CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

5.1.1. Data Preparation and Processing

First of all, we need to review the sample of the dataset that will be used for the further processes. The total of the dataset is 853 images, that at least have 1 class between wearing a mask, not wearing a mask, or wearing a mask but wrong. To review the dataset, the researcher uses the LabelImg application for this step. LabelImg is a graphical image annotation tool that is written in Python and uses Qt for its graphical interface. An example of the existing dataset is as follow:

Figure 5.1: Example of the data

After the data is ready, the researcher will convert the RAW data to the CSV. For this step, the researcher use code bellow, save as xml_to_csv.py, and call “python xml_to_csv.py --type train” :

```
1. import os
2. import glob
3. import pandas as pd
4. import xml.etree.ElementTree as ET
5. import argparse
6. parser = argparse.ArgumentParser()
7. parser.add_argument('--type', help='test, val, or train',
   required=True)
8. args = parser.parse_args()
9. 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('ym
in').text),int(member.find("bndbox").find('xmax').text),int(member.fi
nd("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.')
```

The output of the previous step is train_labels.csv file. The samples of the results of the CSV will show below :

train_labels.csv

1 to 10 of 4072 entries Filter

filename	width	height	class	xmin	ymin	xmax	ymax
maksssksksss0.png	512	366	without_mask	79	105	109	142
maksssksksss0.png	512	366	with_mask	185	100	226	144
maksssksksss0.png	512	366	without_mask	325	90	360	141
maksssksksss1.png	400	156	with_mask	321	34	354	69
maksssksksss1.png	400	156	with_mask	224	38	261	73
maksssksksss1.png	400	156	with_mask	299	58	315	81
maksssksksss1.png	400	156	with_mask	143	74	174	115
maksssksksss1.png	400	156	with_mask	74	69	95	99
maksssksksss1.png	400	156	with_mask	191	67	221	93
maksssksksss1.png	400	156	with_mask	21	73	44	93

Show 10 per page

1 2 10 100 300 400 408

Figure 5.2: Example of train_label.csv content

For further processes, we also need the label map file to recap the classes. For that, the researcher use code below, save as generate_labelmap.py, and call “python generate_labelmap.py” :

```
1. import pandas as pd
2. df = pd.read_csv("train_labels.csv") # asumsi label paling lengkap
3. idx = 1
4. dic = {}
5. with open("label_map.pbtxt", "w") as f:
6.     for idx, label in enumerate(df["class"].unique()):
7.         idx+=1
8.         f.write("item{\n")
9.         f.write("id: %d\n" % (idx))
10.        f.write("name: '" + label + "'\n")
11.        f.write("}\n\n")
12.print("DONE")
```

The output of the previous step is label_map.pbtxt file. The results of the previous step will show below:

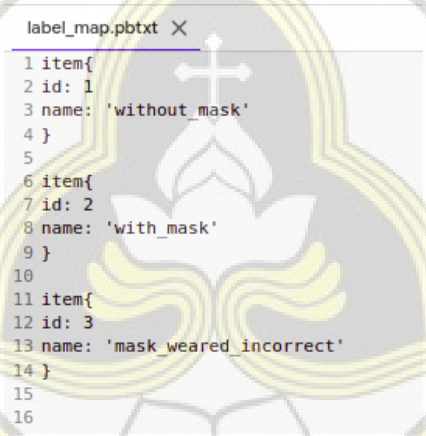

Figure 5.3: Example of label_map.pbtxt content

After that, the train_labels.csv, the images, the label_map.pbtxt will be processed again to the TF Record file. For that, the researcher use code below, save it as generate_tfrecord.py, and call “python generate_tfrecord.py --csv_input train_labels.csv --image_dir train --labelmap_dir label_map.pbtxt --output_path train.tfrecord” :

```
1. from __future__ import division
2. from __future__ import print_function
3. from __future__ import absolute_import
4. import os
5. import io
6. import pandas as pd
7. from tensorflow.python.framework.versions import VERSION
8. if VERSION >= "2.0.0a0":
9.     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:
        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),
        }))
55.     return tf_example
56.def main(_):
57.     writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
58.     path = os.path.join(os.getcwd(), FLAGS.image_dir)
59.     examples = pd.read_csv(FLAGS.csv_input)
60.     grouped = split(examples, 'filename')
61.     for group in grouped:
62.         tf_example = create_tf_example(group, path)
63.         writer.write(tf_example.SerializeToString())
64.     writer.close()
65.     output_path = os.path.join(os.getcwd(), FLAGS.output_path)
66.     print('Successfully created the TFRecords:
    {}'.format(output_path))
67.if __name__ == '__main__':
68.     tf.app.run()
```

After the train.tfrecord is ready, the researcher will validate the total of the dataset and split the dataset to 3 types (20%, 50%, and 80%).

```
1. # 20
2. train28 = dataset.skip(int(80/100*totalData))
3. train28Writer =
   tf.data.experimental.TFRecordWriter('train28.tfrecord')
4. train28Writer.write(train28)
5. #  50
6. train55 = dataset.skip(int(50/100*totalData))
7. train55Writer =
   tf.data.experimental.TFRecordWriter('train55.tfrecord')
8. train55Writer.write(train55)
9. # 80
10.train82 = dataset.skip(int(20/100*totalData))
11.val2 = dataset.take(int(20/100*totalData))
12.train82Writer =
   tf.data.experimental.TFRecordWriter('train82.tfrecord')
13.val2Writer = tf.data.experimental.TFRecordWriter('val2.tfrecord')
14.train82Writer.write(train82)
15.val2Writer.write(val2)
```

