How to setup Colab w/ Google Drive data hosting for Training/Retraining TF models

MOTIVATION: **If you only have a laptop or desktop that does not have a decent GPU that is supported by Tensorflow (only some NVidia GPUs are supported --thinking gaming machine), then training will be quite slow locally. There are some alternatives to running locally - one if you have the $$$ is to run in the Google Cloud (or AWS or XX) environment. If you do not have the funds, a possible alternative is to run on Google Colab environment with your data hosted on Google Drive and this document takes you through the steps to do this.**

Taken from https://towardsdatascience.com/detailed-tutorial-build-your-custom-real-time-object-detector-5ade1017fd2d

1. Create New Colab

- Go to the main Colab interface - https://colab.research.google.com/notebooks/intro.ipynb
- Create a new Notebook.

2. Select Runtime Type for Colab

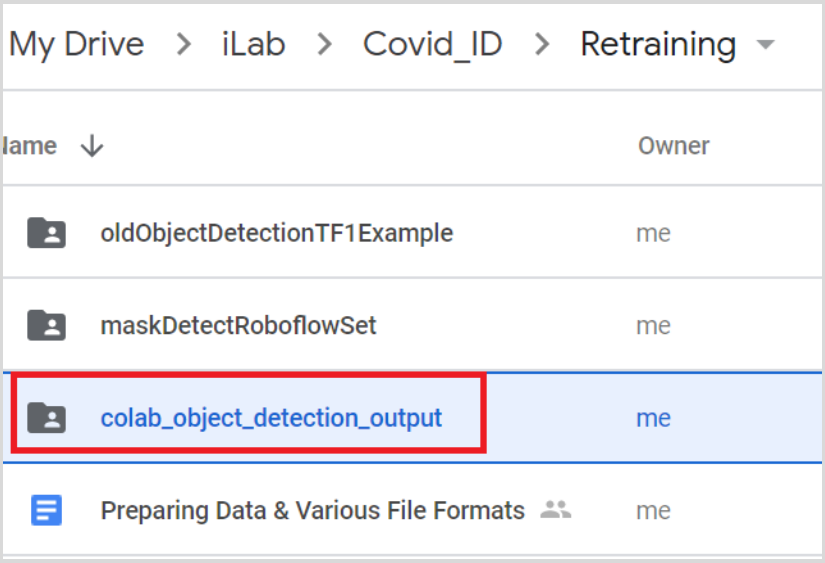
- Colab is run in Google Cloud and you have choice of TPU or GPU and this is great news for us --as Google Colabs with some restrictions are run for free.
- From the top left menu: Go to **Runtime** > **Change runtime type** > select **GPU** from **hardware accelerator.** Some pretrained models support TPU. The pretrained model we are choosing in this project only supports GPU.
  - Read this comparison paper on using TPU versus GPU and the performance for training
  - Here is a colab you can run to compare GPU and TPU performance on colab

3. **Setup Google Backup AND Sync App** (Highly Recommended) to assist with storage of files to Google Drive from your harddrive.....

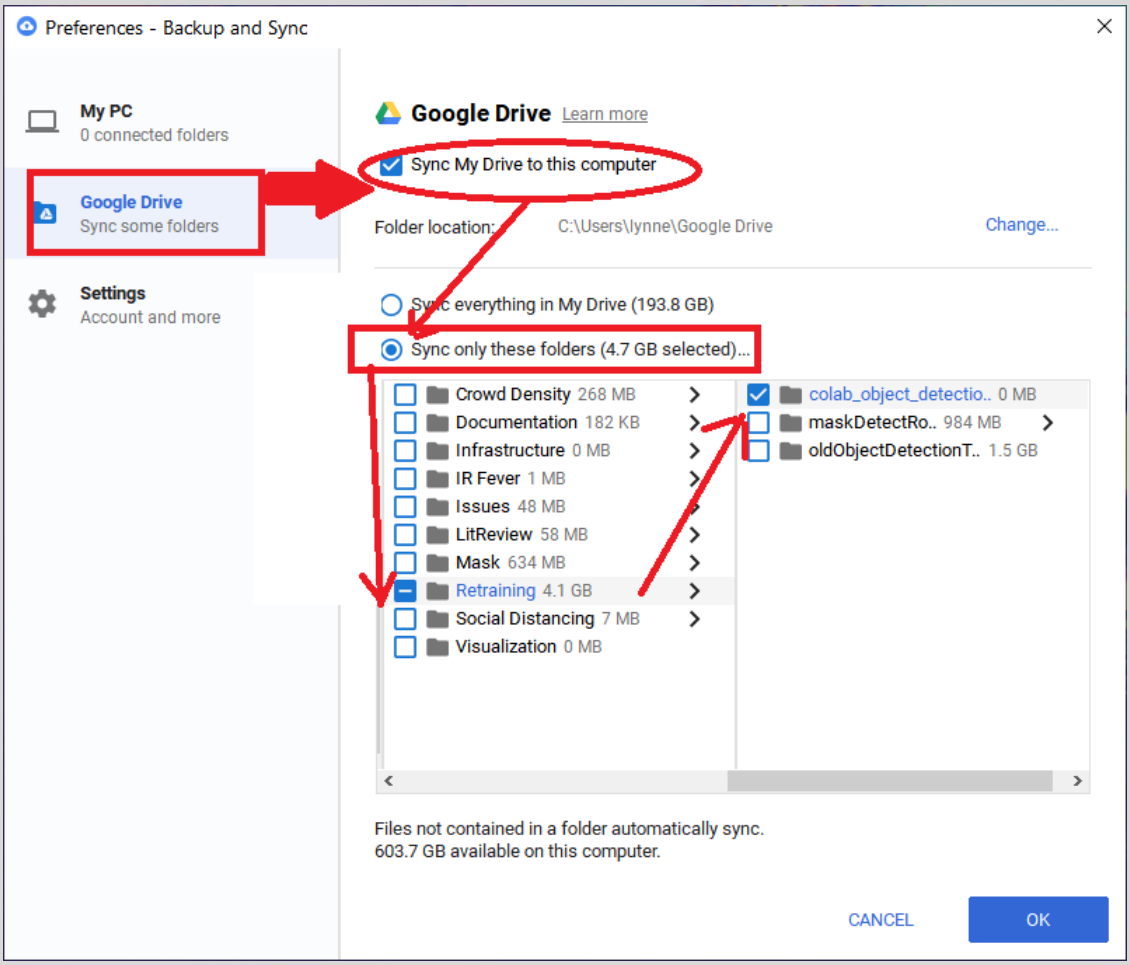
- .

