Using K-ary Logistic Regression for stop sign detection: A Project Review

Yunhai Han
Department of Mechanical and Aerospace Engineering
University of California, San Diego
y8han@eng.ucsd.edu

***Abstract*—This paper presented approaches using K-ary Logistic Regression for stop sign detection. In this work, I manually label the selected image data into seven different classes. Then, I trained a K-ary Logistic Regression model with all the labelled image data. In order to decrease the loss function, I optimize the parameters, and input other unseen images for classification. Finally, I detect the stop sign from the proceeded images based on the geometric properties.**

***Index Terms*—K-ary Logistic Regression, Machine Learning, optimization, detection algorithm**

I. Introduction

In modern world, the rapid development of the driver assistance system has made driving easier for drivers. However, in order to increase the driving safety and driving experience, we should propose a method which could impart the autonomous driving system the intelligence to make the appropriate reaction based on the surrounding environment [1]. For example, when the car is approaching the crossing road, it should be aware of the positions and even the velocities of pedestrians and other vehicles in front of it. Until they all leave, the car should not start to move. To be more specific, if there exit a stop sign in front of the car, the control system needs to send the stop signal to the motors immediately. Hence, various methods have been proposed to detect stop signs and some of them have been successfully applied into the real-world system [2].

The essential part of the detection algorithm is how to remove the effect of the illumination change and the various point-of-view. If two images of the same object are taken at the same place, one of which is taken during the daytime and the other one is taken in the night. The pixel intensities could be totally different because of the imaging system is not light-invariant. Also, if two images of the same object at the same time are taken from different perspectives, the shape of the object projected in the image plane could not be the same because of the pinhole camera model. In order to handle these two problems, a new method should be proposed on how to learn the variance between different *red*s and the difference between the general red and other colors.

In this project, many different images containing stop sign are given as the train set. Hence, the main purpose of this project is to learn from these training data for a classifier and apply it into any unseen image. In order to obtain a classifier with light invariance, a K-ary Logistic model is proposed for its robustness. Once the color is classified, a stop sign detector would be implemented to locate the stop sign in the image.

II. Problem Formulation

Given a training set of images containing stop signs or other objects like roads or cars, the problem is to build a color classifier that could distinguish the stop-sign red with any other colors, including other red, blue and other colors. Using the K-ary Logistic Regression, segmentation could be implemented by classifying each pixel into seven different classes:

- stop-sign red
- other red
- green
- blue
- yellow
- gray
- white

Then, with the segmented binary images(stopsign-¿255,all other classes-¿0), a detection algorithm is implemented to locate all the objects which satisfy certain requirements. These objects represent the stop signs in each image.

The K-ary Logistic regression with K-classes($y \in 1, ..., K^n$) uses a softmax model with parameter $W \in \mathbb{R}^{K \times d}$. In this project, each pixel intensity could be represented in the RGB color space and there are in total seven different color classes. Hence, the value of K and d could be determined as $K = 7, d = 3$.

The probability function for this model could be described as following:

$$p(\mathbf{y}|X, W) = \prod_{i=1}^{n} \mathbf{e}_{y_i}^T \mathbf{s}(Wx_i) = \prod_{i=1}^{n} \mathbf{e}_{y_i}^T \frac{e^{Wx_i}}{\mathbf{1}^T e^{Wx_i}} \tag{1}$$

In the above equation, e_j is the j-th standard basis vector, for example, e_3 is (0,0,1,0,0,0,0).

$s(\mathbf{z})$ is the *softmax* function, which has the following representation and derivatives:

$$s(\mathbf{z}) = \frac{e^{\mathbf{z}}}{\mathbf{1}^T e^{\mathbf{z}}} \in \mathbb{R}^7 \tag{2}$$

$$\frac{\mathrm{d}s_i}{\mathrm{d}z_j} = \begin{cases} s_i(1 - s_i) & \textbf{if } i = j \\ -s_i s_j & else \end{cases} \tag{3}$$

In the above two equations, **z** is a vector with seven elements in this case and it is the product of W and the ith pixel in the training set.

For the text image, the objective is to find the maximum value of the probability function and classify this pixel x^* with the corresponding label.

$$y = \arg\max_y e_y^T \mathbf{s}(Wx^*) \tag{4}$$

In the above equation, the variable y represents the index of the label in the color list defined before. For example, if y is 1, it means the pixel should be classified as stop sign, which is the first class in the list.

