**https://www.edureka.co/blog/tensorflow-object-detection-tutorial/**

**Object Detection Workflow**

Every Object Detection Algorithm has a different way of working, but they all work on the same principle.

**Feature Extraction:**They extract features from the input images at hands and use these features to determine the class of the image. Be it through MatLab, Open CV, Viola Jones or Deep Learning.

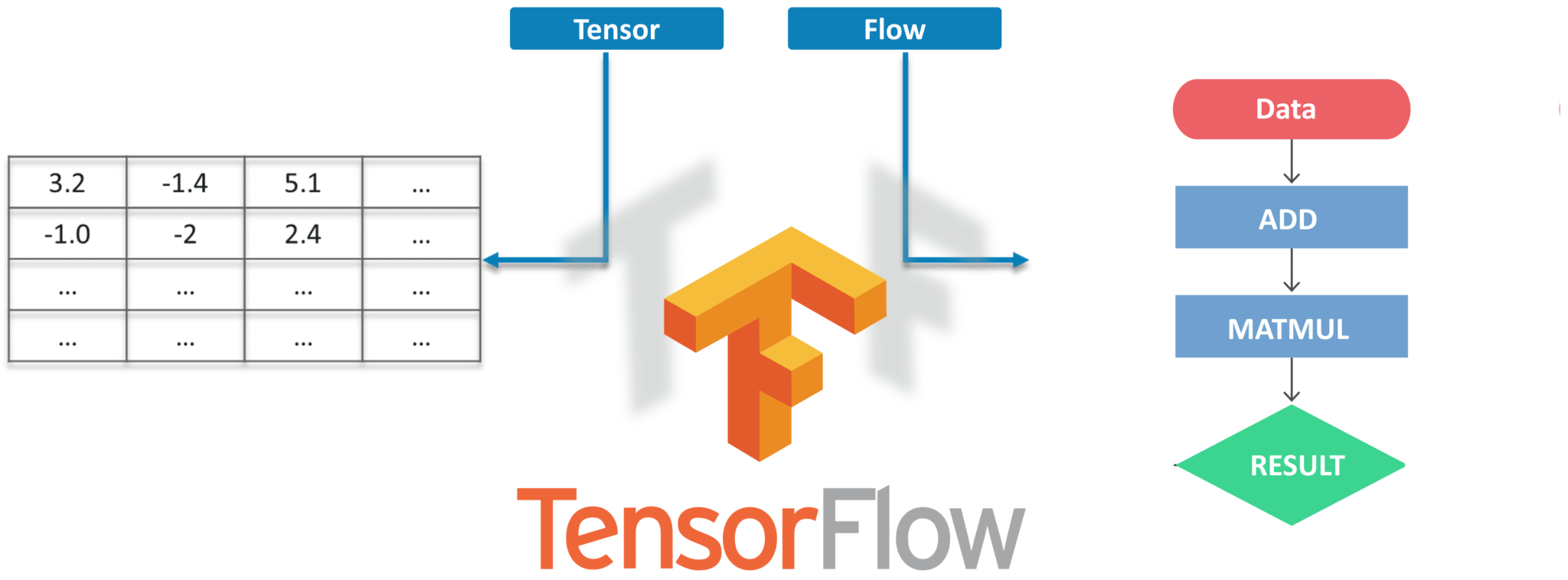
Timeline

Description automatically generated with low confidence

Now that you have understood the basic workflow of Object Detection, let’s move ahead in Object Detection Tutorial and understand what Tensorflow is and what are its components?

**What is TensorFlow?**

Tensorflow is Google’s Open Source Machine Learning Framework for dataflow programming across a range of tasks. Nodes in the graph represent mathematical operations, while the graph edges represent the multi-dimensional data arrays (**tensors**) communicated between them.



Tensors are just multidimensional arrays, an extension of 2-dimensional tables to data with a higher dimension. There are many features of Tensorflow which makes it appropriate for Deep Learning. So, without wasting any time, let’s see how we can implement Object Detection using Tensorflow.

**Object Detection Tutorial**

**Getting Prerequisites**

* Before working on the Demo, let’s have a look at the prerequisites. We will be needing:
  + Python
  + TensorFlow
  + Tensorboard
  + [Protobuf v3.4 or above](https://github.com/google/protobuf/releases)

**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" \t "_blank)*** 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" \t "_blank)***. 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**“.
* Next, we need to go inside the Tensorflow folder and then inside research folder and run protobuf from there using this command:

|  |  |
| --- | --- |
|  | "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.

