ECE 276A Project 1
Stop sign detection using Gaussian Discriminate Analysis

Shiladitya Biswas
Dept. of Electrical and Computer Engineering
Universrity of California,San Diego)
California, USA

Fig. 1. STOP sign

I. Introduction

Traffic Sign board detection is one of the fundamental task that enable self driven cars to follow traffic rules successfully. Hence it is important to develop computer algorithms that can perform Traffic Sign board detection.In this paper I proposed a Stop sign board detector algorithm using Gaussian Discriminate analysis. The algorithm takes in images in BGR format and draws a bounding box on it wherever Stop signs are detected. The mathematical approach of the algorithm is explained in the upcoming sections.

II. Problem Formulation

Our problem statement is to identify/detect all the Stop signs in a given image. In order to do that I identified some of the key features of a Stop sign. Looking at figure 1 one can observe two striking properties. Firstly the stop sign boards are predominantly red in colour with "STOP" written in white on it. Secondly, they have a specific shape i.e. all the stop signs have regular Octagon in shape. Based on the above observations here is my strategy to detect stop signs:

I would like to thank Saurabh Mirani, for letting me use his BGR data. This helped me decide as to which color space should I work with. Eventually, I collected data in BRG Format and worked in BGR Color space. Added to that we also collaborated on analyzing all the test cases on the autograder and discussed the logic behind detecting the Stop signs in each image.

(A) Color Segmentation: Implement a red pixel classifier that detects red regions in a given image.
(B) Shape Detection: Perform a shape study of the detected red regions and match it with the shape properties of the stop sign as discussed above. Draw a bounding rectangle around the regions which satisfy the shape properties.

A. Color Segmentation

Every image is a collection of some definite number of pixels, each pixel is a 3D vector. The elements on the 3D vector represent the intensity (0-255) of Red, Green and Blue channel. Therefore each pixel is a data point in the R^3 vector space. Since the color of a pixel is governed by the intensity values in the R,G and B channel, the reddish pixels forms a cluster in one particular volume of R^3. Similarly other not red pixels forms a cluster at a different location in the colour space. Now, in order to detect stop signs we need to segment this 3D color space into discrete volumes of red and not red regions. This can be done using supervised learning where we try to fit two (one for red and another for not_red pixels) tri-variate Probability Density functions(PDF) on the clustered color space. In other words we try learn the parameters of the PDF for each of the classes. This process is generally referred to as training the classifier.

Let each 3D color pixel be represented by vector $\mathbf{x}$, where $\mathbf{x} \in R^3$ and let each discrete volume be represented by label y, where $y \in \{0, 1\}$. Here 0 represents the Not Red class and 1 represents the Red class. The mathematical objective is to learn a function $F : R^3 \longrightarrow R$ for each class that can assign a label in y to a given data point x, either from the training dataset or from an unseen test set, with some probability. Since I am using generative method to classify pixels, my function F is given as follow:

$$F(\mathbf{x}) = argmax_y \; p(y, \mathbf{x}; \omega, \Theta) = p(\mathbf{x}|y; \omega) \times p(y|\Theta) \quad (1)$$

where $p(y, \mathbf{x}; \omega)$= Joint PDF of y and $\mathbf{x}$ parameterized by ω and Θ.

Let there be total N_{red} data points in red class i.e. y=1 and N_{notred} data points in not red class i.e. y=0. And assuming that all the data points '$\mathbf{x}$' are Independent and Identically Distributed (IID) and are clubbed into a single large matrix $\mathbf{X}$, such that $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N]$, equation 1 can be rewritten

for each class as follows, the resulting function is often called the Likelihood function:

$$p(y, \mathbf{X}; \omega, \Theta) \propto \prod_{i=1}^{N} F(\mathbf{x}_i) \tag{2}$$

$$p(y, \mathbf{X}; \omega, \Theta) \propto argmax_y \prod_{i=1}^{N} p(\mathbf{x}_i | y = 1; \omega) \times p(y_i | \Theta) \tag{3}$$

