APPENDIX

CONVERT XML TO CSV

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--type', help='test, val, or train',
                    required=True)
args = parser.parse_args()
image_types = ["png", "PNG", "jpg", "jpeg", "JPG", "JPEG"]
def xml_to_csv(img_files, xml_files):
    xml_list = []
    for i, xml_file in enumerate(xml_files):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size').find('width').text),
                     int(root.find('size').find('height').text),
                     member[0].text,
int(member.find("bndbox").find('xmin').text),
int(member.find("bndbox").find('ymin').text),
int(member.find("bndbox").find('xmax').text),
                     int(member.find("bndbox").find('ymax').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():
    state = args.type
    image_path              =              os.path.join(os.getcwd(),
''+state).replace("\\","/")
    all_files_xml = []
    all_files_img = []
    all_path = []
    for root, subdirs, files in os.walk(image_path):
        for f in files:
            if len(files)>0:
                if f.split(".")[-1] in image_types:
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()
```

GENERATE TF RECORD

```
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
def class_text_to_int(row_label):
    if row_label == 'without_mask':
        return 1
    elif row_label == 'with_mask':
        return 2
    elif row_label == 'mask_weared_incorrect':
        return 3
    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(), 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()
```

PIPELINE CONFIG

```
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"
  }
}
```

TRAINING MODEL (MODEL_MAIN_TF2.PY)

```
1. from absl import flags
2. import tensorflow.compat.v2 as tf
3. from object_detection import model_lib_v2
4. flags.DEFINE_string('pipeline_config_path', None, 'Path to pipeline
   config ''file.')
5. flags.DEFINE_integer('num_train_steps',  None,  'Number  of  train
   steps.')
6. flags.DEFINE_bool('eval_on_train_data', False, 'Enable evaluating on
   train ''data (only supported in distributed training).')
7. flags.DEFINE_integer('sample_1_of_n_eval_examples',    None,    'Will
   sample one of ''every n eval input examples, where n is provided.')
```

```
8. 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.')
9. flags.DEFINE_string('model_dir', None, 'Path to output model directory
   ''where event and checkpoint files will be written.')
10. 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`.')
11. flags.DEFINE_integer('eval_timeout', 3600, 'Number of seconds to wait
   for an''evaluation checkpoint before exiting.')
12. flags.DEFINE_bool('use_tpu', False, 'Whether the job is executing on
   a TPU.')
13. flags.DEFINE_string('tpu_name', default=None, help='Name of the Cloud
   TPU for Cluster Resolvers.')
14. flags.DEFINE_integer('num_workers', 1, 'When num_workers > 1,
   training uses ''MultiWorkerMirroredStrategy. When num_workers = 1 it
   uses ''MirroredStrategy.')
15. flags.DEFINE_integer('checkpoint_every_n', 1000, 'Integer defining
   how often we checkpoint.')
16. 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.'))
17. FLAGS = flags.FLAGS
18. def main(unused_argv):
19.   flags.mark_flag_as_required('model_dir')
20.   flags.mark_flag_as_required('pipeline_config_path')
21.   tf.config.set_soft_device_placement(True)
22.   if FLAGS.checkpoint_dir:

   model_lib_v2.eval_continuously(pipeline_config_path=FLAGS.pipeline_co
   nfig_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_tra
   in_examples), checkpoint_dir=FLAGS.checkpoint_dir, wait_interval=300,
   timeout=FLAGS.eval_timeout)
23.   else:
24.     if FLAGS.use_tpu:
25.       resolver = tf.distribute.cluster_resolver.TPUClusterResolver(
26.           FLAGS.tpu_name)
27.       tf.config.experimental_connect_to_cluster(resolver)
28.       tf.tpu.experimental.initialize_tpu_system(resolver)
29.       strategy = tf.distribute.experimental.TPUStrategy(resolver)
30.     elif FLAGS.num_workers > 1:
31.       strategy                                                    =
   tf.distribute.experimental.MultiWorkerMirroredStrategy()
32.     else:
33.       strategy = tf.compat.v2.distribute.MirroredStrategy()
34.     with strategy.scope():
35.
   model_lib_v2.train_loop(pipeline_config_path=FLAGS.pipeline_config_pa
   th,   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)
36. if __name__ == '__main__':
37.   tf.compat.v1.app.run()
```

SYSTEM CODE

