APPENDIX

CONVERT XMT to CSV

```
1. import os
2. import glob
3. import pandas as pd
4. import xml.etree.ElementTree as ET
5.
6. import argparse
7. parser = argparse.ArgumentParser()
8. parser.add_argument('--type', help='test, val, or train',
9.                     required=True)
10. args = parser.parse_args()
11.
12. image_types = ["png", "PNG", "jpg", "jpeg", "JPG", "JPEG"]
13.
14. def xml_to_csv(img_files, xml_files):
15.
16.     xml_list = []
17.     for i, xml_file in enumerate(xml_files):
18.         tree = ET.parse(xml_file)
19.         root = tree.getroot()
20.
21.         for member in root.findall('object'):
22.             value = (root.find('filename').text,
23.                      int(root.find('size').find('width').text),
24.                      int(root.find('size').find('height').text),
25.                      member[0].text,
26.
   int(member.find("bndbox").find('xmin').text),
27.
   int(member.find("bndbox").find('ymin').text),
28.
   int(member.find("bndbox").find('xmax').text),
29.                      int(member.find("bndbox").find('ymax').text)
30.                      )
31.
32.             xml_list.append(value)
33.     column_name = ['filename', 'width', 'height', 'class', 'xmin',
   'ymin', 'xmax', 'ymax']
34.     xml_df = pd.DataFrame(xml_list, columns=column_name)
35.     return xml_df
36.
37.
38. def main():
39.     state = args.type
40.     image_path               =              os.path.join(os.getcwd(),
   ''+state).replace("\\","/")
41.     all_files_xml = []
42.     all_files_img = []
43.     all_path = []
44.     for root, subdirs, files in os.walk(image_path):
45.         for f in files:
46.             if len(files)>0:
47.                 if f.split(".")[-1] in image_types:
48.
   all_files_img.append(os.path.join(root,f).replace("\\","/"))
```

```
                if f.split(".")[-1] in ["xml"]:
                    all_files_xml.append(os.path.join(root,f).replace("\\","/"))
    all_files_img = sorted(all_files_img)
    all_files_xml = sorted(all_files_xml)
    xml_df = xml_to_csv(all_files_img, all_files_xml)
    xml_df.to_csv(state+'_labels.csv', index=None)
    print('Successfully converted xml to csv.')


main()
```

LABELING MAP

```
import pandas as pd

df = pd.read_csv("train_labels.csv") # asumsi label paling lengkap
idx = 1
dic = {}

with open("label_map.pbtxt", "w") as f:
    for idx, label in enumerate(df["class"].unique()):
        idx+=1
        f.write("item{\n")
        f.write("id: %d\n" % (idx))
        f.write("name: '" + label + "'\n")
        f.write("}\n\n")

print("DONE")
```

TFRECORD

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

import os
import io
import pandas as pd

from tensorflow.python.framework.versions import VERSION
if VERSION >= "2.0.0a0":
    import tensorflow.compat.v1 as tf
else:
    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('image_dir', '', 'Path to the image directory')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
FLAGS = flags.FLAGS


# TO-DO replace this with label map
```

```
27. def class_text_to_int(row_label):
28.     if row_label == 'Carrot':
29.         return 1
30.     elif row_label == 'Tomato':
31.         return 2
32.     elif row_label == 'Cabbage':
33.         return 3
34.     elif row_label == 'Pisang':
35.         return 4
36.     else:
37.         return 0
38.
39.
40. def split(df, group):
41.     data = namedtuple('data', ['filename', 'object'])
42.     gb = df.groupby(group)
43.     return [data(filename, gb.get_group(x)) for filename, x in
    zip(gb.groups.keys(), gb.groups)]
44.
45.
46. def create_tf_example(group, path):
47.     with tf.gfile.GFile(os.path.join(path,
    '{}'.format(group.filename)), 'rb') as fid:
48.         encoded_jpg = fid.read()
49.     encoded_jpg_io = io.BytesIO(encoded_jpg)
50.     image = Image.open(encoded_jpg_io)
51.     width, height = image.size
52.
53.     filename = group.filename.encode('utf8')
54.     image_format = b'jpg'
55.     xmins = []
56.     xmaxs = []
57.     ymins = []
58.     ymaxs = []
59.     classes_text = []
60.     classes = []
61.
62.     for index, row in group.object.iterrows():
63.         xmins.append(row['xmin'] / width)
64.         xmaxs.append(row['xmax'] / width)
65.         ymins.append(row['ymin'] / height)
66.         ymaxs.append(row['ymax'] / height)
67.         classes_text.append(row['class'].encode('utf8'))
68.         classes.append(class_text_to_int(row['class']))
69.
70.     tf_example =
    tf.train.Example(features=tf.train.Features(feature={
71.         'image/height': dataset_util.int64_feature(height),
72.         'image/width': dataset_util.int64_feature(width),
73.         'image/filename': dataset_util.bytes_feature(filename),
74.         'image/source_id': dataset_util.bytes_feature(filename),
75.         'image/encoded': dataset_util.bytes_feature(encoded_jpg),
76.         'image/format': dataset_util.bytes_feature(image_format),
77.         'image/object/bbox/xmin':
    dataset_util.float_list_feature(xmins),
78.         'image/object/bbox/xmax':
    dataset_util.float_list_feature(xmaxs),
79.         'image/object/bbox/ymin':
    dataset_util.float_list_feature(ymins),
80.         '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(), 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()
```

TRAINING MODEL

