CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Experiment Set-Up

This research combines articial intelligence models namely Nvidia Jetson Nano and the SSD algorithm. The Nvidia Jetson Nano model used is the Nvidia Jetson Nano Developer Kit New B01 Revision - B01 4GB with an ofcial operating system of Linux4Tegra which has a computing GPU of 128-core Maxwell, Quad-core ARM A57 @ 1.43 GHz CPU, and RAM of 4 GB. We need to download Python version 3.7 and Tensorflow-2.10.0 to run the software that will run on Nvidia. In this research software use, SSD MobileNet V2 320x320 was obtained from Tensorow 2 for the training process. The testing, was carried out in a room that has bright enough light and a good internet network.

5.2. Data Collecting

The data used in this experiment obtained from https://www.kaggle.com/andrewmvd/face-mask-detection . The following is a sample image for testing showed on Figure 5.1:

Figure 5.1 Example of Data Test

5.3. Pre-Processing Data

Before the image will be processed by the system, the image will rst be labeled. The rst thing to do is convert the xml data to csv. Code to convert xml to csv running by the le "xml_to_csv.py".

```
1. def xml_to_csv(img_files, xml_files):
2.     xml_list = []
3.     for i, xml_file in enumerate(xml_files):
```

```
4.          tree = ET.parse(xml_file)
5.          root = tree.getroot()
```

Below is a snippet of code to name the column with the name filename, width, height,etc.

```
6.          for member in root.findall('object'):
7.              value = (root.find('filename').text,
8.                       int(root.find('size').find('width').text),
9.                       int(root.find('size').find('height').text),
10.                        member[0].text,
11.                       int(member.find("bndbox").find('xmin').text),
12.                       int(member.find("bndbox").find('ymin').text),
13.                      int(member.find("bndbox").find('xmax').text),
14.                      int(member.find("bndbox").find('ymax').text)
15.                      )
16.
17.             xml_list.append(value)
```

Below is the code to set data using pandas DataFrame.

```
18.     column_name = ['filename', 'width', 'height', 'class', 'xmin',
                       'ymin', 'xmax', 'ymax']
19.      xml_df = pd.DataFrame(xml_list, columns=column_name)
20.      return xml_df
```

The output of “xml_to_csv.py” as follow on Figure 5.2:

	filename	width	height	class	xmin	ymin	xmax	ymax
2	makssekskses0.png	512	366	without_mask	79	105	109	142
3	makssekskses0.png	512	366	with_mask	185	100	226	144
4	makssekskses0.png	512	366	without_mask	325	90	360	141
5	makssekskses1.png	400	156	with_mask	321	34	354	69
6	makssekskses1.png	400	156	with_mask	224	38	261	73
7	makssekskses1.png	400	156	with_mask	299	58	315	81
8	makssekskses1.png	400	156	with_mask	143	74	174	115
9	makssekskses1.png	400	156	with_mask	74	69	95	99
10	makssekskses1.png	400	156	with_mask	191	67	221	93
11	makssekskses1.png	400	156	with_mask	21	73	44	93
12	makssekskses1.png	400	156	with_mask	369	70	398	99
13	makssekskses1.png	400	156	without_mask	83	56	111	89
14	makssekskses10.png	301	400	with_mask	98	267	194	383

Figure 5.2 Ouput of xml_to_csv.py

Next will be a labeling map that is used to store the class. Label map is done by running the file "generate_labelmap.py" with the output in the form of a pbtxt file named label_map.pbtxt below:

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

The following is the output of "generate_labelmap.py" on Figure 5.3:

label_map.pbtxt | OprekDeteksi.py | nindaprojek1.py

```
item{
id: 1
name: 'Tanpa Masker'
}

item{
id: 2
name: 'Menggunakan Masker'
}

item{
id: 3
name: 'Penggunaan Masker Salah'
}
```

Figure 5.3 Output of generate_labelmap.py

Next do TFRecord by running the file "generate_tfrecord.py" . In this step there is an adjustment to the map label, about value and id with the map label in the previous step. The code is written below:

```
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
```

5.4. Software

Next, after preprocessing is done, do the training model. The training model using SSD MobileNet V2 320x320. Which is to prepare the ".config" file by preparing the required configurations such as num_classes=3 with the code below:

```
  ssd {
    num_classes: 3
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
```

