

CHAPTER 5

IMPLEMENTATION AND TESTING

Implementation

All of the modules used in this research are created by using Python programming language, with the help of matplotlib, Image and numpy library in order to open and view the values of each cells in the image. Some other libraries are also used, and some of the modules used are created based on the journals as reference.

As for the neural network, some additional conditions are created in the neural network and input module order to make them able to save the outputs of its predictions into text file for each image directory.

Fuzzy Edge Extraction

In order to be able to recognize every objects in a number plate, the first thing to do is to obtain the features of each object in the plate. Since the majority of colors found in chassis plates are mostly white, gray, and black, the values of hue and saturation in the image are close to 0, which means the fuzzy hue and saturation map of the image is more likely to appear blank. It may reduce the quality of the extracted plate features. To the point that some objects can be missing after extracting the decision map, and later it will affect the performance of the neural network.

The fuzzy edge image can be used as a better option in extracting the chassis plate features since it can contain most of the edge features in the image with a fairly good quality. Thus, the other fuzzy map extraction and integration processes are skipped. The codes for the fuzzy edge extraction module can be seen in images below. And the fuzzy edge extraction can be seen starting from line 39.

```

1
2
3 import os,sys
4 from PIL import Image as Im
5 import math
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import matplotlib.image as mping
9 import cv2
10 import statistics
11
12 for num in range(1,57):
13     img = mping.imread('dataset-vin/vin%d.jpg' % (num), 0)
14
15     width, height = img.shape[:2]
16
17     FEimg = np.zeros_like(img)
18
19     grayimg = np.zeros_like(img)
20
21     delta1=64
22
23     for x in range(0,width):
24         for y in range(0,height):
25             sample = img[x][y]
26             gray = np.amax(sample)
27             grayimg[x][y] = [gray,gray,gray]
28
29
30     for x in range(1, width-1):
31         for y in range(1, height-1):
32             arr_g = np.array([1])
33             g = grayimg[x][y][0]
34
35             for i in range(x-1, x+2):
36                 for j in range(y-1, y+2):
37                     sample = grayimg[i][j][0]
38                     arr_g = np.append(arr_g, sample)
39
40             gmax = np.amax(arr_g)
41             gmin = np.amin(arr_g)

```

Illustration 5.1. 1 fuzzyedge code1

```

42
43     val1 = float((gmax-gmin)/delta1 )
44     m1 = np.array([1,val1])
45     res1 = np.amin(m1)
46
47     gmean = np.mean(arr_g)
48     val2 = (g-gmean)/delta1
49     m2 = np.array([1,val2])
50     res2 = 1- np.amin(m2)
51
52     arr_f = np.array([res1,res2])
53     fmin = np.amin(arr_f)
54
55     fmin = fmin*255
56     if(g == 0):
57         result=0
58     else:
59         result = (fmin/g)
60         if(result >=1):
61             result=1
62         fuz = int(result*255)
63
64         FEimg[x][y] = [fuz,fuz,fuz]
65
66     ImFE = Im.fromarray(FEimg)
67     ImFE.save('chassis_fuzzy/chassis_fedge/fe_vin%d.jpg' % (num), 0)
68
69     print("done")

```

Illustration 5.1. 2: fuzzyedge code2

As seen on the images 5.1.1 and 5.1.2 above, the gray value of the image is obtained by picking the maximum value of the pixel between the red, green, and blue values (line 26). And the maximum and minimum value are obtained by first converting the image into a 1 dimensional array then pick the max and min value by using numpy. (line 40).

The steps of obtaining the memberships of the image can be seen starting from line 43 to line 50. And the normalization starting from line 51.

After the fuzzy edge image is obtained, the next step is to erase any pixels in the image with the value less than 255, since these pixels will be considered as noise that can increase data complexity and reduce the performance of the neural network.



