LAMPIRAN

Lampiran 1 *Script Label Map*

```
item {
   id: 1
   name: 'honda'
}
item {
   id: 2
   name: 'yamaha'
}
```

Lampiran 2 *Script XML to CSV*

```
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 directory in ['train', 'test']:
        image_path = os.path.join(os.getcwd(),
'annotations/{}'.format(directory))
        xml_df = xml_to_csv(image_path)

xml_df.to_csv('data/{}_labels.csv'.format(directory),
index=None)
        print('Successfully converted xml to csv.')
main()
```

Lampiran 3 *Script Generate TFRecord*

```
"""
python generate_tfrecord.py --type=train --
csv_input=data/train_labels.csv  --
output_path=data/train.record --img_path=images/train
python generate_tfrecord.py --type=test --
csv_input=data/test_labels.csv  --
output_path=data/test.record --img_path=images/test
"""
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

import sys
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research\\obj
ect_detection')

from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict
flags = tf.app.flags
flags.DEFINE_string('type', '', 'Type of CSV input
(train/test)')
flags.DEFINE_string('csv_input', '', 'Path to the CSV
input')
flags.DEFINE_string('output_path', '', 'Path to output
TFRecord')
FLAGS = flags.FLAGS
def class_text_to_int(row_label):
    if row_label == 'honda':
        return 1
    elif row_label == 'yamaha':
        return 2
    else:
        return 0

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(os.getcwd(),
'images/{}'.format(FLAGS.type))
    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()
```

Lampiran 4 *Training*

```
model {
  ssd {
    num_classes: 2
    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
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    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
      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: 0.9997,
          epsilon: 0.001,
        }
      }
    }
}
feature_extractor {
  type: 'ssd_mobilenet_v1'
  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: 0.9997,
          epsilon: 0.001,
        }
      }
    }
    loss {
      classification_loss {
        weighted_sigmoid {
          anchorwise_output: true
        }
      }
      localization_loss {
        weighted_smooth_l1 {
          anchorwise_output: true
        }
      }
      hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: 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: {
  batch_size: 4
  optimizer {
    rms_prop_optimizer: {
      learning_rate: {
        exponential_decay_learning_rate {
          initial_learning_rate: 0.004
          decay_steps: 800720
          decay_factor: 0.95
        }
      }
```

```
      momentum_optimizer_value: 0.9
      decay: 0.9
      epsilon: 1.0
    }
  }
  fine_tune_checkpoint:
"ssd_mobilenet_v1_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  num_steps: 30000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
}
train_input_reader: {
  tf_record_input_reader {
    input_path: "C:/Users/Rakhil
Khaeriyah/Documents/Tensorflow/models/research/object_d
etection/train.record"
  }
  label_map_path: "C:/Users/Rakhil
Khaeriyah/Documents/Tensorflow/models/research/object_d
etection/datafix/sign_label_map.pbtxt"
}
eval_config: {
  num_examples: 200
  max_evals: 10
}
eval_input_reader: {
  tf_record_input_reader {
    input_path: "C:/Users/Rakhil
Khaeriyah/Documents/Tensorflow/models/research/object_d
etection/test.record"
  }
  label_map_path: "C:/Users/Rakhil
Khaeriyah/Documents/Tensorflow/models/research/object_d
etection/datafix/sign_label_map.pbtxt"
  shuffle: false
  num_readers: 1
  num_epochs: 1
}
```