III. Technical Approach

There are in total nearly 200 images in the dataset. Some of them contain one of more stop signs taken from different perspectives. And the pixel intensity for different stop signs are not the same duo to the different lighting conditions. I use 13 images(from 88.jpg to 100.jpg) as the training set and all the left images are separated into the test set.

A. Color Segmentation

In order to generalize the segmentation model, I use more than two classes instead of only stop-sign red and other colors as a whole. Because using more different classes refines the classifier, which make it more robust to different conditions. As I defined in the previous section, there are in total seven classes, I restate them here for the convenience of further explanation.

$y = [\text{stop-sign red, other red, green, blue, yellow, gray, white}]$

I hereby tell the reason why I choose these seven classes. s̈top-sign redr̈epresents the specific red areas on each stop signs. Only if any pixel are classified as this class, they would be fed into the detection algorithm. öther redr̈epresents any other red colors, for example, the red clothing of the pedestrians or the red-painted cars. g̈reenr̈epresents the color of grass or the trees, which frequently appear in these training set. b̈luer̈epresents the color of the sky, which cover the largest area in most images as the background. ÿellowr̈epresents some other signs or any yellow objects. g̈rayr̈epresents the color of road or the buildings, which also cover most areas. ẅhiter̈epresents the color of zebra crossings and clouds. Indeed, we start with the six color classes except öhter, but we found in fact there exits some other red objects. Hence, later we add this color class into the list because we think it may refine the model improving its accuracy.

Each pixel is represented in its **RGB** color space($x_i \in \mathbb{R}^3$). we hand label these training images using *roipoly* function in Python and capture the region of interest(ROI). Then we store these data using Python module *pickle* for the next training. We don't convert the pixels into other color spaces because we think the performance is not bad.

Besides, we acknowledge the kindly collaboration from Zhirui Dai, Qinruo Li, Yuhan Liu, Yaosen Lin, Zhijin Liang for data labeling process.

The model we select here is a discriminative classification model. After training, we could obtain one parameter W, which has the dimension of 7*3. Further, we could find the optimal class for each pixel on order to achieve the largest probability.

To be more specific, the parameter could be computed by the non-linear optimization method via MLE(Maximum likelihood estimation). We have to compute the gradient of the data log-likelihood:

$$\begin{aligned} W_{\mathbf{MLE}}^{(t+1)} &= W_{\mathbf{MLE}}^{(t)} + \alpha(\bigtriangledown_W [\log p(\mathbf{y}|X,W)]|_{W=W_{\mathbf{MLE}}^{(t)}}) \\ &= W_{\mathbf{MLE}}^{(t)} + \alpha(\sum_{i=1}^{n}(\mathbf{e}_{y_i} - \mathbf{s}(W_{\mathbf{MLE}}^{(t)}\mathbf{x}_i))\mathbf{x}_i^{\mathrm{T}}) \end{aligned} \tag{5}$$

In the above equation, α is a parameter which we have to tune for many times. The value of α is mainly determined by the number of training data. In this case, as you can imagine, we could extract a great amount of pixels from each image. Hence, finally, I set α as a small number:0.00001.

The computed parameter W are given below:

W			
color channel	red(**R**)	blue(**B**)	green(**G**)
stop-sign red	7.97998502	-5.52467351	-3.12123777
other red	-2.29260393	-2.99410614	-2.45656097
green	2.64526509	11.27600299	-9.97913918
blue	-12.05650543	-0.04008605	18.02165188
yellow	0.90202575	-1.19887349	-2.44773507
gray	2.19011698	-1.6565441	-0.59845608
white	0.63171651	0.1382803	0.58147718

