IBM INTERSCHOOL HACKATHON 2020

Project statement: Biohazard waste/ Hospital waste need to be properly treated and managed. It is often a task for authorities to keep a track of it and this leads to contamination of our environment.

Team number: 4

Team Name: Team no. 4

Team Leader: Arindom Sharma

Team members: Abhilasha Sharma, Saurav Sharma, Sudipa Roy, Sirajul Islam

#The steps needed are:

1. Gathering data
2. Labeling data
3. Generating TFRecords for training
4. Configuring training
5. Training model
6. Exporting inference graph
7. Testing object detector

(**Here, Due to limited time we are only showing a prototype that can identify only one biohazard object that is syringe however using similar method more names of biohazard waste can be added also more substance can be added to the dataset.**)

#1.Gathering data

 # script to transform the resolution of images.

```
from PIL import Image

import os

import argparse

def rescale_images(directory, size):

    for img in os.listdir(directory):

        im = Image.open(directory+img)

        im_resized = im.resize(size, Image.ANTIALIAS)

        im_resized.save(directory+img)

if __name__ == '__main__':
```

```
    parser = argparse.ArgumentParser(description="Rescale images")
    parser.add_argument('-d', '--directory', type=str, required=True, help='Directory containing the images')
    parser.add_argument('-s', '--size', type=int, nargs=2, required=True, metavar=('width', 'height'), help='Image size')
    args = parser.parse_args()
    rescale_images(args.directory, args.size)
```

#2.Labeling data

#Now that we have our images we need to move about 80 percent of the images into the object_detection/images/train directory and the other 20 percent in the object_detection/images/test directory.

#In order to label our data, we need some kind of image labeling software. LabelImg is a great tool for that purpose. We label the pictures of syrings as biohazard here.

#3. Generating TFRecords for training

xml_to_csv.py

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
```

```
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
    for folder in ['train', 'test']:
        image_path = os.path.join(os.getcwd(), ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/'+folder+'_labels.csv'), index=None)
        print('Successfully converted xml to csv.')



main()
```

#Now we can transform our xml files to csvs by opening the command line and typing:

python xml_to_csv.py

generate_tfrecord.py

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
flags.DEFINE_string('image_dir', '', 'Path to images')
FLAGS = flags.FLAGS



# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'syringe':
        return 1
    else:
        None
```

```
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)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    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']))
```

```
    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


def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

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

```
if __name__ == '__main__':
    tf.app.run()
```

#Now the TFRecords can be generated by typing:

python generate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images\train --output_path=train.record

python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images\test --output_path=test.record

#These two commands generate a train.record and a test.record file which can be used to train our object detector.

#4. Configuring training

#The last thing we need to do before training is to create a label map and a training configuration file.

#Creating a label map

#The label map maps an id to a name. We will put it in a folder called training, which is located in the object_detection directory. The labelmap for my detector can be seen below.

```
item {
    id: 1
    name: 'Biohazard syringe'
}
```

Creating a training configuration

#Now we need to create a training configuration file. Because as my model of choice I will use faster_rcnn_inception, which just like a lot of other models can be downloaded

from https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md page

#I will start with a sample config (faster_rcnn_inception_v2_pets.config), which can be found in the sample folder.

#First of I will copy the file into the training folder and then I will open it using a text editor in order to change a few lines in the config.

#Line 9: change the number of classes to number of objects you want to detect (1 in my case)

#Line 106: change *fine_tune_checkpoint* to the path of the *model.ckpt* file:

```
fine_tune_checkpoint:
"C:/Users/Gilbert/Downloads/Other/models/research/object_detecti
on/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
```

#Line 123: change *input_path* to the path of the *train.records* file:

```
input_path:
"C:/Users/Gilbert/Downloads/Other/models/research/object_detecti
on/train.record"
```

#Line 135: change *input_path* to the path of the *test.records* file:

```
input_path:
"C:/Users/Gilbert/Downloads/Other/models/research/object_detecti
on/test.record"
```

#Line 125–137: change *label_map_path* to the path of the label map:

```
label_map_path:
"C:/Users/Gilbert/Downloads/Other/models/research/object_detecti
on/training/labelmap.pbtxt"
```

#Line 130: change *num_example* to the number of images in your test folder.(6 in my case)

#5.Training model

#To train the model we will use the *train.py* file, which is located in the *object_detection/legacy* folder. We will copy it into the *object_detection* folder and then we will open a command line and type:

#Update: Use the model_main file in the object_detection folder instead

```
python model_main.py --logtostderr --model_dir=training/ --
pipeline_config_path=training/faster_rcnn_inception_v2_pets.conf
ig
```

#About every 5 minutes the current loss gets logged to Tensorboard. We can open Tensorboard by opening a second command line, navigating to the *object_detection* folder and typing:

```
tensorboard --logdir=training
```

#This will open a webpage at localhost:6006.

#6. Exporting inference graph

#Now that we have a trained model we need to generate an inference graph, which can be used to run the model. For doing so we need to first of find out the highest saved step number. For this, we need to navigate to the training directory and look for the model.ckpt file with the biggest index.

#Then we can create the inference graph by typing the following command in the command line.

```
python export_inference_graph.py --input_type image_tensor --
pipeline_config_path
training/faster_rcnn_inception_v2_pets.config --
trained_checkpoint_prefix training/model.ckpt-XXXX --
output_directory inference_graph
```

#7. Testing object detector

```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow as tf

import zipfile


from distutils.version import StrictVersion

from collections import defaultdict

from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image


# This is needed since the notebook is stored in the object_detection folder.

sys.path.append("..")

from object_detection.utils import ops as utils_ops


if StrictVersion(tf.__version__) < StrictVersion('1.9.0'):
```

```
  raise ImportError('Please upgrade your TensorFlow installation to v1.9.* or later!')



# This is needed to display the images.
%matplotlib inline


from utils import label_map_util


from utils import visualization_utils as vis_util




MODEL_NAME = 'inference_graph'
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = 'training/labelmap.pbtxt'




detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')



category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)
```

```
def run_inference_for_single_image(image, graph):
  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
In [ ]:
import cv2
cap = cv2.VideoCapture(0)
try:
  with detection_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)

        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)
          # 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)
            cv2.imshow('object_detection', cv2.resize(image_np, (800, 600)))
            if cv2.waitKey(25) & 0xFF == ord('q'):
                cap.release()
                cv2.destroyAllWindows()
                break
except Exception as e:
    print(e)
    cap.release()
```

(#After this add all the biohazard objects to the list of biohazard waste and also make the dataset of the following list.)