



## 基于 TensorFlow 实现的视频中物体的识别系统

学院名称\_\_\_\_\_计算机学院\_\_\_\_\_

班 级\_\_\_\_\_1 班\_\_\_\_\_

学生姓名\_\_\_\_\_张眸\_\_\_\_\_

学 号\_\_\_\_\_3015216030\_\_\_\_\_

签 名\_\_\_\_\_

## 目录

一、概述 .....	3
1、项目名称.....	3
2、项目介绍.....	3
3、项目目标.....	3
4、核心原理.....	3
5、相关开源技术与库.....	3
二、文件解释.....	3
三、程序描述.....	4
四、进一步展望.....	11

## 一、概述

1、项目名称：基于 TensorFlow 实现的视频中物体的识别系统

2、项目介绍：

谷歌公司在 2017 CVPR 上推出了全新的机器学习算法，在新的算法的指导下，图像中物体检测的准确率不断提升。谷歌公司将他们的算法开源，并嵌入 TensorFlow 中，供全世界使用。在本课设中，我利用谷歌公司提供的 API，实现图像的物体检测功能，并在其之上，进一步实现对于视频中物体的检测。此外，我还实现了通过 opencv 调用系统摄像头，实现了现场录制视频中物体的检测。

3、项目目标：（1）通过调用谷歌 api 进行视频中的物体识别。（2）通过 opencv 调用系统摄像头，采集观察点周围的数据，并对这些数据进行物体识别分类。

4、核心原理：（1）通过谷歌在 TensorFlow 上提供的 API，可以对图片进行物体检测。在此基础上，我使用 moviepy，将视频分解为图片，并将每张图片都进行物品检测，最终再将检测完成的图片全部拼接起来，就得到可视频的物体检测。（2）通过 python 调用 opencv 的 api，接入摄像头并录像，再通过视频物体检测系统进行识别。

5、相关开源技术与库：TensorFlow, OpenCV , moviepy, ipython, jupyter 等。

## 二、文件解释

1、文件夹 objective\_detection:谷歌 api 自带文件，其中保存了示例图片和训练模型。

2、Object\_Detection\_In\_Video.ipynb: 主程序，使用 ipython 进行编写，可以进行图像中的物体检测（示例图像，位于 objective\_detection 中，无需改动）和视频中的物体检测（需将待检测视频命名为 video1.mp4 或 video2.mp4，并置于与此文件同一目录下）（为了保证检测时间不过长，建议的视频长度为 5s 之内）