To classify a pixel x_i, the log probability could be determined as below:

$$y = \arg\max_y e_y^{\mathbf{T}} \mathbf{s}(Wx^*) \tag{6}$$

For example, if a pixel value is represented as (x_1, x_2, x_3), the prediction value for stop-sign class could be computed as

$$P_{\textbf{ss-red}} = 7.97998502x_1 - 5.52467351x_2 - 3.12123777x_3$$

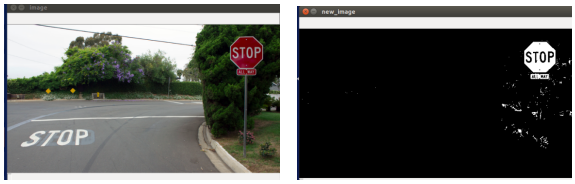
and the prediction value for blue class could be computed as

$$P_{\textbf{blue}} = -12.05650543x_1 - 0.04008605x_2 + 18.02165188x_3$$

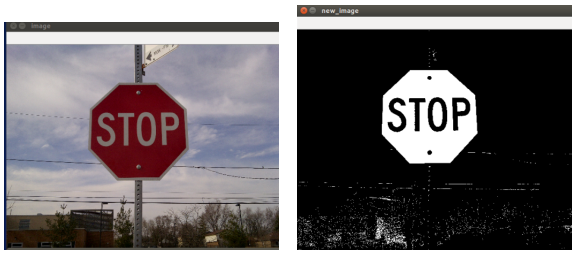
We could simply compare the different prediction values for each class, and then we classify the pixel into the class corresponding to the largest prediction value.

For every pixel in each image, we implement this algorithm to obtain a color segmented image. This image is a binary image with pixels assigned a value of 255 if it is classified stop-sign red and 0 for all other classes. An example of the binary mask image with its original **RGB** representation are displayed in Figure 1.

(a) RGB image (b) Binary image



(c) RGB image (d) Binary image



(e) RGB image (f) Binary image

Fig. 1. Example color segmented image and binary mask image

B. Stop sign Detection

The binary image mask obtained from the previous operation contains some random noise, which may result in poor performance for the detection algorithm. Hence, we first have to find a way to overcome this problem. Hence, additional pre-processing operations are needed to remove the random noise, which could be considered false positive results of the segmentation algorithm. Morphological methods are the first method that come into our mind. The opening operation erodes an image and then dilates the eroded image, using the same structuring element for both operations. Morphological opening is useful for removing small objects from an image while preserving the shape and size of larger objects in the image. In this case, it also shows great effect of removing random noisy pixel. In python-opencv2, all the operations have been encapsulated in the function *cv2.morphologyEx*. By simply calling the function, we could compare the results before and after the opening operation.

$$kernel = cv2.get..Element(cv2.MORPH_RECT, (3,3))$$
$$closed = cv2.mor.Ex(img, cv2.MORPH_OPEN, kernel)$$

In the above expressions, due to the limit space, the name of the two function are not expressed completely. The first function is $cv2.getStructuringElement$ and the second function is $cv2.morphologyEx$.

(a) Before Opening (b) After Opening

Fig. 2. Results before and after Opening

From Figure 2, you can that the small white holes could be removed smoothly. And the shape of the stop-sign almost remains the same.

After removing the random noisy pixels, we now could find all the instances for further distinguish. These operations could be done with the opencv2 function $cv2.findcontours$. It would return all the points on the contours of each instance. Based on the prior knowledge that the area of stop sign should be too small, we could simply set a threshold to filter all the instances whose area is smaller than the threshold. After some test, I set the threshold as 700. With another opencv2 function $cv2.boundingRect$, we could simply draw a bounding box containing each instances. Because the shape of all the stop signs is octagon and its projection on the image plane would not change drastically, the ratio of the width and height of the bounding box would not be too large or too small. From the above two criteria, two simple requirements must be satisfied in order to be considered as the candidates of stop signs:

- area > 700
- $\frac{width}{height} > 0.6$ and $\frac{width}{height} < 1.4$

The area of each contour could be easily obtained from the function $cv2.contourArea$ and the width and height are two member variables of each bounding box.

Besides these two simple requirements, I figure out another criterion based on its geometry property. In Figure 3, you can see there are four right triangles at the four corners of the bounding box and in most cases, we could consider these triangles nearly Isosceles. In other words, the lengths of all the **X** segments and of all the **Y** segments should be nearly the same. In the image plane, it could mean the sum of pixels should be nearly the same. Suppose m represents the sum of **X** pixels and n represents the sum of **Y** pixels, the following requirement must be satisfied:

- $min = \min(n, m)$ $max = \max(n, m)$
- $\frac{max}{min} < 1.5$

The threshold 1.5 is obtained from many tests.

