\*These codes are written in python 3.7\*

Code of Object Detection with Counter:

**import** tensorflow **as** tf

**import** csv

**import** cv2

**import** numpy **as** np

**from** utils **import** visualization\_utils **as** vis\_util

​

**import** glob

**import** os

**from** utils **import** label\_map\_util

model\_name**=**'ssd\_mobilenet\_v1\_coco\_2018\_01\_28' *#model name*

fps **=** 24 *# change it with your input video fps*

width **=** 650*#change it with your input video width*

height **=** 550 *# change it with your input vide height*

is\_color\_recognition\_enabled **=** 0 *# set it to 1 for enabling the color prediction for the detected objects*

roi **=** 250*#roi line position*

deviation **=** 3 *# the constant that represents the object counting area*

input\_video **=** "Vclip.mp4"

model\_found **=** 0

​

**for** file **in** glob.glob("\*"):

**if** (file **==** model\_name):

model\_found **=** 1

model\_name **=** model\_name

model\_file **=** model\_name **+** '.tar.gz'

download\_base **=** 'http://download.tensorflow.org/models/object\_detection/'

​

path\_to\_ckpt **=** model\_name **+** '/frozen\_inference\_graph.pb'

​

path\_to\_labels **=** os.path.join('data', 'mscoco\_label\_map.pbtxt')

​

num\_classes **=** 90

​

**if** (model\_found **==** 0):

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())

​

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** object\_detect\_count(input\_video, detection\_graph, category\_index, is\_color\_recognition\_enabled, fps, width, height, roi, deviation):

total\_passed\_vehicle **=** 0

​

​

cap **=** cv2.VideoCapture(input\_video)

​

total\_passed\_vehicle **=** 0

counting\_mode **=** "..."

width\_heigh\_taken **=** **True**

**with** detection\_graph.as\_default():

**with** tf.Session(graph**=**detection\_graph) **as** sess:

image\_tensor **=** detection\_graph.get\_tensor\_by\_name('image\_tensor:0')

​

detection\_boxes **=** detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')

​

detection\_scores **=** detection\_graph.get\_tensor\_by\_name('detection\_scores:0')

detection\_classes **=** detection\_graph.get\_tensor\_by\_name('detection\_classes:0')

num\_detections **=** detection\_graph.get\_tensor\_by\_name('num\_detections:0')

​

**while**(cap.isOpened()):

ret, frame **=** cap.read()

​

**if** **not** ret:

print("end of the video file...")

**break**

input\_frame **=** frame

​

image\_np\_expanded **=** np.expand\_dims(input\_frame, axis**=**0)

​

(boxes, scores, classes, num) **=** sess.run(

[detection\_boxes, detection\_scores, detection\_classes, num\_detections],

feed\_dict**=**{image\_tensor: image\_np\_expanded})

​

font **=** cv2.FONT\_HERSHEY\_SIMPLEX

​

counter, csv\_line, counting\_mode **=** vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array\_y\_axis(cap.get(1),

input\_frame,

2,

is\_color\_recognition\_enabled,

np.squeeze(boxes),

np.squeeze(classes).astype(np.int32),

np.squeeze(scores),

category\_index,

y\_reference **=** roi,

deviation **=** deviation,

use\_normalized\_coordinates**=True**,

line\_thickness**=**4)

​

**if** counter **==** 1:

cv2.line(input\_frame, (0, roi), (width, roi), (0, 0xFF, 0), 5)

**else**:

cv2.line(input\_frame, (0, roi), (width, roi), (0, 0, 0xFF), 5)

total\_passed\_vehicle **=** total\_passed\_vehicle **+** counter

​

font **=** cv2.FONT\_HERSHEY\_SIMPLEX

cv2.putText(

input\_frame,

'Detected Vehicles: ' **+** str(total\_passed\_vehicle),

(10, 35),

font,

0.8,

(0, 0xFF, 0xFF),

2,

cv2.FONT\_HERSHEY\_SIMPLEX,

)

cv2.putText(

input\_frame,

'ROI Line',

(545, roi**-**10),

font,

0.6,

(0, 0, 0xFF),

2,

cv2.LINE\_AA,

)

cv2.imshow('object counting',input\_frame)

​

**if** cv2.waitKey(1) **&** 0xFF **==** ord('q'):

**break**

​

​

cap.release()

cv2.destroyAllWindows()

​

​

​

​

object\_detect\_count(input\_video, detection\_graph, category\_index, is\_color\_recognition\_enabled, fps, width, height, roi, deviation) *# counting all the objects*

In [ ]:

​

Code for Object detection live through webcam:

………………………………………………………………………………...

In [2]:

**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

**import** cv2

cap **=** cv2.VideoCapture(0)

sys.path.append("..")