Lampiran 5 *Script Training*

```
# Copyright 2017 The TensorFlow Authors. All Rights
Reserved.
#
# Licensed under the Apache License, Version 2.0 (the
"License");
# you may not use this file except in compliance with
the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in
writing, software
# distributed under the License is distributed on an
"AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied.
# See the License for the specific language governing
permissions and
# limitations under the License.
#
==========================================================
=======================

r"""Training executable for detection models.

This executable is used to train DetectionModels. There
are two ways of
configuring the training job:

1) A single pipeline_pb2.TrainEvalPipelineConfig
configuration file
can be specified by --pipeline_config_path.

Example usage:
    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --pipeline_config_path=pipeline_config.pbtxt

2) Three configuration files can be provided: a
model_pb2.DetectionModel
configuration file to define what type of
DetectionModel is being trained, an
input_reader_pb2.InputReader file to specify what
training data will be used and
```

```
a train_pb2.TrainConfig file to configure training
parameters.

Example usage:
    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --model_config_path=model_config.pbtxt \
        --train_config_path=train_config.pbtxt \
        --input_config_path=train_input_config.pbtxt
"""

import functools
import json
import os
import tensorflow as tf
import sys
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research\\obj
ect_detection')
from object_detection.builders import dataset_builder
from object_detection.builders import
graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util

tf.logging.set_verbosity(tf.logging.INFO)

flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the
TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones
to deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on
CPU.  Note that even if '
                     'set to False (allowing ops to run
on gpu), some ops may '
                     'still be run on the CPU if they
have no GPU kernel.')
```

```
flags.DEFINE_integer('worker_replicas', 1, 'Number of
worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks.
If None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints
and training summaries.')

flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a
pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs
are ignored')

flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig
config file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an
input_reader_pb2.InputReader config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel
config file.')

FLAGS = flags.FLAGS


@tf.contrib.framework.deprecated(None, 'Use
object_detection/model_main.py.')
def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0:
tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs =
config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir,
'pipeline.config'),
                    overwrite=True)
  else:
    configs =
config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
```

```
        train_config_path=FLAGS.train_config_path,

train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config',
FLAGS.model_config_path),
                           ('train.config',
FLAGS.train_config_path),
                           ('input.config',
FLAGS.input_config_path)]:
        tf.gfile.Copy(config,
os.path.join(FLAGS.train_dir, name),
                      overwrite=True)

  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']

  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
      is_training=True)

  def get_next(config):
    return dataset_builder.make_initializable_iterator(
        dataset_builder.build(config)).get_next()

  create_input_dict_fn = functools.partial(get_next,
input_config)

  env = json.loads(os.environ.get('TF_CONFIG', '{}'))
  cluster_data = env.get('cluster', None)
  cluster = tf.train.ClusterSpec(cluster_data) if
cluster_data else None
  task_data = env.get('task', None) or {'type':
'master', 'index': 0}
  task_info = type('TaskSpec', (object,), task_data)

  # Parameters for a single worker.
  ps_tasks = 0
  worker_replicas = 1
  worker_job_name = 'lonely_worker'
  task = 0
  is_chief = True
  master = ''

  if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s
and the "master".
```

```
    worker_replicas = len(cluster_data['worker']) + 1
  if cluster_data and 'ps' in cluster_data:
    ps_tasks = len(cluster_data['ps'])

  if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for
distributed training.')

  if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server =
tf.train.Server(tf.train.ClusterSpec(cluster),
protocol='grpc',
                                job_name=task_info.type,

task_index=task_info.index)
    if task_info.type == 'ps':
      server.join()
      return

    worker_job_name = '%s/task:%d' % (task_info.type,
task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target

  graph_rewriter_fn = None
  if 'graph_rewriter_config' in configs:
    graph_rewriter_fn = graph_rewriter_builder.build(
        configs['graph_rewriter_config'],
is_training=True)

  trainer.train(
      create_input_dict_fn,
      model_fn,
      train_config,
      master,
      task,
      FLAGS.num_clones,
      worker_replicas,
      FLAGS.clone_on_cpu,
      ps_tasks,
      worker_job_name,
      is_chief,
      FLAGS.train_dir,
      graph_hook_fn=graph_rewriter_fn)

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

Lampiran 6 *Script Export Inference Graph*

```
# Copyright 2017 The TensorFlow Authors. All Rights
Reserved.
#
# Licensed under the Apache License, Version 2.0 (the
"License");
# you may not use this file except in compliance with
the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in
writing, software
# distributed under the License is distributed on an
"AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied.
# See the License for the specific language governing
permissions and
# limitations under the License.
#
==========================================================
=======================

r"""Tool to export an object detection model for
inference.

Prepares an object detection tensorflow graph for
inference using model
configuration and a trained checkpoint. Outputs
inference
graph, associated checkpoint files, a frozen inference
graph and a
SavedModel
(https://tensorflow.github.io/serving/serving_basic.htm
l).

The inference graph contains one of three input nodes
depending on the user
specified option.
  * `image_tensor`: Accepts a uint8 4-D tensor of shape
[None, None, None, 3]
  * `encoded_image_string_tensor`: Accepts a 1-D string
tensor of shape [None]
    containing encoded PNG or JPEG images. Image
resolutions are expected to be
    the same if more than 1 image is provided.
```

```
  * `tf_example`: Accepts a 1-D string tensor of shape
[None] containing
    serialized TFExample protos. Image resolutions are
expected to be the same
    if more than 1 image is provided.

and the following output nodes returned by the
model.postprocess(..):
  * `num_detections`: Outputs float32 tensors of the
form [batch]
      that specifies the number of valid boxes per
image in the batch.
  * `detection_boxes`: Outputs float32 tensors of the
form
      [batch, num_boxes, 4] containing detected boxes.
  * `detection_scores`: Outputs float32 tensors of the
form
      [batch, num_boxes] containing class scores for
the detections.
  * `detection_classes`: Outputs float32 tensors of the
form
      [batch, num_boxes] containing classes for the
detections.
  * `raw_detection_boxes`: Outputs float32 tensors of
the form
      [batch, raw_num_boxes, 4] containing detection
boxes without
      post-processing.
  * `raw_detection_scores`: Outputs float32 tensors of
the form
      [batch, raw_num_boxes,
num_classes_with_background] containing class score
      logits for raw detection boxes.
  * `detection_masks`: Outputs float32 tensors of the
form
      [batch, num_boxes, mask_height, mask_width]
containing predicted instance
      masks for each box if its present in the
dictionary of postprocessed
      tensors returned by the model.

Notes:
 * This tool uses `use_moving_averages` from
eval_config to decide which
   weights to freeze.