5.1.2. Model Preparation

After the dataset is ready, the researcher will prepare the model that will be trained in the next step. The existing config is adjusted to the needs of the researcher (num_classes = 3, the dataset will be adjusted depending on the target, etc) and saves the file in ".config" format. For the "faster R-CNN ResNet50 V1 640x640" model, config is written below :

```
model {
  faster_rcnn {
    num_classes: 3
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 640
        max_dimension: 640
        pad_to_max_dimension: true
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet50_keras'
      batch_norm_trainable: true
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    initial_crop_size: 14
    maxpool_kernel_size: 2
    maxpool_stride: 2
    second_stage_box_predictor {
```

```
45.      mask_rcnn_box_predictor {
46.        use_dropout: false
47.        dropout_keep_probability: 1.0
48.        fc_hyperparams {
49.          op: FC
50.          regularizer {
51.            l2_regularizer {
52.              weight: 0.0
53.            }
54.          }
55.          initializer {
56.            variance_scaling_initializer {
57.              factor: 1.0
58.              uniform: true
59.              mode: FAN_AVG
60.            }
61.          }
62.        }
63.        share_box_across_classes: true
64.      }
65.    }
66.    second_stage_post_processing {
67.      batch_non_max_suppression {
68.        score_threshold: 0.0
69.        iou_threshold: 0.6
70.        max_detections_per_class: 100
71.        max_total_detections: 300
72.      }
73.      score_converter: SOFTMAX
74.    }
75.    second_stage_localization_loss_weight: 2.0
76.    second_stage_classification_loss_weight: 1.0
77.    use_static_shapes: true
78.    use_matmul_crop_and_resize: true
79.    clip_anchors_to_image: true
80.    use_static_balanced_label_sampler: true
81.    use_matmul_gather_in_matcher: true
82.  }
83.}
84.train_config: {
85.  batch_size: 8
86.  sync_replicas: true
87.  startup_delay_steps: 0
88.  replicas_to_aggregate: 8
89.  num_steps: 5000
90.  optimizer {
91.    momentum_optimizer: {
92.      learning_rate: {
93.        cosine_decay_learning_rate {
94.          learning_rate_base: 8e-3
95.          total_steps: 4000
96.          warmup_learning_rate: 0.0001
97.          warmup_steps: 1000
98.        }
99.      }
100.       momentum_optimizer_value: 0.9
```

```
      }
      use_moving_average: false
    }
    fine_tune_checkpoint_version: V2
    fine_tune_checkpoint: ""
    fine_tune_checkpoint_type: "detection"
    data_augmentation_options {
      random_horizontal_flip {
      }
    }
    max_number_of_boxes: 100
    unpad_groundtruth_tensors: false
  }
  train_input_reader: {
    label_map_path: "label_map.pbtxt"
    tf_record_input_reader {
      input_path: "train.tfrecord"
    }
  }
  eval_config: {
    metrics_set: "coco_detection_metrics"
    use_moving_averages: false
    batch_size: 1;
  }
  eval_input_reader: {
    label_map_path: "label_map.pbtxt"
    shuffle: false
    num_epochs: 1
    tf_record_input_reader {
      input_path: "val.tfrecord"
    }
  }
```

For the “SSD ResNet50 V1 FPN 640x640 (RetinaNet50)” model, config written below :

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 3
    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
```

```
21.         use_matmul_gather: true
22.       }
23.     }
24.     similarity_calculator {
25.       iou_similarity {
26.       }
27.     }
28.     encode_background_as_zeros: true
29.     anchor_generator {
30.       multiscale_anchor_generator {
31.         min_level: 3
32.         max_level: 7
33.         anchor_scale: 4.0
34.         aspect_ratios: [1.0, 2.0, 0.5]
35.         scales_per_octave: 2
36.       }
37.     }
38.     image_resizer {
39.       fixed_shape_resizer {
40.         height: 640
41.         width: 640
42.       }
43.     }
44.     box_predictor {
45.       weight_shared_convolutional_box_predictor {
46.         depth: 256
47.         class_prediction_bias_init: -4.6
48.         conv_hyperparams {
49.           activation: RELU_6,
50.           regularizer {
51.             l2_regularizer {
52.               weight: 0.0004
53.             }
54.           }
55.           initializer {
56.             random_normal_initializer {
57.               stddev: 0.01
58.               mean: 0.0
59.             }
60.           }
61.           batch_norm {
62.             scale: true,
63.             decay: 0.997,
64.             epsilon: 0.001,
65.           }
66.         }
67.         num_layers_before_predictor: 4
68.         kernel_size: 3
69.       }
70.     }
71.     feature_extractor {
72.       type: 'ssd_resnet50_v1_fpn_keras'
73.       fpn {
74.         min_level: 3
75.         max_level: 7
76.       }
```

```
      min_depth: 16
      depth_multiplier: 1.0
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.0004
          }
        }
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          scale: true,
          decay: 0.997,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
      loss {
        classification_loss {
          weighted_sigmoid_focal {
            alpha: 0.25
            gamma: 2.0
          }
        }
        localization_loss {
          weighted_smooth_l1 {
          }
        }
        classification_weight: 1.0
        localization_weight: 1.0
      }
      normalize_loss_by_num_matches: true
      normalize_loc_loss_by_codesize: 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: {
    fine_tune_checkpoint_version: V2
    fine_tune_checkpoint: ""
    fine_tune_checkpoint_type: "detection"
    batch_size: 8
    sync_replicas: true
```

```
    startup_delay_steps: 0
    replicas_to_aggregate: 8
    use_bfloat16: true
    num_steps: 5000
    data_augmentation_options {
      random_horizontal_flip {
      }
    }
    data_augmentation_options {
      random_crop_image {
        min_object_covered: 0.0
        min_aspect_ratio: 0.75
        max_aspect_ratio: 3.0
        min_area: 0.75
        max_area: 1.0
        overlap_thresh: 0.0
      }
    }
    optimizer {
      momentum_optimizer: {
        learning_rate: {
          cosine_decay_learning_rate {
            learning_rate_base: 8e-3
            total_steps: 4000
            warmup_learning_rate: 0.0001
            warmup_steps: 1000
          }
        }
        momentum_optimizer_value: 0.9
      }
      use_moving_average: false
    }
    max_number_of_boxes: 100
    unpad_groundtruth_tensors: false
  }

  train_input_reader: {
    label_map_path: "label_map.pbtxt"
    tf_record_input_reader {
      input_path: "train.tfrecord"
    }
  }
  eval_config: {
    metrics_set: "coco_detection_metrics"
    use_moving_averages: false
    batch_size: 1;
  }
  eval_input_reader: {
    label_map_path: "label_map.pbtxt"
    shuffle: false
    num_epochs: 1
    tf_record_input_reader {
      input_path: "val.tfrecord"
    }
  }
```

For the “SSD MobileNet V2 320x320” model, config is written below :

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 3
    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 {
      }
    }
    encode_background_as_zeros: true
    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
        class_prediction_bias_init: -4.6
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            random_normal_initializer {
              stddev: 0.01
              mean: 0.0
            }
          }
          batch_norm {
            train: true,
            scale: true,
            center: true,
            decay: 0.97,
            epsilon: 0.001,
          }
        }
      }
    }
    feature_extractor {
      type: 'ssd_mobilenet_v2_keras'
      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: 0.97,
             epsilon: 0.001,
           }
         }
        override_base_feature_extractor_hyperparams: true
      }
      loss {
        classification_loss {
```

```
          weighted_sigmoid_focal {
            alpha: 0.75,
            gamma: 2.0
          }
        }
        localization_loss {
          weighted_smooth_l1 {
            delta: 1.0
          }
        }
        classification_weight: 1.0
        localization_weight: 1.0
      }
      normalize_loss_by_num_matches: true
      normalize_loc_loss_by_codesize: 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: {
    fine_tune_checkpoint_version: V2
    fine_tune_checkpoint: ""
    fine_tune_checkpoint_type: "detection"
    batch_size: 8
    sync_replicas: true
    startup_delay_steps: 0
    replicas_to_aggregate: 8
    num_steps: 5000
    data_augmentation_options {
      random_horizontal_flip {
      }
    }
    data_augmentation_options {
      ssd_random_crop {
      }
    }
    optimizer {
      momentum_optimizer: {
        learning_rate: {
          cosine_decay_learning_rate {
            learning_rate_base: 8e-3
            total_steps: 4000
            warmup_learning_rate: 0.0001
            warmup_steps: 1000
          }
        }
        momentum_optimizer_value: 0.9
      }
      use_moving_average: false
```

```
    }
    max_number_of_boxes: 1
    unpad_groundtruth_tensors: false
  }
  train_input_reader: {
    label_map_path: "/content/dataset/label_map.pbtxt"
    tf_record_input_reader {
      input_path: "/content/datasetPecah/train28.tfrecord"
    }
  }
  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/datasetPecah/val28.tfrecord"
    }
  }
```

