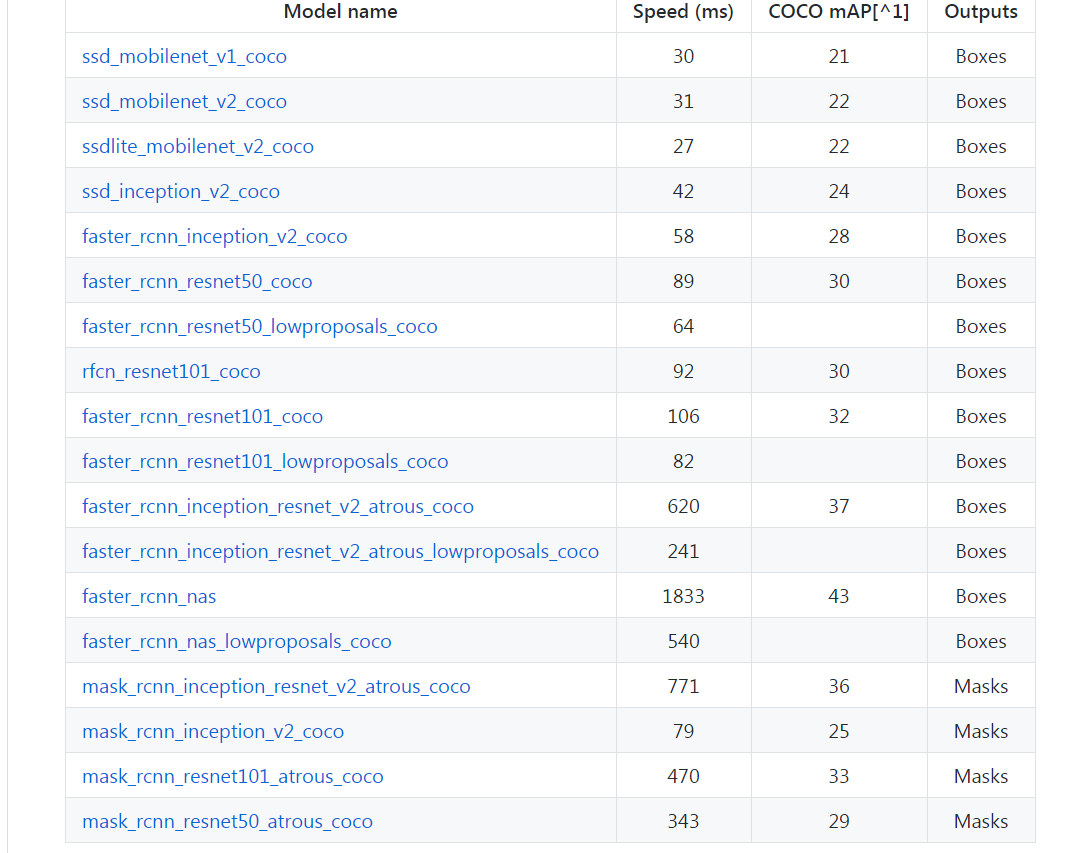
本文档的目的是针对在Windows平台下，使用google公司公布的TensorFlow Object Detection API，使用已经训练好的模型进行图片和视频物体识别；以及使用现有的网络和模型进行微调，训练可以适用于自己新数据的模型。

# 第一部分：google object detection API简介

TensorFlow Object Detection API 的代码库是一个建立在 TensorFlow 之上的开源框架，基于各种网络框架可以帮助人们快速地完成目标检测。为了体现准确率和效率，API使用各种不同的网络架构和决策方法，以及不同的数据集训练出了各种模型。用户可以基于自己的需求，选择使用不同的模型。

到目前为止，google已经训练好的模型包括基于COCO数据集、Kitti数据集以及Open Images的多个模型，譬如



例如，SSD 模型使用了轻量化的 MobileNet，这意味着它可以在移动设备中实时使用，显而易见，这个模型的速度快，相应地，精度不够。在 2016 年 COCO 挑战的研究中，谷歌使用了 Fast RCNN 模型，需要更多计算资源，但结果更为准确。

如需了解更多细节，请参阅谷歌发表在 CVPR 2017 上的论文：https://arxiv.org/abs/1611.10012

Google的模型很多是基于 COCO 数据集训练过的。COCO 数据集包含 30 万张图片，90 中常见事物类别。其中的类别包括：

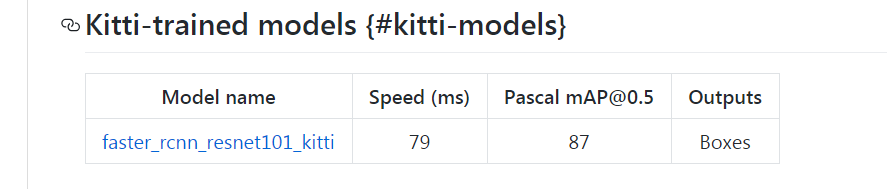


*COCO 数据集的部分类别*

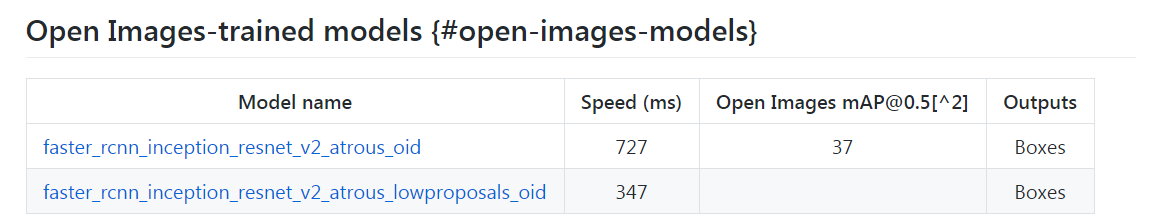
TensorFlow Object Detection API 的 GitHub：https://github.com/tensorflow/models/tree/master/object\_detection

COCO 数据集：<http://mscoco.org/>

基于Kitti训练的模型有



基于Open Image训练的模型包括：



同时，这些模型还在不断更新。

模型中的speed表示处理速度，mAP（mean average precision）表示检测框的精确度和召回率的乘积，mAP越高，表示效果越好。

每个模型保存格式我是tar.gz，解压之后可以看到：

graph proto (grrah.pbtxt)

checkpoint (model.ckpt.data-00000-of-00001, model.ckpt.index, model.ckpt.meta)

frozen graph proto，其中包括了网络图和权重，可以直接使用（将用于第一个练习）

config文件，可以用于生成网络图，保存在samples/comfigs文件夹中。

**第二部分：使用google object detection API进行图片和视频中的目标检测**

Google object detection API依赖于

Protobuf 3+

Python-tk

Pillow 1.0

Lxml

Tf Slim

Jypyter notebook

Matplotlib

Tensorflow

Cython

Cocoapi （可选）

在windows下的安装如下：

1. 确保在之前的课程中已经安装好了python和tensorflow，python版本3.6, tensorflow版本1.7.0。
2. 接下来需要安装Protoc，Google Protocol Buffer( 简称 Protobuf) 是 Google 公司内部的混合语言数据标准，目前已经正在使用的有超过 48,162 种报文格式定义和超过 12,183 个 .proto 文件。Protocol Buffers 是一种轻便高效的结构化数据存储格式，可以用于结构化数据串行化，或者说序列化。它很适合做数据存储或 RPC 数据交换格式。可用于通讯协议、数据存储等领域的语言无关、平台无关、可扩展的序列化结构数据格式。目前提供了 C++、Java、Python 三种语言的 API。

进入下载页面https://github.com/google/protobuf/releases，下载

，或者更新版本

解压之后将protoc.exe的路径添加到环境变量中，或者直接将Protoc.exe拷贝到c:\windows\system32.

1. 安装git，用于从github中下载所需的源代码和model。Git官网（https://www.git-scm.com/download/）
2. 使用pip安装所需组件：

pip install Cython

pip install pillow

pip install lxml

pip install jupyter

pip install matplotlib

1. 下载tensorlfow/models代码

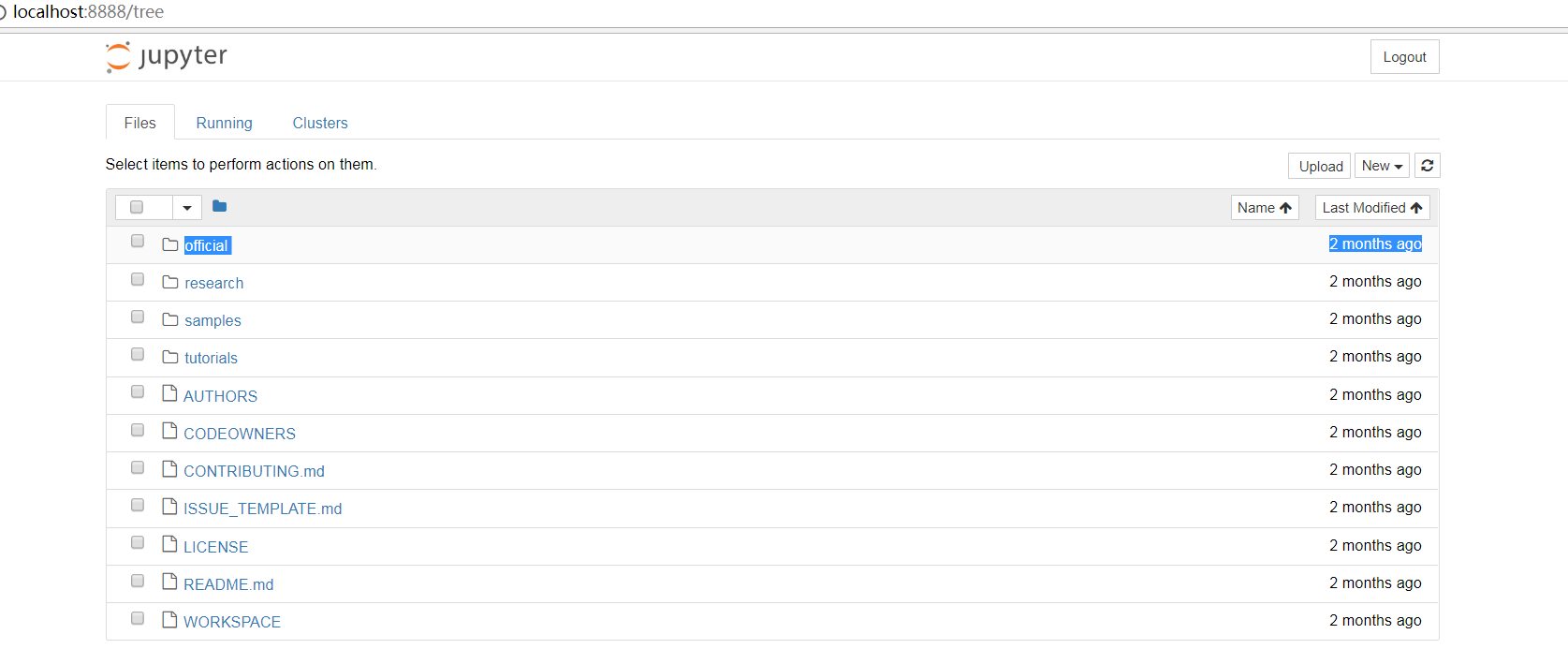
git clone <https://github.com/tensorflow/models.git>

将下载好的代码拷贝到合适的位置，进入Models文件夹，进行编译

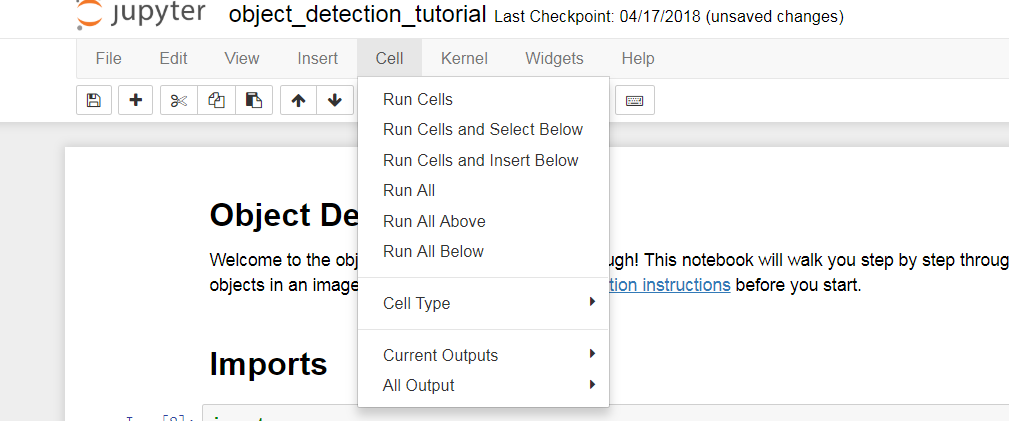
protoc object\_detection/protos/\*.proto –python\_out=.

（使用shift+右键可以在当前文件夹下打开cmd）

1. 在models文件夹下运行命令 jupyter-notebook，此时会在8888端口开启服务程序，并出现如下的界面。

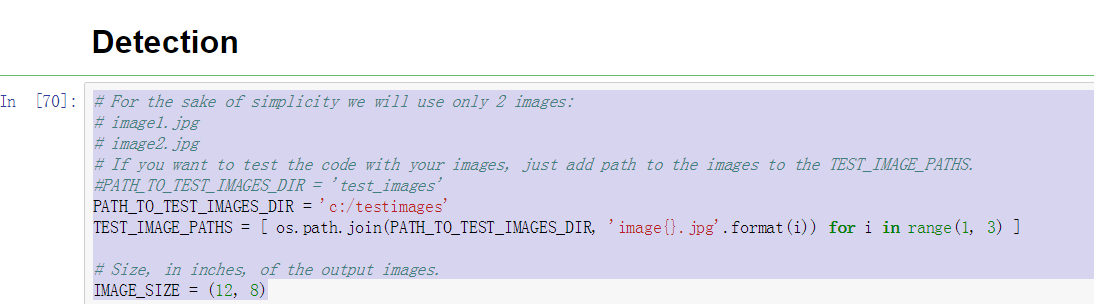


1. 进入research文件夹，进入object\_detection文件夹，打开object\_detection\_tutorial.ipynb。

在

点击Run All，此时系统开始使用默认的配置（所使用的model，测试图片等）自动运行。依赖于网络下载速度（如果本地没有模型），所采用的模型和电脑配置，在几分钟内应该能看到类似于以下图片的结果。

在如下的代码部分修改检测图片，其中图片名字必须是image1.jpg,image2.jpg…..

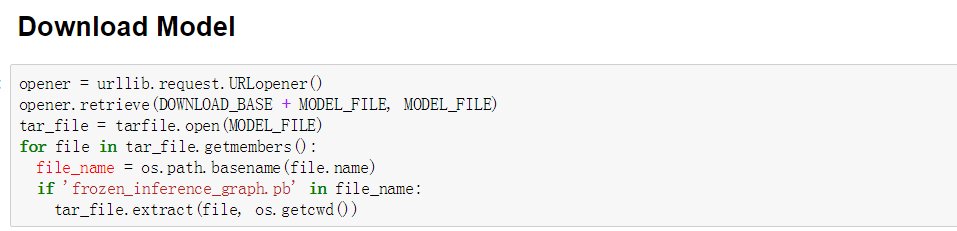


读者可以自己尝试使用不同的model，可以看到不同的model效果差别很大。有的模型识别出来的对象多，精确度高；有的识别对象少；有的模型运行时间需要很长，有的则较短。可以多尝试，并体会每种不同模型的特点。

修改模型：

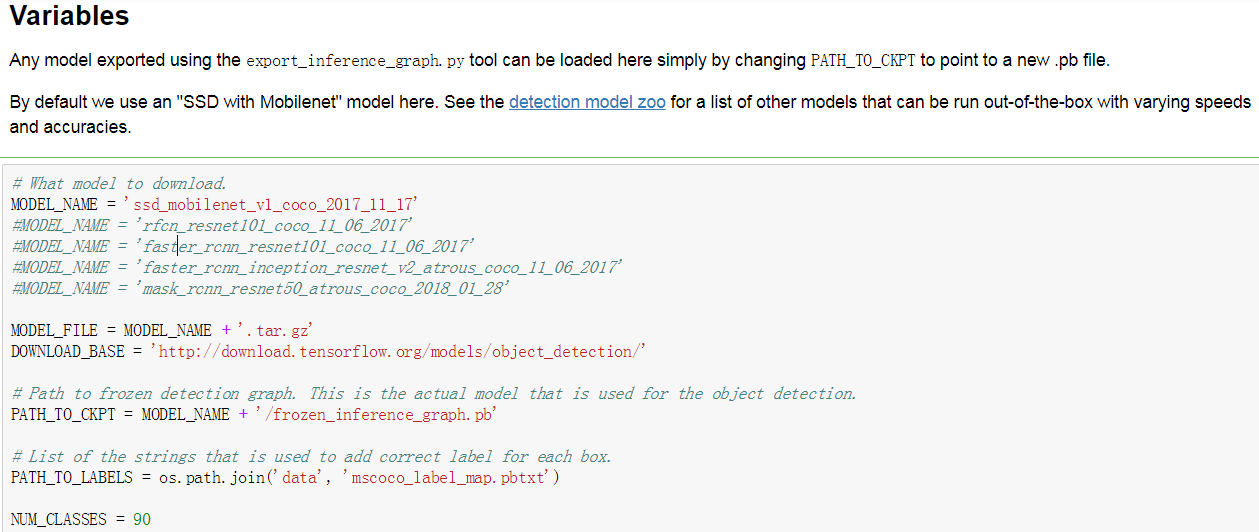
可以在修改所使用的模型，使用的模型如果没有下载到本地，则需要现场下载，会比较耗时。

如果下载出错，在download model处会报错。

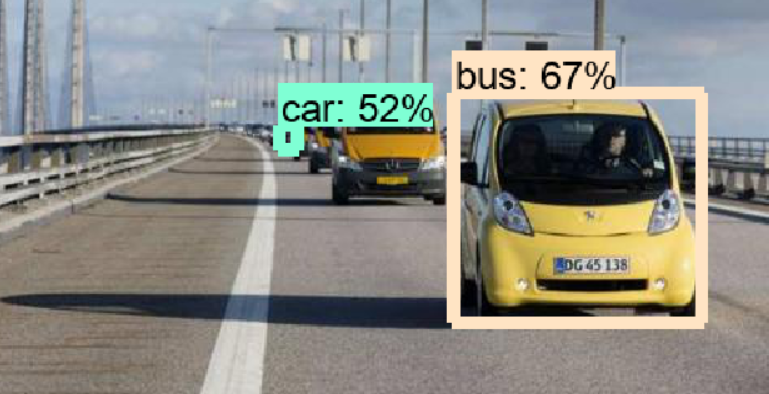


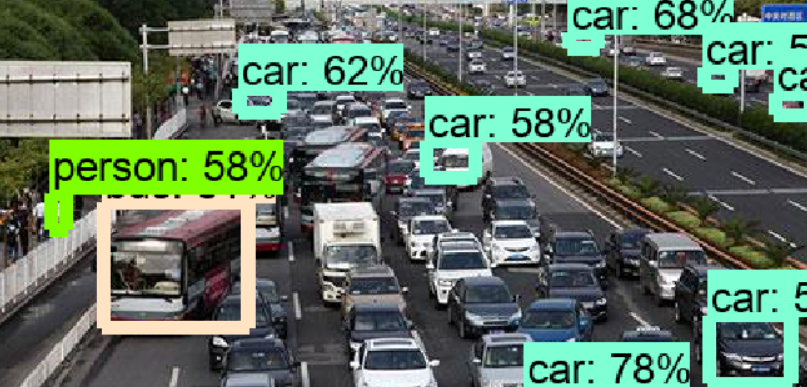
或者可以从以下model zoo（google公布的各种model，不断更新）地址<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md>中提前下载到object\_detection文件夹下。

可以在以下代码处修改所使用的model。

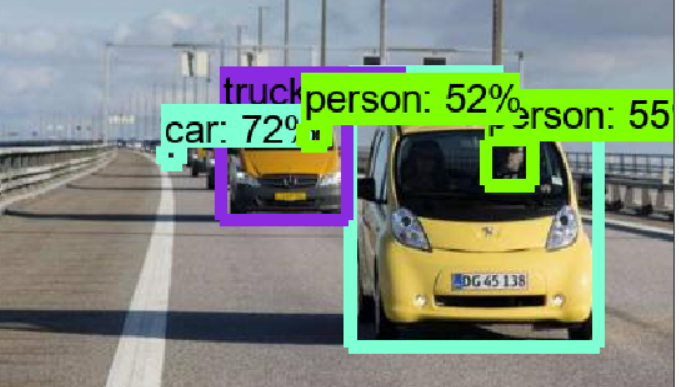


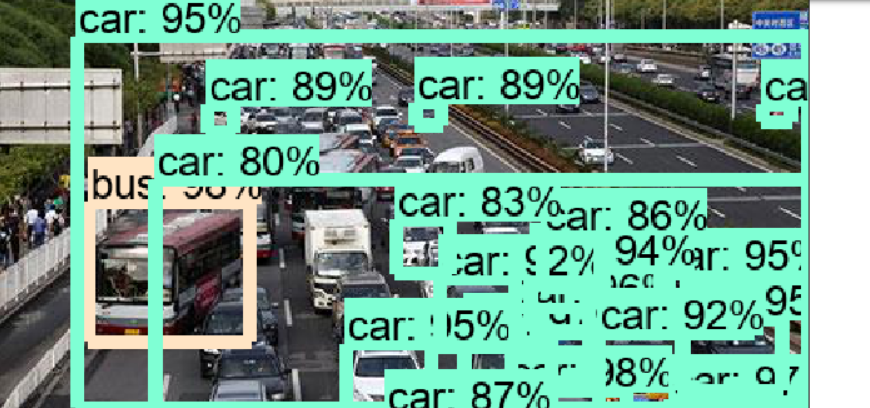
（ssd\_mobilenet\_v1）



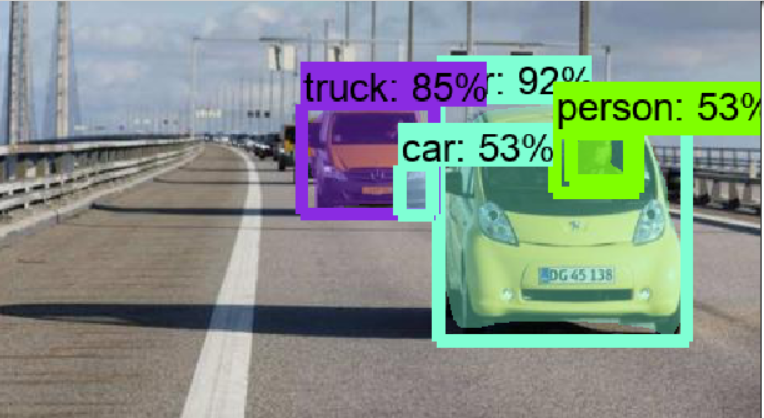


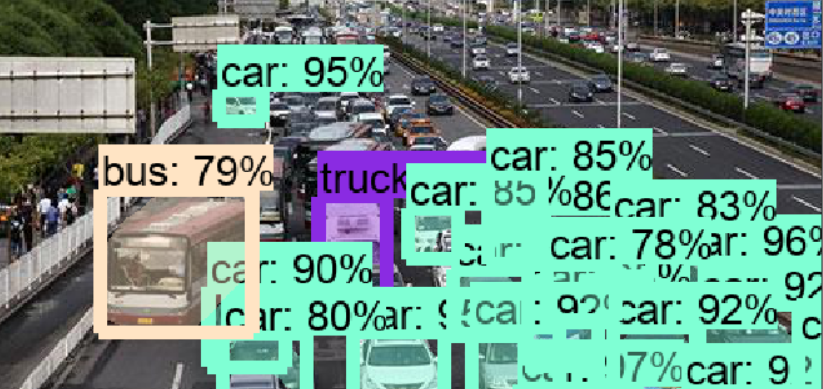
（rfcn\_resnet101\_coco\_11\_06\_2017）





（mask\_rcnn\_resnet50\_atrous\_coco\_2018\_01\_28）





# 视频中的物体识别

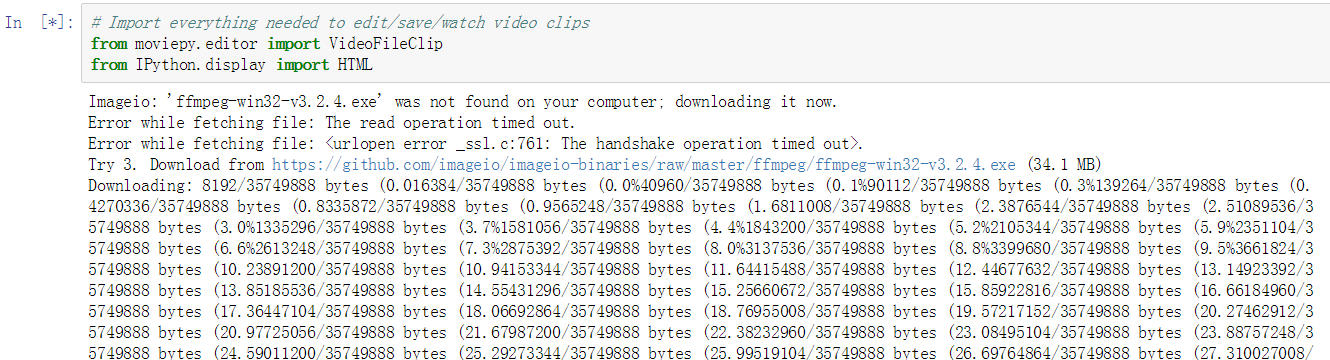
接下来进行视频中的物体识别。进行视频中的物体识别的思路如下：

使用 VideoFileClip 函数从视频中抓取图片；

对抓取的图片进行目标识别，fl\_image 函数可以用来将原图片替换为修改后的图片。

最后，所有修改的剪辑图像被组合成为一个新的视频。

1. 安装opencv2。
2. pip install moviepy
3. 自动安装ffmpeg-win32-v3.2.4.exe；



1. 在图片目标识别的基础上，增加以下代码，实现视频中的目标识别。

# Import everything needed to edit/save/watch video clips

from moviepy.editor import VideoFileClip

from IPython.display import HTML

以上两行代码，引入视频剪辑工具和网页上展示视频的工具。在执行这两行代码时，会自动下载工具ffmpeg，保证网络畅通。

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),

category\_index,

use\_normalized\_coordinates=True,

line\_thickness=8)

return image\_np

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

以上两个函数调用接口实现对图像中的物体的识别。

from moviepy.editor import \*

clip\_output = 'video\_clip.mp4'

clip1 = VideoFileClip("video.mp4").subclip(145,150)

clip\_result = clip1.fl\_image(process\_image) #NOTE: this function expects color images!!s

%time clip\_result.write\_videofile(clip\_output, audio=False)

以上几行代码的目的是从给定的”video.mp4”中截取从145秒到150秒的一小段（subclip(145,150)），然后调用上面的process\_image函数，对图片进行目标识别。然后，再将识别好的图像重新写入，形成视频video\_clip.mp4。从5秒的视频中可以截取100+的图片，如果使用较复杂的模型，非常耗时。其中，video.mp4应该放在object\_detection文件夹中，生成的结果video\_clip也保存在同一文件夹下。

HTML("""

<video width="960" height="540" controls>

<source src="{0}">

</video>

""".format(“video\_clip.mp4”))

这行代码可以在页面上显示新生成的实现目标识别之后的视频。

同样的，可以尝试使用不同的模型，有的速度快，但是效果一般；有的速度慢，但是效果比较好。

# 第三部分：基于google object detection API训练新数据

Google API提供了在MSCOCO、Kitti、AVA v2.1数据集上训练好的模型,并没有基于PASCALVOC数据集训练的模型，下面的例子如何使用Google Tensorflow Detection API训练自己的模型，这里选用的是ssd\_inception\_v2模型。

这里需要注意的是，要对新数据进行训练，首先需要对数据进行处理，将数据转换为tf-record形式。API中提供了将PASCALVOC数据集以及Oxford-IIIT Pet数据集转换为tf-record格式的方法。

详情可参考：

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/preparing_inputs.md>

如果不是这两个数据集，从图片中构造tf-record比较麻烦，具体可参考

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/using_your_own_dataset.md>

对PASCALVOC数据集的准备工作：

从pascalvoc官网下载voc2012数据集：<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar>

将下载下来的tar压缩包解压到任意路径。Tensorflow Object Detection API，实现了将PascalVOC转换为TFRecord格式的代码，源码在models/research/object\_detection/dataset\_tools/create\_pascal\_tf\_record.py。

执行以下命令完成train，和eval数据集的转换：

# From tensorflow/models/research/object\_detection

python dataset\_tools/create\_pascal\_tf\_record.py \

--data\_dir=path to VOCdevkit \

--year=VOC2012 \

--set=train \

--output\_path=path to train record

python dataset\_tools/create\_pascal\_tf\_record.py \

--data\_dir=path to VOCdevkit \

--year=VOC2012 \

--set=val \

--output\_path=path to train record

譬如可以将TFRecord数据存放在${CUSTOMPATH}/voc2012目录下，读者可按自己喜好选择存放目录。

然后下载训练模型，这里选择训练好的[ssd\_inception\_v2\_coco](http://download.tensorflow.org/models/object_detection/ssd_inception_v2_coco_2017_11_17.tar.gz) 模型。下载好模型后，解压到任意文件夹，这里将其存放在${CUSTOMPATH}/voc2012/checkpoints/目录下。这里相当于是准备好了训练的网络和部分的初始参数。

接下来进行配置选择，API源码中提供了很多默认配置，这里选用基于models/research/object\_detection/samples/configs/ssd\_inception\_v2\_coco.config配置文件进行修改，并保存到指定目录，这里将其保存在${CUSTOMPATH}/voc2012 目录下。配置文件主要是针对具体的数据集进行一些配置，如

num\_classes：20 #因为PASCALVOC数据集共有20类数据

fine\_tune\_checkpoint: "{CUSTOMPATH}/voc2012/checkpoints/ssd\_inception\_v2\_coco\_2017\_11\_17/model.ckpt"

from\_detection\_checkpoint: true

train\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "{CUSTOMPATH}/voc2012/pascal.record"