3、video1.mp4, video2.mp4: 待检测视频

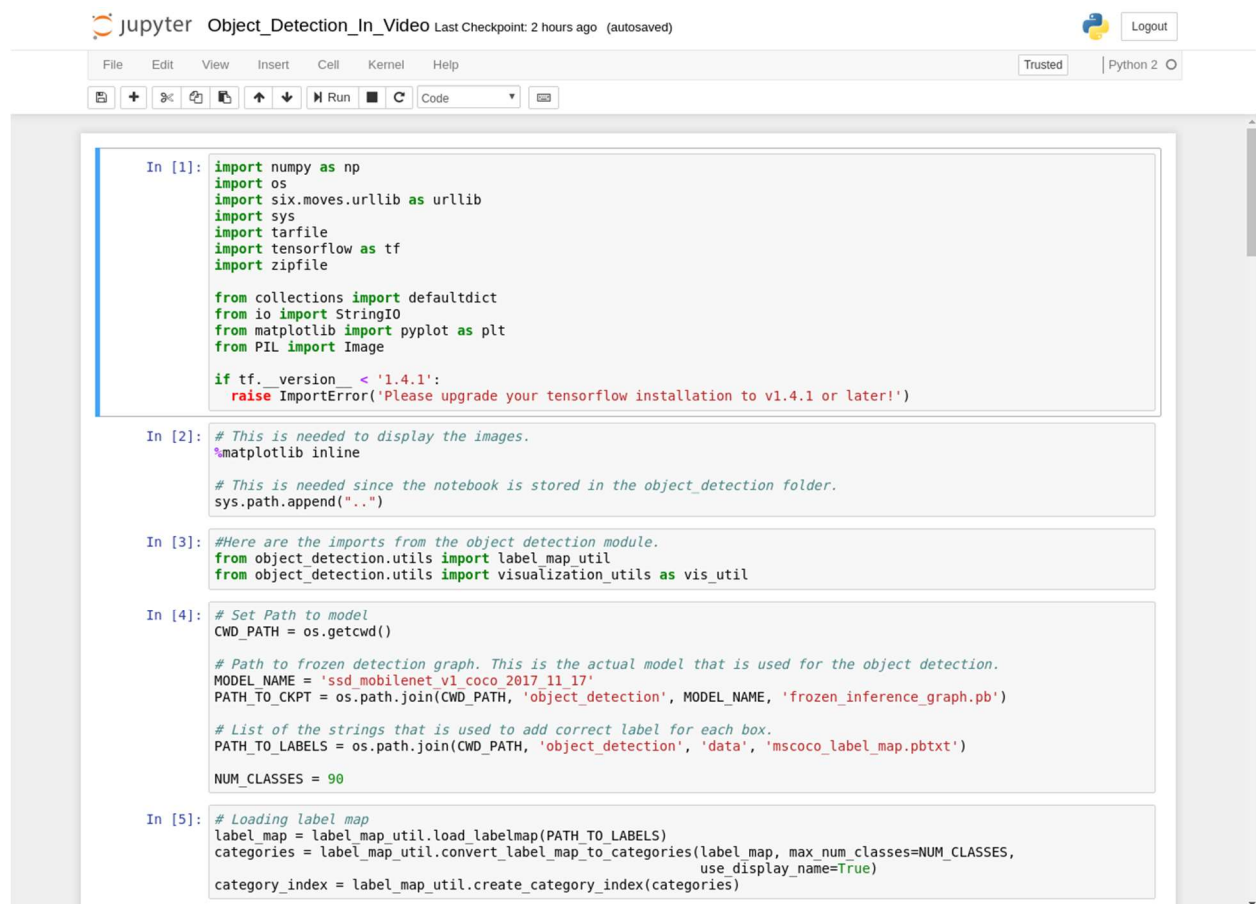
4、video1\_out.mp4, video2\_out.mp4: 检测完成的视频。

5、record.py: 通过调用 opencv 的函数, 打开摄像头进行录像, 并将录像保存在 video1.mp4 中。按 q 可以退出, 或者会在一段时间之后自动退出。

注意: 原文件由于大小所限, 没有将模型进行上传, 运行时, 应先在 [https://github.com/tensorflow/models/blob/477ed41e7e4e8a8443bc633846eb01e2182dc68a/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/477ed41e7e4e8a8443bc633846eb01e2182dc68a/object_detection/g3doc/detection_model_zoo.md) 下载模型, 并将其解压后的文件夹命名为 ssd\_mobilenet\_v1\_coco\_2017\_11\_17 方可使用。)

### 三、程序描述

Object\_Detection\_In\_Video.ipynb 程序如下图所示:



```
In [1]: 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

if tf.__version__ < '1.4.1':
    raise ImportError('Please upgrade your tensorflow installation to v1.4.1 or later!')
```

```
In [2]: # 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("..")
```

```
In [3]: #Here are the imports from the object detection module.
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

```
In [4]: # Set Path to model
CWD_PATH = os.getcwd()

# Path to frozen detection graph. This is the actual model that is used for the object detection.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
PATH_TO_CKPT = os.path.join(CWD_PATH, 'object_detection', 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(CWD_PATH, 'object_detection', 'data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90
```