5.1.3. Model Training

After all the models are already ready, the researcher will train the model. For that, the researcher use code below, save it as “model_main_tf2.py” and execute “python model_main_tf2.py --alsologtostderr --model_dir=dirOfModel --pipeline_config_path=file.config” and start training. Code of the “model_main_tf2.py” written below :

```
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:
      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()
```

The output of this training step is log about train performance like the image below :

```
INFO:tensorflow:Step 200 per-step time 0.666s
I1019 06:16:41.754344 140033910335360 model_lib_v2.py:700] Step 200 per-step time 0.666s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.10587779,
 'Loss/BoxClassifierLoss/localization_loss': 0.10602659,
 'Loss/RPNLoss/localization_loss': 0.098664895,
 'Loss/RPNLoss/objectness_loss': 0.0589003,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.36946958,
 'learning_rate': 0.00168}
I1019 06:16:41.754679 140033910335360 model_lib_v2.py:701] {'Loss/BoxClassifierLoss/classification_loss': 0.10587779,
 'Loss/BoxClassifierLoss/localization_loss': 0.10602659,
 'Loss/RPNLoss/localization_loss': 0.098664895,
 'Loss/RPNLoss/objectness_loss': 0.0589003,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.36946958,
 'learning_rate': 0.00168}
INFO:tensorflow:Step 300 per-step time 0.665s
I1019 06:17:48.223021 140033910335360 model_lib_v2.py:700] Step 300 per-step time 0.665s
INFO:tensorflow:{'Loss/BoxClassifierLoss/classification_loss': 0.1886513,
 'Loss/BoxClassifierLoss/localization_loss': 0.16511232,
 'Loss/RPNLoss/localization_loss': 0.06433578,
 'Loss/RPNLoss/objectness_loss': 0.059956163,
 'Loss/regularization_loss': 0.0,
 'Loss/total_loss': 0.47805557,
 'learning_rate': 0.0024700002}
```

Figure 5.4: Output of training steps

5.1.4. Model Evaluation & Analysis

After the training step is done, the researcher will evaluate the model performance using code that is the same as the training section. But in this step, the researcher needs to set a checkpoint directory. The researcher execute “python model_main_tf2.py --alsologtostderr --model_dir=dirOfModel --pipeline_config_path=file.config --checkpoint_dir=dirOfCP” and evaluation will start. The output of this step is the log of some evaluation metrics. The example is shown below :

```
I1020 04:09:14.638730 139772195035008 model_lib_v2.py:1007] Eval metrics at step 5000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.263549
I1020 04:09:14.649824 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP: 0.263549
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 0.486900
I1020 04:09:14.651375 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP@.50IOU: 0.486900
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 0.240624
I1020 04:09:14.652833 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP@.75IOU: 0.240624
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): 0.185986
I1020 04:09:14.654451 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP (small): 0.185986
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): 0.360528
I1020 04:09:14.655997 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP (medium): 0.360528
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.780212
I1020 04:09:14.657443 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Precision/mAP (large): 0.780212
INFO:tensorflow:        + DetectionBoxes_Recall/AR@1: 0.166491
I1020 04:09:14.658984 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@1: 0.166491
INFO:tensorflow:        + DetectionBoxes_Recall/AR@10: 0.380103
I1020 04:09:14.660473 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@10: 0.380103
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100: 0.440287
I1020 04:09:14.661875 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@100: 0.440287
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (small): 0.362511
I1020 04:09:14.663313 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@100 (small): 0.362511
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (medium): 0.557883
I1020 04:09:14.664731 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@100 (medium): 0.557883
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (large): 0.799815
I1020 04:09:14.666323 139772195035008 model_lib_v2.py:1010]     + DetectionBoxes_Recall/AR@100 (large): 0.799815
INFO:tensorflow:        + Loss/RPNLoss/localization_loss: 0.049334
I1020 04:09:14.667443 139772195035008 model_lib_v2.py:1010]     + Loss/RPNLoss/localization_loss: 0.049334
```

Figure 5.5: Output of evaluation steps

5.1.5. Implement the Best Architecture to the Existing System

After the evaluation and analysis, the best model implemented to the existing system that already been developed before. The existing system was developed using python language. Code of the existing system written below :

```
import numpy as np
import tensorflow as tf
import cv2
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
import mysql.connector
db =
mysql.connector.connect(host="IP",user="Username",password="Password"
,database="Database")
eksekusiDb = db.cursor()
sql = "INSERT INTO tblDataPelanggar (time, pelanggar_masker,
pelanggar_jarak, bukti_ss) VALUES (%s, %s, %s, %s)"
import base64
import requests
import json
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")
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):
    distanceViolations = 0
    maskViolations = 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:
            maskViolations+=1
    box_05 =
    output_dict['detection_boxes'][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)
                    distanceViolations+=1
    if distanceViolations!=0 or maskViolations!=0:
        mixer.init()
        mixer.music.load('alert.ogg')
        mixer.music.play()
    return(hasil, distanceViolations, maskViolations)
cap = cv2.VideoCapture(2)
terakhir = datetime.datetime.now()
while 1:
    _, img = cap.read()
    img = cv2.resize(img, (640,360))
    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=(20,285), 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=(20,285), 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=(20,335), 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=(20,335), 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":"api",
                  "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()
```

5.2. Results

5.2.1. RAW Data

After doing some training and evaluation, the researcher got some RAW data that will be explained in the next steps. To evaluate the consistency of the model, the researcher done 2 times experiments. First experiment using 2:8, 5:5 & 8:2 data scale, the second experiment using 2:2, 5:2 & 8:2 data scale.