HOW TO DO

1. **Create a folder on Google Drive** that you wish all of the training data like checkpoints that are generated during training to be stored to.
   - Example: I am going to call it ***colab_object_detection_output*** and I am putting this folder in the desired location in my GDrive

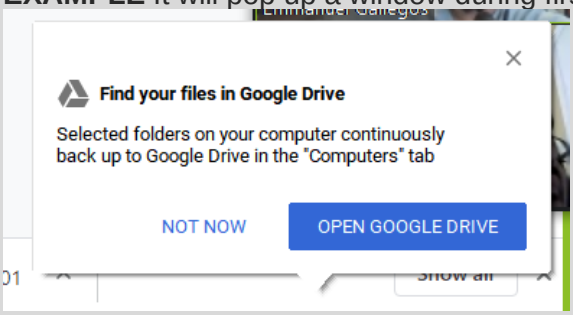


2. **Install [Google Backup and Sync app] & Launch App and Select Folder you created in previous step on Google Drive**
    - Run it and sign in with google account via option for browser (see bottom of app if having trouble sign in with browser)
    - select the ***colab_object_detection_output*** created in the previous step 1. [Google Drive ->Synch My Drive to this computer -> Sync only these folders -> Find the folder you want]

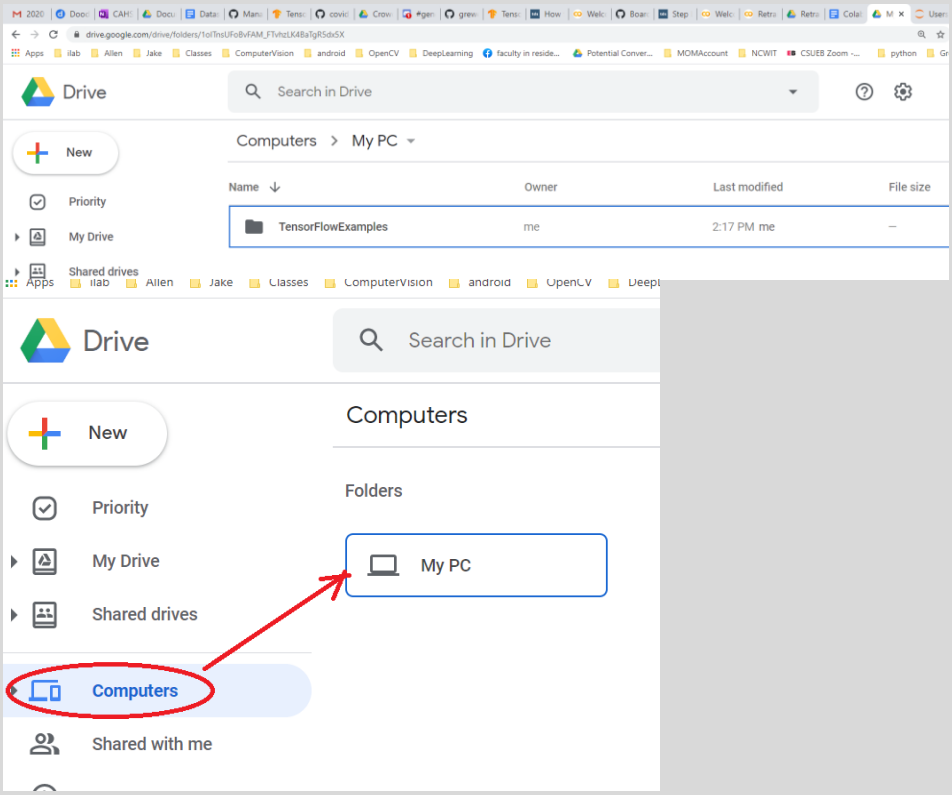


3. Setup up for PC and point ONLY to the folder containing your project you wish to synch (save on Google), it will generate a folder with THAT name on google Drive under (gdrive/'My PC'/YOUR_FOLDER)

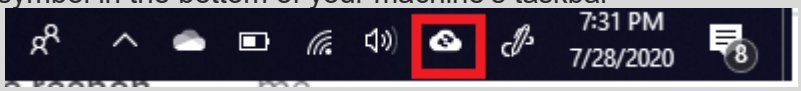
EXAMPLE It will pop up a window during first sync



IF you go to your Google Drive (or follow this button) it will take you to a new folder **"My PC"** that will contain the uploaded (synced from now on automatically???) data you specified as shown here for me for the folder named TensorFlowExamples.