The log of the likelihood functions/equations 3 is written for both the red and not red class. Both the equations are then maximized w.r.t the parameter vector ω and Θ. This is also termed as the Maximum Likelihood Estimate (MLE) of the parameter vector ω and Θ for each class. We define the MLEs as (ω_{red},Θ) and (ω_{Not_red},Θ) for the red and not red class respectively. Once the classifier is trained and we are given a new pixel vector **x_new**, we can determine as to which class does it belong to.
If the probability

$$p(\mathbf{x}_{new} | y = 1; \omega_{red}, \Theta_{red}) > \tag{4}$$

$$p(\mathbf{x}_{new} | y = 0; \omega_{Not\_red}, \Theta_{Not\_red}) \tag{5}$$

then x_{new} belongs to the red pixel class, else it belongs to Not Red pixel class. So now, given any image, we can check it pixel by pixel and accordingly classify the complete image into read and not red regions.

B. Shape Detection

Once red coloured regions in an image are segmented, we can calculate several properties of these contours. These properties include area, area ratio, aspect ratio etc. We can also approximate a polygon around the contour and count the number of edges. By fixing appropriate bounds on these parameter we can properly detect a stop sign. In ideal case if our color segmentation model works well and the stop sign in the image is very prominent, then we can easily fit a 8 sided polygon to the contour and detect it as stop sign. But this is not the case in general, hence during implementation we need to vary the bounds on these properties and accordingly find out what fits best for the general situation. For example, if we are in Yuma, Arizona where the weather is very bright throughout the year, lighting conditions are good, so it is expected that our image segmentation classifier will work very well there, and hence we will get well defined contours, hence stop sign detection will be relatively easy. But this might not be the case in Michagan, where weather is gloomy.

III. Technical Approach

The Gaussian Discriminate model was used to build the classifier. For such a classifier

$$p(\mathbf{y}, \mathbf{X} | \omega, \Theta) = p(\mathbf{y} | \Theta) p((X) | \mathbf{y}, \omega) = \prod_{i=1}^{N} p(y_i | \Theta) p(x_i | y_i, \omega)$$

$$p(y_i | \Theta) := \prod_{k=0}^{1} \Theta_k^{1\{y_i = k\}} \ \& \ p(\mathbf{x}_i | y_i = k, \omega) = \phi(\mathbf{x}_i; \mu_k, \Sigma_k)$$

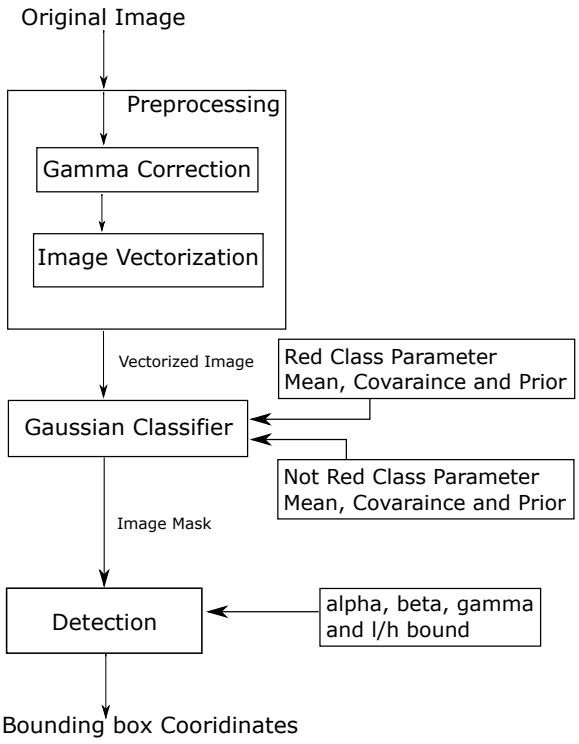


Fig. 2. Stop Sign Detector flow chart

where, k=0,1 represents Red and Not Red class, $\omega := \{\mu_k, \Sigma_k\}$ and MLE estimates of Θ and ω are obtained from:

$$\Theta_k^{MLE} = \frac{1}{N} \sum_{i=1}^{N} 1\{y_i = k\}$$

$$\mu_k^{MLE} = \frac{\sum_{i=1}^{N} \mathbf{x} i 1\{y_i = k\}}{\sum_{i=1}^{N} 1\{y_i = k\}}$$

$$\Sigma_k^{MLE} = \frac{\sum_{i=1}^{N} (\mathbf{x} i - \mu_k^{MLE})(\mathbf{x}_i - \mu_k^{MLE})^T 1\{y_i = k\}}{\sum_{i=1}^{N} 1\{y_i = k\}}$$

In order to train the classifier (i.e. determine the MLE estimates of each class) I extracted data point; both red and not-red from the dataset provided to us. The dataset has total 200 images, 100 of which are with stop signs and rest without stop signs. The dataset was divided into training and validation set. I used 180 (80 with stop sign + 100 without stop sign) as my training set and the remaining 20 images as my validation set. The **roipoly.py** function was used to manually select/extract red pixels from all the images in the training set and stored in a Numpy file named **"STOP.npy"** in BGR format. The same procedure was repeated for not red pixels and the data was stored in another Numpy file named **"Not_STOP.npy"**

The complete algorithm for detecting stop signs can be summarized by the flow chart in figure 2. In short, the algorithm takes in a $l \times b$ still image as input prepossesses it and reshapes it into $(l * b) \times 1$ vector sends it to the classifier. The

Classifier takes in the preprocessed image and segments the image as per the rule in equation 4 and 5 and outputs a masked image. The masked image has '1' values where red pixels were detected and '0' elsewhere. The Masked image is sent into the detection module, which performs a geometric test by looks into several geometric properties of each contour to determine if it is a STOP sign or not. Finally, it draws a bounding box on those contours which has passed the geometric property test and also outputs the coordinates of the bounding boxes.

A. *Prepossessing*

Looking at the training set provided to us, it was observed that some stop signs in an image were very dark. As a result the position of theses pixels are far away from the mean of the red class Gaussian function that we fitted. In other words these dark pixels are very close to the decision boundary of the classifier and hence are at a risk of being wrongly classified as not red. Thus preprocessing the image is very essential. In order to counter this problem **Gamma correction** was performed on the input image before giving it to the classifier. The original image and the image after gamma correction is shown in figure 3 and 4. Figure 5 and 6 shows the segmentation mask given by the classifier when input image is "not gamma corrected" and "gamma corrected" respectively. Clearly we see a lot of improvement in the performance of the classifier.

Furthermore, from previous experience it has been observed that using nested for-loop to access each 3D pixel and then classify them has a **time complexity** $O(n^2)$, Big Oh of n^2. Thus it is a very time consuming process and will lead to Autograder "Timeout". Thus, right after applying gamma correction, the 3D image array was vectorized/reshaped into a $3 \times (l * b)$ array. This array was then fed to the classifier.

B. *Colour Segmentation*

This is one of the most important module of this system. This is where equation 4,5 comes into play. Each column from the $3 \times (l * b)$ array is plugged into equation 4 and checked. If $p(\mathbf{X}|y = 1; \omega_{red}, \Theta_{red}) > p(\mathbf{X}|y = 0; \omega_{Not\_red}, \Theta_{Not\_red})$, (Here X is 2D array of size $3 \times (l * b)$, each column representing a pixel) then the pixel is red else the pixel is not-red. Since X is a vector, both the LHS and RHS of the above inequality is also a vector hence the comparison is no longer time consuming. I used "numpy.greater(A,B)" function for this purpose. This function returns a array with "**1**" at i^{th} location if i^{th} element of A is greater than i^{th} Element of B, and "**0**" elsewhere. The resulting vector is again reshaped backed to the original image dimension i.e. $l \times b \times 3$. An example of original image and its mask is shown in figure 7