5.2.1.1. Experiment 1 (2:8, 5:5, 8:2 data scale)

Faster R-CNN ResNet50 V1 640x640 (5000 steps, 8 Batch Size)								
Data Scale	**mAP**	**mAP Large**	**mAP Medium**	**mAP Small**	**AR @100**	**AR @100 Large**	**AR @100 Medium**	**AR @100 Small**
2:8	0,1516	0,5235	0,2565	0,07701	0,3129	0,6565	0,4781	0,2134
5:5	0,221	0,6474	0,3246	0,1544	0,4149	0,7512	0,571	0,3195
8:2	0,2635	0,7802	0,3605	0,186	0,4403	0,7998	0,5579	0,3625

Table 5.1: Experiment 1 - RAW output of Faster R-CNN ResNet50 V1 architecture

SSD ResNet50 V1 FPN 640x640 (5000 steps, 8 Batch Size)								
Data Scale	**mAP**	**mAP Large**	**mAP Medium**	**mAP Small**	**AR @100**	**AR @100 Large**	**AR @100 Medium**	**AR @100 Small**
2:8	0,2322	0,5058	0,297	0,1795	0,3694	0,6983	0,4847	0,2898
5:5	0,3364	0,6123	0,3701	0,3022	0,5105	0,7886	0,6007	0,4423
8:2	0,3985	0,7733	0,4461	0,3633	0,5471	0,7883	0,6389	0,4806

Table 5.2: Experiment 1 - RAW output of SSD ResNet50 V1 FPN architecture

SSD MobileNet V2 320x320 (5000 steps, 8 Batch Size)								
Data Scale	**mAP**	**mAP Large**	**mAP Medium**	**mAP Small**	**AR @100**	**AR @100 Large**	**AR @100 Medium**	**AR @100 Small**
2:8	0,153	0,5913	0,2655	0,07084	0,2981	0,7017	0,4983	0,1775
5:5	0,1676	0,6234	0,2615	0,09134	0,3437	0,7163	0,5458	0,2177
8:2	0,1604	0,5273	0,2696	0,09545	0,3602	0,8016	0,5307	0,2447

Table 5.3: Experiment 1 - RAW output of SSD MobileNet V2 architecture

5.2.1.2. Experiment 2 (2:2, 5:2, 8:2 data scale)

Faster R-CNN ResNet50 V1 640x640 (5000 steps, 8 Batch Size)

Data Scale	mAP	mAP Large	mAP Medium	mAP Small	AR @100	AR @100 Large	AR @100 Medium	AR @100 Small
2:2	0,1626	0,4342	0,2823	0,08499	0,3363	0,6423	0,4845	0,2318
5:2	0,2359	0,7342	0,3034	0,2097	0,4146	0,7525	0,5181	0,3506
8:2	0,2604	0,7699	0,3485	0,1938	0,417	0,7911	0,5078	0,3536

Table 5.4: Experiment 2 - RAW output of Faster R-CNN ResNet50 V1 architecture

SSD ResNet50 V1 FPN 640x640 (5000 steps, 8 Batch Size)

Data Scale	mAP	mAP Large	mAP Medium	mAP Small	AR @100	AR @100 Large	AR @100 Medium	AR @100 Small
2:2	0,2377	0,4762	0,2831	0,2065	0,392	0,6563	0,4621	0,347
5:2	0,3437	0,6705	0,3268	0,347	0,5145	0,7084	0,5538	0,488
8:2	0,3967	0,8006	0,4261	0,371	0,5753	0,8296	0,6233	0,5324

Table 5.5: Experiment 2 - RAW output of SSD ResNet50 V1 FPN architecture

SSD MobileNet V2 320x320 (5000 steps, 8 Batch Size)

Data Scale	mAP	mAP Large	mAP Medium	mAP Small	AR @100	AR @100 Large	AR @100 Medium	AR @100 Small
2:2	0,1376	0,4714	0,2282	0,07639	0,3381	0,7661	0,4985	0,2249
5:2	0,1725	0,5703	0,2924	0,1004	0,3765	0,7687	0,5422	0,2701
8:2	0,1667	0,5233	0,2785	0,1002	0,3626	0,7736	0,502	0,2745

Table 5.6: Experiment 2 - RAW output of SSD MobileNet V2 architecture

5.2.2. Explanation

5.2.2.1. Experiment 1 (2:8, 5:5, 8:2 data scale)

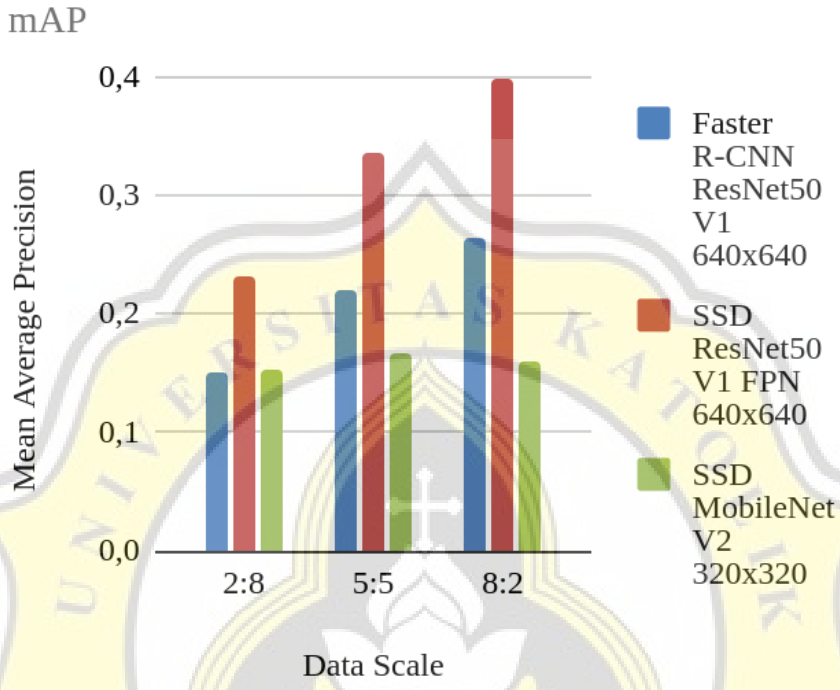

Figure 5.6: Experiment 1 - Mean average precision of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.6 shows the data of mean average precision from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision for detection. In a specific situation (2:8 data scale), Faster R-CNN ResNet50 V1 and SSD MobileNet V2 classification loss approximately the same. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate. That's because data can know about variate types of data (also avoid overfit).