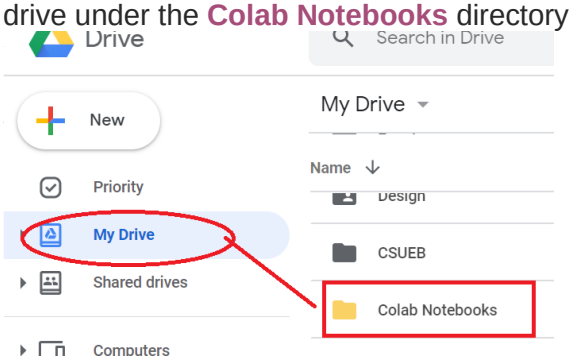


NOTE: you can always bring up the Google Backup and Sync app (it appears as a kind of Cloud symbol in the bottom of your machine's taskbar



4. **IMPORTANT --periodically save your Colab file working on to Drive**

As you create new collabs and work on the code, you need to periodically do a **File->Save** It will appear in your drive under the **Colab Notebooks** directory



5. ## Choose model for retraining

> using the following cell in your colab to select one of the existing pre-trained models that come with Object Detection API for retraining. Below we are selected SSD Mobilenet V2.
TIP: there are MORE models possible --see local install of your Object Detection API for tf2 --- >models/research/object_detection/configs/tf2 for a listing or go online.

```
# Some models to train on
MODELS_CONFIG = {
    'ssd_mobilenet_v2': {
        'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
        'pipeline_file': 'ssd_mobilenet_v2_coco.config',
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
        'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
    },
    'rfcn_resnet101': {
        'model_name': 'rfcn_resnet101_coco_2018_01_28',
        'pipeline_file': 'rfcn_resnet101_pets.config',
    }
}

# Select a model in `MODELS_CONFIG`.
# Here we choose ssd_mobilenet_v2 for this project, you could choose any
selected_model = 'ssd_mobilenet_v2
```

6. ## Setup your Data - Organization:

This tutorial assumes you have setup your directory with the following structure. Note that as this is a detection example, all the images are stored in the images sub-directory and the indication of which are used for training versus testing are given by the directories train_labels and test_labels containing an xml for each corresponding image. So do some kind of sort randomly to extract say the 20% (or whatever %) you will use for testing.

```
object detection
            └── data
                    ├── images
```

```
│       ├── image 1.jpg
│       └── ...
│
├── annotations
│       ├── image 1.xml
│       └── ...
│
├── train labels //contains labels only
│       ├── image 1.xml
│       └── ...
│
└── test labels //contains labels only
        ├── image 50.xml
        └── ...
```

TIP: you can do this random selection of xml files using the following kind of code--read it to understand:

```
#creating two dir for training and testing

!mkdir test_labels train_labels

# lists the files inside 'annotations' in a random order (not really
# random, by their hash value instead)

# Moves the first 400/2000 labels (20% of the labels) to the testing
dir: `test_labels`

!ls annotations/* | sort -R | head -400 | xargs -I{} mv {} test_labels/

# Moves the rest of labels '1600' labels to the training dir: `train_labels

!ls annotations/* | xargs -I{} mv {} train_labels/
```

7. ## Mounting Google Drive: for Dataset loading + storage of created files:

You have a few options for this Option1 mounting the Google Drive via code in colab (option 2 similar) Option 3 mounts the Google Drive via GUI interface in colab. It is assumed (see previous step) you have already uploaded to Google Drive your Data. We will also be using the Google Drive as a location to store our created files like our training files -checkpoints, model, etc.

Tip (after mounting): You can view the full working directory on **Google Colab** Notebook by: Open the left panel by clicking on the top left arrow. Or use ⌘/**Ctrl**+**Alt**+**P** Then Click on **Files** from the top left menu.

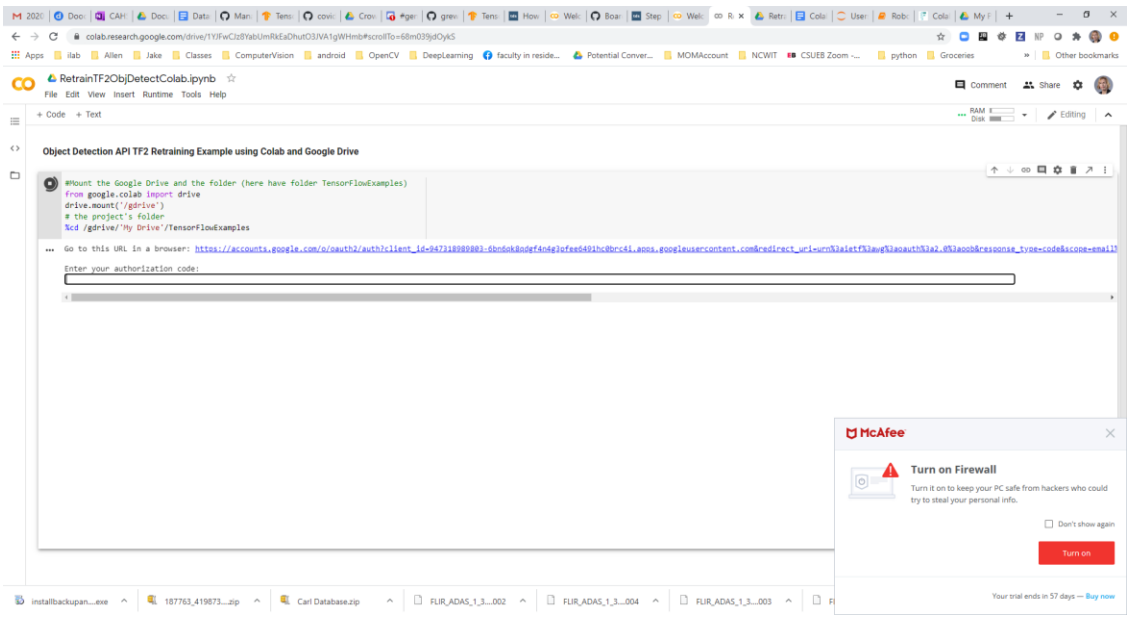
OPTION 1: In Colab cell have code to Mount your Google Drive

