# Tensorflow Object Detection API的使用

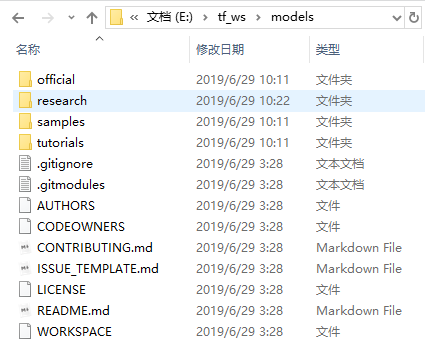
这个文档基于以下教程编写：

https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10

## 环境准备

首先，安装好Tensowflow环境，我的环境是：

下载Tensorflow/models仓库，地址：<https://github.com/tensorflow/models.git>，在E:\tf\_ws目录下：



下载仓库：

https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10，解压所有文件到models/research/object\_detection目录下。

添加环境变量：PYTHONPATH



编译protobuf。在\models\research目录下执行：

|  |
| --- |
| protoc --python\_out=. .\object\_detection\protos\anchor\_generator.proto .\object\_detection\protos\argmax\_matcher.proto .\object\_detection\protos\bipartite\_matcher.proto .\object\_detection\protos\box\_coder.proto .\object\_detection\protos\box\_predictor.proto .\object\_detection\protos\eval.proto .\object\_detection\protos\faster\_rcnn.proto .\object\_detection\protos\faster\_rcnn\_box\_coder.proto .\object\_detection\protos\grid\_anchor\_generator.proto .\object\_detection\protos\hyperparams.proto .\object\_detection\protos\image\_resizer.proto .\object\_detection\protos\input\_reader.proto .\object\_detection\protos\losses.proto .\object\_detection\protos\matcher.proto .\object\_detection\protos\mean\_stddev\_box\_coder.proto .\object\_detection\protos\model.proto .\object\_detection\protos\optimizer.proto .\object\_detection\protos\pipeline.proto .\object\_detection\protos\post\_processing.proto .\object\_detection\protos\preprocessor.proto .\object\_detection\protos\region\_similarity\_calculator.proto .\object\_detection\protos\square\_box\_coder.proto .\object\_detection\protos\ssd.proto .\object\_detection\protos\ssd\_anchor\_generator.proto .\object\_detection\protos\string\_int\_label\_map.proto .\object\_detection\protos\train.proto .\object\_detection\protos\keypoint\_box\_coder.proto .\object\_detection\protos\multiscale\_anchor\_generator.proto .\object\_detection\protos\graph\_rewriter.proto .\object\_detection\protos\flexible\_grid\_anchor\_generator.proto .\object\_detection\protos\calibration.proto |

该命令会在models\object\_detection\protos目录下生成pb2.py文件，如果后面执行训练命令时出现pb2找不到的情况，可能是生成的pb2.py文件不完全，查看object\_detection\protos文件夹下还有什么文件没有加入到以上的命令上，加上云再生成一遍。