With the stop sign detected, the coordinates of the bottom-left and top-right corners of the bounding box could be easily obtained from the four member variables of it:

- x-> the x-coordinate of the top-left corner
- y-> the y-coordinate of the top-left corner
- w-> the width of the bounding box
- h-> the height of the bounding box

Fig. 3. The illustration of its geometry property

What we have to pay attention is the origin of image in opencv2 is different from the the required one. In opencv2, the origin is the top-left corner of the image but the required origin is the bottom-left corner of the image. Hence, we need to convert the y coordinate.

- $y_{required}$ = height of the image - y

IV. Results and Discussion

Using 13 images from training and 30 images for validation, the accuracy is given below:

$$Accuracy = \frac{26}{30} = 86.7\%$$

However, for the test images on the autograder, I was only able to detect the stop-signs in two of them. I have tried many methods, but they didn't work. Right now, I think the main reason is that during training process, the volume training data for each class is unbalanced. For example, I feed much more pixels classified as gray than pixels classified as stop-sign red, because the areas of road are much larger than the areas of stop sign. For Logistics regression, data unbalance may pose a threat to erode its generalization ability. However, it works well on the validation images, which makes me feel a little confused. Maybe the main reason is that the test images on the autograder are in extremely bad lighting conditions. And the best way to solve this problem is to feed more balanced date into the training model and make sure that the training images are taken with different lighting conditions.

A. Stop sign Detection Results

Using the validation images set, the good cases far outnumber the bad cases. You can see the results shown starting from the next page.

V. Acknowledgment

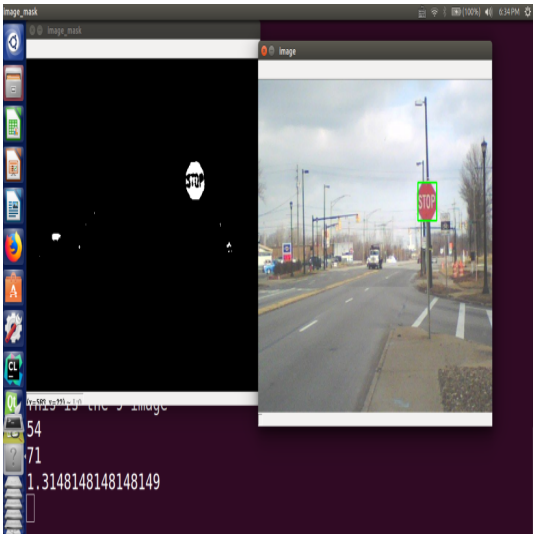
I am grateful to the whole Python communities for providing us with so many powerful tools.

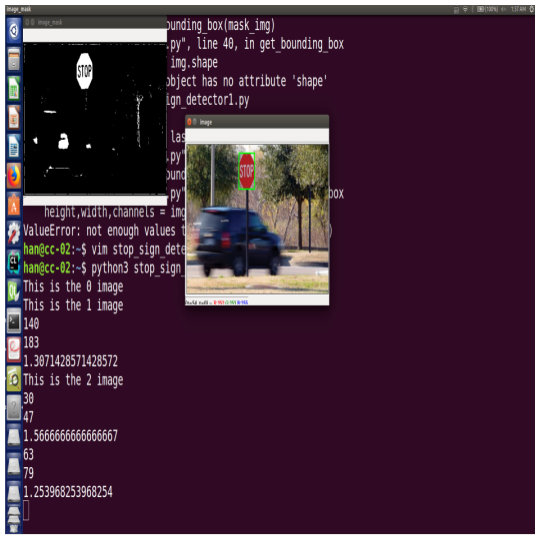
References

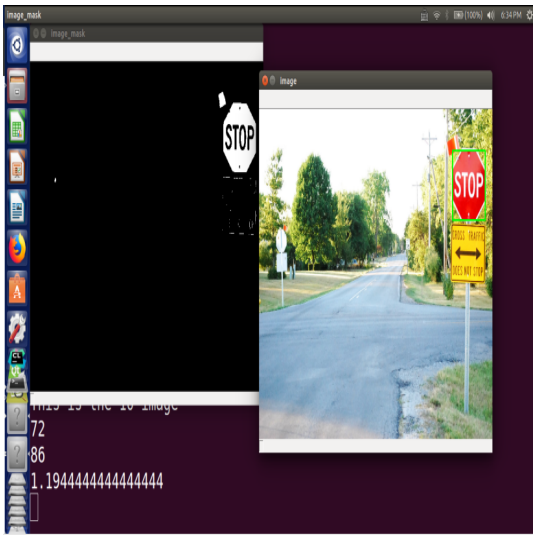
[1] Liu, Henry Ran, Bin. (2001). Vision-Based Stop Sign Detection and Recognition System for Intelligent Vehicles. Transportation Research Record. 1748. 161-166. 10.3141/1748-20.