```
1. import numpy as np
2. import tensorflow as tf
3. import cv2
4. import os
5. os.chdir('models/research')
6. import datetime
7. from object_detection.utils import ops as utils_ops
8. from object_detection.utils import label_map_util
9. from object_detection.utils import visualization_utils as vis_util
10. from pygame import mixer
11. from threading import Thread
12. import mysql.connector
13. db = mysql.connector.connect(
14.   host="localhost",
15.   user="root",
16.   password="",
17.   database="db_pkm2"
18. )
19. eksekusiDb = db.cursor()
20. sql = "INSERT INTO tblDataPelanggar (time, pelanggar_masker,
    pelanggar_jarak, bukti_ss) VALUES (%s, %s, %s, %s)"
21. import base64
22. import requests
23. import json
24. utils_ops.tf = tf.compat.v1
25. tf.gfile = tf.io.gfile
26. os.chdir('../..')
27. PATH_TO_LABELS = 'label_map.pbtxt'
28. category_index =
    label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
    use_display_name=True)
29. detection_model = tf.saved_model.load("inference_graph/saved_model")
30. def deteksi_center(ymin, xmin, ymax, xmax, w, h):
31.     cx = ((xmin+xmax)/2)*w
32.     cy = ((ymin+ymax)/2)*h
33.     return int(cx),int(cy)
34. def run_inference_for_single_image(model, image):
35.     image = np.asarray(image)
36.     input_tensor = tf.convert_to_tensor(image)
37.     input_tensor = input_tensor[tf.newaxis,...]
38.     model_fn = model.signatures['serving_default']
39.     output_dict = model_fn(input_tensor)
40.     num_detections = int(output_dict.pop('num_detections'))
41.     output_dict = {key:value[0, :num_detections].numpy()
42.                    for key,value in output_dict.items()}
43.     output_dict['num_detections'] = num_detections
44.     output_dict['detection_classes'] =
    output_dict['detection_classes'].astype(np.int64)
45.     if 'detection_masks' in output_dict:
46.         detection_masks_reframed =
    utils_ops.reframe_box_masks_to_image_masks(
47.             output_dict['detection_masks'],
    output_dict['detection_boxes'],
48.             image.shape[0], image.shape[1])
49.         detection_masks_reframed = tf.cast(detection_masks_reframed
    >= 0.5, tf.uint8)
50.         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 = []
    for index in range(len(box_05)):
        ymin, xmin, ymax, xmax = box_05[index]
        rekapKoordinat.append(deteksi_center(ymin, xmin, ymax, xmax, w, h))
    for i in range(len(rekapKoordinat)):
        hasil = cv2.circle(hasil, rekapKoordinat[i], 5, (255, 0, 0), -5)
    rekapJarak = {}
    if len(rekapKoordinat)>1:
        for i in range(len(rekapKoordinat)-1):
            rekapJarak[i] = {}
            for j in range(i+1, len(rekapKoordinat)):
                rekapJarak[i][j] = (np.linalg.norm(np.array(rekapKoordinat[i]) - np.array(rekapKoordinat[j])))
                if rekapJarak[i][j] < 300:
                    hasil = cv2.line(hasil, rekapKoordinat[i], rekapKoordinat[j], (255, 0, 0), 2)
                    totalMelanggarJarak+=1
    if totalMelanggarJarak!=0 or totalMelanggarMasker!=0:
        mixer.init()
        mixer.music.load('alert.ogg')
        mixer.music.play()

    return(hasil, totalMelanggarJarak, totalMelanggarMasker)
cap = cv2.VideoCapture(2)
terakhir = datetime.datetime.now()
while 1:
    _, img = cap.read()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    inferencehasil = show_inference(detection_model, img)
    final_img = inferencehasil[0]
    final_img = cv2.cvtColor(final_img, cv2.COLOR_RGB2BGR)
    if inferencehasil[1] != 0:
```

```
        final_img = cv2.putText(final_img, "Total Kasus Jarak = " +
    str(inferencehasil[1]),            org=(20,460),            fontFace=
    cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(0,0,255),
            thickness=3, lineType=cv2.LINE_AA)
    else:
        final_img = cv2.putText(final_img, "Total Kasus Jarak = " +
    str(inferencehasil[1]),            org=(20,460),            fontFace=
    cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(0,255,0),
            thickness=3, lineType=cv2.LINE_AA)
    if inferencehasil[2] != 0:
        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),
            thickness=3, lineType=cv2.LINE_AA)
    else:
        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),
            thickness=3, lineType=cv2.LINE_AA)
    if                  datetime.datetime.now()                  >=
    terakhir+datetime.timedelta(seconds=5):
        terakhir = datetime.datetime.now()

        if inferencehasil[1]!=0 or inferencehasil[2]!=0:
            _, buffer = cv2.imencode(".jpg", final_img)
            url = "https://api.imgbb.com/1/upload"
            payload = {
                "key":"e23ce021eeb07060c36d1cf80c0d5c67",
                "image":base64.b64encode(buffer)
            }
            res = requests.post(url, payload)
            respon = json.loads(res.text)
           print("Terdeteksi    pelanggaran.    Link    :    "    +
    respon["data"]["display_url"])
            valueku  =  (datetime.datetime.now(),  inferencehasil[2],
    inferencehasil[1], respon["data"]["display_url"])
            eksekusiDb.execute(sql, valueku)
            db.commit()
        else:
            valueku  =  (datetime.datetime.now(),  inferencehasil[2],
    inferencehasil[1], "-")
            eksekusiDb.execute(sql, valueku)
            db.commit()
            print("Kondisi aman")
    cv2.imshow('img', final_img)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Similarity Report

PAPER NAME

TA-19.K1.0011.docx

WORD COUNT	CHARACTER COUNT
7988 Words	**41672 Characters**
PAGE COUNT	FILE SIZE
30 Pages	**1.7MB**
SUBMISSION DATE	REPORT DATE
Jan 4, 2023 2:33 PM GMT+7	**Jan 4, 2023 2:34 PM GMT+7**

17% Overall Similarity

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

- 12% Internet database
- Crossref database
- 12% Submitted Works database
- 5% Publications database
- Crossref Posted Content database

Summary