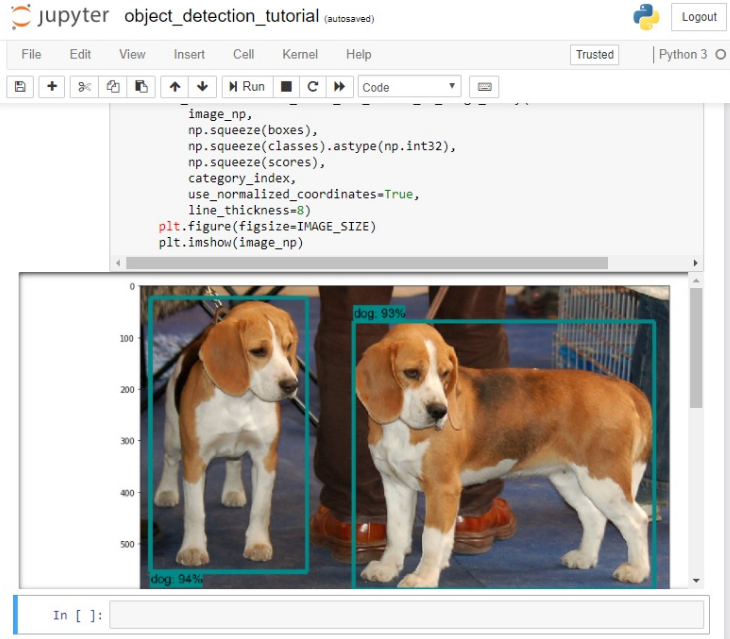
执行setup.py，在models\research目录下，执行：

|  |
| --- |
| python setup.py build  python setup.py install |

确认环境是否正常工作，在object\_detection目录下，执行：

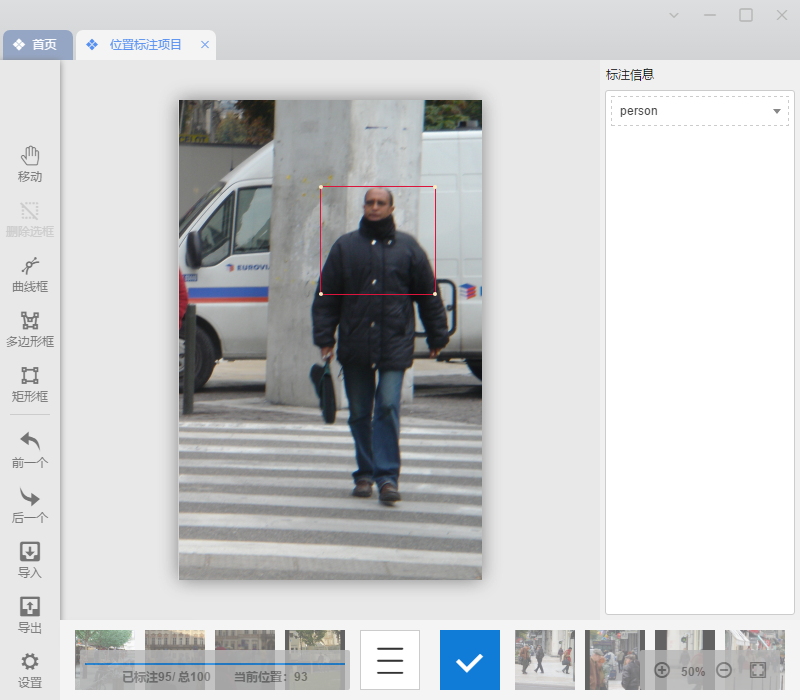
|  |
| --- |
| jupyter notebook object\_detection\_tutorial.ipynb |