C. *Detection*

This module takes in the masked array and the original image as input from the segmentation module. The mask is then multiplied element by element with all the channels of the original image to reveal the red areas only. This image is then converted to gray scale binary image. This image is

Fig. 3. Original Image

Fig. 4. Image after Gamma Correction

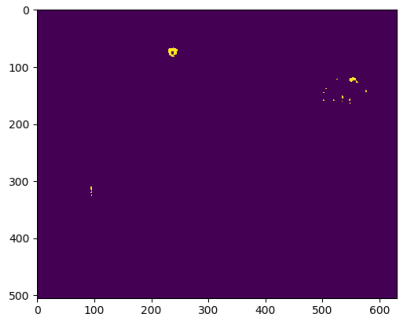


Fig. 5. Segmentation mask created when input is original image

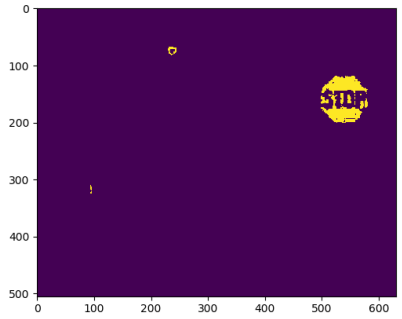


Fig. 6. Segmentation mask created when input is gamma corrected image image

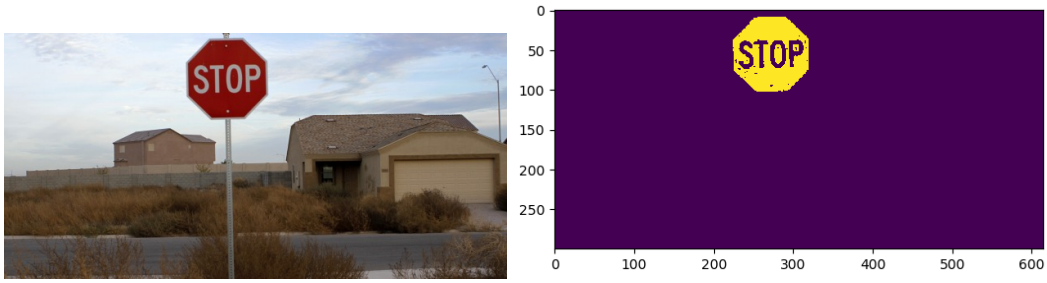


Fig. 7. Image and its mask returned by the classifier

given as input to the cv2.FindContour function. This function returns a list of all the contours (i.e. all the red regions, identified classified by my classifier in the previous module) in the image. Generally it is a trend to filter out noise from the masked array before using it in the detection module, but in my case it worked pretty well without using the filter command.

Once the contours were all detected, certain properties of the contours were evaluated. These properties are:

1) **Contour Area** α: Noisy images after segmentation can have several small specs of pixels wrongly classified as red pixels here and there on the image. These specs will obviously not be a stop sign. On the other hand the STOP sign board is ought to take some considerable area of in the whole image. Thus we look into the area of all the detected contours in a the image and discard those contours whose area is less than some threshold.
2) **Contour area ratio** β: The above filtering method might not be suitable for small images. In a small image the stop sign area will obviously be small. On using the above method, we run into the risk of discarding small stop signs. Thus we also look into the area ratio i.e. (contour area / Image area). If the area ratio is higher than some threshold then it is a stop sign.
3) **No of edges** γ: We try to fit a polygon around the contour and then count the number of edges this polygon has. Ideally a stop sign board is octagon in shape so we expect to have a polygon of exactly 8 edges. But this is not the case in reality. If the STOP is far in the image then it looks circular in shape. Similarly, if the stop sign is behind some obstacle like a tree branch etc then the approximated polygon might have less than 8 edges. Hence we put some range on γ we detect stop signs
4) **Height/width ratio** (h/w): Here w is the width and h is the height of a rectangle that can bound each contour. This parameter is easily made available by the cv2.boundingRect function. This function returns w, h of the bounding rectangle and also the x,y coordinates of the top left corner of the rectangle. This is a very important parameter because it allows us to filter out arbitrary and irregular contours which might have both area and area ratio in range. Ideally a stop sign has $h/w = 1$, but again this might not appear to be the case in segmented image. Hence again we keep a bound on h/w ratio keep it between slightly greater and lesser than 1.

Now, we look into the test cases in the autograder as well as the training and validation set to determine the correct bound on α, β, γ and h/w. Once the stop signs are detected we draw a bounding box around it and append a list variable "boxes" with the coordinates of bottom left corner $(x, l-(y+h))$ and top right $(x+w, l-y)$ corner of the bounding box. In doing so I was able to pass 12 out of 16 test cases on the autograder.

IV. Results and Discussions

The classifier performed as expected on majority of the training and validation set images. The MLE parameters of both the classes were obtained as follows:

$$\theta_{red}^{MLE} = 0.012098 \qquad \theta_{Not\_red}^{MLE} = 0.987901$$

$$\mu_{red}^{MLE} = [53.85255293, 46.50559933, 160.38144411]$$

$$\mu_{Not\_red}^{MLE} = [134.67835746, 127.82135958, 117.79590885]$$

$$\Sigma_{red}^{MLE} = \begin{pmatrix} 3635.2179 & 3850.2715 & 1267.4629 \\ 3850.2715 & 4325.5668 & 1268.4174 \\ 1267.4629 & 1268.4174 & 3123.2352 \end{pmatrix}$$

$$\Sigma_{Not\_red}^{MLE} = \begin{pmatrix} 5152.3090 & 3929.185 & 3032.51 \\ 3929.185 & 3596.1 & 3231.5266 \\ 3032.51 & 3231.5266 & 3489.2491 \end{pmatrix}$$

Some examples of successful stop sign detection are shown in figure 8,9,10,11,12,13,. The coordinates of the bounding boxes are shown on top left corner of the image with bounding rectangle

Some examples of unsuccessful detection are shown in figure 14, 15,16, 17,18. The reason as to why they were unsuccessful are written below each one of them.

Therefore, in general we observe that, the Gaussian discriminate model classifier performs pretty well to segment a given image into red and not red region. This is endorsed by the fact that the segmentation masks obtained for most of the images are correct i.e. we get a bit mask with '1' at locations where we have red pixels and '0' otherwise. But the shape detection algorithm is not very robust. This is

Fig. 8. Image number 91 output and its segmentation Mask. Although the stop signs are very far, thus they are approximated with polygon of sides ¿ 8. Yet my algorithm is able to draw a bounding box on both of them. Thus my algorithm is robust

Fig. 9. Image number 79 output and its segmentation Mask. This was a success because of (i)Gamma correction and (ii) The background of the image has less red areas, hence we get a clean segmentation mask

because the shape detection algo works on the masked image directly. It only considers the contour properties like Area, Area ratio, no of side of an approximate polygon that can enclose the contour, h/w etc into consideration. Hence it can't distinguish between a stop sign and other objects. For example, if we have a simple red octagon (without STOP written on it) in the image, the classifier will classify it as red, but the detection algo will wrongly detect it as a stop sign, since the image has all the properties of a stop sign being octagon in shape. This is what makes Human Intelligence different from computer algorithms. Maybe Optical Character recognition might work well here. Furthermore, the prerpocessing i.e. gamma correction I did on the image before sending it to the classifier worked very well for dark images, as discussed previously and shown in figure. This highlights the fact that preprocessing is an essential part for Supervised learning algorithms.

Fig. 10. Image number 89 output and its segmentation Mask. Although the stand is also red, it has a very high h/w ratio, thus the also does consider it to be a stop sign.

