Gorillas Masking Code

Author: Huichun Yang

- **Date:** 7/16/2020
- **Affiliation:** George Washington University

Object Detection

This Object Detection is based on Google Tensorflow object detection inference. instructions

Detection Neural Network is using Mask RCNN.

This Object Detection part is only used for gorillas detection.

Imports ¶

In [ ]:
```
%pylab inline
%matplotlib inline
import time
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
from sklearn.svm import SVC

import skimage.io as skio
import skimage.color as skcolor
from scipy import ndimage as ndi
from skimage import measure
import math


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

# 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('1.9.0'):
    raise ImportError('Please upgrade your TensorFlow installation to v1.9.* or later!')
```

Object detection imports

Here are the imports from the object detection module.

In [ ]:
```
from utils import label_map_util

from utils import visualization_utils as vis_util

from utils import ops as utils_ops
```

Model preparation

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 [ ]:
```
# What model to download.
# MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_NAME = 'mask_rcnn_inception_v2_coco_2018_01_28'
# MODEL_NAME = 'faster_rcnn_resnet101_coco_2018_01_28'
# MODEL_NAME = 'ssd_mobilenet_v1_coco_2018_01_28'

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

# Path to frozen detection graph. This is the actual model that is used for the object detection.
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', 'mscoco_label_map.pbtxt')
```

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 [ ]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.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='')
```

Loading label map

The `Label map` tool maps indices to category names. For example, 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 [ ]:
```
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

Defining functions

In [ ]:
```
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 [ ]:
```
# Input actual laser distance in cm
LASER_DIS = 4

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

In [ ]:
```
def run_inference_for_single_image(image, graph):
    with tf.device('/device:GPU:1'):
        with graph.as_default():
            with tf.compat.v1.Session() as sess:

                # Get handles to input and output tensors.
                ops = tf.compat.v1.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.compat.v1.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 coordinates and fit the imag
e size.
                    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, -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.compat.v1.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 [ ]:
```
PATH_TO_TEST_IMAGES_DIR = 'G:/image/'
time0 = time.time()
os.mkdir('output/')
for folder in os.listdir(PATH_TO_TEST_IMAGES_DIR)[:]:
    if not os.path.exists('output/'+folder):
        os.mkdir('output/'+folder)
    img_folder = os.path.join(PATH_TO_TEST_IMAGES_DIR,folder)
    TEST_IMAGE_PATHS = [ os.path.join(img_folder,img ) for img in os.listdir(img_folder) ]
    for idx,image_path in enumerate(TEST_IMAGE_PATHS[:]):
        print('Picture:', str(idx),os.listdir(img_folder)[idx])
        time2 = time.time()
        image = Image.open(image_path)
        # the array based representation of the image will be used 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 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=18)

        masked = skio.imread(image_path)
        total_mask = np.zeros_like(masked[:,:,0])
        for single_mask in output_dict.get('detection_masks'):
            total_mask = np.logical_or(total_mask,single_mask)
        masked[:,:,0] *= total_mask
        masked[:,:,1] *= total_mask
        masked[:,:,2] *= total_mask

        plt.imsave('output/' + folder + '/' + os.listdir(img_folder)[idx] + '.jpg', masked)
        time3 = time.time()
        print("Single time"+str(time3-time2))
time1 = time.time()
print("Total time"+str(time1-time0))
```