008]
                [ 0.99755412  0.99750346  0.99380219  0.99067008]
                [ 0.99755412  0.99750346  0.99380219  0.99067008]

        Running on GPU #1
                Top 4 box scores:
                [ 0.96881998  0.96441168  0.96164131  0.96006596]
                [ 0.9377929   0.91686022  0.80374646  0.79758978]
                [ 0.90396696  0.89217037  0.85456908  0.85334581]

        Running on GPU #2
                Top 4 box scores:
                [ 0.9377929   0.91686022  0.80374646  0.79758978]
                [ 0.9377929   0.91686022  0.80374646  0.79758978]
                [ 0.9377929   0.91686022  0.80374646  0.79758978]
```

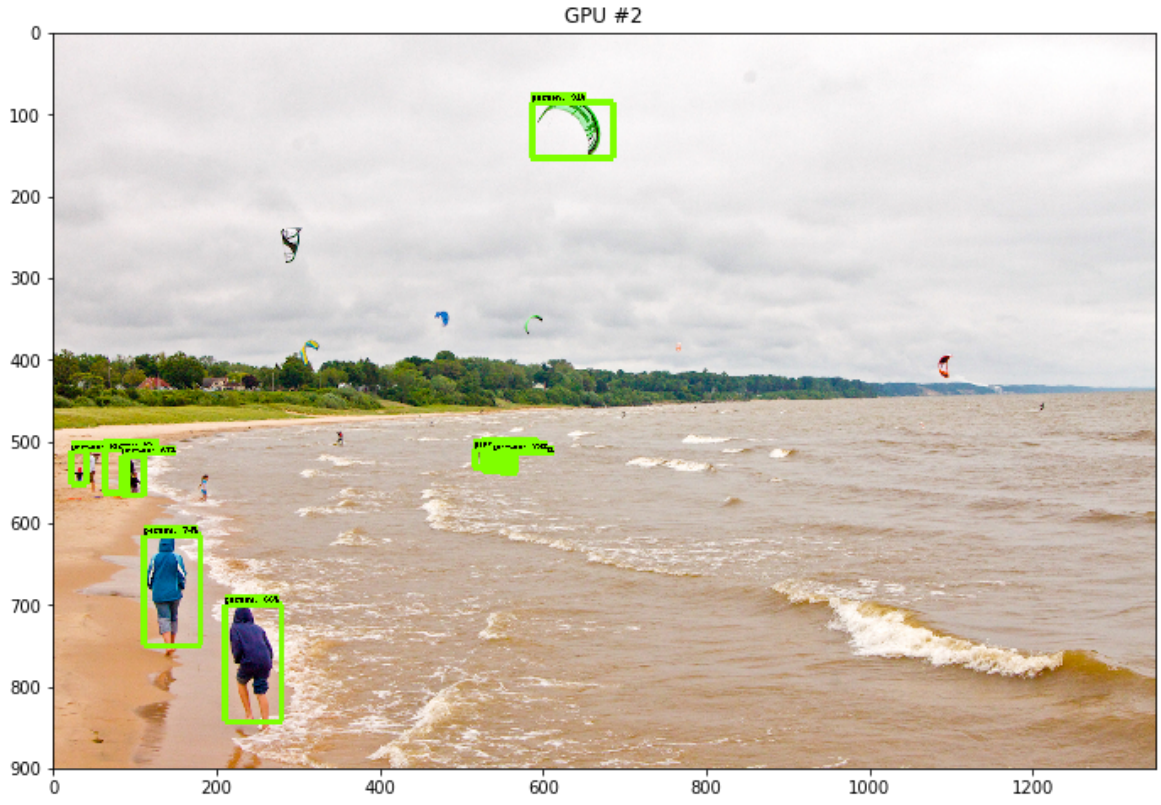

GPU #0
0
100
200
300
400
500
600
0
200
400
600
800
1000


GPU #1
0
100
200
300
400
500
600
0
200
400
600
800
1000


GPU #2
0
100
200
300
400
500
600
0
200
400
600
800
1000


GPU #0
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200


GPU #1
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200


GPU #2
0
100
200
300
400
500
600
700
800
900
0
200
400
600
800
1000
1200

In [ ]: