CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

The code is splitted into 2 parts, the first part contains the process to acquire and train the CNN, and the second part contains the process to turn CNN from image classification algorithm to object detection algorithm.

5.1.1. Train CNN

```
      function preload(){
1.    let j = 817;
2.    let a = 9;
3.      for(let i = 0; i < 300 ; i++){
4.  jerukbusuk[i]=loadImage(`dataset/800x600/train/rottenoranges/
jerukbusuk(${j}).png`);
5.  jeruksegar[i]=loadImage(`dataset/800x600/train/freshoranges/
jeruksegar(${j}).png`);
6.        j++;
7.        a++;
8.      }
9.  }
```

Line 1 to Line 9 contains a function to load the images to train the CNN. Line 1 and 2 declare from which order the images want to be loaded, the i < 300 in line 3 declare how many images that are gonna be loaded. Line 4 and 5 are the variables that contain all the loaded images.

```
10. function setup(){
11.   createCanvas(400400);
12.   let options = {
13.     task: 'imageClassification',
14.     inputs:[IMAGE_WIDTH,IMAGE_HEIGHT,IMAGE_CHANNELS],
15.     debug: true,
16.   };
17.   cnn = ml5.neuralNetwork(options);
18.   for(let f = 0; f < jerukbusuk.length; f++){
19.     let i = 0;
20.     cnn.addData({image : jerukbusuk[i]},{label : "Busuk"});
21.     cnn.addData({image : jeruksegar[i]},{label : "Segar"});
22.     i++;
23.   }
24.   console.log('add data selesai');
25.   cnn.normalizeData();
26.   console.log('normalizeData jalan');
27.   cnn.train({epochs : 50},trainingselesai);
28. }
```

Line 10 to 28 contains the function to set up the CNN and train the CNN. Line 12 to 16 are used to declare which machine learning to use from the ml5 library, in this case the CNN was chosen and saved in the option variable. In line 17 the machine learning that were declared in the previous line are acquired and inserted into the CNN variable. Line 18 to 23 is a loop to label the loaded images and add it into CNN. Line 25 is a function to normalize the data so it can be used to train the CNN. Line 27 is the line to start training the data, epochs in line 27 function as how many times the CNN is going to be trained.

```
29. function trainingselesai(){
30.   console.log('training selesai');
31.   cnn.save();
32. }
```

Line 30 is an indicator to indicate if the training process is done, and line 31 is a function to save the trained CNN.

5.1.2. Object Detection

```
33. groundtruth = [
34.   ['jeruksegar(1).png',
35.     [
36.       ['Segar','100%',58,99,455,439]
37.
38.     ]
39.   ],
40.   ['jeruksegar(2).png',
41.     [
42.       ['Segar','100%',25,31,354,306]
43.
44.     ]
45.   ],
46.   ['jeruksegar(3).png',
47.     [
48.       ['Segar','100%',40,47,407,383]
49.
50.     ]
51.   ],
52.   ['jeruksegar(22).png',
53.     [
54.       ['Segar','100%',89,31,291,121],
55.       ['Segar','100%',46,122,286,355],
56.       ['Segar','100%',287,63,501,278]
57.
58.     ]
59.   ],
60.   ['jeruksegar(16).png',
61.     [
62.       ['Segar','100%',1,17,563,446]
63.
64.     ]
65.   ],
66.   ['jeruksegar(243).png',
```

```
67.     [
68.       ['Segar','100%',8,21,171,147],
69.       ['Segar','100%',9,147,175,280],
70.       ['Segar','100%',5,280,171,416]
71.
72.     ]
73.   ],
74.   ['jeruksegar(226).png',
75.     [
76.       ['Segar','100%',28,21,314,296]
77.
78.     ]
79.   ],
80.   ['jeruksegar(240).png',
81.     [
82.       ['Segar','100%',25,13,349,333],
83.       ['Segar','100%',175,104,539,432],
84.       ['Segar','100%',382,41,680,354]
85.
86.     ]
87.   ],
88.   ['jeruksegar(246).png',
89.     [
90.       ['Segar','100%',11,8,397,341]
91.
92.     ]
93.   ],
94.   ['jeruksegar(248).png',
95.     [
96.       ['Segar','100%',25,21,272,269],
97.       ['Segar','100%',157,79,439,368]
98.
99.     ]
100.  ],
101.  ['jerukbusuk(1).png',
102.    [
103.      ['Segar','100%',43,32,365,198],
104.      ['Segar','100%',82,202,237,347],
105.      ['Segar','100%',242,202,324,288],
106.      ['Busuk','100%',372,124,419,196],
107.      ['Busuk','100%',329,203,419,288],
108.      ['Busuk','100%',243,294,389,346],
109.      ['Busuk','100%',165,349,371,418]
110.
111.
112.    ]
113.  ],
114.  ['jerukbusuk(2).png',
115.    [
116.      ['Busuk','100%',26,16,290,277]
117.    ]
118.  ],
119.
120.  ['jerukbusuk(3).png',
121.    [
122.      ['Busuk','100%',113,49,313,196],
```

```
123.       ['Segar','100%',33,57,107,197],
124.       ['Segar','100%',249,3,456,47],
125.       ['Segar','100%',318,50,515,201],
126.       ['Segar','100%',29,201,502,356]
127.
128.     ]
129.   ],
130.
131.   ['jerukbusuk(6).png',
132.     [
133.       ['Busuk','100%',25,14,203,230]
134.     ]
135.   ],
136.   ['jerukbusuk(8).png',
137.     [
138.       ['Busuk','100%',29,34,230,185],
139.       ['Busuk','100%',231,54,449,247],
140.       ['Busuk','100%',29,133,259,356]
141.     ]
142.   ],
143.   ['jerukbusuk(222).png',
144.     [
145.       ['Busuk','100%',52,1,310,247]
146.     ]
147.   ],
148.   ['jerukbusuk(229).png',
149.     [
150.       ['Segar','100%',24,9,310,247],
151.       ['Busuk','100%',23,128,252,252]
152.     ]
153.   ],
154.   ['jerukbusuk(270).png',
155.     [
156.       ['Busuk','100%',70,14,343,301]
157.     ]
158.   ],
159.   ['jerukbusuk(248).png',
160.   [
161.     ['Segar','100%',31,273,397,410],
162.     ['Busuk','100%',30,15,409,268]
163.   ]
164.   ],
165.   ['jerukbusuk(233).png',
166.   [
167.     ['Busuk','100%',51,29,265,242]
168.
169.   ]
170.   ]
171.];
```

Line 33 to 171 declare the ground truth data such as coordinate, filename, and label, ground truth data is used to evalute the algorithm.

```
172.let imgwidth = imgElement.width / 2;
173.let imgheight = imgElement.height / 2;
```

Line 172 and 173 determine how many parts the image will be divided, in this case the image is going to be divided into 3 parts in width and 3 parts in height resulting in a total of 9 parts.

```
174. while(roix1 != imgElement.width && roix2 != imgElement.height){
175.       let rect = new cv.Rect(roix1,roix2,roiy1,roiy2);
176.       dst = src.roi(rect);
177.       cv.imshow('canvasOutput', dst);
178.       resizeimg();
179.       i++;
180.       klasifikasi();
181.
182.       if(roix1 != imgElement.width){
183.         roix1 = roix1 + imgwidth;
184.
185.       }
186.       x[i] = roix1;
187.       w[i] = roix1 + imgwidth;
188.
189.       if(roix1 == imgElement.width && roix2 <= imgElement.height){
190.         roix1 = imgwidth;
191.         roix2 = roix2 + imgheight;
192.
193.       }
194.       y[i] = roix2;
195.       h[i] = roix2 + imgheight;
196.
197.       f++;
198.     }
```

Line 174 to 198 is a while loop process and save image coordinate. Line 175 declares a rectangle with the coordinate of roix1,roix2,roiy1,roiy2. Line 176 extracted a part of the image with the coordinate that was declared by line 175 and sent to html as output in line 177. Line 178 is a function to resize the image that is already outputted by line 177 so it can be classified by CNN in a function in line 180, this was done because CNN can only classify an image that has the same resolution as the images that CNN gets trained with. Line 182 to line 185 are used to check if the roix1 value is the same as the input image width. This was done so the code in line 175 and 176 can extract the next image to the right of the previous image until it reaches the right end of the input image. Line 186 and 187 are arrays that are used to save the x and width coordinate of the image. Line 189 to line 193 check if the roix1 has already reached the end of the input image, and check if the roix2 has a smaller value than the input image height. If the conditions are met then it reset the roix1 and increase the value of roix2 so the code in line 175 and 176 can extract the next lower parts of the input image from left to right. Line 194 and 195

are arrays that are used to save the y and height coordinate of the extracted image. If every part of the input image is already extracted then the loop will stop.

```
function pyramid(){
  let src = cv.imread(imgElement);
  let dst = new cv.Mat();
   x = [];
   y = [];
  recordnama = []
  recordprediksi = []
  cv.pyrDown(src, dst, new cv.Size(0, 0), cv.BORDER_DEFAULT);
  cv.imshow('canvasOutput', dst);
  src.delete(); dst.delete();
  src = cv.imread('canvasOutput');
  dst = new cv.Mat();
    let imgwidth = src.size().width / width;
    let imgheight = src.size().height / height;
    console.log('pyramid height ' , imgheight);
    let roix1 = 0;
    let roix2 = 0;
    let roiy1 = imgwidth;
    let roiy2 = imgheight;
    let i = 0;
    x[i] = 0;
    w2[i] = x[i] + imgwidth;
    y[i] = 0;
    h2[i] = y[i] + imgheight;
    while(roix1 != src.size().width && roix2 != src.size().height){
      let rect = new cv.Rect(roix1,roix2,roiy1,roiy2);
      dst = src.roi(rect);
      cv.imshow('canvasOutput', dst);
      i++;
      resizeimg();
        klasifikasi();
      if(roix1 != src.size().width){
        roix1 = roix1 + imgwidth;
      }
      x[i] = roix1;
      w2[i] = x[i] + imgwidth;
      if(roix1 == src.size().width && roix2 <= src.size().height){
        roix1 = imgwidth;
        roix2 = roix2 + imgheight;

      }
      y[i] = roix2;
      h2[i] = y[i] + imgheight;

      f++;
    }
    src.delete();
    dst.delete();
}
```

Line 199 to 250 have the works the same as line 172 to 198 the only difference is in the line 206. Line 206 is a function to make the input image smaller in size.

```
248.function setup() {
249.  createCanvas(400, 400);
250.
251.  let options = {
252.    task: "imageClassification",
253.    input: [64, 64, 4],
254.  };
255.  cnn = ml5.neuralNetwork(options);
256.  const modelDetails = {
257.    model: "model/64x64/300 data/model.json",
258.    metadata: "model/64x64/300 data/model_meta.json",
259.    weights: "model/64x64/300 data/model.weights.bin",
260.  };
261.  hasildiv = createDiv("Loading model...");
262.
263.  cnn.load(modelDetails, modelReady);
264.
265.}
```

line 248 to line 265 are used to load the CNN. Line 252 declares what kind of machine learning that are gonna be used, imageClassificatin is the code name for CNN. Line 253 declares the image size that can be classified with the CNN. Line 255 acquired the CNN from the ml5 library and inserted it into the cnn variable. Line 256 to 260 are used to get the trained CNN that are saved before. Line 261 outputs a text to indicate the model is still being loaded. Line 263 is a command to load the CNN. Everything in function setup will be the first to be executed when the program starts, because setup function is a function from the p5 library that is used to insert some line of code that wants to be executed first when the program starts.

```
266.function record(){
267.  let n = 0;
268.  if(c == 0 ){
269.  for(i = 0 ; i < x.length ; i ++){
270.    if (recordprediksi[i] > 60){
271.      recordx[n] = Math.round(x[i]);
272.      recordy[n] = Math.round(y[i]);
273.      label[n] = recordnama[i];
274.      prediksi[n] = recordprediksi[i];
275.      recordw[n] = w[i];
276.      recordh[n] = h[i];
277.      n++;
278.    }
279.  }
280.  console.log('mulai pyramid');
281.  z = 0;
282.  f = 0;
283.  recordnama=[];
284.  recordprediksi = [];
285.  x = [];
286.  y = [];
```

```
287.  c++;
288.  pyramid();
289.  }else{
290.    n = 0;
291.    for(i = 0 ; i < x.length ; i ++){
292.      if (recordprediksi[i] > 60 ){
293.        recordx2[n] = Math.round(x[i]);
294.        recordy2[n] = Math.round(y[i]);
295.        label2[n] = recordnama[i];
296.        prediksi2[n] = recordprediksi[i];
297.        recordw2[n] = w2[i];
298.        recordh2[n] = h2[i];
299.        n++;
300.      }
301.    }
302.    nms();
303.}
304.}
```

Line 266 to 304 is a function that records all the coordinates and classification results. Line 269 to 279 records all the coordinates and classification results from line 174 to 198. Line 268 is a function to make sure all the process from line 269 to 279 is done before executing line 290 to 303 to execute and record all the coordinates and classification results from the pyramid function. Line 270 and 292 works as a filter to filter out all the prediction results that have prediction values smaller than 60%. Line 302 executes the NMS function.

```
305.function nms(){
306.//mencari file name
307.let tempgroundtruthlength;
308. var inp = document.getElementById('fileInput');
309.for (i = 0; i < inp.files.length; i++) {
310.    let file = inp.files[i];
311.      filename = file['name'];
312.}
313.//mencari panjang groundtruth yang mempunyai nama sama dengan filename
314.for ( let z = 0  ; z < groundtruth.length ; z++){
315.  groundtruthname = groundtruth[z][0];
316.  if(filename == groundtruthname){
317.    console.log('ground truth full',groundtruth[z]);
318.      for(let f = 0 ; f < groundtruth[z][1].length; f++ ){
319.        groundtruthlabel = groundtruth[z][1][f][0];
320.        tempgroundtruthlength = groundtruth[z][1].length;
321.      }
322.      if(b > 0){
323.         groundtruthlength = groundtruthlength + tempgroundtruthlength
   ;
324.      }else{
325.        groundtruthlength = tempgroundtruthlength  ;
326.      }
327.    }
328.  }
329.
330.
331.  prediksi =  prediksi.map(str => {
```

```
332.    return Number(str);
333.  });;
334.  prediksi2 = prediksi2.map(str => {
335.    return Number(str);
336.  });;
337.  let boxes = [];
338.  let recordxfinal = recordx.concat(recordx2);
339.  let recordyfinal = recordy.concat(recordy2);
340.  let recordwfinal = recordw.concat(recordw2);
341.  let recordhfinal = recordh.concat(recordh2);
342.  let labelfinal = label.concat(label2);
343.
344.  const prediksinomor = prediksi.concat(prediksi2);
345.  for ( i = 0 ; i < recordxfinal.length ; i ++){
346.boxes[i]=
   [recordyfinal[i],recordxfinal[i],recordhfinal[i],recordwfinal[i]];
347.  }
348.  let text = prediksinomor.map(String);
349.  //merubah data koordinat menjadi tensor agar dapat digunakan di nms
350.  const shape = [boxes.length ,4];
351.  const boxes1 = tf.tensor2d(boxes,shape);
352.//menghilangkan kotak prediksi yg overlap
353.const        nms        =        tf.image.nonMaxSuppression(boxes1,
   prediksinomor,tempgroundtruthlength + 1 ,0.5);
354.let mat = cv.imread(imgElement);
355.hasil = nms.arraySync();
356.let font = cv.FONT_HERSHEY_SIMPLEX
357.//visualisasi hasil prediksi
```