```
In [5]: # Loading label map
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NUM_CLASSES,
    use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

```
In [6]: # Load a frozen TF model
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='')

In [7]: # Helper code
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)

In [8]: # First test on images
PATH_TO_TEST_IMAGES_DIR = 'object_detection/test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3) ]

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

In [9]: # Detection function
def detect_objects(image_np, sess, detection_graph):
    # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
    image_np_expanded = np.expand_dims(image_np, axis=0)
    image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')

    # Each box represents a part of the image where a particular object was detected.
    boxes = detection_graph.get_tensor_by_name('detection_boxes:0')

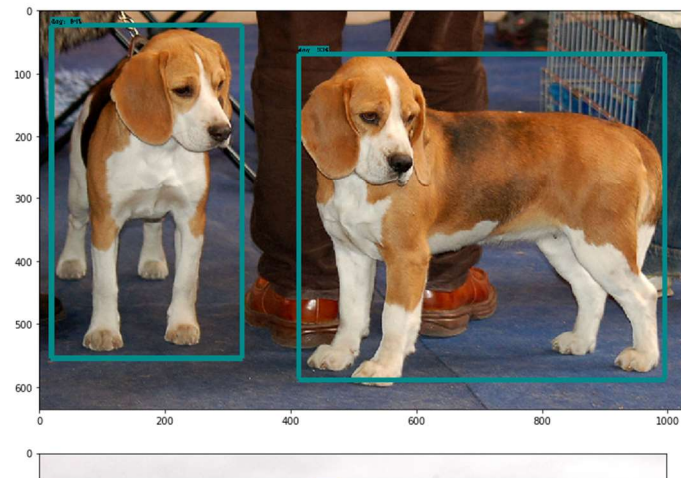
    # Each score represent how level of confidence for each of the objects.
    # Score is shown on the result image, together with the class label.
    scores = detection_graph.get_tensor_by_name('detection_scores:0')
    classes = detection_graph.get_tensor_by_name('detection_classes:0')
    num_detections = detection_graph.get_tensor_by_name('num_detections:0')

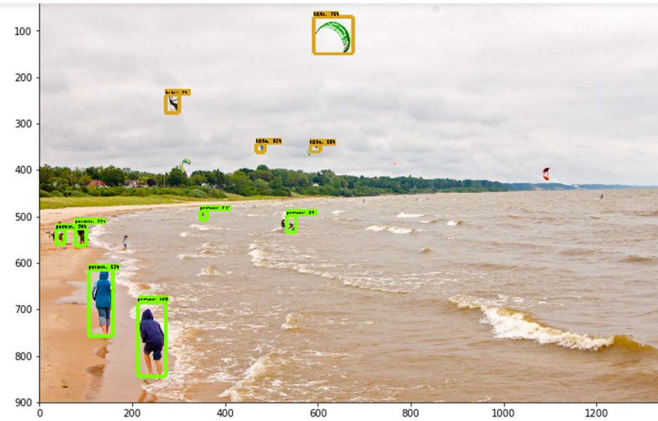
    # Actual detection.
    (boxes, scores, classes, num_detections) = sess.run(
        [boxes, scores, classes, num_detections],
        feed_dict={image_tensor: image_np_expanded})

    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        np.squeeze(boxes),
        np.squeeze(classes).astype(np.int32),
        np.squeeze(scores),
        None,
        True,
        True)
```

```
np.squeeze(boxes),
np.squeeze(classes).astype(np.int32),
np.squeeze(scores),
category_index,
use_normalized_coordinates=True,
line_thickness=8)
return image_np
```

```
In [10]: #Detection
with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:
        for image_path in TEST_IMAGE_PATHS:
            image = Image.open(image_path)
            image_np = load_image_into_numpy_array(image)
            image_process = detect_objects(image_np, sess, detection_graph)
            plt.figure(figsize=IMAGE_SIZE)
            plt.imshow(image_process)
```





```
In [11]: import imageio
imageio.plugins.ffmpeg.download()
from moviepy.editor import VideoFileClip
from IPython.display import HTML
```

```
In [12]: def process_image(image):
# NOTE: The output you return should be a color image (3 channel) for processing video below
# you should return the final output (image with lines are drawn on lanes)
with detection_graph.as_default():
with tf.Session(graph=detection_graph) as sess:
image_process = detect_objects(image, sess, detection_graph)
return image_process
```

```
In [13]: import gc
gc.collect();
white_output = 'video1_out.mp4'
clip1 = VideoFileClip("video1.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects color images!!
time white_clip.write_videofile(white_output, audio=False)
```