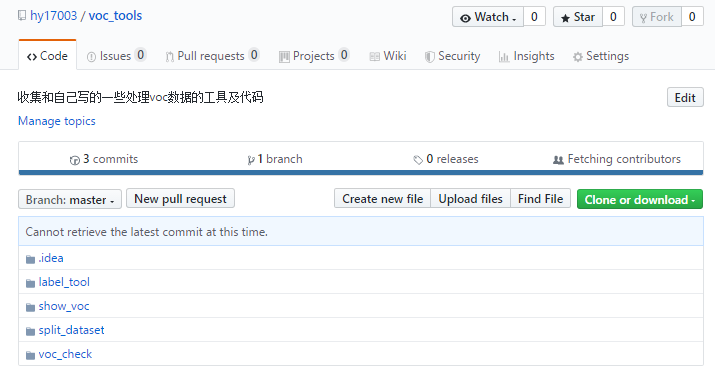
执行各段代码：



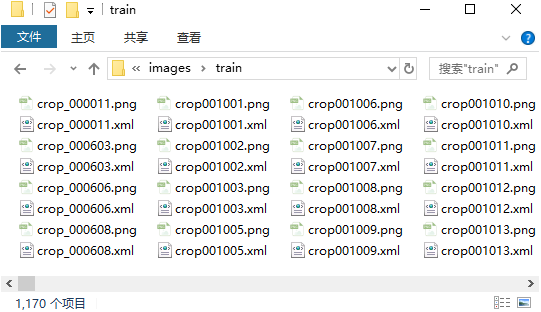
## 数据准备

使用标注精灵助手对图像进行标注，保存框为VOC格式，对标注好的数据，进行检查，如一些标注框超出图像范围，空标注文件等，进行检查修正，最后将数据按一定比例分成训练集和测试集，这里有参考的处理程序和代码：https://github.com/hy17003/voc\_tools





将训练数据包括图像和标注文件放到models\research\object\_detection\images\train目录下：



测试数据包括图像和文件放到models\research\object\_detection\images\test目录下。

## 生成tfRecord文件

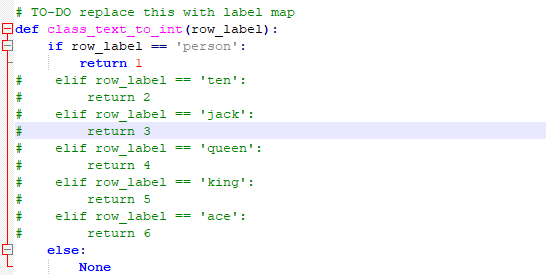
生成csv文件，在object\_detection目录下执行：

|  |
| --- |
| python xml\_to\_csv.py |

在images目录下生成csv文件：



转成tfrecord文件，根据实际的目标类型，修改object\_detection目录下的generate\_tfrecord.py文件：



我这里只检测person这一类型。

执行：

|  |
| --- |
| python generate\_tfrecord.py --csv\_input=images\train\_labels.csv --image\_dir=images\train --output\_path=train.record  python generate\_tfrecord.py --csv\_input=images\test\_labels.csv --image\_dir=images\test --output\_path=test.record |

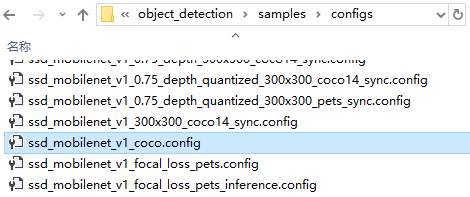
在\object\_detection目录下生成**train.record**和**test.record**文件。

在object\_detection/trainingh目录下创建**labelmap.pbtxt**文件，内容如下：



## 修改训练配置文件

这里使用ssd\_mobilenet\_v1模型进行微调，在object\_detection/samples/configs中找到ssd\_mobilenet\_v1\_coco.config文件，复制到object\_detection\training文件夹中。

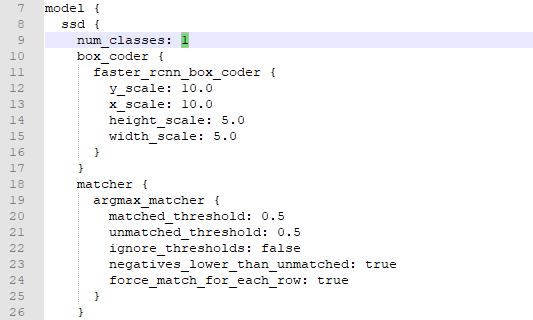


到以下地址下载相应的模型，解压到object\_detecion/training文件夹中：

https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/detection\_model\_zoo.md



编辑ssd\_mobilenet\_v1\_coco.config文件，主要修改类型的数量，这里只有一类，修改为1，相关数据的地址PATH\_TO\_BE\_CONFIGURED，批大小batch\_size，这埯改成16