Line 308 to 312 used to get the input filename. Line 314 to 329 is a loop used to check if the filename has the same name as one of the ground truth data. Line 315 used to acquire the ground truth name of current iteration in the loop. Line 316 if the filename has the same value as the current ground truth name, if it is correct then line 319 will collect the label of the ground truth and line 320 will collect the current ground truth length. The data collected by line 320 is used for evaluation later. Line 331 to 336 remove the '.' from the data and convert the string data of both prediction value from the original input image and pyramid into integer so it can be processed. Line 338 to 344 merge all the recorded data of coordinates, label, and prediction value. Line 345 to 347 is a loop to put all the coordinates data into 1 array value so it can be processed into a tensor by line 350 to determine the tensor shape and line 351 to turn the data into tensor. Line 348 converts the prediction value into a string and puts it into a text array. The data from line 320, 344,and 346 is used to do Non Max Surpression function in line 353. It removes the overlap in the bounding boxes and chooses the bounding boxes that have higher value. Line 354 is to get the original input image and put it in the mat variable. The result from line 353 is a tensor, it gets converted in line 355 into an array so it can be used more easily. Line 356 determines a text font that is going to be used to visualize the object detection results.

```
358.for(let f = 0 ; f < hasil.length ; f++){
359.         let    point1    =    new    cv.Point(recordxfinal[hasil[f]],
   recordyfinal[hasil[f]]);
360.         let    point2    =    new    cv.Point(recordwfinal[hasil[f]],
   recordhfinal[hasil[f]]);
361.         let    point3    =    new    cv.Point(recordxfinal[hasil[f]],
   recordyfinal[hasil[f]] - 10 );
362.      let   point4   =   new   cv.Point(recordwfinal[hasil[f]]   +   10,
   recordhfinal[hasil[f]]);
363.      let   point5   =   new   cv.Point(recordwfinal[hasil[f]]   -   20,
   recordhfinal[hasil[f]] - 20);
364.  let z = f.toString();
365.          cv.putText(mat,labelfinal[hasil[f]],point3,     font,     0.5,
   [0,0,255,0],1,cv.LINE_AA);
366.             cv.putText(mat,text[hasil[f]],point4,      font,      0.5,
   [0,0,255,0],1,cv.LINE_AA)
367.  cv.putText(mat,z,point5, font, 0.5,[0,0,255,0],1,cv.LINE_AA)
368.  cv.rectangle(mat, point1,point2, [0, 0, 30, 255 ], 3);
369.}
370.//visualisasi groundtruth
371.for ( let z = 0  ; z < groundtruth.length ; z++){
372.  groundtruthname = groundtruth[z][0];
373.  if(filename == groundtruthname){
374.      for(let f = 0 ; f < groundtruth[z][1].length; f++ ){
375.                  let  point1  =    new  cv.Point(groundtruth[z][1][f]
   [2],groundtruth[z][1][f][3]);
376.                   let  point2  =  new  cv.Point(groundtruth[z][1][f]
   [4],groundtruth[z][1][f][5]);
377.        cv.rectangle(mat, point1,point2, [0, 0, 255, 255 ], 3);
378.      }
379.
380.  }
381.}
382.console.log('b',b);
383.cv.imshow('canvasOutput', mat);
384.mat.delete();
```

Line 358 to 368 is a loop that visualize the results of the algorithm and sent it back as an output to html. To visualize a value in opencvjs the value needs to be turned into a point value, line 359 turn the x and y coordinate into point1 value, and line 360 turn the width and height coordinate into point2 value. Line 361 and 362 is the same as line 359 and 360 the difference is line 361, 362, and 363 is used as a point to visualize as a text the classification label and the order of the classification. Line 364 used to convert the loop counter into string so it can be visualize. Line 365 to 367 are where the text got visualize using the point from variable from line 361 to 363. The results of line 355 is used to select which label, coordinates, and prediction value got visualize. The predicted object label is visualized in line, the prediction value is visualize in line 366 and the loop counter is visualize in line 367. Line 368 create a rectangle based on the coordinate from line 359 and 360. The visualized text from line 365 to 367 are placed around the rectangle from line 368. Line 371 to 381 will visualize the groundtruth data if the condition in

line 373 is met. This was done by looping all the ground truth data and matching it with input file data. If it match then line 374 will start a loop based on the matching ground truth data to grab the coordinate data. Line 375 and 376 grabs the coordinate data and insert it into a point function so it can be visualize by line 377. Line 383 sends all the visualize data into html as an output. After sending the data into html line 384 will delete the visualize data.

```
385.let koordinatprediksi = [];
386.var inp = document.getElementById('fileInput');
387. f = 0;
388. //memasukan semua data prediksi ke dalam 1 array
389.for (i = 0; i < inp.files.length; i++) {
390.    let file = inp.files[i];
391.    for (let k = 0 ; k  < hasil.length ; k++){
392.                                                  koordinatprediksi[k]       =
   [file['name'],labelfinal[hasil[f]],text[hasil[f]],
393.      recordxfinal[hasil[f]], recordyfinal[hasil[f]],
394.      recordwfinal[hasil[f]], recordhfinal[hasil[f]]];
395.      filename = file['name'];
396.      f++;
397.    }
398.}
399.  koordinatfinal[b] = koordinatprediksi
400.  for ( i = 0 ; i < groundtruth.length ; i++){
401.    if(filename == groundtruth[i][0]){
402.      iou();
403.    }
404. }
```

Line 386 get the inputted file from html and puts it in inp variable. Line 389 to 398 is a loop to put all the processed data that have been selected by the non max suppression method in line 353 into a global array. All the data temporarily stored in array in line 392 then stored into global array in 399. The variable in line 399 indicates how many times the image has been inputted and how many times the object detection process is already done. Line 400 to 404 check if the inputted file name have a match in ground truth data. If the conditions are met then execute the iou function, which is the first step to evaluate the algorithm.

```
405.for ( let z = 0  ; z < groundtruth.length ; z++){
406.        groundtruthname = groundtruth[z][0];
407.
408.        if(filename == groundtruthname){
409.          console.log('ground truth full',groundtruth[z]);
410.          for(let i = 0 ; i < koordinatfinal[b].length ; i ++){
411.            for(let f = 0 ; f < groundtruth[z][1].length; f++ ){
412.              groundtruthlabel = groundtruth[z][1][f][0];
413.                  console.log('groundtruth[z][1].length',groundtruth[z]
   [1].length);
414.               xa = max([groundtruth[z][1][f][2] , koordinatfinal[b][i]
   [3]]);
415.               ya = max([groundtruth[z][1][f][3] , koordinatfinal[b][i]
   [4]]);
```

```
416.                xb = min([groundtruth[z][1][f][4] , koordinatfinal[b][i]
   [5]]);
417.                yb = min([groundtruth[z][1][f][5] , koordinatfinal[b][i]
   [6]]);
418.               interArea = max(0, xb - xa +1 ) * max(0, yb - ya +1);
419.                boxAArea = (groundtruth[z][1][f][4] - groundtruth[z][1]
   [f][2] +1) *
420.               (groundtruth[z][1][f][5] - groundtruth[z][1][f][3] +1);
421.                boxBArea = (koordinatfinal[b][i][5] - koordinatfinal[b]
   [i][3] +1) *
422.                 (koordinatfinal[b][i][6] - koordinatfinal[b][i][4]+1);
423.                 labeliou[i] = koordinatfinal[b][i][1];
424.                      iou[f] = interArea / float(boxAArea + boxBArea -
   interArea);
425.                 console.log('iou',iou);
426.             }
427.             //cari tp fp dan fn
428.             totaliou[i] = max(iou);
```