[MoviePy] >>> Building video video1\_out.mp4



Run Code

[MoviePy] Writing video video1\_out.mp4

100% | 84/84 [02:02<00:00, 1.48s/it]

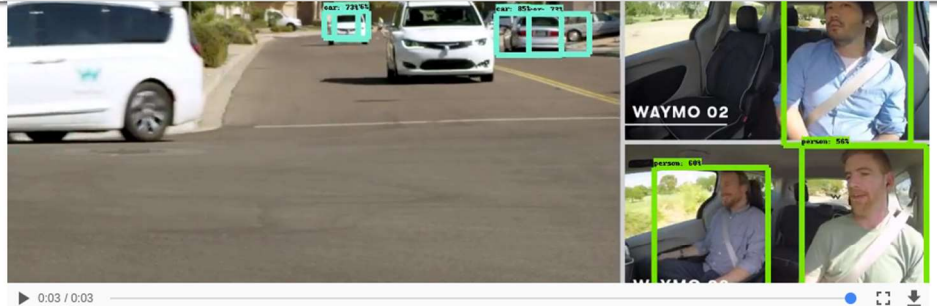
[MoviePy] Done.

[MoviePy] >>> Video ready: video1\_out.mp4

CPU times: user 2min 26s, sys: 7.66 s, total: 2min 34s

Wall time: 2min 3s

```
In [14]: HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""").format(white_output)
```



```
In [15]: gc.collect()
white_output1 = 'video2_out.mp4'
clip1 = VideoFileClip("video2.mp4")
white_clip = clip1.fl_image(process_image) #NOTE: this function expects color images!!
%time white_clip.write_videofile(white_output1, audio=False)
```