```
model {
  ssd {
    num_classes: 3
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.03
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.97
          center: true
          scale: true
          epsilon: 0.001
          train: true
```

```
      }
    }
    override_base_feature_extractor_hyperparams: true
  }
  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 {
    }
  }
  box_predictor {
    convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.97
          center: true
          scale: true
          epsilon: 0.001
          train: true
        }
      }
      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
    }
  }
  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
      }
    }
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-08
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: false
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
          delta: 1.0
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.75
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 16
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    ssd_random_crop {
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.008
          total_steps: 120000
          warmup_learning_rate: 0.0001
```

```
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint:        "ssd_mobilenet_v2_320x320_coco17_tpu-8/cp/ckpt-0"
  num_steps: 120000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/content/dataset/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/content/dataset/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "/content/dataset/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/content/dataset/test.record"
  }
}
```

FILE CONFIG

```
from absl import flags
import tensorflow.compat.v2 as tf
from object_detection import model_lib_v2

flags.DEFINE_string('pipeline_config_path', None, 'Path to pipeline config '
                    'file.')
flags.DEFINE_integer('num_train_steps', None, 'Number of train steps.')
flags.DEFINE_bool('eval_on_train_data', False, 'Enable evaluating on train '
                  'data (only supported in distributed training).')
flags.DEFINE_integer('sample_1_of_n_eval_examples', None, 'Will sample one of '
                     'every n eval input examples, where n is provided.')
flags.DEFINE_integer('sample_1_of_n_eval_on_train_examples', 5, 'Will sample '
                     'one of every n train input examples for evaluation, '
                     'where n is provided. This is only used if '
                     '`eval_training_data` is True.')
```

```
flags.DEFINE_string(
    'model_dir', None, 'Path to output model directory '
                       'where event and checkpoint files will be written.')
flags.DEFINE_string(
    'checkpoint_dir', None, 'Path to directory holding a checkpoint. If '
    '`checkpoint_dir` is provided, this binary operates in eval-only mode, '
    'writing resulting metrics to `model_dir`.')

flags.DEFINE_integer('eval_timeout', 3600, 'Number of seconds to wait for an'
                     'evaluation checkpoint before exiting.')

flags.DEFINE_bool('use_tpu', False, 'Whether the job is executing on a TPU.')
flags.DEFINE_string(
    'tpu_name',
    default=None,
    help='Name of the Cloud TPU for Cluster Resolvers.')
flags.DEFINE_integer(
    'num_workers', 1, 'When num_workers > 1, training uses '
    'MultiWorkerMirroredStrategy. When num_workers = 1 it uses '
    'MirroredStrategy.')
flags.DEFINE_integer(
    'checkpoint_every_n', 1000, 'Integer defining how often we checkpoint.')
flags.DEFINE_boolean('record_summaries', True,
                     ('Whether or not to record summaries defined by the model'
                      ' or the training pipeline. This does not impact the'
                      ' summaries of the loss values which are always'
                      ' recorded.'))

FLAGS = flags.FLAGS


def main(unused_argv):
  flags.mark_flag_as_required('model_dir')
  flags.mark_flag_as_required('pipeline_config_path')
  tf.config.set_soft_device_placement(True)

  if FLAGS.checkpoint_dir:
    model_lib_v2.eval_continuously(
        pipeline_config_path=FLAGS.pipeline_config_path,
        model_dir=FLAGS.model_dir,
        train_steps=FLAGS.num_train_steps,
        sample_1_of_n_eval_examples=FLAGS.sample_1_of_n_eval_examples,
        sample_1_of_n_eval_on_train_examples=(
            FLAGS.sample_1_of_n_eval_on_train_examples),
        checkpoint_dir=FLAGS.checkpoint_dir,
        wait_interval=300, timeout=FLAGS.eval_timeout)
  else:
    if FLAGS.use_tpu:
      # TPU is automatically inferred if tpu_name is None and
      # we are running under cloud ai-platform.
      resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
          FLAGS.tpu_name)
```