Line 410 to 426 is a function to count the iou between ground truth and the prediction. It loops the ground truth data that have the same name as the input file name and compares the ground truth data to every prediction data. Line 414 and 415 compare the x and y of ground truth data and prediction data to find the maximum value between two data and save it as xa and ya. Line 416 and 417 compare the x and y of ground truth data and prediction data to find the minimum value between the two data and save it into xb and yb. Line 418 is a mathematical process to find the intersection value between the ground truth data and prediction data. Line 419 and 421 creates boxes based on the ground truth and prediction data. The box in line 419 is based on ground truth data and line 421 is based on prediction data. Line 423 acquired the current loop prediction label and inserted it to the labeliou array. Line 424 is the mathematical process to find the iou by dividing the intersection area from line 418 and the boxes area from line 419 and 421. Since the loop works by looping the prediction data to every ground truth data, the author needs to find the prediction data that are intersecting with one of the ground truth data. To do it the iou data needs to be compared and find the max value, because the highest the iou value means that the prediction data and ground truth data are intersecting, this function is done line 428 to find the maximum value of the iou.

```
429.    if(totaliou[i] > threshold && labeliou[i] == groundtruthlabel){
430.                  tp.push([filename,i,labeliou[i],totaliou[i]]);
431.
432.               }else if(totaliou[i] > 0 && totaliou[i] <threshold)
433.               {
434.                 fp.push ([filename,i,labeliou[i],totaliou[i]]);
435.
436.               }
437.               else{
438.                 fn.push ([filename,i,labeliou[i],totaliou[i]]);
439.
440.               }
441.
```

```
442.                tptotal[i] = tp.length;
443.                fptotal[i] = fp.length;
444.                fntotal[i] = fn.length;
445.                predictionlength = predictionlength + m;
446.                precision[i] = tp.length / (tp.length + fp.length);
447.                precision[i] = precision[i] || 0;
448.                recall[i] = tp.length / (tp.length + fn.length);
449.                recall[i] = recall[i] || 0;
450.                precisiontotal[i] = tp.length / predictionlength;
451.                precisiontotal[i] = precisiontotal[i] || 0;
452.                recalltotal[i] = tp.length /groundtruthlength;
453.                recalltotal[i] = recalltotal[i] || 0;
454.                console.log('precisiontotal[i]',precisiontotal[i])
455.                console.log('totaliou.length',totaliou.length)
456.                console.log('tp.length',tp.length)
457.                console.log(' predictionlength', predictionlength)
458.              }
459.            }
460.          }
```

The process from line 429 is still inside the loop of line 410. Line 429 to 440 is a if function to find the tp,fp and fn based on the if condition. Line 442 to 444 is an array variable to save the current loop tp, fp, and fn length. Line 445 is a global variable to save the prediction data length, the m in line 445 is an integer with a value of 1. Line 446 is a mathematical process to get the precision value and save it in a precision array. Line 447 will convert the value of line 446 to 0 if the value is NaN. Line 448 is a mathematical process to get the recall value. Line 449 will convert the value of line 448 to 0 if the value is NaN. Line 450 and 452 are a mathematical process to get the precision and recall that are needed for the evaluation process using MAP. Line 451 and 453 will convert the value of line 450 and 452 to 0 if the value is NaN. Line 460 is the end of the loop.

```
461.    tabeliou[b]=[labeliou,confidence,tptotal,fptotal,fntotal,precision
   total,recalltotal,iou];
462.    prediksifinal.push( [precisiontotal,recalltotal,labeliou]);
463.        refresh();
464.      ap();
465.      b++;
466.    }
```

Line 416 is used to insert all the processed data from line 405 to 460 into a global array. Line 462 add the precision,recall,and iou label into prediksifinal array. Line 463 will execute a function to reset most of the global variable. Line 464 will execute the last function to do the evaluating process. Line 465 add 1 to the value of variable b.

```
467. for(let  i = 0 ; i < prediksifinal.length ; i++){
468.    if(i > 0 ){
469.    precisiontotal = precisiontotal.concat(prediksifinal[i][0]);
470.    recalltotal = recalltotal.concat(prediksifinal[i][1]);
471.    labeltotal = labeltotal.concat(prediksifinal[i][2]);
472.    }else{
```

```
473.     precisiontotal = prediksifinal[i][0];
474.     recalltotal = prediksifinal[i][1];
475.     labeltotal = prediksifinal[i][2];
476.     }
477.
478.   }
479.
```

Line 467 to 479 is a loop to acquired the precision, recall, and label that are saved in prediksifinal array into precisiontotal, recalltotal, and labeltotal array.

```
480.   for ( let i = 0 ; i < precisiontotal.length ; i ++){
481.    if(temprecall.includes(recalltotal[i]) == false){
482.     // temprecall[m] = recalltotal[i];
483.     temprecall.push(recalltotal[i]);
484.     tempprecision.push(precisiontotal[i]);
485.     templabel.push(labeltotal[i]);
486.     // console.log('i',i);
487.
488.    }else{
489.     let f = temprecall.indexOf(recalltotal[i]);
490.     let z = precisiontotal[f];
491.     if(precisiontotal[i] > z){
492.       precisiontotal[f] = precisiontotal[i];
493.     }
494.    }
495.   }
```