The feature_extractor hyperparameter below contains the SSD type and other requirements for this step.

```
    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
      }
```

The box_coder hyperparameter below is used for image scale initialization:

```
      box_coder {
        faster_rcnn_box_coder {
          y_scale: 10.0
          x_scale: 10.0
          height_scale: 5.0
          width_scale: 5.0
        }
      }
```

Hyperparameter of matcher to adjust thresholds:

```
      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
        }
      }
```

Hyperparameter box predictor for customizing desired image data:

```
      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
        }
      }
```

Hyperparameter anchor generator is used to adjust the ratio:

```
      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
        }
      }
```

Adjust the normalization loss as needed:

```
      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 {
    }
  }
  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
  }
```

The hyperparameter to train the ssd model using code below:

```
  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
}
```

Read the input train from the previous step, namely from the "label_map.pbtxt" file:

```
train_input_reader {
  label_map_path: "/content/dataset/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "/content/dataset/train.record"
  }
}
```

Specifies the type of metric to use:

```
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
```

Evaluate the accuracy of the previous step of each training:

```
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"
  }
}
```

To start training model use the command “!python /model_main_tf2.py --alsologtostderr --model_dir=dirOfModel --pipeline_config_path=file.config”. Below is output of the training step on Figure 5.4 :

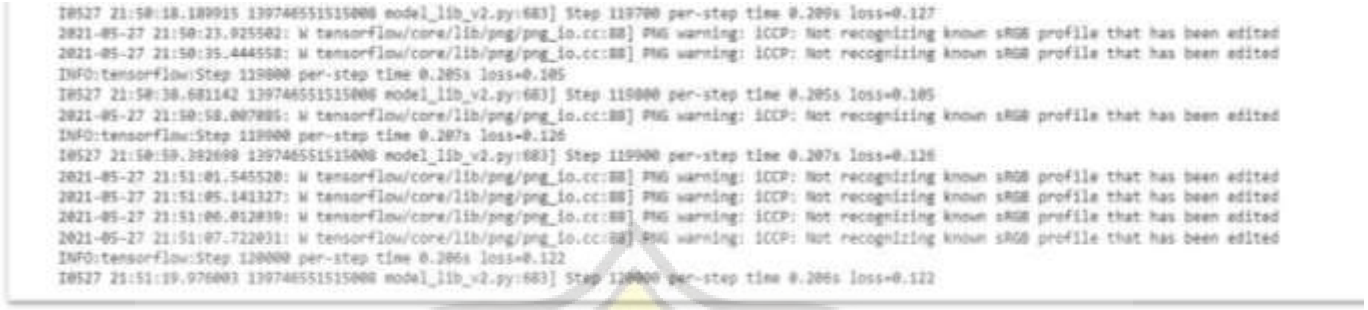

Figure 5.4 Output of Training Step

5.5. Implementation

After preprocessing and training data, the next step is to connect the software and hardware so the system can be used. To connect software and hardware using python language.

To determine the center point use the function below:

```
186. def deteksi_center(ymin, xmin, ymax, xmax, w, h):
187.     cx = ((xmin+xmax)/2)*w
188.     cy = ((ymin+ymax)/2)*h
189.     return int(cx),int(cy)
```

Below is the code that uses the library from Tensorflow to preprocess data:

```
190. def run_inference_for_single_image(model, image):
191.     image = np.asarray(image)
192.     input_tensor = tf.convert_to_tensor(image)
193.     input_tensor = input_tensor[tf.newaxis,...]
194.     model_fn = model.signatures['serving_default']
195.     output_dict = model_fn(input_tensor)
196.     num_detections = int(output_dict.pop('num_detections'))
197.     output_dict = {key:value[0, :num_detections].numpy()
198.                    for key,value in output_dict.items()}
199.     output_dict['num_detections'] = num_detections
200.     output_dict['detection_classes'] = output_dict ['detecti on_cl
   asses'].astype(np.int64)
201.     if 'detection_masks' in output_dict:
202.         detection_masks_reframed =
   utils_ops.reframe_box_masks_to_image_masks(
203.             output_dict['detection_masks'],
   output_dict['detection_boxes'],
204.             image.shape[0], image.shape[1])
205.
206.         detection_masks_reframed = tf.cast(detection_masks_reframed
   >= 0.5, tf.uint8)
207.         output_dict['detection_masks_reframed'] =
   detection_masks_reframed.numpy()
208.     return output_dict
```