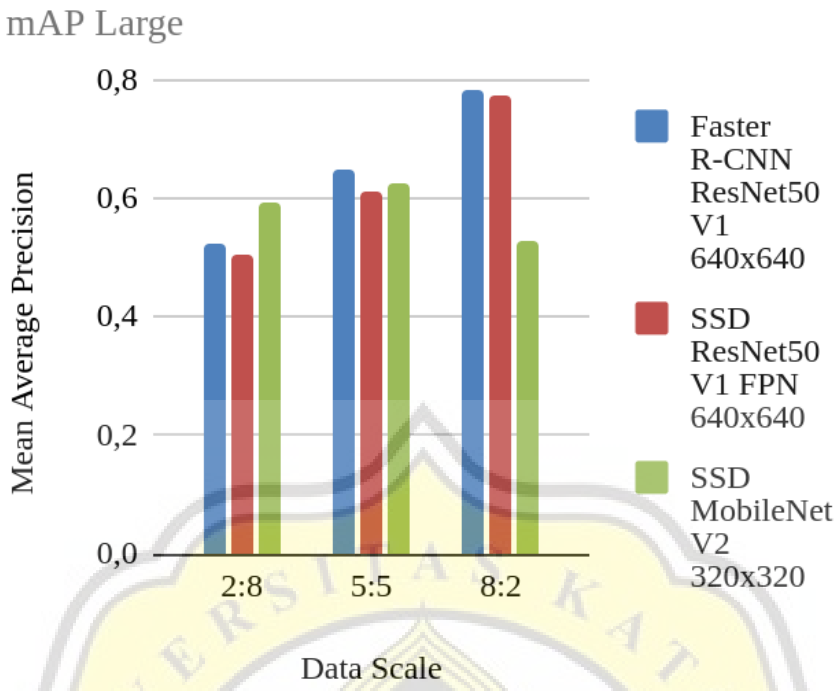

Figure 5.7: Experiment 1 - Mean average precision of large images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.7 shows the data of mean average precision of large images size detection from the evaluation section. From the chart above, we know that SSD MobileNet V2 architecture has the biggest mean average precision (specifically for large-size images) in the 2:8 data scale. On another scale, Faster R-CNN ResNet50 V1 has the biggest mean average precision (specifically for large images). From the overall experiment, we know that **Faster R-CNN ResNet50 V1 is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect large images (only SSD MobileNet V2 that gave the different result). That's because data can know about variate types of data (also avoid overfit).

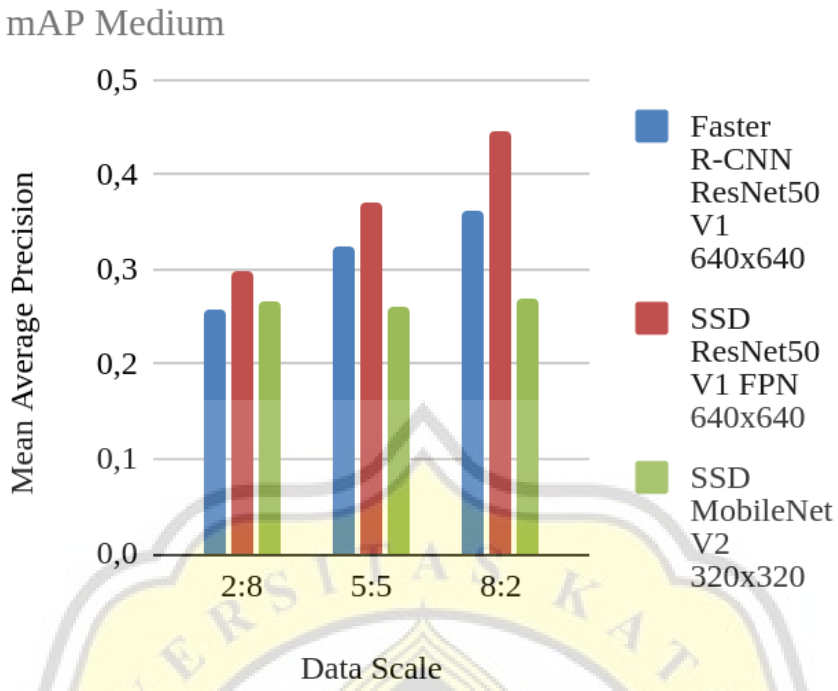

Figure 5.8: Experiment 1 - Mean average precision of medium images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.8 shows the data of mean average precision of medium images size detection from the evaluation section. This data is the data from the evaluating section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision (specifically for medium size images) for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect medium size images (only SSD MobileNet V2 that gave the different result). That's because data can know about variate types of data (also avoid overfit).

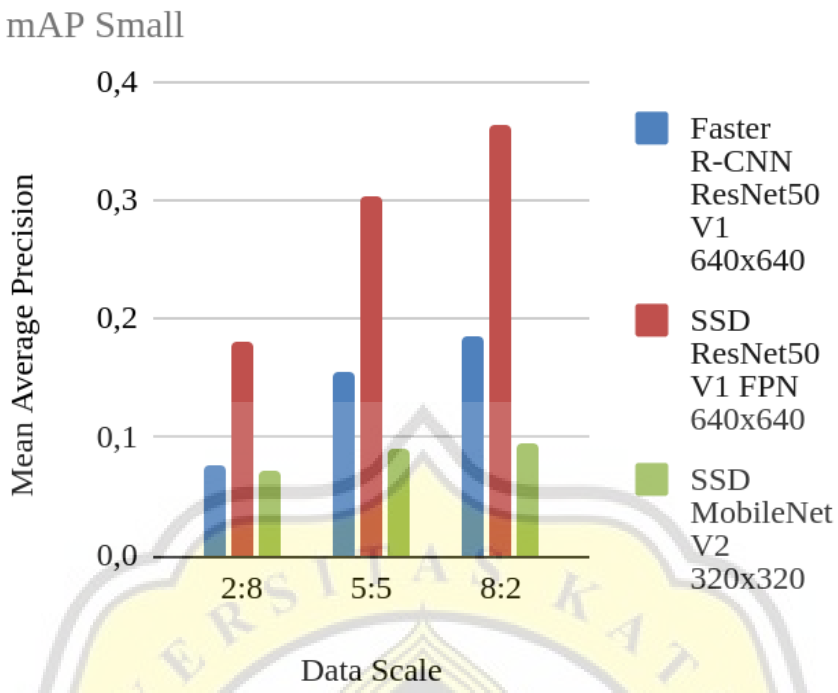

Figure 5.9: Experiment 1 - Mean average precision of small images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.9 shows the data of mean average precision of small images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision (specifically for small size images) for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect small-size images. That's because data can know about variate types of data (also avoid overfit).