Line 480 to 495 is a loop to remove the duplicate in recall and get the highest precision based on the order of the duplicate recall. I variable will function as a counter to count the loop iteration. Line 481 get the value of recalltotal array in the order of i and checks if that value match any value in temprecall array. If the condition is met, then it stores the current recalltotal value in the order of i into temp recall array, current precisiontotal value in the order of i into tempprecision, and curent labeltotal value the order of i into templabel array. If the condition didn't met, then it execute line 489 to 493 to get the precision that have the same order as the recall in the recalltotal array that match the recall value in the temprecall variable. Line 489 finds the order of the recall value in the order of i in recalltotal array that match the value in temprecall array then stores the order into f variable. Line 490 get the precision value in precisiontotal array in f order and store the value into z variable. Line 491 checks if the precisiontotal value in the order of ti is higher than z, if the condition is correct then the value in precisiontotal in f order get replaced by the precisiontotal value in the order of i.

```
496.   for(let i = 0 ; i < tempprecision.length ; i++){
497.     n =  n + tempprecision[i];
498.   }
499.
500.   ap = n / 11; //ap for 11 point interpolation for all class
```

Line 496 to 500 is a 11 point interpolation method to calculate average precision. This is 1 of 2 ways to calculate average precision. Line 496 to 498 is a loop to sum all the values in temp precision and saves it into a variable. Line 500 divided the n value by 11 to get the ap value using the 11 point interpolation method. This method won't be used to calculate MAP, the author chooses to use the second method to calculate the MAP. This line was created as a reference for the author to see which method is better.

```
501.  for(let i = 0 ; i < tempprecision.length ; i++){
502.    if(interprecision.includes(tempprecision[i]) == false){
503.      interprecision[m] = tempprecision[i];
504.      interrecall[m] = temprecall[i];
505.      interlabel [m] = templabel[i];
506.      console.log('interrecall m',interrecall[m]);
507.      m++;
508.
509.    }else{
510.      z= interrecall.length -1 ;
511.
512.      if(temprecall[i] > interrecall[z]){
513.        interrecall[z] = temprecall[i];
514.      }
515.      console.log('i',i)
516.
517.    }
518.    // console.log('interrecall m',interrecall[m])
519.  }
```

Line 501 to 519 works the same as line 480 to 495 the difference is line 480 to 495 process the precision while line 501 to 519 process the recall value.

```
520.  for(let a = 0 ; a < kelas.length ; a++){
521.    interaptotal = 0;
522.    interap = [];
523.    let k = 0
524.  //calculating area under curve precision x recall
525.      let l = 0;
526.      for(let i = 0 ; i < interprecision.length ; i++){
527.        if(interlabel[i] == kelas[a]){
528.        interap[k] = interprecision[i] * (interrecall[i] - l);
529.        l = interrecall[i];
530.        interap[k] = interap[k] || 0;
531.        console.log('kelas a',kelas[a])
532.        console.log('interlabel i',interlabel[i])
533.        console.log('interprecision[i] ',interprecision[i] )
534.        k++;
535.       }
536.      }
537.      console.log('interap',interap)
538.      //hitung hasil all point interpolation ap
539.      for ( let i = 0; i < interap.length; i++){
540.        interaptotal = interaptotal + interap[i];
541.      }
542.
```

```
543.  interapkelas[a] = interaptotal || 0;
544.}
```

Line 520 to 544 is a second method to calculate average precision, this method called all point interpolation. Line 520 creates a loop based on the number of classes to calculate average precision for each class of the data; the loop ends in line 544. Line 526 creates a loop based on inter precision array length with i as a loop counter. Is a condition to make sure the loop only processes the data that have the same label as kelas array in order of a. Line 528 is the process to calculate the average precision and saves it into an interap array in order of k variables. L is a variable that have default value of 0 but the value will be replaced by interrecall value after each loop iteration. Line 530 used to make sure the value of the interap array isn't NaN, if it NaN then the value is replaced with 0. Line 539 to 541 is a loop to sum every value in an interap array and save it to an interaptotal variable. Line 543 saves the value of the interaptotal into an interapkelas array in the order of a and makes sure the value isn't NaN.

```
545.  let map;
546.  let aptotal = 0;
547.  for(let i = 0 ; i < interapkelas.length; i++){
548.    aptotal = aptotal + interapkelas[i];
549.  }
550.  map = aptotal / interapkelas.length;
551.  // console.log('prediksifinallength',prediksifinal.length);
552.  console.log('label total',labeltotal);
553.  console.log('precision total' , precisiontotal);
554.  console.log('recall total',recalltotal);
555.  console.log('templabel',templabel);
556.  console.log('temprecall',temprecall);
557.  console.log('tempprecision',tempprecision);
558.  console.log('interlabel', interlabel);
559.  console.log('interprecision',interprecision);
560.  console.log('interrecall',interrecall);
561.  console.log('ap',aptotal);
562.  console.log('interap',interap);
563.  console.log('interaptotal',interaptotal)
564.  console.log('interapkelas',interapkelas)
565.  console.log('MAP',map);
566.}
```

Line 547 is a loop to sum all the values in interapkelas. This was done because the formula to calculate a map is the sum of all average precision classes divided by the number of classes. Line 550 is the process to calculate the map.

5.2. Results

This test is done using the CNN that is trained using a total of 300 images in 64 x 64 and 200 x 100 resolution across 20 images in total, 10 rotten orange images and 10 fresh orange images with the IoU threshold of 0.3. This test compares the results of the object detection between using NMS and without using NMS. Without using NMS the filter in line 270 and 292

needs to be changed to >50 and < 99. And remove the ‘hasil’ from line 392 to 394 and in line 359 to 368.