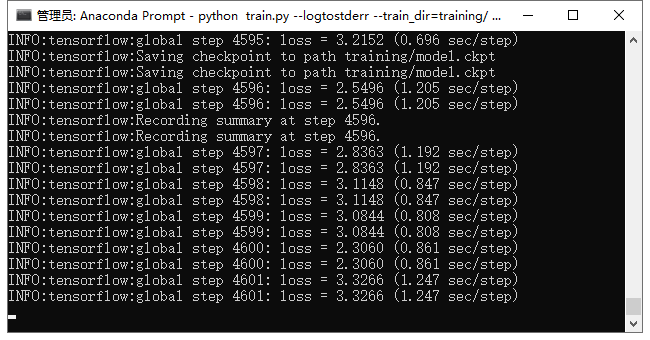


## 训练

将object\_detection\legacy目录下的train.py文件拷贝到object\_detection目录下。

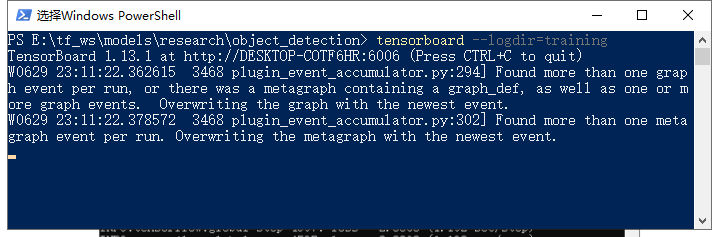
在E:\tf\_ws\models\research\object\_detection目录下执行：

|  |
| --- |
| python train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssd\_mobilenet\_v1\_coco.config |