In Colab mount Google Drive for access to data

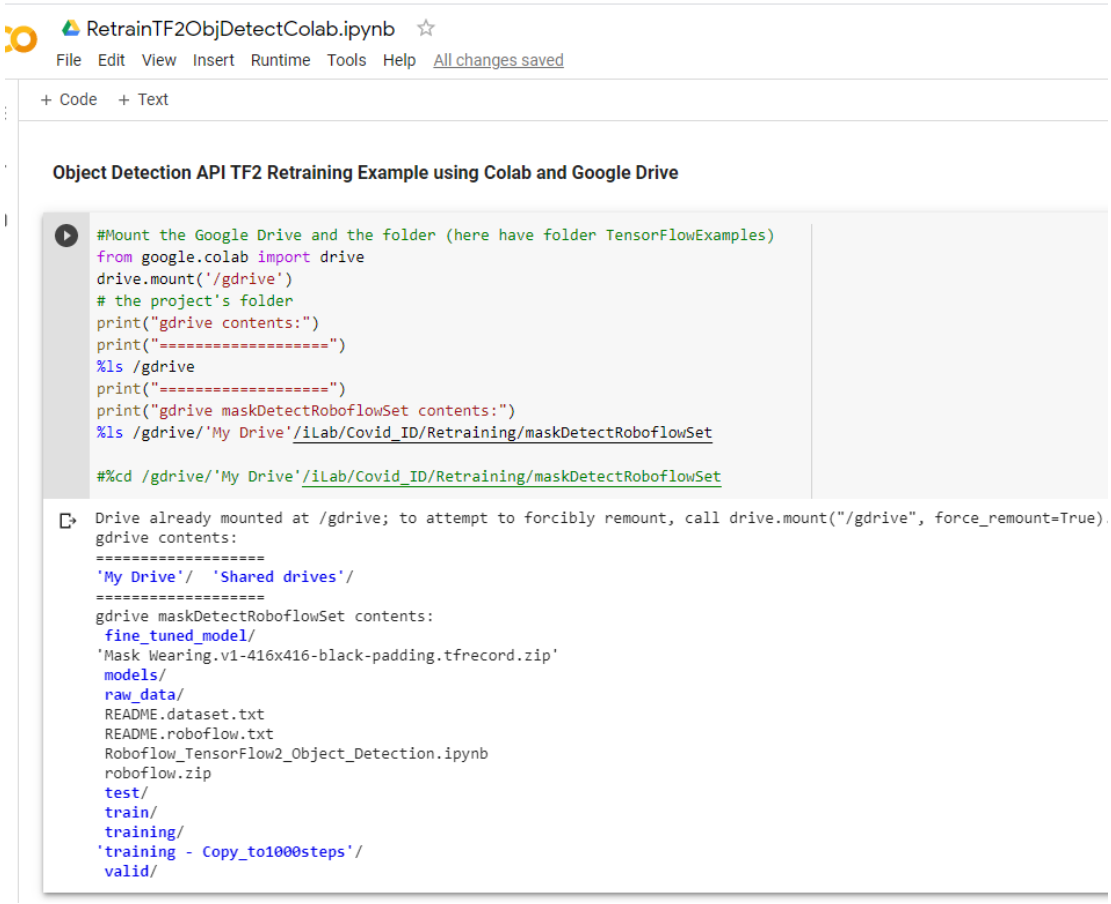
On the Colab Notebook, Mount Gdrive and navigate to the data folder, you will be asked for the authorization code each time you run this code:

```
from google.colab import drive
drive.mount('/gdrive')
# the project's folder
%cd /gdrive/'My Drive'/object_detection
```

EACH time you run this you will be prompted for an authorization code you will get by logging into your Google Drive account



Output of STEP2

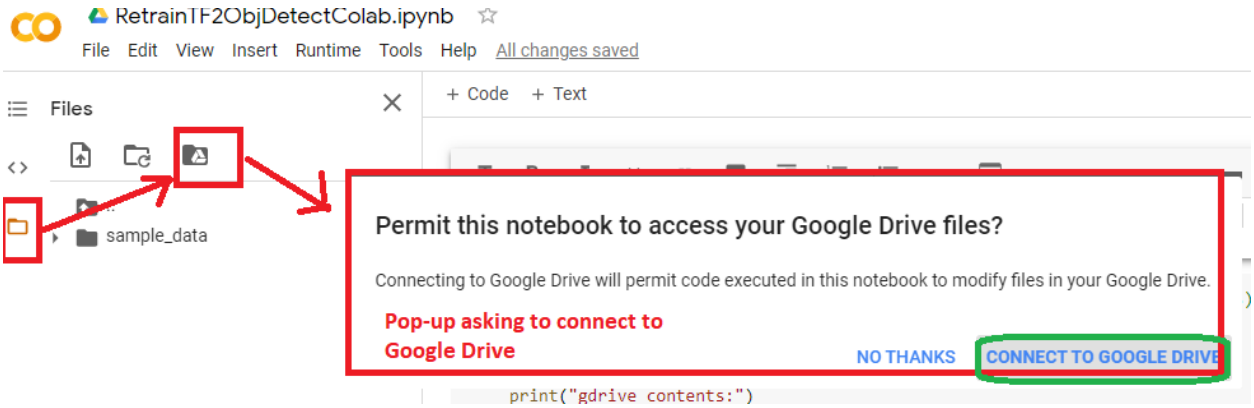


More: Read [here for Another reference regarding using Google Drive for hosting datasets in colab]