}

label\_map\_path: "{CUSTOMPATH}/models/research/object\_detection/data/pascal\_label\_map.pbtxt"

}

eval\_config: {

num\_examples: 8000

# Note: The below line limits the evaluation process to 10 evaluations.

# Remove the below line to evaluate indefinitely.

max\_evals: 10

}

eval\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "{CUSTOMPATH}/voc2012/pascal\_eval.record"

}

label\_map\_path: "{CUSTOMPATH}/models/research/object\_detection/data/pascal\_label\_map.pbtxt"

shuffle: false

num\_readers: 1

}

注意其中的所有路径设置都必须根据自己的文件位置进行设置

接下来就可以执行以下命令进行模型训练

# From the tensorflow/models/research/

python object\_detection/train.py \

--logtostderr \

--pipeline\_config\_path=${PATH\_TO\_YOUR\_PIPELINE\_CONFIG} \

--train\_dir=${PATH\_TO\_TRAIN\_DIR}

由于代码中使用了COCO evaluation metrics，所以需要安装COCO API，官方给出的安装方式如下：

git clone https://github.com/cocodataset/cocoapi.gitcd cocoapi/PythonAPI

make

cp -r pycocotools <path\_to\_tensorflow>/models/research/

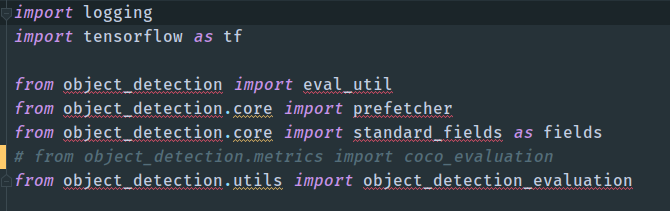
但在尝试过程中这一步骤失败了，所以直接使用 Pascal VOC evaluation，要使用 Pascal VOC evaluation，需要对API源码做修改，不然会出现找不到模块，修改如下：

修改research/object\_detection/eval\_util.py文件，将coco\_evaluation模块导入注释掉：

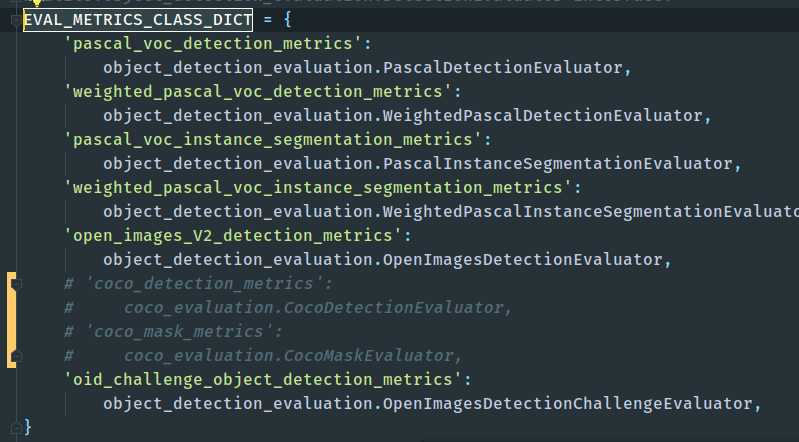
直接使用 Pascal VOC evaluation ! n(\*≧▽≦\*)n。要使用 Pascal VOC evaluation，需要对API源码做一定更改，不然会出现找不到模块，修改如下：