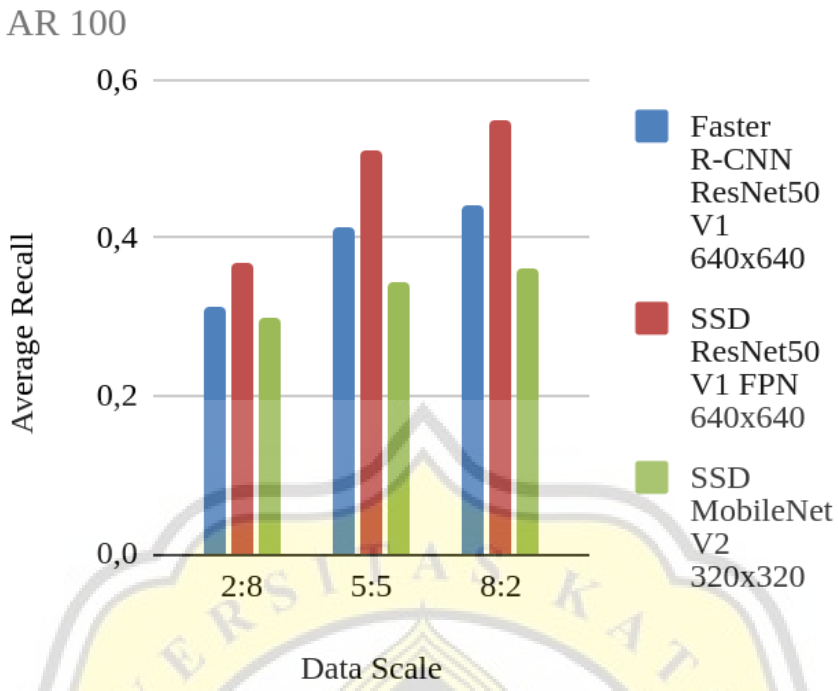

Figure 5.10: Experiment 1 - Average recall of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.10 shows the data of average recall from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect images. That's because data can know about variate types of data (also avoid overfit).

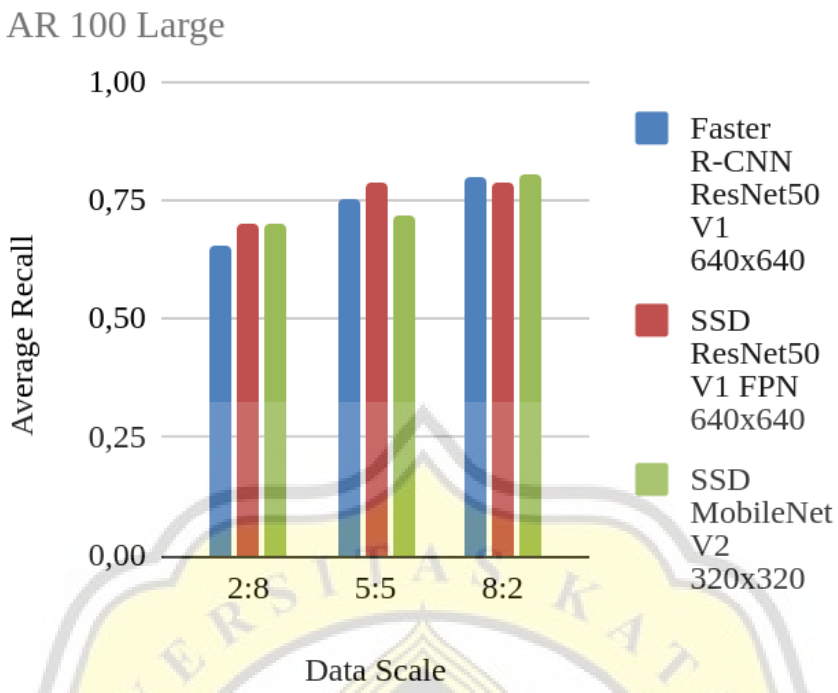

Figure 5.11: Experiment 1 - Average recall of large image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.11 shows the data of average recall of large images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for detection in the 2:8 and 5:5 data scales. But on the 8:2 scale, SSD MobileNet has the biggest average recall. Overall from this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect large-size images (only SSD ResNet50 V1 FPN that gave the different result). That's because data can know about variate types of data (also avoid overfit).

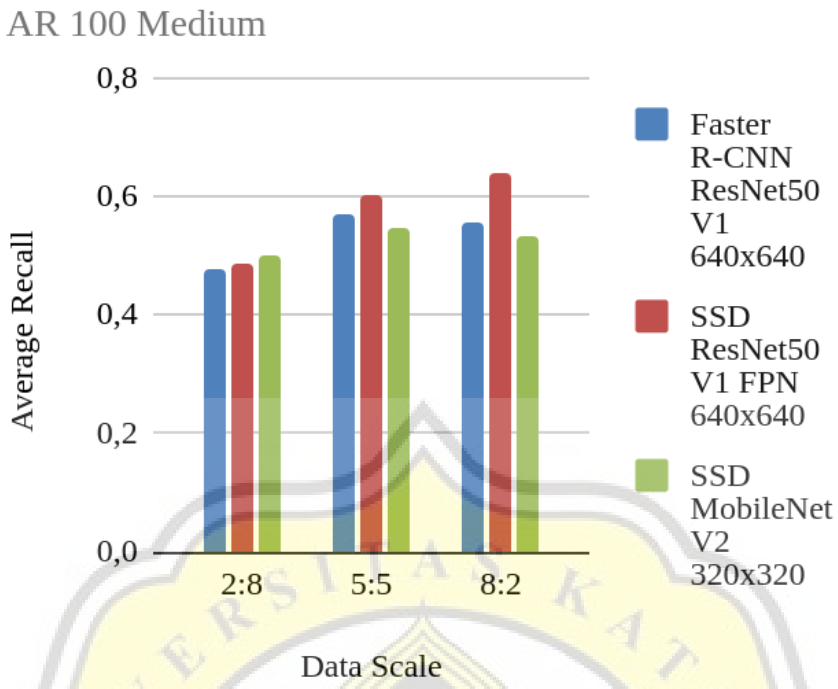

Figure 5.12: Experiment 1 - Average recall of medium image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.12 shows the data of average recall of medium images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for detection overall. But in the 2:8 scale, SSD MobileNet V2 more accurate. Overall from this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case. From another perspective, for the Faster R-CNN ResNet 50 V1 & SSD MobileNet V2, the data scale didn't do much to increase recall.