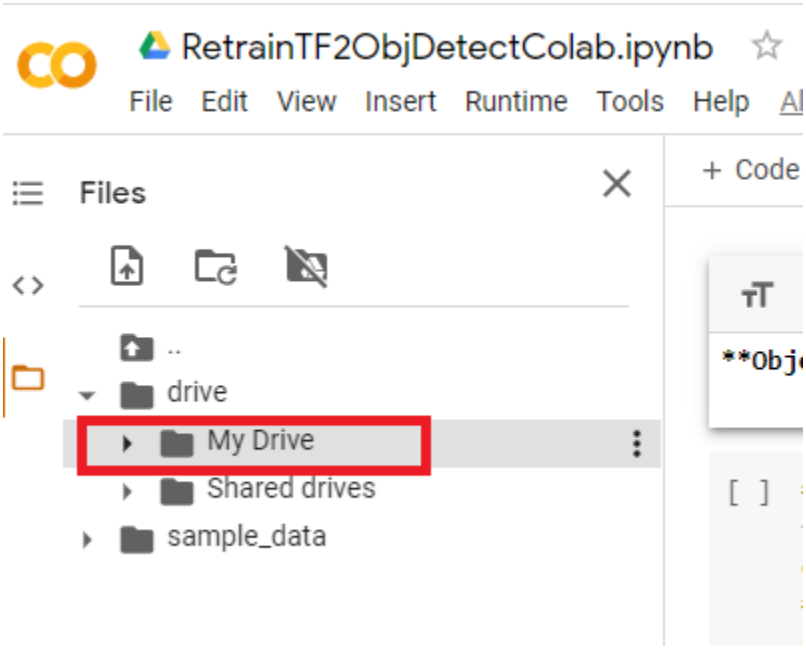
Option 2 you can [import a single file at a time from Google Drive into colab.]

Option 3: MOUNT Google Drive VIA Colab GUI: Select FIles Icon in left -> Mound Google Drive Icon -> will ask you to connect and select folder

**** LYNNE NOTE: I am currently having trouble with this option --it seems to drop the connection to the mounted drive quickly ****



You will be asked to log in and it will take some time to mount it



Now you can add some code in a cell in colab to access the data. NOTE with this method of mounding Google Drive you access with path ***/drive/'My Drive'***

```
#using the GUI way to mount a drive you access not with #drive/'My Drive' to
top directory of the GDrive mounted
# To mount select Files icon on left then Mount Google Drive
root_dir = "drive/'My Drive'/"
base_dir = root_dir + 'iLab/Covid_ID/Retraining/maskDetectRoboflowSet'
%ls {base_dir}
```

Here you can see the output:

```
#using the GUI way to mount a drive you access not with drive/'My Drive' to top
# To mount select Files icon on left then Mount Google Drive
root_dir = "drive/'My Drive'/"

base_dir = root_dir + 'iLab/Covid_ID/Retraining/maskDetectRoboflowSet'
%ls {base_dir}
```

```
 fine_tuned_model/
'Mask Wearing.v1-416x416-black-padding.tfrecord.zip'
 models/
 raw_data/
 README.dataset.txt
 README.roboflow.txt
 Roboflow_TensorFlow2_Object_Detection.ipynb
 roboflow.zip
 test/
 train/
 training/
'training - Copy_to1000steps'/
 valid/
```

EXTENDED CODE EXAMPLE --will display more info

```
#using the GUI way to mount a drive you access not with drive/'My Drive' to top directory of the GDrive mounted
# To mount select Files icon on left then Mount Google Drive
print("You must first mount drive via Colab GUI or this cell will not work")
root_dir = "drive/'My Drive'/"

base_dir = root_dir + 'iLab/Covid_ID/Retraining/DetectionWeaponsExample'
print("base directory")
%ls {base_dir}

print("\n\ndata directory")
data_dir = base_dir + '/data'
print("\n\ntrain dir")
%ls {data_dir}

#tell number of train label files
print("\n\ntrain_labels_dir")
train_labels_dir = data_dir + '/train_labels'
#%ls {train_labels_dir}
%ls -1 {train_labels_dir} | wc -l

print("\n\ntest_labels_dir")
test_labels_dir = data_dir + '/test_labels'
#ls {test_labels_dir}
%ls -1 {test_labels_dir} | wc -l

print("\n\nimages_dir")
images_dir = data_dir + '/images'
#%ls {images_dir}
%ls -1 {images_dir} | wc -l
```

8. ## Shared drive:
If the data you're using is in a shared google drive,this is how you access the path:

```
#Mount the Google Drive and the folder (here have folder
DetectionWeaponsExample_CodeLabBased in the shared drive RetrainTF2DataAndModels)

from google.colab import drive
drive.mount('/gdrive')

%cd /gdrive/Shared\ drives/RetrainTF2DataAndModels/DetectionWeaponsExample_CodeLabBased
```

9. ## During Training Save Checkpoints to Drive

Colab environment shuts down (due to timeouts, in activity,etc) -so it is up to you to save your data throughout the colab processing including training to keep it.

MORE:
When training starts, checkpoints, logs and many other important files will be created. When the Colab kernel disconnects, these files, along with everything else, will be deleted if they don't get saved on your Google Drive or somewhere else.

The kernel disconnects shortly after your computer sleeps or after using the Colab **GPU** for 12 hours. Training will need to be restarted from zero if the trained model did not get saved.[2] --> THIS MEANS you may be training in time "segments" meaning you have 12 hours and you should before the end (ideally near it) save a check point and then restart kernel/colab but, now not from beginning but this saved checkpoint. You will then train for another 12 hours and so on until you are satisfied or reached the total number of steps in training desired.