[MoviePy] >>> Building video video2\_out.mp4

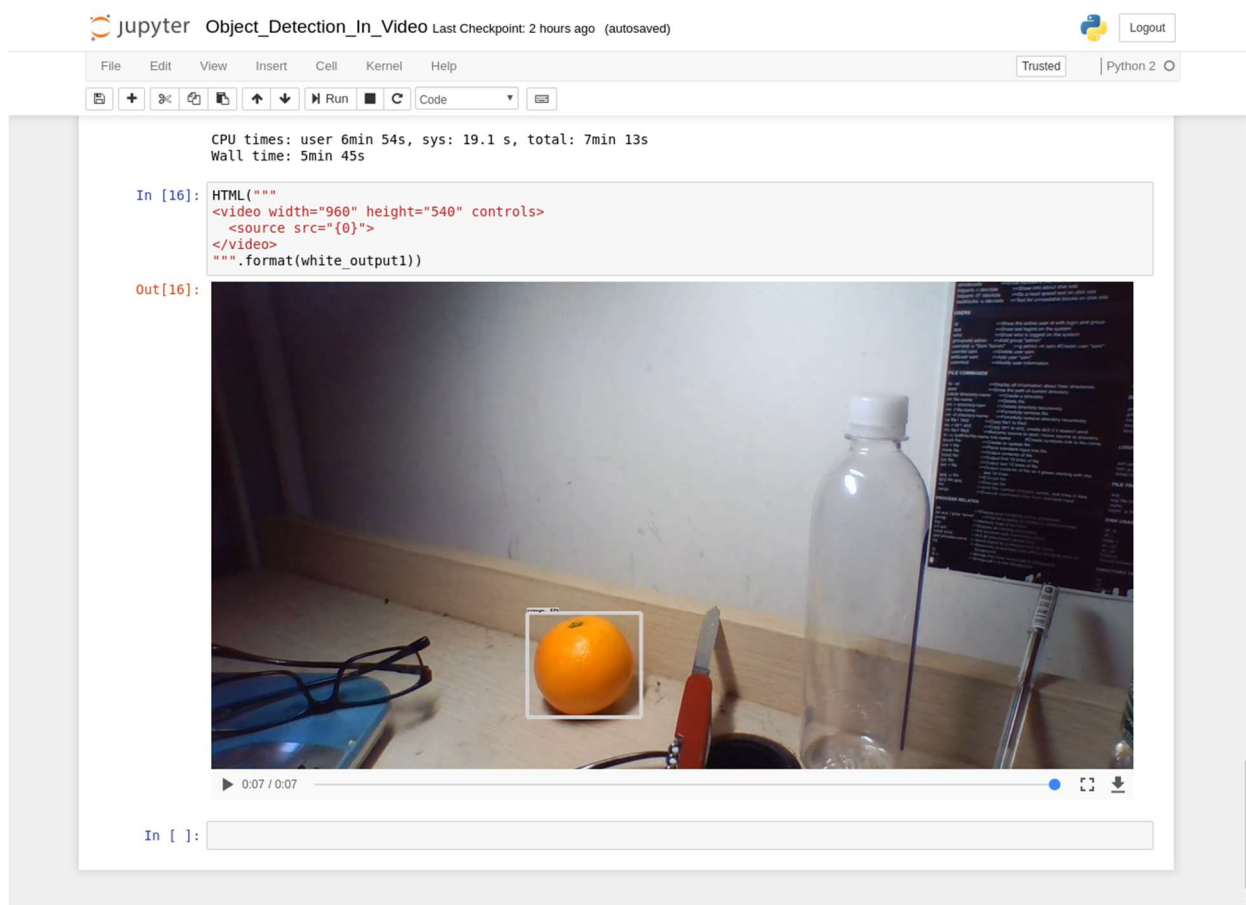
[MoviePy] Writing video video2\_out.mp4

100% | 215/215 [05:41<00:00, 1.61s/it]

[MoviePy] Done.