Below is the code to detect every 1 image frame:

```
209. def show_inference(model, image_np):
210.     totalMelanggarMasker = 0
211.
212.     output_dict = run_inference_for_single_image(model, image_np)
213.
214.     hasil = vis_util.visualize_boxes_and_labels_on_image_array(
215.         image_np,
216.         output_dict['detection_boxes'],
217.         output_dict['detection_classes'],
218.         output_dict['detection_scores'],
219.         category_index,
220.         instance_masks=output_dict.get('detection_masks_reframed',
    None),
221.         use_normalized_coordinates=True,
222.         line_thickness=8)
```

Below is the code to determine if x ==1 or x==3 then the totalViolatingMaker is increased by 1:

```
223.     for x in
    output_dict['detection_classes'][output_dict['detection_scores']>=0.5]
    :
224.         if x==1 or x==3:
225.             totalMelanggarMasker+=1
226.
227.     box_05 =
    output_dict['detection_boxes'][output_dict['detection_scores']>=0.5]
228.     score_05 =
    output_dict['detection_scores'][output_dict['detection_scores']>=0.5]
```

Below is the code to print the results from the system:

```
229.     # Masker
230.     if inferencehasil[2] != 0:
231.         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),
232.             thickness=3, lineType=cv2.LINE_AA)
233.     else:
234.         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),
235.             thickness=3, lineType=cv2.LINE_AA)
```

5.6. Integrate Hardware

After combining Nvidia Jetson Nano 4GB, IMX219 camera module, Sandisk extreme pro 64GB MicroSD, Jetson Nano acrylic case, TL-WN722N usb wi adapter, HDMI video capture USB 2.0, buzzer, 5V 4A adapter, cooling fan. And the results can be seen in the following image on Figure 5.5 .

Figure 5.5 Shows Hardware Assembly

5.7. Testing and Evaluation

After implementing the system, the next step is to test the system. The results of the test are presented in the table below:

Table 5.1. Results of Testing

No	Actual		Predicted		TP	TN	FP	FN
	Non Violation	Violation	Non Violation	Violation				
1	0	6	1	5	5	0	1	0
2	2	0	2	0	0	2	2	2
3	2	1	3	0	0	2	3	2
4	2	0	2	0	0	2	4	2
5	0	3	0	3	3	0	5	0
6	2	0	2	0	0	2	6	2
7	0	1	0	1	1	0	7	0
8	0	1	0	1	1	0	8	0
9	1	1	1	1	1	1	9	1
10	2	0	2	0	0	2	10	2
SUM					11	10	0	2

Based on the calculations from table 5.1 above, the accuracy with the formula is obtained by 91.3%, Precision by 100%, and Recall by 84.6%. Below is a sample capture of the experimental results :

(a) (b)

(c) (d)

(e)

Figure 5.6 (a)Example of Results 1,(b) Example of Results 2, (c) Example of Results 3, (d) Example of Results 4 , and (e) Example of Results 5

Showed in Figure 5.6(a) is able to detect the using mask but is wrong, Figure 5.6(b) and (e) use mask properly, and Figure 5.6(c) and (d) not using the mask.

5.8. Discussion

Based on calculations, accuracy is 91.3%, precision is 100%, and recall is 84,6%. The results above can be concluded that the SSD MobileNet V2 320x320 is able to detect violations use of masks. Several factors cause accuracy and recall not to be 100 % because the lighting is not bright, the distance from the camera, and the position of the object is not facing the camera. For lighting, it is required to be in a room with enough light so that the camera can capture images. The brighter, the clearer object can be captured by the camera. The distance to the camera showed on Figure 5.5(e) must not be too far from the camera's reach so that it can be captured by the camera. And the position of the object with the camera must be visible so that it can be detected clearly. If the position of the object faces the camera, the camera will capture the image maximum and the system can detect it optimally.