object_detection_tutorial

March 28, 2019

1 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 before you start.

2 Imports

In [21]:
```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to v1.12.*.')
```

2.1 Env setup

In [22]:
```
# This is needed to display the images.
%matplotlib inline
```

2.2 Object detection imports

Here are the imports from the object detection module.

```
In [23]: from utils import label_map_util

         from utils import visualization_utils as vis_util
```

3 Model preparation

3.1 Variables

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

By default we use an "SSD with Mobilenet" model here. See the detection model zoo 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 = 'dino_chrome'

         # Path to frozen detection graph. This is the actual model that is used for the object
         PATH_TO_FROZEN_GRAPH = 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', 'dino-chrome.pbtxt')
```

3.2 Download Model

```
In [ ]:
```

3.3 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_FROZEN_GRAPH, 'rb') as fid:
             serialized_graph = fid.read()
             od_graph_def.ParseFromString(serialized_graph)
             tf.import_graph_def(od_graph_def, name='')
```

3.4 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]: category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use
```

3.5 Helper code

```
In [27]: def load_image_into_numpy_array(image):
           (im_width, im_height) = image.size
```

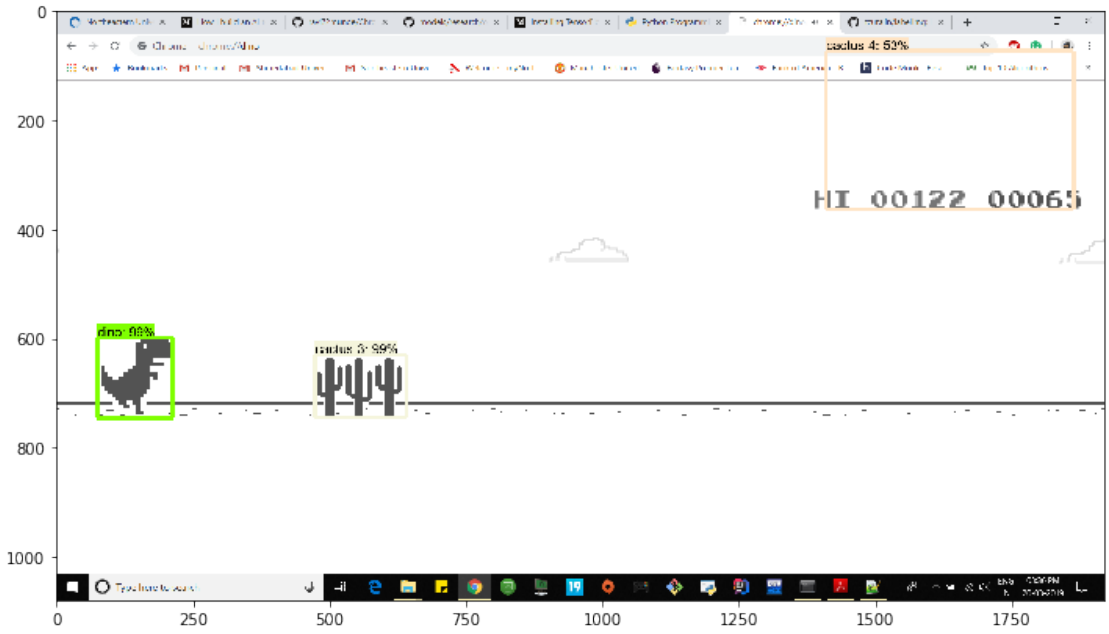
```
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)
```

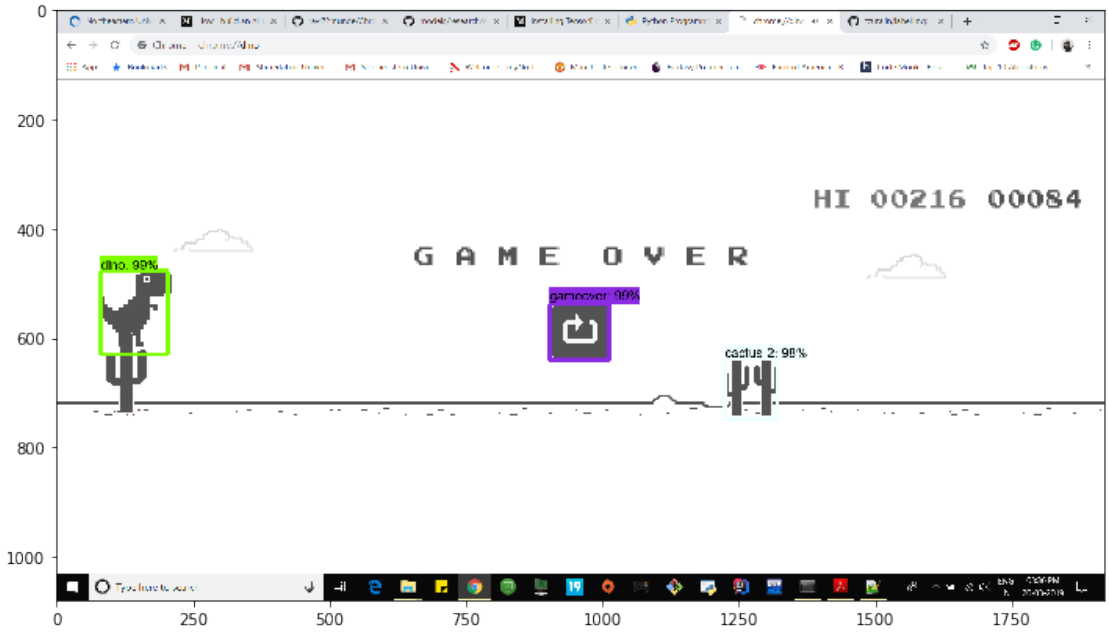
4 Detection

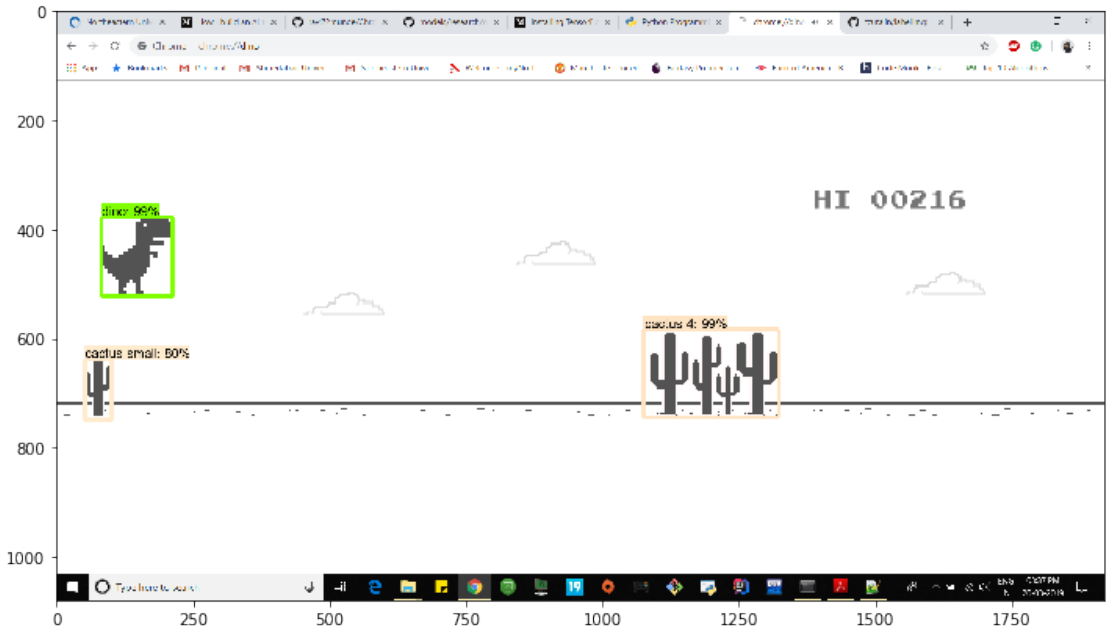
In [28]:
```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the images to the TES
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, '{}.png'.format(i))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