Illustration 5.1. 3: thresholding

The image above shows the fuzzy edge image before the thresholding(top) and after thresholding (bottom)

```

1  import os,sys
2  import cv2
3  import numpy as np
4  from matplotlib import pyplot as plt
5  import matplotlib.image as mpimg
6
7  DIR = "chassis_fuzzy/chassis_fedge/"
8  for source in os.listdir(DIR):
9      DIR2 = DIR + str(source)
10     img = cv2.imread(DIR2, 0)
11     maxi = np.amax(img)
12     av = np.average(img)
13     thres = 5 + ((maxi+av)/2)
14     ret,thresh1 = cv2.threshold(img,thres,255,cv2.THRESH_BINARY)
15     savedir = "fedge_binary/vin/" + str(source)
16     cv2.imwrite(savedir, thresh1)
17
18  print("done")
19

```

Illustration 5.1. 4: threshold code

As seen in the code above, starting from line 8, the module will loop through every images in the directory, and apply thresholding on them. And after the process is finished, the resulting binary image will be saved in a new directory, as seen on line 18.

Neural Network

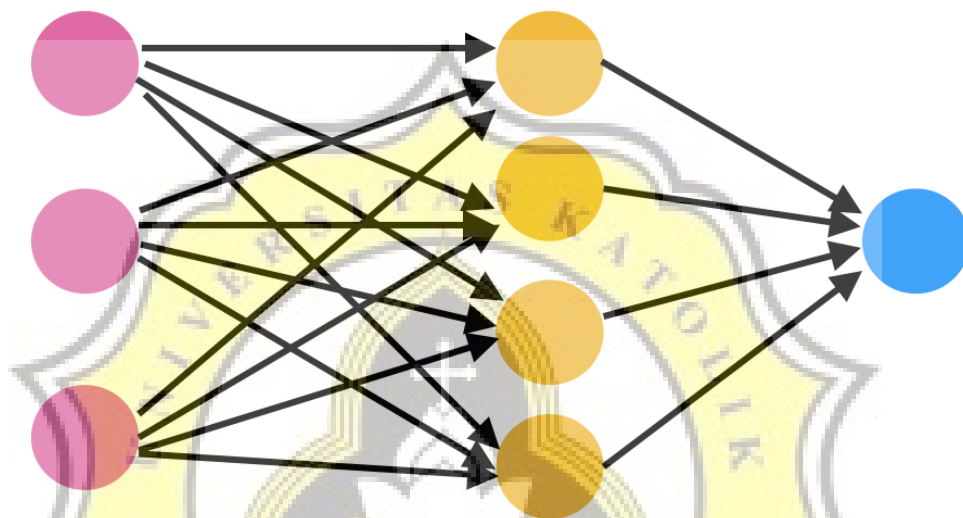


Illustration 5.1. 5: NN illustration

The image above is the illustration of the neural network and it shows three (3) kinds of nodes that makes up its structure. Which consists of:

- 3 input nodes (pink)
- 1 hidden layer that consists of 4 hidden nodes (yellow)
- 1 output node (blue)

The processes of the neural network consists of:

1. Initializing the neural network

This step is only done once when the neural network is initialized.

The neural network will initialize random weights for each nodes.