5.2.1. Object Detection

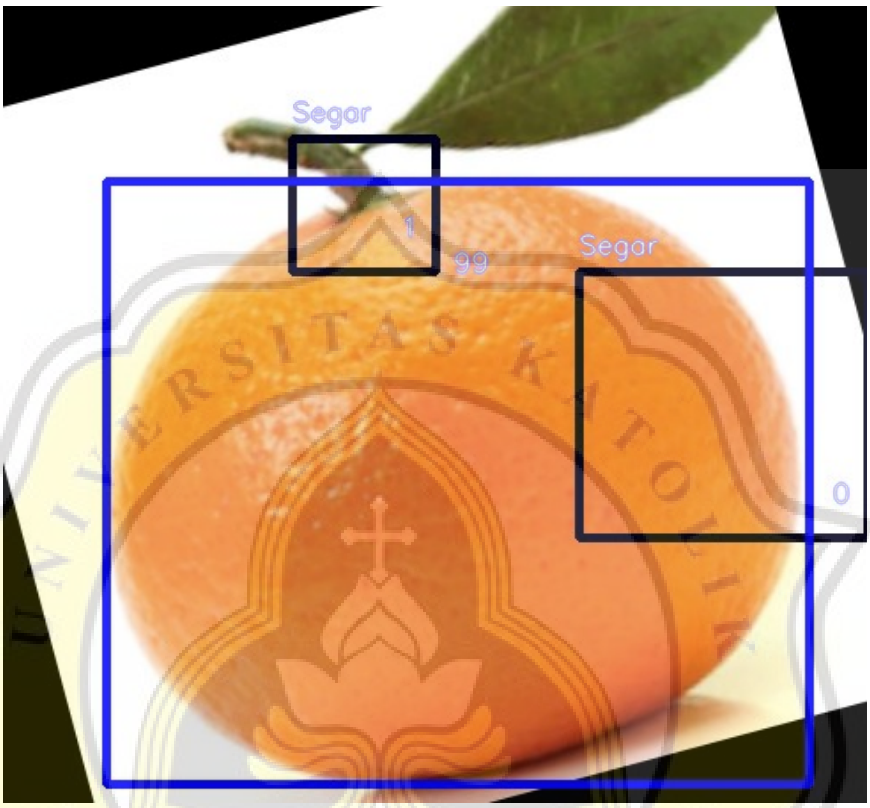

Figure 12: Object Detection Results With NMS

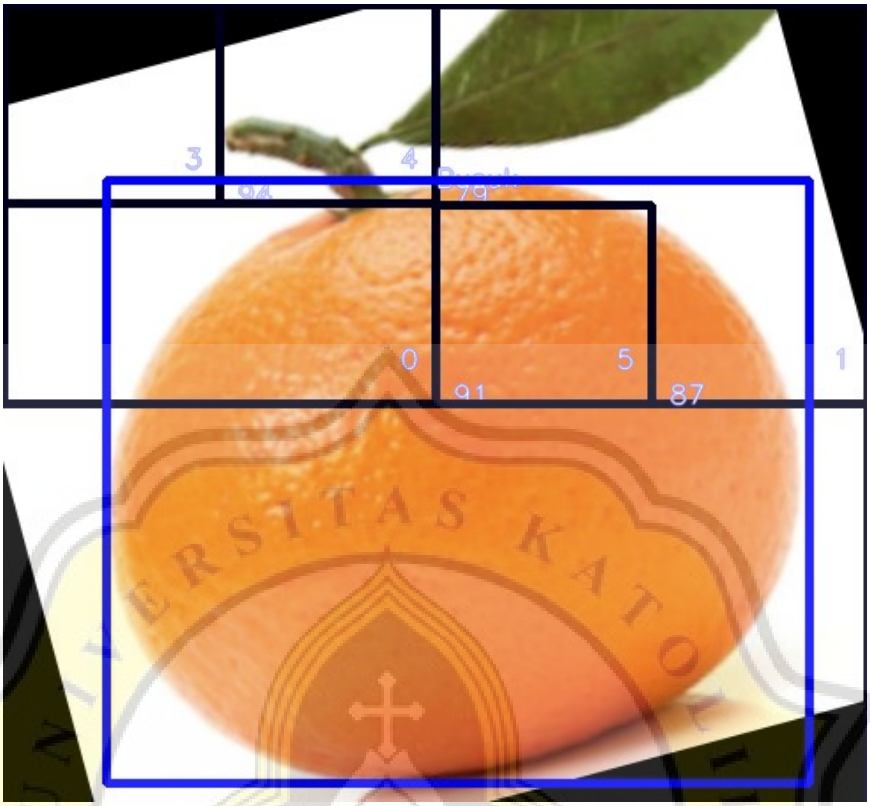

Figure 13: Object Detection Results Without NMS

Figure 12 and 13 show the object detection results. The blue rectangle indicates the ground truth while the black rectangle indicates the prediction made by the CNN it shows the prediction label and score. The object detection prediction doesn't fully cover the object area, this happens because the image is divided into small parts and the ground truth area is bigger than the divided parts. Figure 13 is the results of object detection without using NMS function and only using filters to remove the prediction boxes that have prediction scores lower than 50% and the prediction boxes that have the score of 99%. Figure 13 have more prediction boxes than Figure 12 that are using NMS because the NMS function also filters out the prediction boxes that are intersected with one another or overlap while without NMS the all prediction boxes that meet the requirement of the filter get visualized resulting in more prediction boxes.