Example Usage:
--------------
python export_inference_graph \
```

```
    --input_type image_tensor \
    --pipeline_config_path
path/to/ssd_inception_v2.config \
    --trained_checkpoint_prefix path/to/model.ckpt \
    --output_directory path/to/exported_model_directory

The expected output would be in the directory
path/to/exported_model_directory (which is created if
it does not exist)
with contents:
 - inference_graph.pbtxt
 - model.ckpt.data-00000-of-00001
 - model.ckpt.info
 - model.ckpt.meta
 - frozen_inference_graph.pb
 + saved_model (a directory)

Config overrides (see the `config_override` flag) are
text protobufs
(also of type pipeline_pb2.TrainEvalPipelineConfig)
which are used to override
certain fields in the provided pipeline_config_path.
These are useful for
making small changes to the inference graph that differ
from the training or
eval config.

Example Usage (in which we change the second stage
post-processing score
threshold to be 0.5):

python export_inference_graph \
    --input_type image_tensor \
    --pipeline_config_path
path/to/ssd_inception_v2.config \
    --trained_checkpoint_prefix path/to/model.ckpt \
    --output_directory path/to/exported_model_directory
\
    --config_override " \
            model{ \
              faster_rcnn { \
                second_stage_post_processing { \
                  batch_non_max_suppression { \
                    score_threshold: 0.5 \
                  } \
                } \
              } \
            }"
"""
```

```
import tensorflow as tf
import sys
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research')
sys.path.append('C:\\Users\\Rakhil
Khaeriyah\\Documents\\Tensorflow\\models\\research\\obj
ect_detection')
from google.protobuf import text_format
from object_detection import exporter
from object_detection.protos import pipeline_pb2

slim = tf.contrib.slim
flags = tf.app.flags

flags.DEFINE_string('input_type', 'image_tensor', 'Type
of input node. Can be '
                    'one of [`image_tensor`,
`encoded_image_string_tensor`, '
                    '`tf_example`]')
flags.DEFINE_string('input_shape', None,
                    'If input_type is `image_tensor`,
this can explicitly set '
                    'the shape of this input tensor to
a fixed size. The '
                    'dimensions are to be provided as a
comma-separated list '
                    'of integers. A value of -1 can be
used for unknown '
                    'dimensions. If not specified, for
an `image_tensor, the '
                    'default shape will be partially
specified as '
                    '`[None, None, None, 3]`.')
flags.DEFINE_string('pipeline_config_path', None,
                    'Path to a
pipeline_pb2.TrainEvalPipelineConfig config '
                    'file.')
flags.DEFINE_string('trained_checkpoint_prefix', None,
                    'Path to trained checkpoint,
typically of the form '
                    'path/to/model.ckpt')
flags.DEFINE_string('output_directory', None, 'Path to
write outputs.')
flags.DEFINE_string('config_override', '',
```

```
'pipeline_pb2.TrainEvalPipelineConfig '
                    'text proto to override
pipeline_config_path.')
flags.DEFINE_boolean('write_inference_graph', False,
                     'If true, writes inference graph
to disk.')
tf.app.flags.mark_flag_as_required('pipeline_config_pat
h')
tf.app.flags.mark_flag_as_required('trained_checkpoint_
prefix')
tf.app.flags.mark_flag_as_required('output_directory')
FLAGS = flags.FLAGS


def main(_):
  pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
  with tf.gfile.GFile(FLAGS.pipeline_config_path, 'r')
as f:
    text_format.Merge(f.read(), pipeline_config)
  text_format.Merge(FLAGS.config_override,
pipeline_config)
  if FLAGS.input_shape:
    input_shape = [
        int(dim) if dim != '-1' else None
        for dim in FLAGS.input_shape.split(',')
    ]
  else:
    input_shape = None
  exporter.export_inference_graph(
      FLAGS.input_type, pipeline_config,
FLAGS.trained_checkpoint_prefix,
      FLAGS.output_directory, input_shape=input_shape,

write_inference_graph=FLAGS.write_inference_graph)


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

Lampiran 7 *Script* Pengujian

```
# coding: utf-8
# In[1]:
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
# In[2]:
from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
# In[16]:
import cv2
cap = cv2.VideoCapture(0)
# In[5]:
# This is needed since the notebook is stored in the
object_detection folder.
sys.path.append("..")
# In[7]:
from utils import label_map_util
from utils import visualization_utils as vis_util
# In[8]:
# What model to download.
MODEL_NAME = 'motor_33002_2'
#MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE =
'http://download.tensorflow.org/models/object_detection
/'
# 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',
'sign_label_map.pbtxt')
NUM_CLASSES = 2
# In[7]:
# In[9]:
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[10]:
label_map =
label_map_util.load_labelmap(PATH_TO_LABELS)
categories =
label_map_util.convert_label_map_to_categories(label_ma
p, max_num_classes=NUM_CLASSES, use_display_name=True)
category_index =
label_map_util.create_category_index(categories)
# In[11]:
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)
# In[12]:
# If you want to test the code with your images, just
add path to the images to the TEST_IMAGE_PATHS.
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, 3) ]
# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
# In[ ]:
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, (1000,800)))
      if cv2.waitKey(25) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        break
```