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

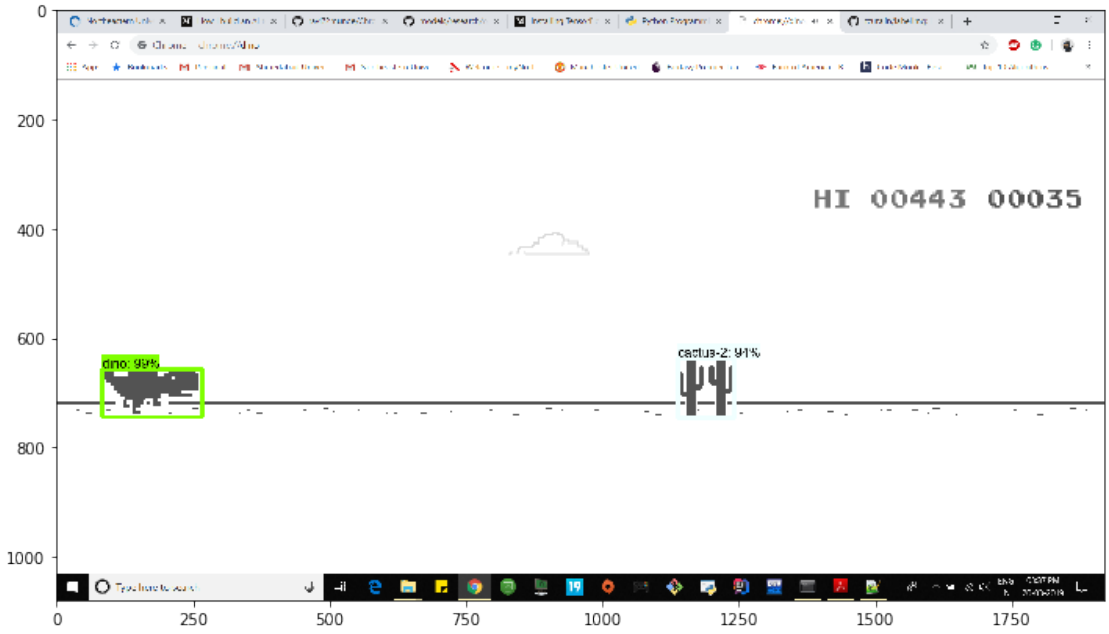
In [29]:
```
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.outputs}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
        # Reframe is required to translate mask from box coordinates to image coordinat
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num_detection, -1,
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')
```

```
            # Run inference
            output_dict = sess.run(tensor_dict,
                                   feed_dict={image_tensor: np.expand_dims(image, 0)})

            # all outputs are float32 numpy arrays, so convert types as appropriate
            output_dict['num_detections'] = int(output_dict['num_detections'][0])
            output_dict['detection_classes'] = output_dict[
                'detection_classes'][0].astype(np.uint8)
            output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
            output_dict['detection_scores'] = output_dict['detection_scores'][0]
            if 'detection_masks' in output_dict:
              output_dict['detection_masks'] = output_dict['detection_masks'][0]
        return output_dict

In [30]: for image_path in TEST_IMAGE_PATHS:
            image = Image.open(image_path)
            # the array based representation of the image will be used later in order to prepare
            # result image with boxes and labels on it.
            image_np = load_image_into_numpy_array(image)
            # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
            image_np_expanded = np.expand_dims(image_np, axis=0)
            # Actual detection.
            output_dict = run_inference_for_single_image(image_np, detection_graph)
            # Visualization of the results of a detection.
            vis_util.visualize_boxes_and_labels_on_image_array(
                image_np,
                output_dict['detection_boxes'],
                output_dict['detection_classes'],
                output_dict['detection_scores'],
                category_index,
                instance_masks=output_dict.get('detection_masks'),
                use_normalized_coordinates=True,
                line_thickness=8)
            plt.figure(figsize=IMAGE_SIZE)
            plt.imshow(image_np)
```


cactus 4: 53%
HI 00122 00065
dino: 98%
cactus 3: 99%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00216 00084
G A M E O V E R
dino: 98%
gameover: 99%
cactus 2: 98%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