Fig. 11. Image number 15 output and its segmentation Mask

Fig. 12. Image number 9 output and its segmentation Mask

Fig. 13. Image number 9 output and its segmentation Mask

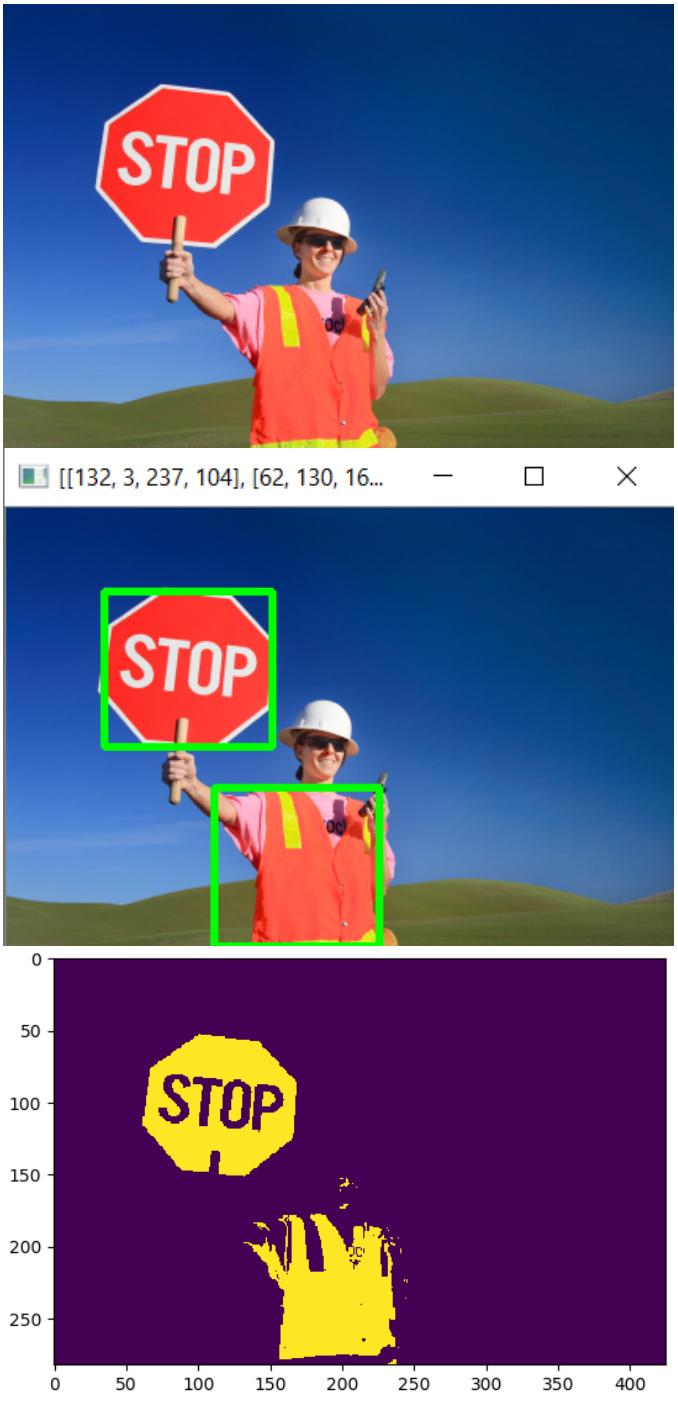


Fig. 14. Image number 6 output and its segmentation Mask. We observe that our pixel classifier has worked classifier, but it can't distinguish between red and orange colour properly. It assigns the pixels on the shirt to red class too. Which is partly true, the shirt pixels are somewhat red. Hence we get such a segmentation mask. Since the contour area of the STOP sign and the shirt are nearly equal, the algorithm fits a bounding box on the shirt as well. Hence the detection fails