**Main Code**

After the environment is set up, you need to go to the “**object\_detection**” directory and then create a new python file. You can use **Spyder** or **Jupyter** to write your code.

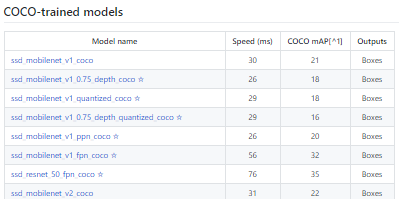
Next

* First of all, we need to import all the libraries

|  |  |
| --- | --- |
|  | **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 |

Next, we will download the model which is trained on the [***COCO dataset***](http://cocodataset.org/#home). COCO stands for **Common Objects in Context,**this dataset contains around 330K labeled images. Now the model selection is important as you need to make an important tradeoff between **Speed and Accuracy**. Depending upon your requirement and the system memory, the correct model must be selected.

Inside “*models>research>object\_detection>g3doc>detection\_model\_zoo*” contains all the models with different speed and accuracy(mAP).



* Next, we provide the required model and the frozen inference graph generated by Tensorflow to use.

|  |  |
| --- | --- |
|  | MODEL\_NAME **=** 'ssd\_mobilenet\_v1\_coco\_2017\_11\_17'  MODEL\_FILE **=** MODEL\_NAME **+** '.tar.gz'  DOWNLOAD\_BASE **=** '<a href="http://download.tensorflow.org/models/object\_detection/">http://download.tensorflow.org/models/object\_detection/</a>'    PATH\_TO\_CKPT **=** MODEL\_NAME **+** '/frozen\_inference\_graph.pb'    PATH\_TO\_LABELS **=** os.path.join('data', 'mscoco\_label\_map.pbtxt')    NUM\_CLASSES **=** 90 |

* This code will download that model from the internet and extract the frozen inference graph of that model.

|  |  |
| --- | --- |
|  | 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**=**'') |

* Next, we are going to load all the labels

|  |  |
| --- | --- |
|  | 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) |

* Now we will convert the images data into a numPy array for processing.

|  |  |
| --- | --- |
|  | **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) |

* The path to the images for the testing purpose is defined here. Here we have a naming convention “**image[i]**” for i in (1 to n+1), n being the number of images provided.

|  |  |
| --- | --- |
|  | PATH\_TO\_TEST\_IMAGES\_DIR **=** 'test\_images'  TEST\_IMAGE\_PATHS **=** [ os.path.join(PATH\_TO\_TEST\_IMAGES\_DIR, 'image{}.jpg'.format(i)) **for** i **in** range(1, 8) ] |

* This code runs the inference for a single image, where it detects the objects, make boxes and provide the class and the class score of that particular object.

|  |  |
| --- | --- |
|  | **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 |

* Our Final loop, which will call all the functions defined above and will run the inference on all the input images one by one, which will provide us the output of images in which objects are detected with labels and the percentage/score of that object being similar to the training data.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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)   |  |  | | --- | --- | | A person flying a kite  Description automatically generated with medium confidence | Detected-Pics-Object Detection Tutorial | |

**Live Object Detection Using Tensorflow**

For this Demo, we will use the same code, but we’ll do a few **tweakings**. Here we are going to use ***OpenCV*** and the camera Module to use the live feed of the webcam to detect objects.

* Add the OpenCV library and the camera being used to capture images. Just add the following lines to the import library section.

|  |  |
| --- | --- |
|  | **import** cv2  cap **=** cv2.VideoCapture(0) |

* Next, we don’t need to load the images from the directory and convert it to numPy array as OpenCV will take care of that for us

**Remove This**

|  |  |
| --- | --- |
|  | **for** image\_path **in** TEST\_IMAGE\_PATHS:  image **=** Image.open(image\_path)  # the array based representation of the image will be used later in..  # result image with boxes and labels on it.  image\_np **=** load\_image\_into\_numpy\_array(image) |

**With**

|  |  |
| --- | --- |
|  | **while** True:  ret, image\_np **=** cap.read() |

* We will not use matplotlib for final image show instead, we will use OpenCV for that as well. Now, for that,

**Remove This**

|  |  |
| --- | --- |
|  | plt.figure(figsize**=**IMAGE\_SIZE)  plt.imshow(image\_np) |