修改research/object\_detection/eval\_util.py文件，将coco\_evaluation模块导入注释掉：

修改research/object\_detection/evaluator.py文件，同样注释掉coco\_evaluation模块导入：



然后将该文件中的EVAL\_METRICS\_CLASS\_DICT字典关于coco的字段注释掉：



最后执行以下命令进行模型验证：

# From the tensorflow/models/research/ directory

python object\_detection/eval.py \

--logtostderr \

--pipeline\_config\_path=${PATH\_TO\_YOUR\_PIPELINE\_CONFIG} \

--checkpoint\_dir=${PATH\_TO\_TRAIN\_DIR} \

--eval\_dir=${PATH\_TO\_EVAL\_DIR}

同样，注意替换文件路径为自己文件的位置

模型训练过程中会创建以下检查点文件到${TRAIN\_DIR}中：

model.ckpt-${CHECKPOINT\_NUMBER}.data-00000-of-00001

model.ckpt-${CHECKPOINT\_NUMBER}.index

model.ckpt-${CHECKPOINT\_NUMBER}.meta

选择一个检查点文件，进行模型导出，执行以下命令：

# From tensorflow/models/research/

python object\_detection/export\_inference\_graph.py \

--input\_type image\_tensor \

--pipeline\_config\_path ${PIPELINE\_CONFIG\_PATH} \

--trained\_checkpoint\_prefix ${TRAIN\_PATH} \

--output\_directory ${EXPORT\_DIR}

执行上面的命令后会产生以下文件在指定目录中：

output\_inference\_graph.pb，输出模型的冻结图形格式

saved\_model/，一个包含导出模型的保存模型格式的目录

model.ckpt.\*，用于导出的模型检查点

checkpoint，一个指定的检查点文件

### 使用导出的模型进行测试

创建测试代码如下：

|  |
| --- |
|  |
| import os  import numpy as np  import tensorflow as tf  from PIL import Image  from matplotlib import pyplot as plt  from object\_detection.utils import ops as utils\_ops  from utils import label\_map\_util  from utils import visualization\_utils as vis\_util  # Path to frozen detection graph. This is the actual model that is used for the object detection.  PATH\_TO\_CKPT = '{CUSTOMPATH}/voc2012/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', 'pascal\_label\_map.pbtxt')  NUM\_CLASSES = 20  if tf.\_\_version\_\_ < '1.4.0':  raise ImportError(  'Please upgrade your tensorflow installation to v1.4.\* or later!')  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='')  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)  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)  # 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) for image in os.listdir(PATH\_TO\_TEST\_IMAGES\_DIR)]  # Size, in inches, of the output images.  IMAGE\_SIZE = (12, 8)  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 coordinates and fit the image 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.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  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 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=8)  plt.figure(figsize=IMAGE\_SIZE)  plt.imshow(image\_np)  plt.show() |

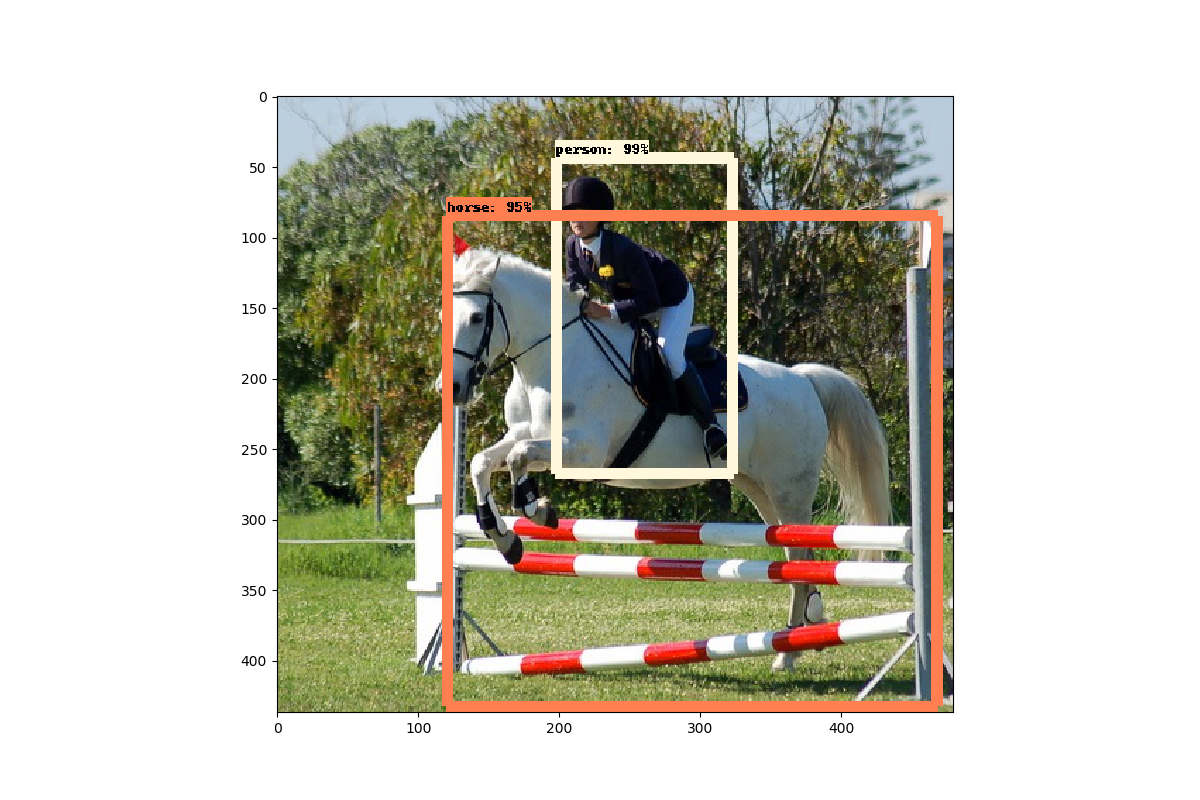
注意将代码中模型路径替换为自己的路径：

PATH\_TO\_CKPT = '{CUSTOMPATH}/voc2012/frozen\_inference\_graph.pb'

将代码保存到research/object\_detection/detection.py中，然后直接执行代码，不加任何参数便可以得到测试结果。research/object\_detection/test\_images中默认只有两张用于测试的照片，可以将目录指定到自己想要测试的照片路径，但是这份测试代码要求目录下只能存放图片文件，不然执行会报错。默认的目录下有个txt文件，如果不修改代码，记得删除该文件。

**在本地运行时，tensorflow/models/research/和slim目录应附加到环境变量中。**

最后，使用自己的模型目标检查结果如下：



[**https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/installation.md**](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md)

[**https://github.com/protocolbuffers/protobuf/releases?after=v3.0.0-javalite**](https://github.com/protocolbuffers/protobuf/releases?after=v3.0.0-javalite)

[**https://blog.csdn.net/chenkjiang/article/details/22159407/**](https://blog.csdn.net/chenkjiang/article/details/22159407/)