```
import numpy as np
import pandas as pd
import tensorflow as tf
from PIL import Image
from matplotlib import pyplot as plt
from tensorflow.python.util import compat
from tensorflow.core.protobuf import saved_model_pb2
from google.protobuf import text_format
import pprint
import json
import os
```

```
# needed to install object_detection library and enlarge labels
! rm -rf ./models && git clone https://github.com/tensorflow/models.git \
    && sed -i "s#ImageFont.truetype('arial.ttf', 24)#ImageFont.truetype('arial.ttf', 50)#g" ./models/research/object_detection/utils/visualiza
    && cp /usr/share/fonts/truetype/dejavu/DejaVuSans.ttf /usr/share/fonts/truetype/dejavu/arial.ttf
```

```
Cloning into 'models'...
remote: Enumerating objects: 84934, done.
remote: Counting objects: 100% (562/562), done.
remote: Compressing objects: 100% (253/253), done.
remote: Total 84934 (delta 339), reused 514 (delta 304), pack-reused 84372
Receiving objects: 100% (84934/84934), 597.51 MiB | 19.19 MiB/s, done.
Resolving deltas: 100% (60833/60833), done.
cp: cannot stat '/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf': No such file or directory
```

```
# install object_detection library
! cd models/research \
    && protoc object_detection/protos/*.proto --python_out=. \
    && cp object_detection/packages/tf2/setup.py . && \
    python3 -m pip install --use-feature=2020-resolver .
```

```
Usage:
  /usr/bin/python3 -m pip install [options] <requirement specifier> [package-index-options] ...
  /usr/bin/python3 -m pip install [options] -r <requirements file> [package-index-options] ...
  /usr/bin/python3 -m pip install [options] [-e] <vcs project url> ...
  /usr/bin/python3 -m pip install [options] [-e] <local project path> ...
  /usr/bin/python3 -m pip install [options] <archive url/path> ...

option --use-feature: invalid choice: '2020-resolver' (choose from 'fast-deps', 'truststore', 'no-binary-enable-wheel-cache')
```

```
!pip install tensorflow-object-detection-api
```

```
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.9/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.9/dist-packages (from jsonschem
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-auth
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from requests-oauthlib>=0.7.0->google-auth-o
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packages (from cffi>=1.12->cryptography>=2.0->SecretStorage>
Building wheels for collected packages: tensorflow-object-detection-api
  Building wheel for tensorflow-object-detection-api (setup.py) ... done
  Created wheel for tensorflow-object-detection-api: filename=tensorflow_object_detection_api-0.1.1-py3-none-any.whl size=844510 sha25
  Stored in directory: /root/.cache/pip/wheels/38/6f/73/3c0c70f9a8219c31c96c495b686af1416154b60707fd96a4cd
Successfully built tensorflow-object-detection-api
Installing collected packages: rfc3986, qtpy, pkginfo, jeepney, jedi, jaraco.classes, requests-toolbelt, readme-renderer, SecretStorag
```

```
from object_detection.utils import visualization_utils as vis_util
from object_detection.utils import dataset_util, label_map_util
from object_detection.protos import string_int_label_map_pb2
```

```
# reconstruct frozen graph
def reconstruct(pb_path):
    if not os.path.isfile(pb_path):
        print("Error: %s not found" % pb_path)

    print("Reconstructing Tensorflow model")
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        od_graph_def = tf.compat.v1.GraphDef()
        with tf.io.gfile.GFile(pb_path, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
    print("Success!")