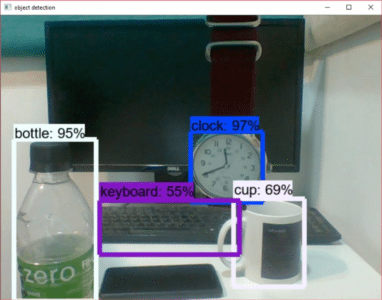
**With**

|  |  |
| --- | --- |
|  | cv2.imshow('object detection', cv2.resize(image\_np, (800,600)))  **if** cv2.waitKey(25) & 0xFF **==** ord('q'):    cv2.destroyAllWindows()  **break** |

This code will use OpenCV that will, in turn, use the camera object initialized earlier to open a new window named “**Object\_Detection**” of the size “800×600”. It will wait for 25 milliseconds for the camera to show images otherwise, it will close the window.

**Final Code with all the changes:**

|  |  |
| --- | --- |
|  | **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    Click snippet panel ( <> ) a la izquierda  Click Camera Capture. Click insert  Run script: Shift + Enter  **import** cv2  cap **=** cv2.VideoCapture(0)    sys.path.append("..")  !pip install tensorflow-object-detection-api    **from** utils **import** label\_map\_util  **from** utils **import** visualization\_utils as vis\_util  MODEL\_NAME **=** 'ssd\_mobilenet\_v1\_coco\_11\_06\_2017'  MODEL\_FILE **=** MODEL\_NAME **+** '.tar.gz'  DOWNLOAD\_BASE **=** '<a href="http://download.tensorflow.org/models/object\_detection/">http://download.tensorflow.org/models/object\_detection/</a>'    # Path to frozen detection graph. This is the actual model that is used for the object detection.  PATH\_TO\_CKPT **=** 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')    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'):        cv2.destroyAllWindows()  **break** |



*Now that you have understood the basics of****Object Detection****, check out the****[Tensorflow Certification](https://www.edureka.co/ai-deep-learning-with-tensorflow" \t "_blank)****by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. This Certification Training is curated by industry professionals as per the industry requirements & demands. You will master the concepts such as SoftMax function, Autoencoder Neural Networks, Restricted Boltzmann Machine (RBM), and work with libraries like Keras & TFLearn.*

Stay ahead of the curve in technology with Edureka’s Post Graduate Program in AI and Machine Learning in partnership with E&ICT Academy, National Institute of Technology, Warangal. This [Artificial Intelligence Course](https://www.edureka.co/executive-programs/machine-learning-and-ai) is curated to deliver the best results.

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

Click snippet panel ( <> ) a la izquierda

Click Camera Capture. Click insert

Run script: Shift + Enter

from IPython.display import display, Javascript

from google.colab.output import eval\_js

from base64 import b64decode

def take\_photo(filename='photo.jpg', quality=0.8):

  js = Javascript('''

    async function takePhoto(quality) {

      const div = document.createElement('div');

      const capture = document.createElement('button');

      capture.textContent = 'Capture';

      div.appendChild(capture);

      const video = document.createElement('video');

      video.style.display = 'block';

      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);

      div.appendChild(video);

      video.srcObject = stream;

      await video.play();

      // Resize the output to fit the video element.

      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.

      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');

      canvas.width = video.videoWidth;

      canvas.height = video.videoHeight;

      canvas.getContext('2d').drawImage(video, 0, 0);

      stream.getVideoTracks()[0].stop();

      div.remove();

      return canvas.toDataURL('image/jpeg', quality);

    }

    ''')

  display(js)

  data = eval\_js('takePhoto({})'.format(quality))

  binary = b64decode(data.split(',')[1])

  with open(filename, 'wb') as f:

    f.write(binary)

  return filename

#!pip install tensorflow-utils

!pip install tensorflow-object-detection-api

cap = cv2.VideoCapture(0)

sys.path.append("..")

#from utils import label\_map\_util

#from utils import visualization\_utils as vis\_util

from object\_detection.utils import label\_map\_util

from object\_detection.utils import visualization\_utils as vis\_util

MODEL\_NAME = 'ssd\_mobilenet\_v1\_coco\_11\_06\_2017'

MODEL\_FILE = MODEL\_NAME + '.tar.gz'

DOWNLOAD\_BASE = '<a href="http://download.tensorflow.org/models/object\_detection/">http://download.tensorflow.org/models/object\_detection/</a>'

# Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH\_TO\_CKPT = 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')

NUM\_CLASSES = 90