10. Setup Google Colab VM environment to have necessary software

>Google colab VM environment has most packages already installed (Python, Tensorflow, etc).
>However you may need to install additional packages..below are the additional packages we will install by adding the ***following code to a codelab cell:***

```
!print("Currently Installed")
!pip list
print("\n\n")
print("installing protobuf-compiler python-pil python-lxml and python-tk")
!sudo apt-get update  # this will update location of packages
```

```
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk
print("\n\n")
print("installing Cython contextlib2 pillow lxml matplotlib")
!pip install -qq Cython contextlib2 pillow lxml matplotlib
print("\n\n")
print("installing pycocotools")
!pip install -qq pycocotools
```

```
!pip install tf_slim
```

Setting up Python Path to /models directory

```
import os
```

```
os.environ['PYTHONPATH'] += "/content/drive/My
Drive/DetectionWeaponsExample_CodeLabBased/models/research:/content/drive/My
Drive/DetectionWeaponsExample_CodeLabBased/models/research/slim"
```

11. Import modules you will use in your colab code AND install Object detection API

by adding the ***following code to a codelab cell:***

```
from __future__ import division, print_function, absolute_import
import pandas as pd
import numpy as np
import csv
import re
import cv2
import os
import glob
```

```
import xml.etree.ElementTree as ET
import io
import tensorflow.compat.v1 as tf
from PIL import Image
from collections import namedtuple, OrderedDict
import shutil
import urllib.request
import tarfile
from google.colab import files
```

To install Object Detection API (and put slim folder in path) do following

```
# MUST do each time restart colab kernel
# INSTALL Object Detection API inside the Colab, as sits above TF2
# this will take several minutes as it will copy over all of the object_detection
files
# you have mounted in project_folder/models/research/object_detection
# as you can see by the output of this cell it copies them into the colab
# environment at build/lib/object_detection

#this this is what I need to make the object_detection get installed
models_research_dir = base_dir + "/models/research"
print({models_research_dir})
%cd {models_research_dir}
%ls
!python setup.py install   # from the models/reasearch or whatever file ---look at
the last ReftrainTF2ObjDetect.ipynb did locally


#this is for setting up path for slim
import os
os.environ['PYTHONPATH'] += ':/content/drive/My
Drive/DetectionWeaponsExample_CodeLabBased/models/research:/content/drive/My
Drive/DetectionWeaponsExample_CodeLabBased/models/research/slim'
```

12. Preprocessing Data

- STEP A: convert images to correct size for model
- take xml files in train_labels and test_labels and make single train_labels.csv and test_labels.csv

- create label_map.pbtext that lists classes and names

Preprocessing Images and Labels

1. Converting the annotations from xml files to two csv files for each `train_labels/` and `train_labels/`.

2. Creating a pbtxt file that specifies the number of class (one class in this case)

3. Checking if the annotations for each object are placed within the range of the image width and height.

```
#checks if the images box position is placed within the image.

#note: while this doesn't checks if the boxes/annotatoins are correctly
# placed around the object, Tensorflow will through an error if this
occured.
%cd /content/gun_detection/data
# path to images
images_path = 'images'

#loops over both train_labels and test_labels csv files to do the check
# returns the image name where an error is found
# return the incorrect attributes; xmin, ymin, xmax, ymax.
for CSV_FILE in ['train_labels.csv', 'test_labels.csv']:
  with open(CSV_FILE, 'r') as fid:
      print('[*] Checking file:', CSV_FILE)
      file = csv.reader(fid, delimiter=',')
      first = True
      cnt = 0
      error_cnt = 0
      error = False
      for row in file:
          if error == True:
              error_cnt += 1
              error = False
          if first == True:
              first = False
              continue
          cnt += 1
          name, width, height, xmin, ymin, xmax, ymax = row[0],
int(row[1]), int(row[2]), int(row[4]), int(row[5]), int(row[6]),
int(row[7])
```

```
            path = os.path.join(images_path, name)
            img = cv2.imread(path)
            if type(img) == type(None):
                error = True
                print('Could not read image', img)
                continue
            org_height, org_width = img.shape[:2]
            if org_width != width:
                error = True
                print('Width mismatch for image: ', name, width, '!=',
org_width)
            if org_height != height:
                error = True
                print('Height mismatch for image: ', name, height, '!=',
org_height)
            if xmin > org_width:
                error = True
                print('XMIN > org_width for file', name)
            if xmax > org_width:
                error = True
                print('XMAX > org_width for file', name)
            if ymin > org_height:
                error = True
                print('YMIN > org_height for file', name)
            if ymax > org_height:
                error = True
                print('YMAX > org_height for file', name)
            if error == True:
                print('Error for file: %s' % name)
                print()
        print()
        print('Checked %d files and realized %d errors' % (cnt,
error_cnt))
        print("-----")
```

- convert the images in images_dir and the single csv files to the TFRecord files train_labels.record and test_labels.record that will be use for faster data presentation in training
  - There is a part of the code below that must be fixed as hard coded (as in original example)