**from** utils **import** label\_map\_util

**from** utils **import** visualization\_utils **as** vis\_util

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\_CKPT **=** MODEL\_NAME **+** '/frozen\_inference\_graph.pb'

PATH\_TO\_LABELS **=** os.path.join('data', 'mscoco\_label\_map.pbtxt')

NUM\_CLASSES **=** 90

*#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())*

​

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)

**with** detection\_graph.as\_default():

**with** tf.Session(graph**=**detection\_graph) **as** sess:

**while** **True**:

ret, image\_np **=** cap.read()

*# 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)

​

cv2.imshow('object detection', cv2.resize(image\_np, (800,600)))

**if** cv2.waitKey(25) **&** 0xFF **==** ord('q'):

cap.release()

cv2.destroyAllWindows()

**break**

…………………………………………………………………………………..

Code for object detection in an image:

**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

sys.path.append("..")

**from** object\_detection.utils **import** ops **as** utils\_ops

**from** utils **import** label\_map\_util

**from** utils **import** visualization\_utils **as** vis\_util

In [2]:

MODEL\_NAME **=** 'faster\_rcnn\_resnet101\_coco\_2018\_01\_28'

MODEL\_FILE **=** MODEL\_NAME **+** '.tar.gz'

DOWNLOAD\_BASE **=** 'http://download.tensorflow.org/models/object\_detection/'

PATH\_TO\_CKPT **=** MODEL\_NAME **+** '/frozen\_inference\_graph.pb'

PATH\_TO\_LABELS **=** os.path.join('data', 'mscoco\_label\_map.pbtxt')

In [3]:

NUM\_CLASSES **=** 90

In [4]:

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 [5]:

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]:

**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

PATH\_TO\_TEST\_IMAGES\_DIR **=** ''

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)

**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)

**from** PIL **import** Image

**for** image\_path **in** TEST\_IMAGE\_PATHS:

image **=** Image.open(image\_path)

image\_np **=** load\_image\_into\_numpy\_array(image)

plt.imshow(image\_np)

print(image.size, image\_np.shape)

​

*#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**=**'')

**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)

print(image\_process.shape)

plt.figure(figsize**=**IMAGE\_SIZE)

plt.imshow(image\_process)

​

………………………………………………………………………………….

In both of the above cases we are using already trained models that are trained using Coco datasets there are alot of models available. You can find it in folder Models>research>object\_detection>g3doc>detection\_model\_zoo.md

The model used in code for object detection in images is very fast and accurate but will require a dedicated gpu or a reasonably good cpu.

………………………………………………………………………………..

\*\*\*\* To set the Environment \*\*\*\*

### Setting up the Environment

* Now to Download TensorFlow and TensorFlow GPU you can use pip or conda commands:

|  |  |
| --- | --- |
|  | # For CPU  pip install tensorflow  # For GPU  pip install tensorflow-gpu |

* For all the other libraries we can use pip or conda to install them. The code is provided below:

|  |  |
| --- | --- |
|  | pip install --user Cython  pip install --user contextlib2  pip install --user pillow  pip install --user lxml  pip install --user jupyter  pip install --user matplotlib |

* Next, we have Protobuf: Protocol Buffers (Protobuf) are Google’s language-neutral, platform-neutral, extensible mechanism for serializing structured data, – think of it like XML, but smaller, faster, and simpler. You need to [*Download Protobuf*](https://github.com/google/protobuf/releases) version 3.4 or above for this demo and extract it.
* Now you need to Clone or Download TensorFlow’s Model from [*Github*](https://github.com/tensorflow/models). Once *downloaded* and *extracted* rename the “models-masters” to just “models“.
* Now for simplicity, we are going to keep “models” and “protobuf” under one folder “Tensorflow“.
* Place the bin folder in model folder.
* Next, we need to go inside the Tensorflow folder and then inside research folder and run protobuf from there using this command:

|  |  |
| --- | --- |
| 1 | "path\_of\_protobuf's bin"./bin/protoc object\_detection/protos/ |

* To check whether this worked or not, you can go to the protos folder inside models>object\_detection>protos and there you can see that for every proto file there’s one python file created.

\*\*\*\*\*Make the code file in “research” Folder \*\*\*\*\*\*

Some References for Theory and Code:

<https://github.com/ahmetozlu/tensorflow_object_counting_api>

<https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4>

<https://github.com/priya-dwivedi/Deep-Learning/blob/master/Object_Detection_Tensorflow_API.ipynb>

<https://towardsdatascience.com/building-a-toy-detector-with-tensorflow-object-detection-api-63c0fdf2ac95>

These are some common problems faced with working with images:

<https://stackoverflow.com/questions/38320722/python-ioerror-errno-2-no-such-file-or-directory>

<https://stackoverflow.com/questions/51569669/python-from-utils-import-label-map-util-importerror-cannot-import-name-lab>

<https://medium.com/@ghimire.aiesecer/custom-object-detection-using-convolutional-neural-network-in-google-colaboratory-a9bc1d6eb8a8>

There is a Tutorial python notebook in folder

models>research>object\_detection>tutorial.ipynb

Most of the references you will find will be written in python 2.X but the tutorial in the models is latest and updated whenever a new version of python is released.