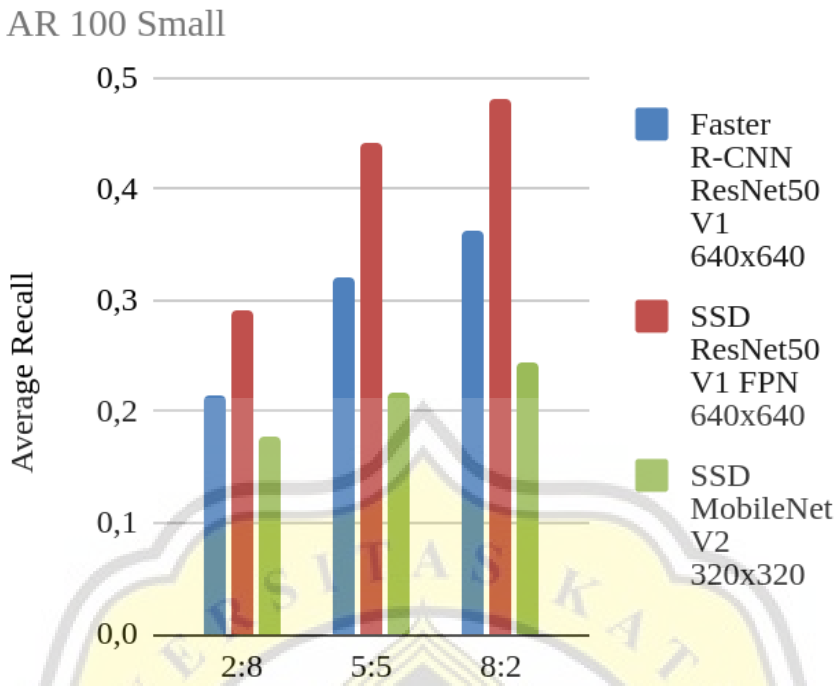

Figure 5.13: Experiment 1 - Average recall of small image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.13 shows the data of average recall of small images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for all of the experiments. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect small-size images. That's because data can know about variate types of data (also avoid overfit).

5.2.2.2. Experiment 2 (2:2, 5:2, 8:2 data scale)

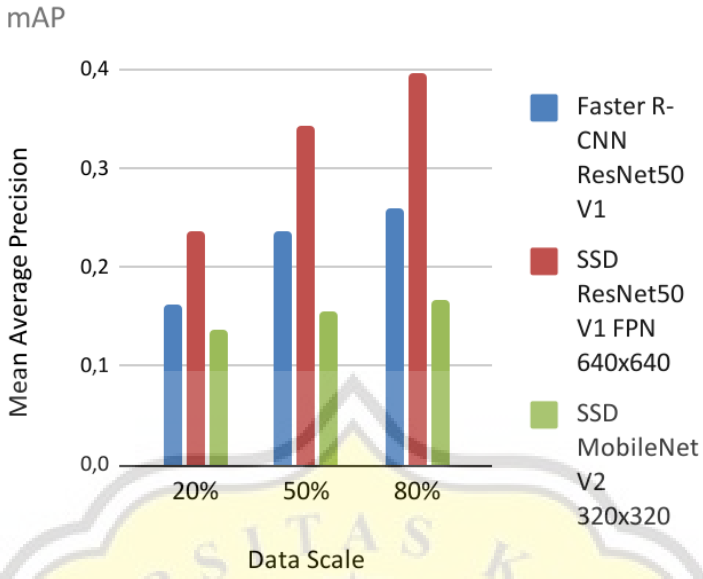

Figure 5.14: Experiment 2 - Mean average precision of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.14 shows the data of mean average precision from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision for detection. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect objects. That's because data can know about various types of data (also avoid overfit).

Figure 5.15: Experiment 2 - Mean average precision of large images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.15 shows the data of mean average precision from the evaluation section. From the chart above we know that overall SSD ResNet50 V1 FPN architecture has the biggest mean average precision for detection (20% & 80% data scale). From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect large images (only SSD MobileNet V2 that gave the different result). That's because data can know about various types of data (also avoid overfit).

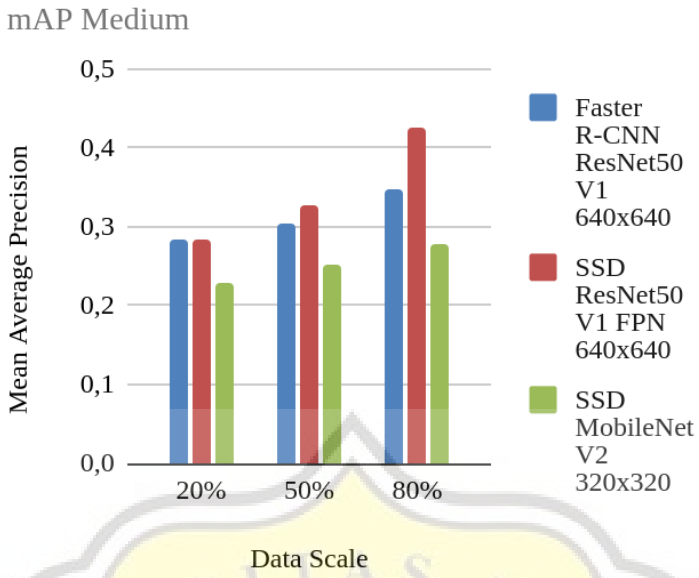

Figure 5.16: Experiment 2 - Mean average precision of medium images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.16 shows the data of mean average precision of medium images size detection from the evaluation section. This data is the data from the evaluating section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision (specifically for medium size images) for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect medium size images. That's because data can know about various types of data (also avoid overfit).

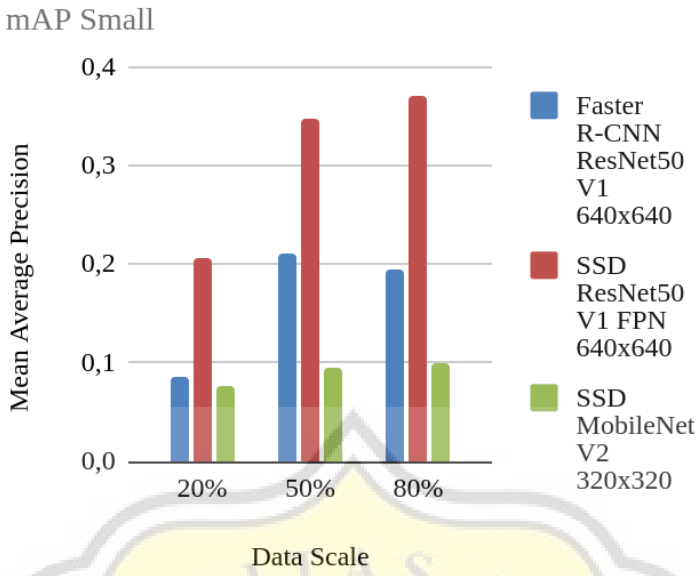

Figure 5.17: Experiment 2 - Mean average precision of small images size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.17 shows the data of mean average precision of small images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest mean average precision (specifically for small size images) for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect small-size images (Only Faster-RNN ResNet50 V1 that gave different results. 50% data scale better than 80% data scale). That's because data can know about various types of data (also avoid overfit).