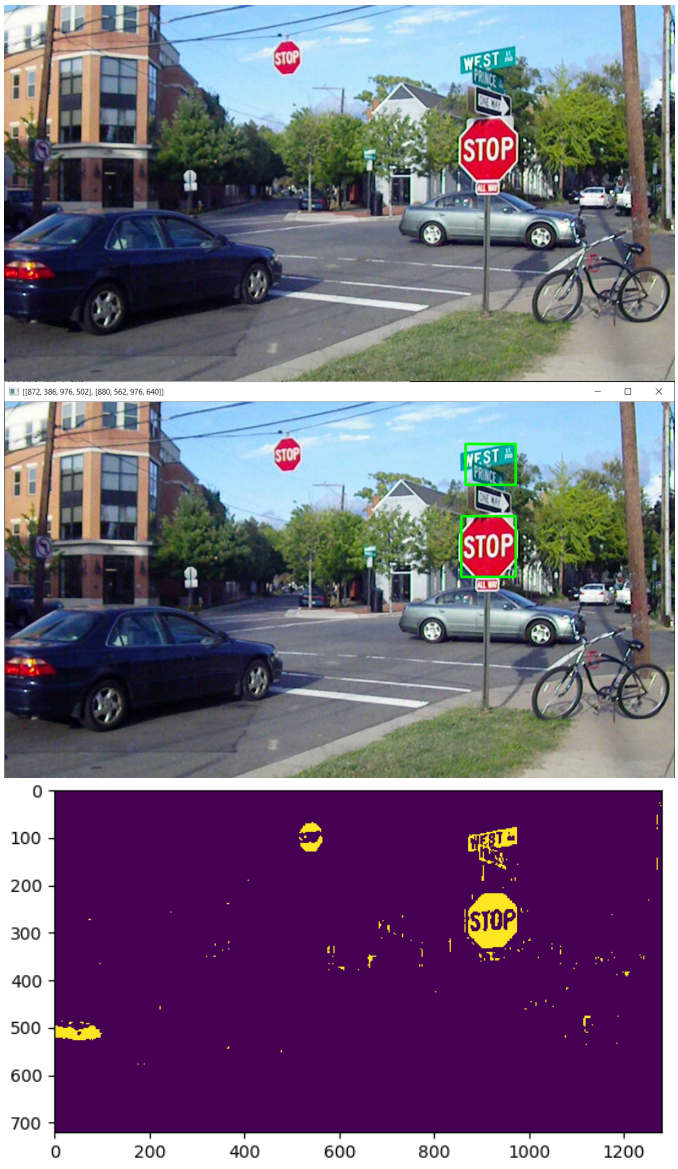


Fig. 15. Image number 9 output and its segmentation Mask. In this image the classifier successfully segments the red pixel in the image. The smaller stop sign in air couldn't pass the bound we set on α, β and γ thus it is not detected as stop signs.