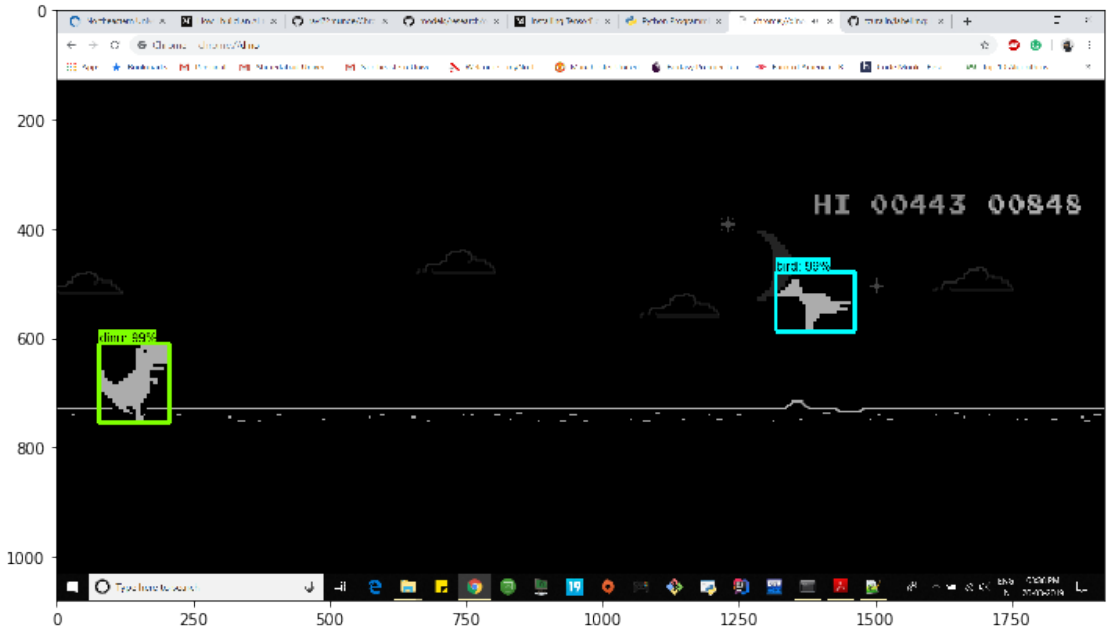

HI 00216
dino: 99%
cactus small: 80%
cactus 4: 99%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00443 00035
dino: 99%
cactus-2: 94%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

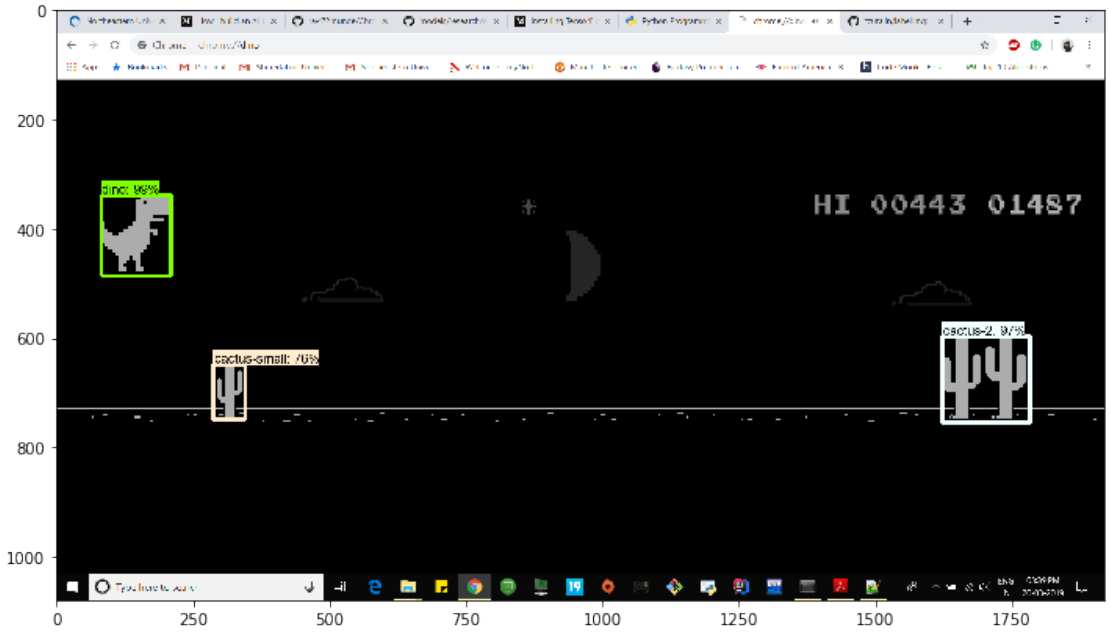

dino: 89%
bird: 99%
HI 00443 00489
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


bird: 97%
dino: 99%
HI 00443 00571
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00443
cactus-small: 95%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00443 00848
bird: 98%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00443
dino: 99%
cactus-3: 78%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 00443 01487
dino: 99%
cactus-small: 76%
cactus-2: 97%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750


HI 01585 01585
GAME OVER
gameover: 99%
dino: 99%
bird: 70%
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750