```
      tf.config.experimental_connect_to_cluster(resolver)
      tf.tpu.experimental.initialize_tpu_system(resolver)
      strategy = tf.distribute.experimental.TPUStrategy(resolver)
    elif FLAGS.num_workers > 1:
      strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy()
    else:
      strategy = tf.compat.v2.distribute.MirroredStrategy()

    with strategy.scope():
      model_lib_v2.train_loop(
          pipeline_config_path=FLAGS.pipeline_config_path,
          model_dir=FLAGS.model_dir,
          train_steps=FLAGS.num_train_steps,
          use_tpu=FLAGS.use_tpu,
          checkpoint_every_n=FLAGS.checkpoint_every_n,
          record_summaries=FLAGS.record_summaries)

if __name__ == '__main__':
  tf.compat.v1.app.run()
```

IMPLEMENTATION

```
import numpy as np
import tensorflow as tf
import cv2
import os
os.chdir('models/research')
import datetime
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
from pygame import mixer
from threading import Thread
# about image
import base64
import requests
import json
utils_ops.tf = tf.compat.v1
tf.gfile = tf.io.gfile
os.chdir('../..')
PATH_TO_LABELS = 'label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
detection_model = tf.saved_model.load("inference_graph/saved_model")
# https://link.springer.com/content/pdf/10.1007/s41870-021-00658-2.pdf
# [ymin, xmin, ymax, xmax]
def deteksi_center(ymin, xmin, ymax, xmax, w, h):
    cx = ((xmin+xmax)/2)*w
    cy = ((ymin+ymax)/2)*h
    return int(cx),int(cy)
def run_inference_for_single_image(model, image):
    image = np.asarray(image)
    input_tensor = tf.convert_to_tensor(image)
    input_tensor = input_tensor[tf.newaxis,...]
    model_fn = model.signatures['serving_default']
    output_dict = model_fn(input_tensor)

```

```
    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

    output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

    if 'detection_masks' in output_dict:
        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


def show_inference(model, image_np):
    totalMelanggarJarak = 0
    totalMelanggarMasker = 0

    output_dict = run_inference_for_single_image(model, image_np)

    hasil = 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)

    for x in output_dict['detection_classes'][output_dict['detection_scores']>=0.5]:
        if x==1 or x==3:
            totalMelanggarMasker+=1

    box_05 = output_dict['detection_boxes'][output_dict['detection_scores']>=0.5]
    score_05 = output_dict['detection_scores'][output_dict['detection_scores']>=0.5]

    h, w, c = hasil.shape
    rekapKoordinat = []

    if totalMelanggarMasker!=0:
        mixer.init()
        mixer.music.load('alert.ogg')
        mixer.music.play()

    return(hasil, totalMelanggarMasker)


```

```
86.
87. cap = cv2.VideoCapture(2)
88.
89. terakhir = datetime.datetime.now()
90.
91. while 1:
92.     _, img = cap.read()
93.     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
94.
95.     inferencehasil = show_inference(detection_model, img)
96.
97.     final_img = inferencehasil[0]
98.     final_img = cv2.cvtColor(final_img, cv2.COLOR_RGB2BGR)
99.
100.    # Masker
101.    if inferencehasil[2] != 0:
102.        final_img = cv2.putText(final_img, "Total Pelanggar Masker =
   " + str(inferencehasil[2]), org=(20,415), fontFace=
   cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(0,0,255),
103.            thickness=3, lineType=cv2.LINE_AA)
104.    else:
105.        final_img = cv2.putText(final_img, "Total Pelanggar Masker =
   " + str(inferencehasil[2]), org=(20,415), fontFace=
   cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(0,255,0),
106.            thickness=3, lineType=cv2.LINE_AA)
107.
108.    if datetime.datetime.now() >=
   terakhir+datetime.timedelta(seconds=5):
109.        terakhir = datetime.datetime.now()
110.
111.        if inferencehasil[1]!=0 or inferencehasil[2]!=0:
112.            _, buffer = cv2.imencode(".jpg", final_img)
113.
114.            url = "https://api.imgbb.com/1/upload"
115.            payload = {
116.                "key":"e23ce021eeb07060c36d1cf80c0d5c67",
117.                "image":base64.b64encode(buffer)
118.            }
119.            res = requests.post(url, payload)
120.            respon = json.loads(res.text)
121.
122.            print("Terdeteksi pelanggaran. Link : " +
   respon["data"]["display_url"])
123.
124.            valueku = (datetime.datetime.now(), inferencehasil[2],
   inferencehasil[1], respon["data"]["display_url"])
125.            eksekusiDb.execute(sql, valueku)
126.            db.commit()
127.
128.        else:
129.            valueku = (datetime.datetime.now(), inferencehasil[2],
   inferencehasil[1], "-")
130.            eksekusiDb.execute(sql, valueku)
131.            db.commit()
132.            print("Kondisi aman")
133.
134.    cv2.imshow('img', final_img)
135.
136.    if cv2.waitKey(1) == ord('q'):
137.        break
138.
```

```
139.cap.release()
140.cv2.destroyAllWindows()
```

Similarity Report

PAPER NAME

TA-19.K1.0059.docx

WORD COUNT	CHARACTER COUNT
5460 Words	**29296 Characters**
PAGE COUNT	FILE SIZE
27 Pages	**1.3MB**
SUBMISSION DATE	REPORT DATE
Jan 5, 2023 12:49 PM GMT+7	**Jan 5, 2023 12:50 PM GMT+7**

17% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 17% Internet database
- 9% Publications database
- Crossref database
- Crossref Posted Content database
- 12% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)
- Manually excluded text blocks

Summary