    return detection_graph
```

```
# visualize detection
def image2np(image):
    (w, h) = image.size
    return np.array(image.getdata()).reshape((h, w, 3)).astype(np.uint8)

def image2tensor(image):
    npim = image2np(image)
    return np.expand_dims(npim, axis=0)

%matplotlib inline
def detect(detection_graph, test_image_path):
```

```
    with detection_graph.as_default():
        gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_memory_fraction=0.01)
        with tf.compat.v1.Session(graph=detection_graph,config=tf.compat.v1.ConfigProto(gpu_options=gpu_options)) 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')

            image = Image.open(test_image_path)
            (boxes, scores, classes, num) = sess.run(
                [detection_boxes, detection_scores, detection_classes, num_detections],
                feed_dict={image_tensor: image2tensor(image)}
            )

            npim = image2np(image)
            vis_util.visualize_boxes_and_labels_on_image_array(
                npim,
                np.squeeze(boxes),
                np.squeeze(classes).astype(np.int32),
                np.squeeze(scores),
                category_index,
                use_normalized_coordinates=True,
                line_thickness=15)
            plt.figure(figsize=(12, 8))
            plt.imshow(npim)
            plt.show()
```

```
directory_path = r'C:\Users\16027\Downloads\archive\data'
#ANNOTATIONS_FILE = os.path.join(directory_path,ANNOTATIONS)
#json_file_path = os.path.normpath(os.path.join(directory_path, 'annotations.json'))
file_name = '\\annotations.json'
json_file_path = directory_path + file_name
#json_file_path = '"' + json_file_path + '"'
print(type(json_file_path))
NCLASSES = 60
print(json_file_path)
#df = pd.read_json(json_file_path)
```

```
<class 'str'>
C:\Users\16027\Downloads\archive\data\annotations.json
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
DATA_DIR = '/content/drive/MyDrive/data'
ANNOTATIONS_FILE = os.path.join(DATA_DIR, 'annotations.json')
NCLASSES = 60
```

```
with open(ANNOTATIONS_FILE) as json_file:
    data = json.load(json_file)

categories = data['categories']
```

```
print('Building label map from examples')

labelmap = string_int_label_map_pb2.StringIntLabelMap()
for idx,category in enumerate(categories):
    item = labelmap.item.add()
    # label map id 0 is reserved for the background label