- OUTPUT = new or updated train_labels.record and test_labels.record files in your Google Drive path specified

```
#### **Generate TFRecords: Convert csv + images to TFRecord files
(train_labels.record, test_lables.record**
1. Read in the train_labels.csv and the corresponding images to create a
train_labels.record a TFRecord file
2. Read in the test_labels.csv and the corresponding images to create a
test_labels.record a TFRecord file

**HARD CODED MUST FIX: ** The following hard codes for the class pistol to convert
to the class number 1. Need to read in the label_map.pbtxt file instead and parse
it for multiple classes and create a map so can look up the id for the passed
row_label ---see code below
```

```
#adjusted from: https://github.com/datitran/raccoon_dataset

# converts the csv files for training and testing data to two TFRecords files.
# places the output in the same directory as the input
# NOTE: WEIRD PROBLEM with the paths for Mounted Google Drive here where need
/gdrive/'My Drive'/*** to do any ls or cat
#        but to open a file for reading or writing need just /gdrive/My Drive/***
#QUESTION: why did the writing of the csv and pbtext work in cell up above with
the 'My Drive' around path???????????


from object_detection.utils import dataset_util
%cd {data_dir}

# problem with the data_dir not evaluating the 'My Drive' into My Drive
#DATA_BASE_PATH = data_dir + '/'
#image_dir = data_dir +'/images/'
DATA_BASE_PATH = "/gdrive/My
Drive/iLab/Covid_ID/Retraining/DetectionWeaponsExample/data/"
images_dir = "/gdrive/My
Drive/iLab/Covid_ID/Retraining/DetectionWeaponsExample/data/images/"
print("Data base path " + DATA_BASE_PATH)

print("Images path " + images_dir)
```

```
#do a list to see if the record file already exists
#this will falue when no 'My Drive' and using only My Drive
#%ls {DATA_BASE_PATH}

#FIX- THIS IS HARDCODED method for converting the class label to its id instead!!!
def class_text_to_int(row_label):
    if row_label == 'pistol':
        return 1
    else:
        None

#some kind of parsing function that create a special DataSet for parsing each
image in a loop
def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in
zip(gb.groups.keys(), gb.groups)]

#This is a function that reads in image from a file (using tf.io package) and its
bounding box information and creates
# and instance of tf.train.Example that can be used to add to a TFRecord
def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb')
as fid:
        encoded_jpg = fid.read()
    #open up io stream to file containing image
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    #open up Image file pointer using the stream previously opened
    image = Image.open(encoded_jpg_io)
    #retrieve size of image from the data in the Image file pointer (stored in the
jpg file)
    width, height = image.size


    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    #setup array to represent all the bounding boxes for this image
    # bounding box i upper left point = (xmins[i],ymins[i])  lower right point
=(xmaxs[i], ymaxs[i])
```

```
    # class label of ith' box stored in classes_text[i]
    # also as building out this array add to classes[] any unique new classes
found
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    #cycle through the rows in the label csv file pased and add in the bounding
box info into arrays
    #    and corresponding class label.
    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    #build out a tf.train.Example using all the read in information for this image
and its bounding boxes
    # this will be used later to create a TFRecord
    # see https://www.tensorflow.org/tutorials/load_data/tfrecord  for information
about tf.train.Example and TFRecord format
    # as you can see includes for each image:
    #              height, width, filename, the actual image pixel values, image
format,
    #              and bounding boxes (as arrays of xmin,ymin and xmax,ymax
representing the lower-left and upper-right points)
    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
```

```
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example

#go through the train_labels.csv and afterwards test_lables.csv and create
TFRecord files for each
#  pd is associated with loaded python Pandas module imported and used to read csv
files
for csv in ['train_labels', 'test_labels']:
  #use TFrecordWriter to write records to a TFRecord file as specified in path
  #see https://www.tensorflow.org/api_docs/python/tf/io/TFRecordWriter
  label_file = DATA_BASE_PATH + csv + '.record'
  print(".........going to save TFRecord to " + label_file)
 # label_file = "/"

  writer = tf.io.TFRecordWriter(DATA_BASE_PATH + csv + '.record')
  #writer = tf.io.TFRecordWriter(dummy2)
  path = os.path.join(images_dir)

  #read in all the rows in the csv file using pandas module into a pandas
DataFrame datastructure
  #see https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.read_csv.html
  #see https://pandas.pydata.org/pandas-docs/stable/reference/frame.html
  # need file to open = "/gdrive/My
Drive/iLab/Covid_ID/Retraining/DetectionWeaponsExample/data/" + csv + ".csv"
  print(DATA_BASE_PATH + csv + '.csv')
  examples = pd.read_csv(DATA_BASE_PATH + csv + '.csv')


  #For each image group it with its bounding boxes
  grouped = split(examples, 'filename')

  #for each image and its bounding boxes create an instance of tf.train.Example
that is
  #  written out into a file that is the created TFRecord file
  # see https://www.tensorflow.org/tutorials/load_data/tfrecord
  # for information about tf.train.Example and TFRecord format
  for group in grouped:
```

```
      print(" group in loop ")
      print(group)
      tf_example = create_tf_example(group, path)
      writer.write(tf_example.SerializeToString())

  writer.close()
  output_path = os.path.join(os.getcwd(), DATA_BASE_PATH + csv + '.record')
  print('Successfully created the TFRecords: {}'.format(DATA_BASE_PATH +csv +
'.record'))
```

- Download ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz model for tf2 and save it

```
#Download ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz

%mkdir /content/drive/My\
Drive/DetectionWeaponsExample_CodeLabBased/models/research/deploy/

%cd /content/drive/My
Drive/DetectionWeaponsExample_CodeLabBased/models/research/deploy/

model_import_name = 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz'

import tarfile

download_tar =
'http://download.tensorflow.org/models/object_detection/tf2/20200711/' +
model_import_name


!wget {download_tar}

tar = tarfile.open(model_import_name)

tar.extractall()

tar.close()
```