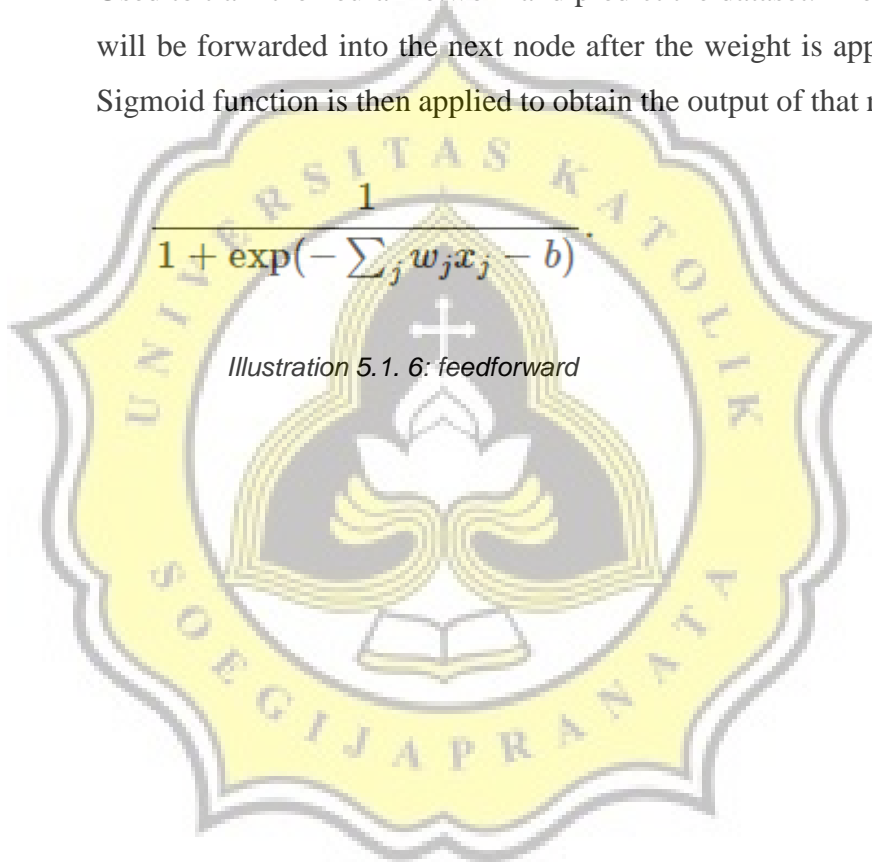
2. Feed forward

Used to train the neural network and predict the dataset. The data will be forwarded into the next node after the weight is applied.

Sigmoid function is then applied to obtain the output of that node.

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

Illustration 5.1. 6: feedforward



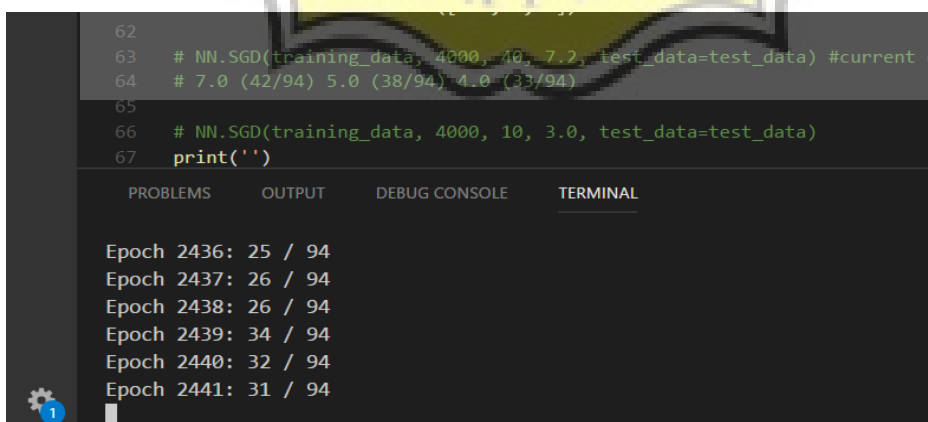
3. Back propagation

Back propagation is the vital part in training the neural network. Since it plays a role in changing the weights based on error in the prediction results. This process starts by calculating the output error by comparing the output with the dataset's label. Sigmoid derivative will then be applied to the error values by using sigmoid prime function. And then added to the weight.

$$\frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}} \right) = y(1-y)$$

Illustration 5.1. 7: backpropagation

In this research, the neural network will be set to having 594 input nodes, which is the same amount as the pixels in the image, 30 hidden nodes, 26 output nodes (for letter or alphabetical input), and 10 output nodes for numeric input. The neural network will be trained for 400 times before beginning to predicting the outputs. And each groups of the inputs (letter and number) will be predicted by using two separate neural networks. The neural network will print the accuracy of each epochs after each training is done.



```

62
63 # NN.SGD(training_data, 4000, 40, 7.2, test_data=test_data) #current c
64 # 7.0 (42/94) 5.0 (38/94) 4.0 (33/94)
65
66 # NN.SGD(training_data, 4000, 10, 3.0, test_data=test_data)
67 print('')

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

Epoch 2436: 25 / 94
Epoch 2437: 26 / 94
Epoch 2438: 26 / 94
Epoch 2439: 34 / 94
Epoch 2440: 32 / 94
Epoch 2441: 31 / 94

```

Illustration 5.1. 8: epoch

Testing

From the total of 92 plate images, 55 fuzzy edge maps are successfully extracted. The images then divided into 2 groups, consisting of 14 images as the testing sample, and the rest becoming the training samples.