    item.id = int(category['id'])+1
    item.name = category['name']

with open('./labelmap.pbtxt', 'w') as f:
    f.write(text_format.MessageToString(labelmap))

print('Label map witten to labelmap.pbtxt')

with open('./labelmap.pbtxt') as f:
    pprint.pprint(f.readlines())
```

```
          id: 19\n',
        '}\n',
        'item {\n',
        '  name: "Pizza box"\n',
        '  id: 20\n',
        '}\n',
        'item {\n',
        '  name: "Paper cup"\n',
        '  id: 21\n',
        '}\n',
        'item {\n',
        '  name: "Disposable plastic cup"\n',
        '  id: 22\n',
        '}\n',
        'item {\n',
        '  name: "Foam cup"\n',
        '  id: 23\n',
        '}\n',
        'item {\n',
        '  name: "Glass cup"\n',
        '  id: 24\n',
        '}\n',
        'item {\n',
        '  name: "Other plastic cup"\n',
        '  id: 25\n',
        '}\n',
        'item {\n',
        '  name: "Food waste"\n',
        '  id: 26\n',
        '}\n',
        'item {\n',
        '  name: "Glass jar"\n',
        '  id: 27\n',
        '}\n',
        'item {\n',
        '  name: "Plastic lid"\n',
        '  id: 28\n',
        '}\n',
        'item {\n',
        '  name: "Metal lid"\n',
        '  id: 29\n',
        '}\n',
        'item {\n',
        '  name: "Other plastic"\n',
        '  id: 30\n',
        '}\n',
```

```
from tensorflow.python.lib.io import file_io
```

```
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.protos import string_int_label_map_pb2
```

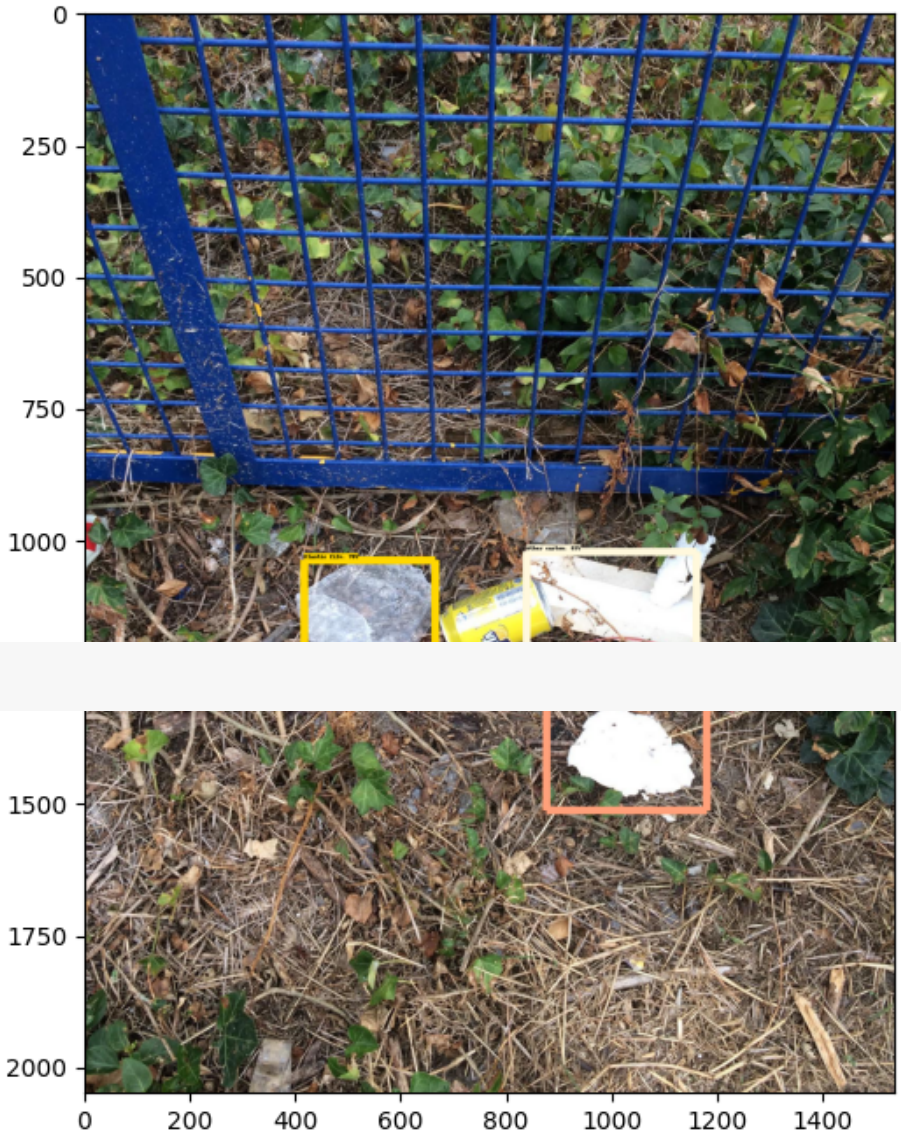
```
with tf.io.gfile.GFile('labelmap.pbtxt', 'r') as f:
    label_map_string = f.read()
    label_map = string_int_label_map_pb2.StringIntLabelMap()
    try:
        text_format.Merge(label_map_string, label_map)
    except text_format.ParseError:
        label_map.ParseFromString(label_map_string)
```

```
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=NCLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

```
detection_graph = reconstruct("/content/drive/MyDrive/data1/ssd_mobilenet_v2_taco_2018_03_29.pb")
```

```
Reconstructing Tensorflow model
Success!
```

```
detect(detection_graph, '/content/drive/MyDrive/data/batch_1/000014.jpg')
```


0
250
500
750
1000

1500
1750
2000
0
200
400
600
800
1000
1200
1400

✓ 11s completed at 3:07 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.