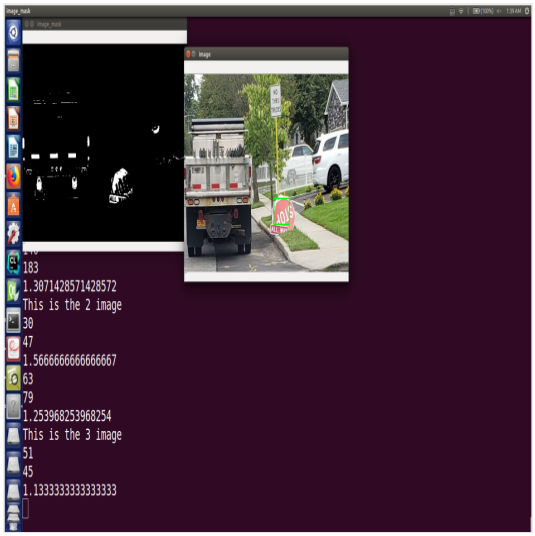
[2] M. A. A. Sheikh, A. Kole and T. Maity, "Traffic sign detection and classification using colour feature and neural network," 2016 International Conference on Intelligent Control Power and Instrumentation (ICICPI), Kolkata, 2016, pp. 307-311.

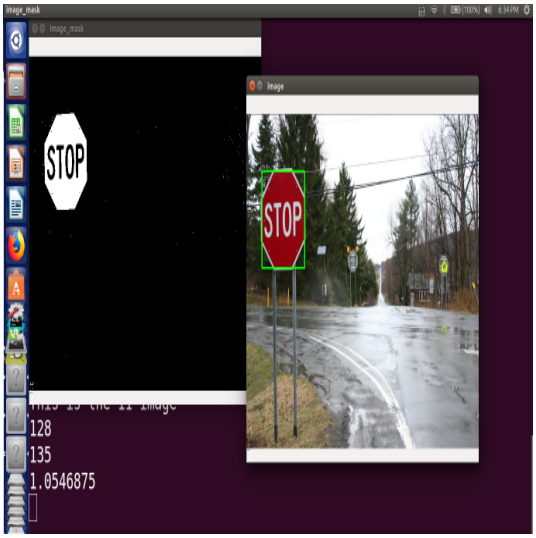

t has no attribute 'shape'
box = my_detector.get_boundin
File "stop_sign_detector1.py",
height,width,channels = img.shape
ValueError: not enough values to unpack (expected 3, got 2)
han@cc-02:~$ vim stop_sign_detector1.py
han@cc-02:~$ python3 stop_sign_detector1.py
This is the 0 image
This is the 1 image
140
183
1.3071428571428572