5.2.2. MAP

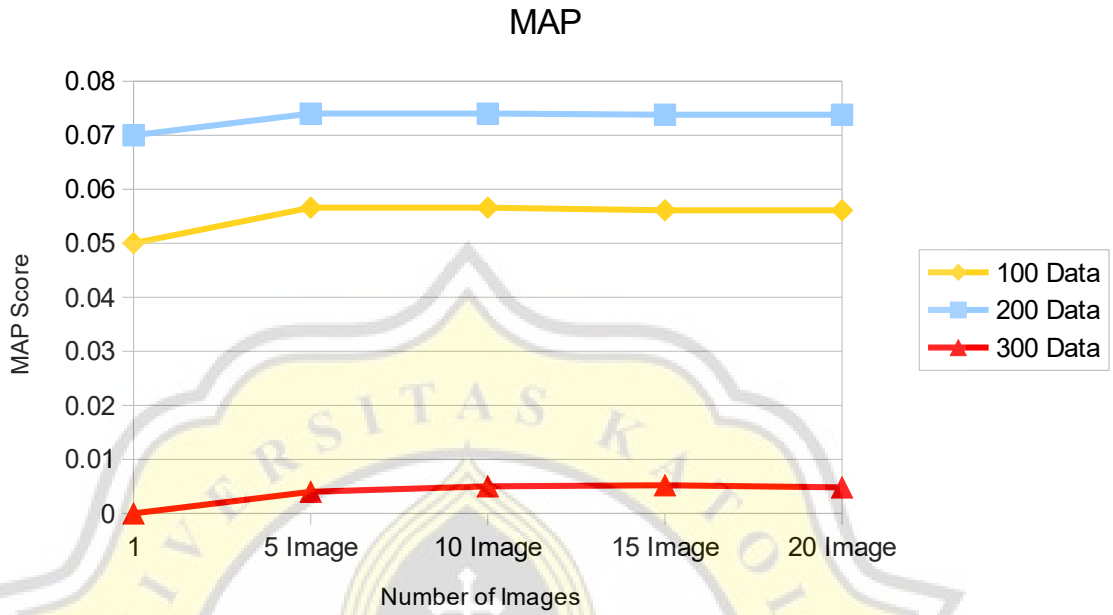

Figure 14: MAP Result With NMS

Figure

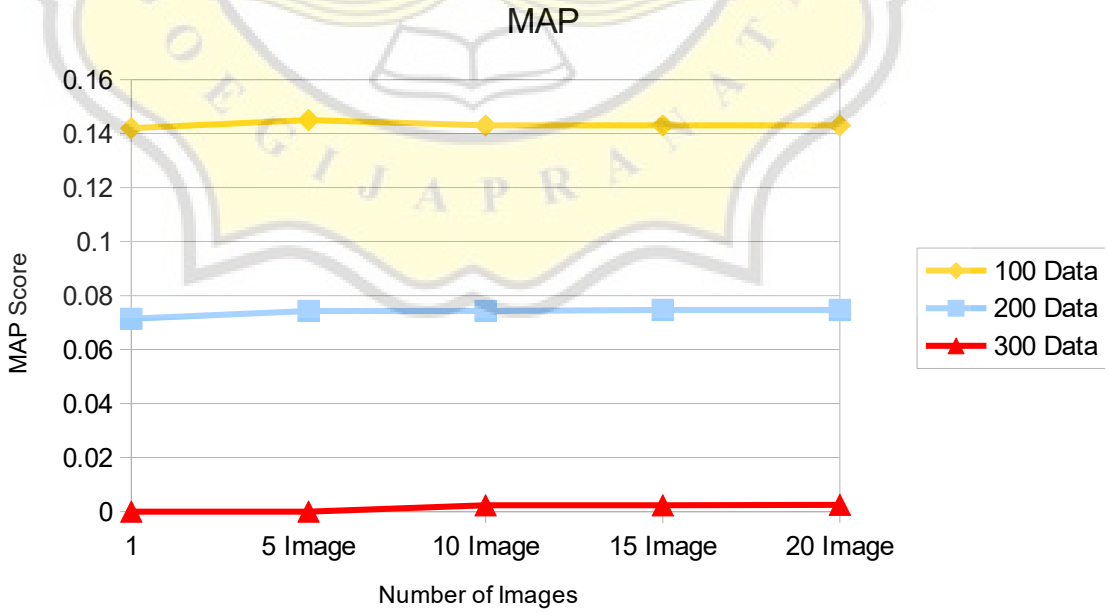

Figure 14 and Figure 15 is a chart of the MAP results. The blue line shows the MAP of the test using CNN that are trained with 200 data in 64x64 resolution; the yellow line shows the MAP results using CNN that are trained with 100 data in 200x100 resolution, and the red line indicates the MAP results using CNN that are trained with 300 data in 64x64 resolution. As shown in Figure 14 and Figure 15, the MAP scores are low there are a few factors that cause this such as :

1. The prediction boxes didn't cover the full area of the ground truth as shown in Figure 14 and Figure 15, and in some cases the prediction boxes are inside the ground truth area. The prediction boxes are set by dividing the height and width of the images, and the current setting of prediction boxes that is shown in Figure 14 and Figure 15 is already set to the largest setting.
2. The prediction selection made by NMS missed the ground truth area.
3. The prediction results made by the filter missed the ground truth area.
4. The prediction label doesn't match the ground truth label. For example the ground truth is 'Segar' while the prediction results is 'Busuk'.

As shown in Figure 14 the CNN trained with 300 data have a lower score than the CNN trained with 200 and 100 data this happened because the CNN that are trained with 300 data give the background of the object with a prediction score more than 95%, this might cause the NMS to remove all the prediction boxes that are on the object because the scores are much lower than the background. As shown in Figure 14 the MAP results with 100 data have a higher score than 200 data. This happened because the 100 data CNN uses higher image resolution in this case 200 x 100. As shown in Figure 15 the MAP results without using NMS have higher scores, this happened because there are more prediction boxes that are intersecting with the ground truth area giving it more score. Based on the results of Figure 15 increasing the resolution of the training image does help improve the accuracy of the object detection.