13. Copy over the config file for your choosen model to project_directory/config subdirectory you create

```
# SSD with Mobilenet v2
# Trained on COCO17, initialized from Imagenet classification checkpoint
# Train on TPU-8
#
# Achieves 22.2 mAP on COCO17 Val

model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 1  [Change it to the number of classes the object detection works on]
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      ssd_anchor_generator {
```

```
    num_layers: 6
    min_scale: 0.2
    max_scale: 0.95
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    aspect_ratios: 3.0
    aspect_ratios: 0.3333
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 300
    width: 300
  }
}
box_predictor {
  convolutional_box_predictor {
    min_depth: 0
    max_depth: 0
    num_layers_before_predictor: 0
    use_dropout: false
    dropout_keep_probability: 0.8
    kernel_size: 1
    box_code_size: 4
    apply_sigmoid_to_scores: false
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
      batch_norm {
        train: true,
        scale: true,
        center: true,
        decay: 0.97,
        epsilon: 0.001,
      }
    }
  }
}
feature_extractor {
```

```
  type: 'ssd_mobilenet_v2_keras'
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
      }
    }
    batch_norm {
      train: true,
      scale: true,
      center: true,
      decay: 0.97,
      epsilon: 0.001,
    }
  }
  override_base_feature_extractor_hyperparams: true
}
loss {
  classification_loss {
    weighted_sigmoid_focal {
      alpha: 0.75,
      gamma: 2.0
    }
  }
  localization_loss {
    weighted_smooth_l1 {
      delta: 1.0
    }
  }
  classification_weight: 1.0
  localization_weight: 1.0
}
normalize_loss_by_num_matches: true
normalize_loc_loss_by_codesize: true
post_processing {
  batch_non_max_suppression {
    score_threshold: 1e-8
    iou_threshold: 0.6
    max_detections_per_class: 100
    max_total_detections: 100
  }
  score_converter: SIGMOID
```

```
    }
  }
}

train_config: {
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/mobilenet_v2__coco17_tpu-8/checkpoint/ckpt-0"  [this is the saved checkpoint of the model]
  fine_tune_checkpoint_type: "detection" [Change it to Detection as we are doing detection rather than classification]
  batch_size: 24  [Change the batch size according to the memory your pc can handle, try changing it to 12 to reduce memory load]
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 50000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .8
          total_steps: 50000 [Change the number of steps accordingly]
          warmup_learning_rate: 0.13333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  label_map_path: "PATH_TO_BE_CONFIGURED/label_map.txt" [Replace it with your google drive path inside /data]
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/train2017-?????-of-00256.tfrecord" [Replace it with your google drive path inside /data]
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
```

```
  use_moving_averages: false
}

eval_input_reader: {
  label_map_path: "PATH_TO_BE_CONFIGURED/label_map.txt" [Replace it with your google drive path inside /data]
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/val2017-?????-of-00032.tfrecord" [Replace it with your google drive path inside /data]
  }
}
```

14. Training calling script

- **For TF1:**

```
!python3 /gdrive/My Drive/iLab/Covid_ID/Retraining/ColabRetraining/DetectionWeaponsExample_CoLabBased/models/research/object_detection/model_main.py \
--pipeline_config_path=/gDrive/My Drive/iLab/Covid_ID/Retraining/ColabRetraining/DetectionWeaponsExample_CoLabBased/config/ssd_mobilenet_v2_coco.config
--model_dir=/gdrive/My Drive/iLab/Covid_ID/Retraining/ColabRetraining/DetectionWeaponsExample_CoLabBased/training
```

- **For TF2:**

```
!python model_main_tf2.py \
    --pipeline_config_path=training/ssd_efficientdet_d0_512x512_coco17_tpu-8.config \
    --model_dir=training \
    --alsologtostderr
```

15. Export saved model for retraining

```
!python exporter_main_v2.py \
    --trained_checkpoint_dir=training \
    --pipeline_config_path=training/ssd_efficientdet_d0_512x512_coco17_tpu-8.config \
    --output_directory=inference_graph \
```

16. Testing images on the saved model

```
%cd /content/drive/My Drive/Google_Colab_Training/Object_detection/models/research
import io
import os
import scipy.misc
import numpy as np
import six
import time
import glob
from IPython.display import display

from six import BytesIO

import matplotlib
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont

import tensorflow as tf
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

%matplotlib inline


def load_image_into_numpy_array(path):
  """Load an image from file into a numpy array.

  Puts image into numpy array to feed into tensorflow graph.
```

```
  Note that by convention we put it into a numpy array with shape
  (height, width, channels), where channels=3 for RGB.

  Args:
    path: a file path (this can be local or on colossus)

  Returns:
    uint8 numpy array with shape (img_height, img_width, 3)
  """
  img_data = tf.io.gfile.GFile(path, 'rb').read()
  image = Image.open(BytesIO(img_data))
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)


labelmap_path = '/content/drive/My
Drive/Google_Colab_Training/Object_detection/models/research/object_detection/training/l
abelmap.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(labelmap_path,
use_display_name=True)



tf.keras.backend.clear_session()

output_directory = 'drive/My
Drive/Google_Colab_Training/Object_detection/models/research/object_detection/inference_
graph'
model = tf.saved_model.load(f'/content/{output_directory}/saved_model')



def run_inference_for_single_image(model, image):
  image = np.asarray(image)
  # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
  input_tensor = tf.convert_to_tensor(image)
  # The model expects a batch of images, so add an axis with `tf.newaxis`.
  input_tensor = input_tensor[tf.newaxis,...]

  # Run inference
  model_fn = model.signatures['serving_default']
```

```
  output_dict = model_fn(input_tensor)

  # All outputs are batches tensors.
  # Convert to numpy arrays, and take index [0] to remove the batch dimension.
  # We're only interested in the first num_detections.
  num_detections = int(output_dict.pop('num_detections'))
  output_dict = {key:value[0, :num_detections].numpy()
                 for key,value in output_dict.items()}
  output_dict['num_detections'] = num_detections

  # detection_classes should be ints.
  output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

  # Handle models with masks:
  if 'detection_masks' in output_dict:
    # Reframe the the bbox mask to the image size.
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
              output_dict['detection_masks'], output_dict['detection_boxes'],
               image.shape[0], image.shape[1])
    detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                       tf.uint8)
    output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

  return output_dict



%cd /content/drive/My
Drive/Google_Colab_Training/Object_detection/models/research/object_detection/

for image_path in glob.glob('data/images/test/*.jpg'):
  image_np = load_image_into_numpy_array(image_path)
  output_dict = run_inference_for_single_image(model, image_np)
  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_reframed', None),
      use_normalized_coordinates=True,
      line_thickness=8)
```

```
display(Image.fromarray(image_np))
```

If everything went good, you will probably see the output here….