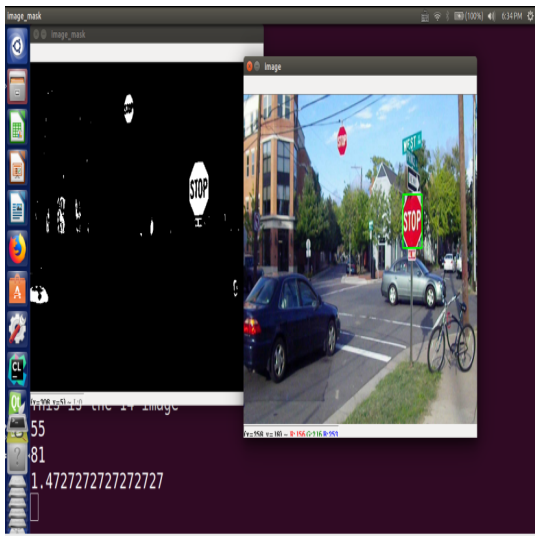

54
71
1.3148148148148149

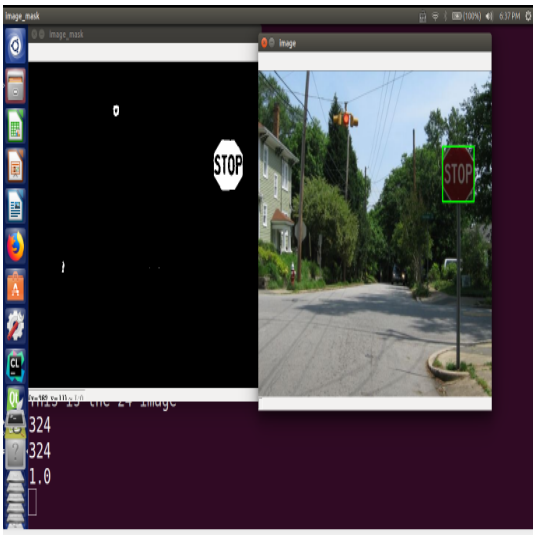

ounding_box(mask_img)
.py", line 40, in get_bounding_box
img.shape
bject has no attribute 'shape'
gn_detector1.py
box
height,width,channels = ima
ValueError: not enough values t
han@cc-02:~$ vim stop_sign_dete
han@cc-02:~$ python3 stop_sign
This is the 0 image
This is the 1 image
140
183
1.3071428571428572
This is the 2 image
30
47
1.5666666666666667
63
79
1.253968253968254

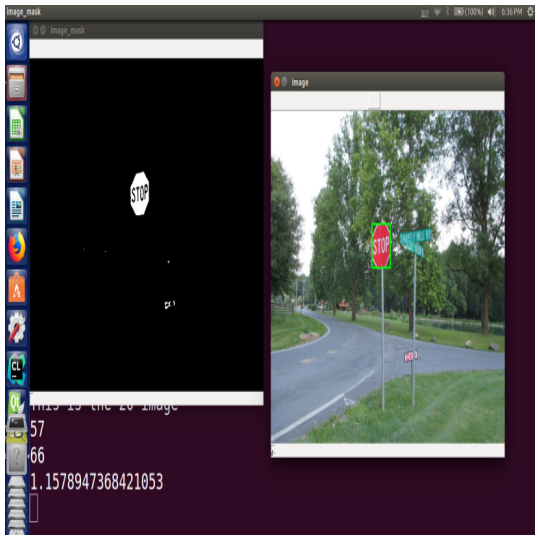

72
86
1.1944444444444444

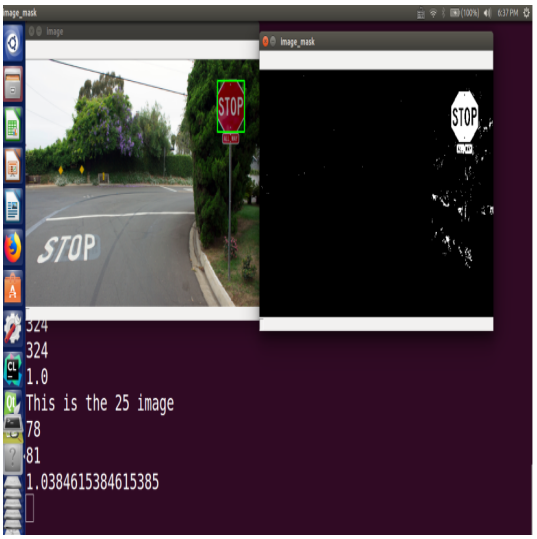

183
1.3071428571428572
This is the 2 image
30
47
1.5666666666666667
63
79
1.253968253968254
This is the 3 image
51
45
1.1333333333333333