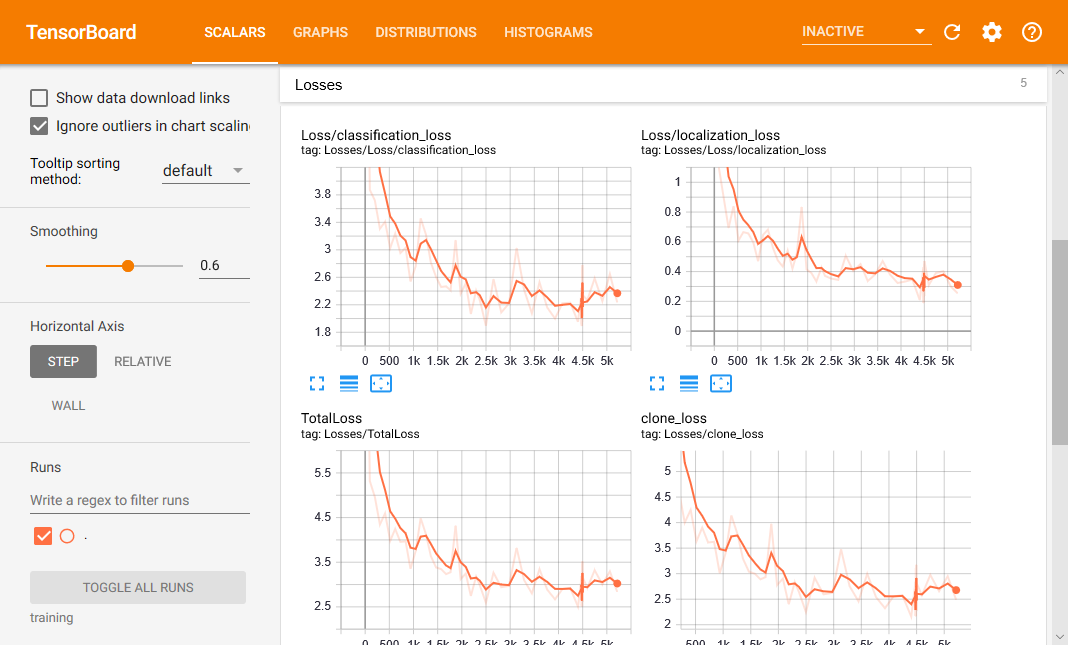


训练过程中，在object\_detection目录下执行：

|  |
| --- |
| tensorboard --logdir=training |



使用浏览器，打开http://desktop-cotf6hr:6006，我这里使用的是Micosoft Edge浏览器：

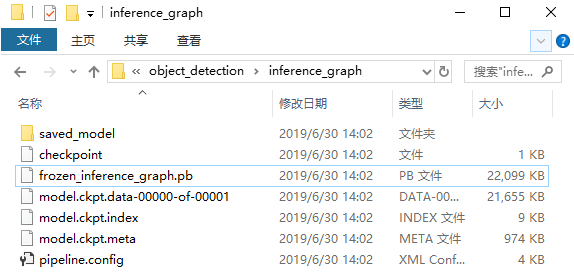


## 生成ffrozen\_inference\_graph.pb文件

在object\_detection目录下，执行：

|  |
| --- |
| python export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path training/ssd\_mobilenet\_v1\_coco.config --trained\_checkpoint\_prefix training/model.ckpt-71395 --output\_directory inference\_graph |

其中71395根据自己情况做修改。



## 测试

|  |
| --- |
| # Import packages  **import** os  **import** cv2  **import** numpy **as** np  **import** tensorflow **as** tf  **import** sys  # This is needed since the notebook is stored in the object\_detection folder.  sys**.**path**.**append**(**".."**)**  # Import utilites  **from** utils **import** label\_map\_util  **from** utils **import** visualization\_utils **as** vis\_util  # Name of the directory containing the object detection module we're using  MODEL\_NAME **=** 'inference\_graph'  # Grab path to current working directory  CWD\_PATH **=** os**.**getcwd**()**  # Path to frozen detection graph .pb file, which contains the model that is used  # for object detection.  PATH\_TO\_CKPT **=** os**.**path**.**join**(**CWD\_PATH**,**MODEL\_NAME**,**'frozen\_inference\_graph.pb'**)**  NUM\_CLASSES **=** 1  # Path to label map file  PATH\_TO\_LABELS **=** os**.**path**.**join**(**CWD\_PATH**,**'training'**,**'labelmap.pbtxt'**)**  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**)**  # Load the Tensorflow model into memory.  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='')  sess = tf.Session(graph=detection\_graph)  # Define input and output tensors (i.e. data) for the object detection classifier  # Input tensor is the image  image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')  # Output tensors are the detection boxes, scores, and classes  # Each box represents a part of the image where a particular object was detected  detection\_boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')  # Each score represents level of confidence for each of the objects.  # The score is shown on the result image, together with the class label.  detection\_scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0')  detection\_classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0')  # Number of objects detected  num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0')  IMAGE\_LIST = os.listdir('test\_images')  for IMAGE\_NAME in IMAGE\_LIST:  # Path to image  PATH\_TO\_IMAGE = os.path.join(CWD\_PATH, 'test\_images', IMAGE\_NAME)  # Load image using OpenCV and  # expand image dimensions to have shape: [1, None, None, 3]  # i.e. a single-column array, where each item in the column has the pixel RGB value  image = cv2.imread(PATH\_TO\_IMAGE)  image\_expanded = np.expand\_dims(image, axis=0)  # Perform the actual detection by running the model with the image as input  (boxes, scores, classes, num) = sess.run(  [detection\_boxes, detection\_scores, detection\_classes, num\_detections],  feed\_dict={image\_tensor: image\_expanded})  # Draw the results of the detection (aka 'visulaize the results')  vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(  image,  np.squeeze(boxes),  np.squeeze(classes).astype(np.int32),  np.squeeze(scores),  category\_index,  use\_normalized\_coordinates=True,  line\_thickness=8,  min\_score\_thresh=0.8)  # All the results have been drawn on image. Now display the image.  cv2.imshow('Object detector', image)  # Press any key to close the image  cv2.waitKey(0)  # Clean up  cv2.destroyAllWindows() |