[MoviePy] >>> Video ready: video2\_out.mp4





块 1, 2, 3: 引入必要的包

块 4, 5, 6: 加载模型

块 7: 图像读入函数

块 8: 图像中物体检测函数, 通过调用谷歌提供的 api 进行图像中物体的检测。

块 9, 10: 用测试图像检测是否识别成功

块 11: 引入处理视频时所需要的库

块 12: 定义图像处理函数

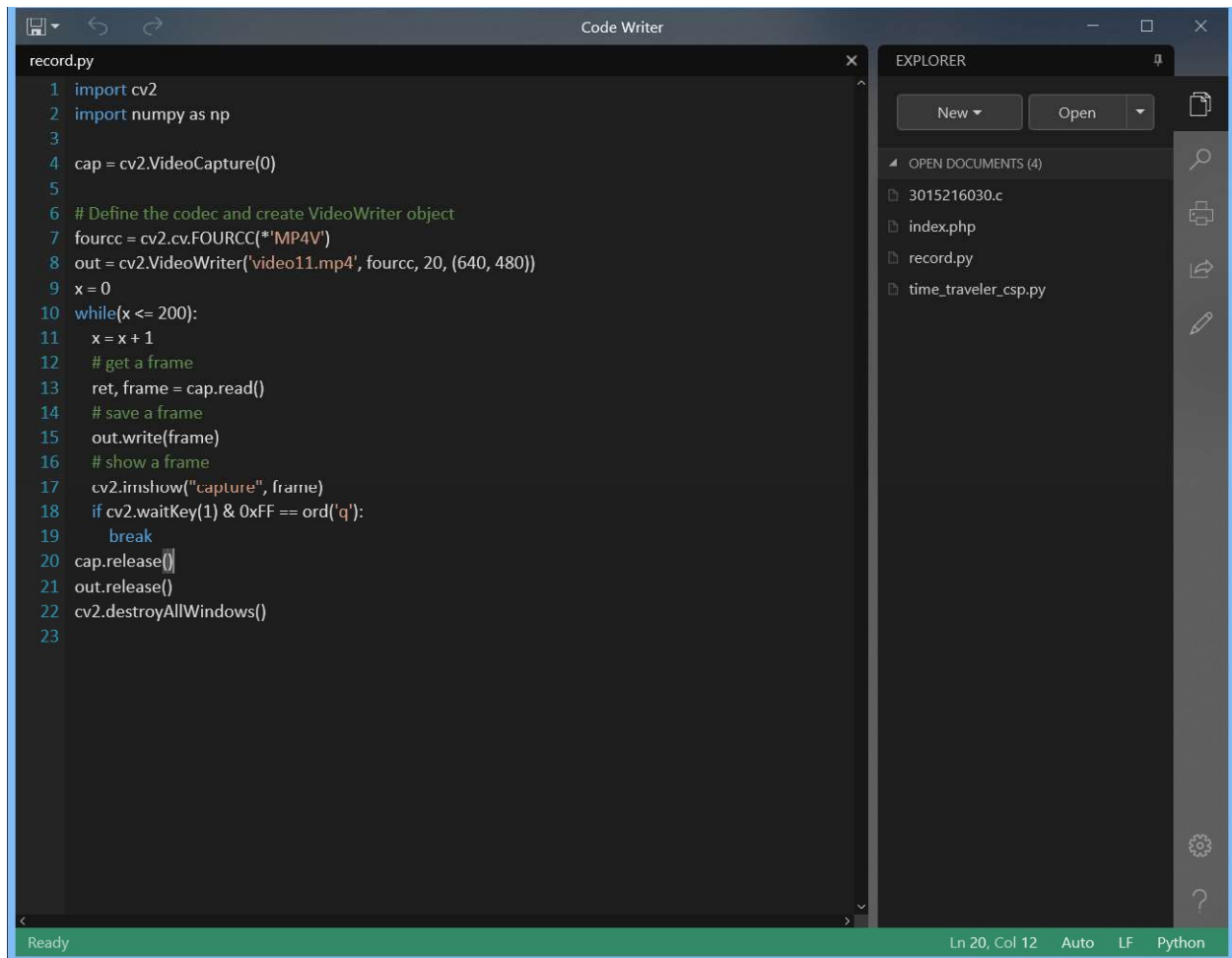
块 13: 对于 video1 进行处理。该部分将原图像按照其本身的模式分开变为大量图片的集合, 再将每一张图片单独用物体检测函数进行识别, 最后将识别的结果拼接在一起变为视频, 该视频即为进行物体检测后的视频。

块 14: 将处理过的 video1 进行展示。

块 15：处理 video2。其过程与块 13 相同。

块 16：将处理过的 video2 进行展示。

record.py:



```
record.py
1 import cv2
2 import numpy as np
3
4 cap = cv2.VideoCapture(0)
5
6 # Define the codec and create VideoWriter object
7 fourcc = cv2.cv.FOURCC(*'MP4V')
8 out = cv2.VideoWriter('video11.mp4', fourcc, 20, (640, 480))
9 x = 0
10 while(x <= 200):
11     x = x + 1
12     # get a frame
13     ret, frame = cap.read()
14     # save a frame
15     out.write(frame)
16     # show a frame
17     cv2.imshow("capture", frame)
18     if cv2.waitKey(1) & 0xFF == ord('q'):
19         break
20 cap.release()
21 out.release()
22 cv2.destroyAllWindows()
23
```