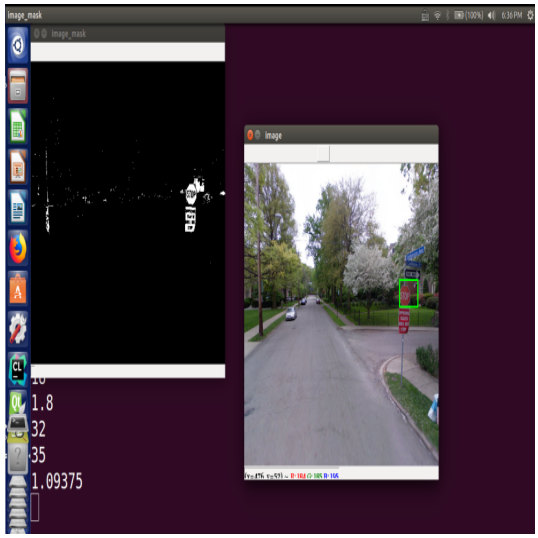

128
135
1.0546875

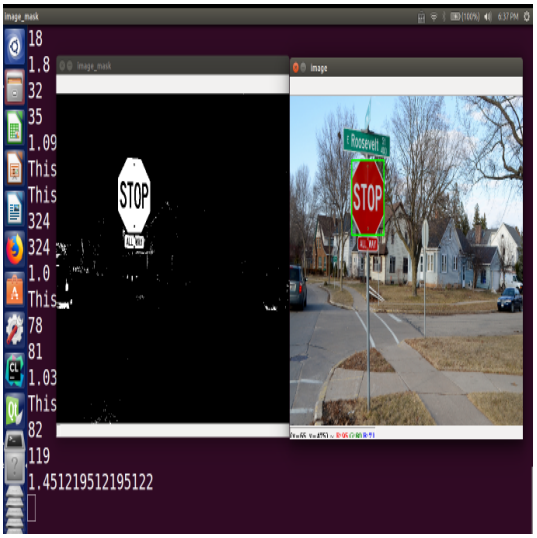

image_mask
image
STOP
55
81
1.4727272727272727


image_mask
image
STOP
324
324
1.0


image_mask
image
STOP
57
66
1.1578947368421053


image_mask
image
STOP
324
324
1.0
This is the 25 image
78
81
1.0384615384615385


image_mask
image
1.8
32
35
1.09375


image_mask
image
STOP
18
1.8
32
35
1.09
This
This
324
324
1.0
78
81
1.03
82
119
1.451219512195122

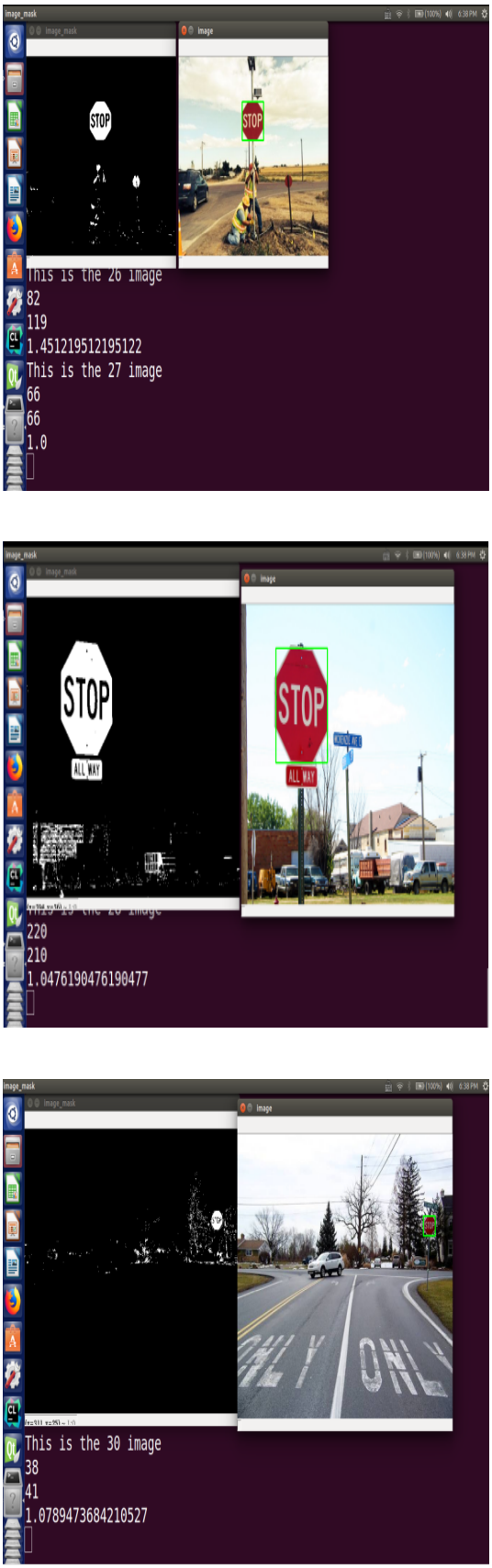


Fig. 4. Detection Results