The training and testing samples then being segmented and fed into the neural network. With the testing samples containing 49 alphabet images and 94 numeric images, the neural network was able to predict the numeric images with a maximum accuracy of 51%, and 44.8% maximum for the alphabet inputs. With that accuracy, still, many of the plates can be read just partially by the neural network. With only few of the objects in the plate being predicted accurately.

Table 5.2. 1 result_table

| No. | Plate number | Predicted | Accuracy |
|-----|--------------------|-------------------|----------|
| 1 | 35500038a | 10505058g | 4/10 |
| 2 | rhs28054 | bmb24554 | 3/8 |
| 3 | 1407 | 1407 | 4/4 |
| 4 | lfobclt13160004833 | bdbbclb0000453081 | 4/18 |
| 5 | ghd5uc274747 | ghd0bb747574 | 3/12 |
| 6 | an5l6685 | nc5l8881 | 3/8 |
| 7 | an5l24133 | ma0a24120 | 3/9 |
| 8 | 1003021463 | 5000502145 | 2/10 |
| 9 | gt400espada | bn400dgbld | 3/11 |
| 10 | 6857om | 4853gm | 3/6 |
| 11 | jhmugd37700s200000 | gdmbb00720t700505 | 5/17 |
| 12 | 1hgcm56683a086782 | 1hghm20505g005702 | 7/17 |
| 13 | vf23vrhyf42356165 | df20bmhmb42055155 | 7/16 |

As seen in the table above, with only 51% and 44.8% of accuracy for overall samples, the resulting predictions is still far from the expectations. This proves that there should be some problems in the chassis plate that should be considered which can contribute to the quality of the extracted features.

While noises in an image can be solved with various approaches, other problems in the chassis plates such as lack of uniformity needs more consideration. Since it may give a greater impact to the accuracy. Therefore, a new, different approach is needed in this application as compared to the recognition systems commonly used in vehicle license plate.

An additional step is also tested in order to obtain a better image quality, such as applying an edge detection kernel to the fuzzy edge image instead of applying thresholding. But instead of improving the quality of the image, it causes the resulting feature image to drop in quality. Not only that, the plates which the quality can be preserved drops from 55 to 28 plates, from all the 92 plate samples.

The image below shows the examples of the plates that dropped in quality, the top picture shows the original image, the middle for the binarization result of the fuzzy edge images, and the fuzzy edge with edge detection kernel at the bottom.

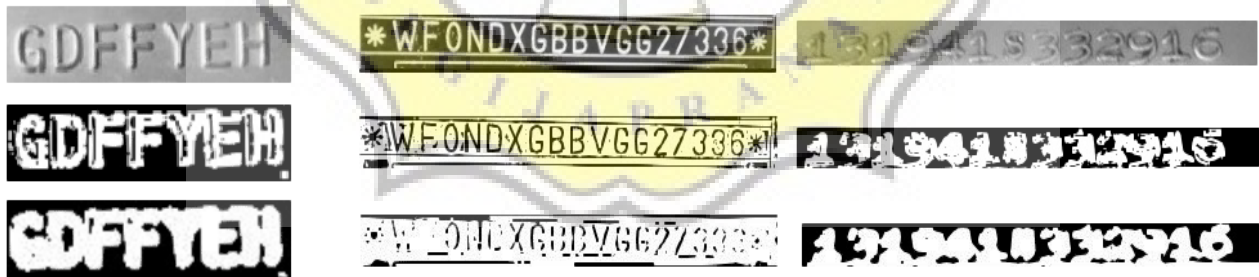


Illustration 5.2. 1: reduced accuracy