`cap = cv2.VideoCapture(0)` :开启主摄像头

`fourcc = cv2.cv.FOURCC(*'MP4V')` : 定义录制的视频类型

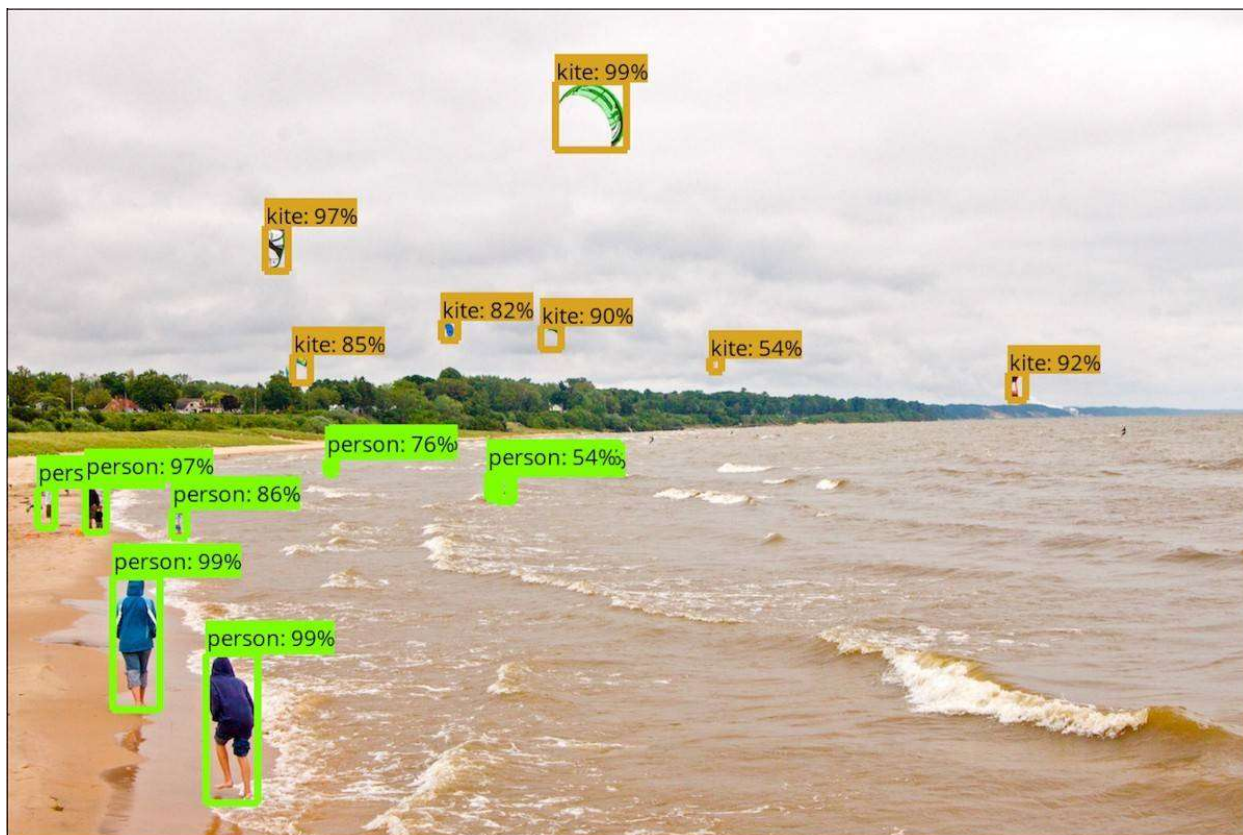
`out = cv2.VideoWriter('video11.mp4', fourcc, 20, (640, 480))` : 存储录像

while 循环：每次都将从摄像头得到的帧存储到 out 中，知道循环了 200 次或者键盘输入 q 推出。

经过如上操作，视频便会被保存到 out 中了。之后再调用视频识别即可。

四、结果展示：

图片效果展示：



动态图效果展示：

[https://github.com/mosquitozm100/OPENSOURCE/blob/master/video2\\_out.gif](https://github.com/mosquitozm100/OPENSOURCE/blob/master/video2_out.gif)

[https://github.com/mosquitozm100/OPENSOURCE/blob/master/video1\\_out\\_2.gif](https://github.com/mosquitozm100/OPENSOURCE/blob/master/video1_out_2.gif)

[https://github.com/mosquitozm100/OPENSOURCE/blob/master/video1\\_out.gif](https://github.com/mosquitozm100/OPENSOURCE/blob/master/video1_out.gif)

#### 四、进一步展望

- 实时周边物体检测系统
- 瓶颈：图像处理速度过慢
- 解决方案：多进程并发处理