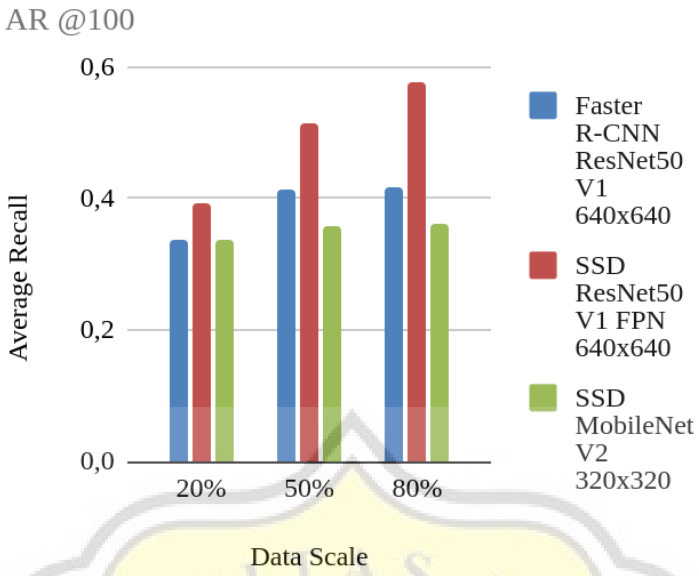

Figure 5.18: Experiment 2 - Average recall of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.18 shows the data of average recall from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for detection in all data scales. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect images. That's because data can know about variate types of data (also avoid overfit).

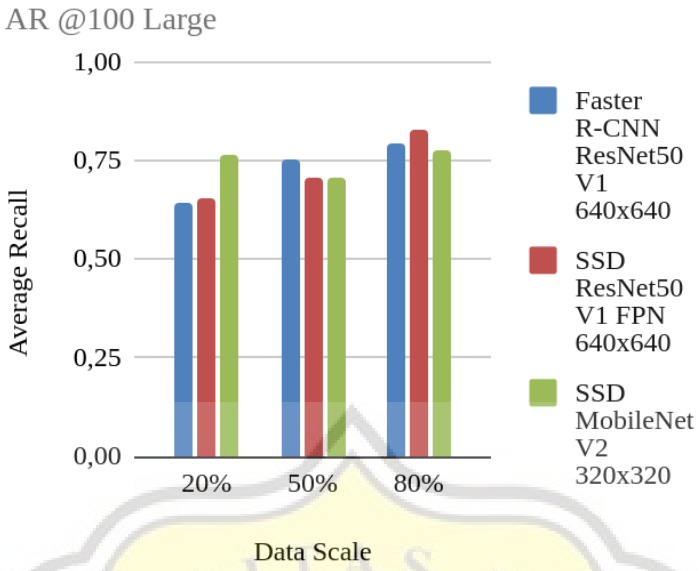

Figure 5.19: Experiment 2 - Average recall of large image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.19 shows the data of average recall of large images size detection from the evaluation section. From the chart above, we know that the results are so random. From the chart above, we know that SSD MobileNet V2 architecture has the biggest average recall for the large images in the 2:2 data scale. Faster R-CNN ResNet 50 V1 architecture has the biggest average recall for the large images in the 2:2 data scale. SSD ResNet50 V1 FPN architecture has the biggest average recall for the large images in the 8:2 data scale. From this situation, we know that the score of the three architectures is approximately the same.

From another perspective, overall we know that more data made the system more accurate to detect images. That's because data can know about variate types of data (also avoid overfit).

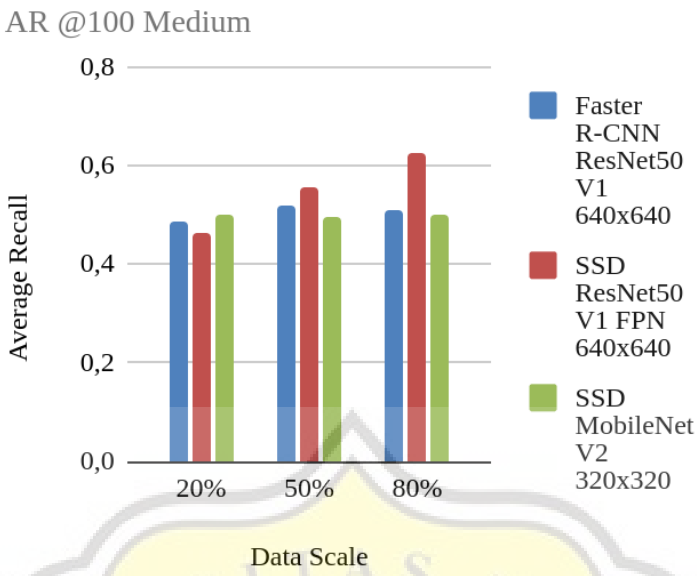

Figure 5.19: Experiment 2 - Average recall of medium image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.19 shows the data of average recall of medium images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for detection overall. But in the 20% data scale experiment, SSD MobileNet V2 is more accurate. Overall from this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case. From another perspective, for the Faster R-CNN ResNet 50 V1 & SSD MobileNet V2, the data scale didn't do much to increase recall.

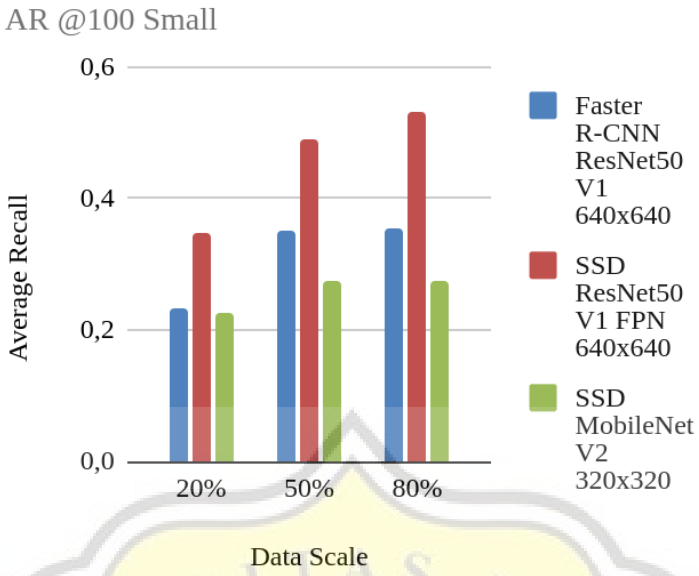

Figure 5.20: Experiment 2 - Average recall of small image size detection of Faster-RNN ResNet50 V1, SSD ResNet50 V1 FPN & SSD MobileNet V2 architectures

Figure 5.20 shows the data of average recall of small images size detection from the evaluation section. From the chart above, we know that SSD ResNet50 V1 FPN architecture has the biggest average recall for all of the experiments. From this situation, overall we know that **SSD ResNet50 V1 FPN is more effective** than others in this case.

From another perspective, overall we know that more data made the system more accurate to detect small-size images. That's because data can know about various types of data (also avoid overfit).