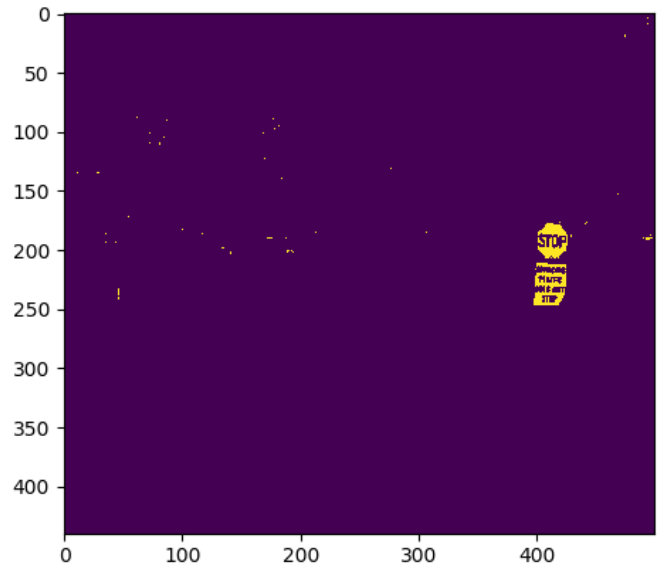
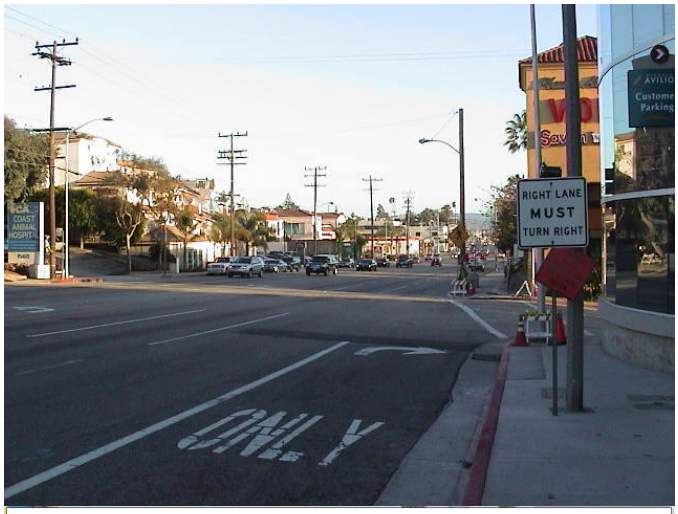
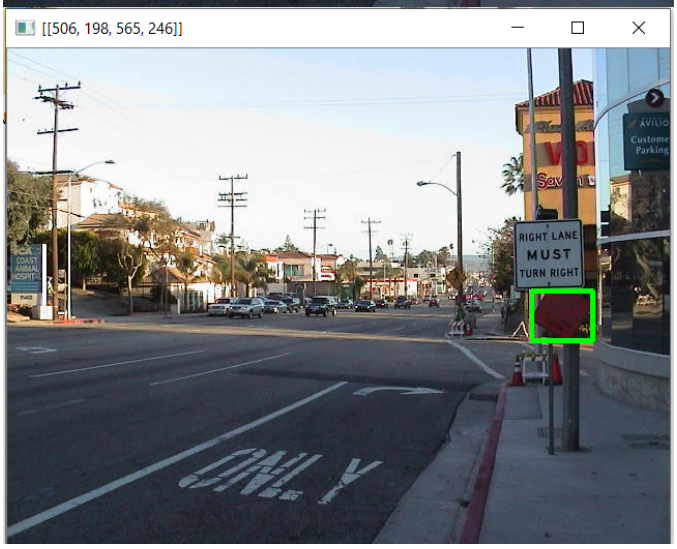


Fig. 16. Image number 38 output and its segmentation Mask. We observe that the red board below the stop sign is also detected by the classifier, this implies that that classifier performs very well. But then the shape detection algo thinks both the contours are stop signs, since both the contours are of comparable area. Thus both the contours are detected as STOP signs





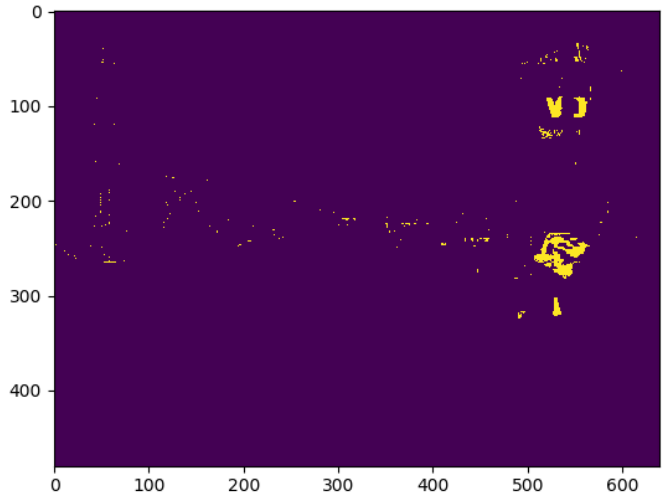


Fig. 17. Image number 13 output and its segmentation Mask. Again the classifier here performs somewhat well. This is partly due to the fact that we perform gamma correction on the image before classifying its pixels. The problem here is that the contour formed here has properties like Area, area ratio etc similar to a stop sign. Hence the detection algorithm forms a bounding box around the deep red colored sign board, and the detection fails completely

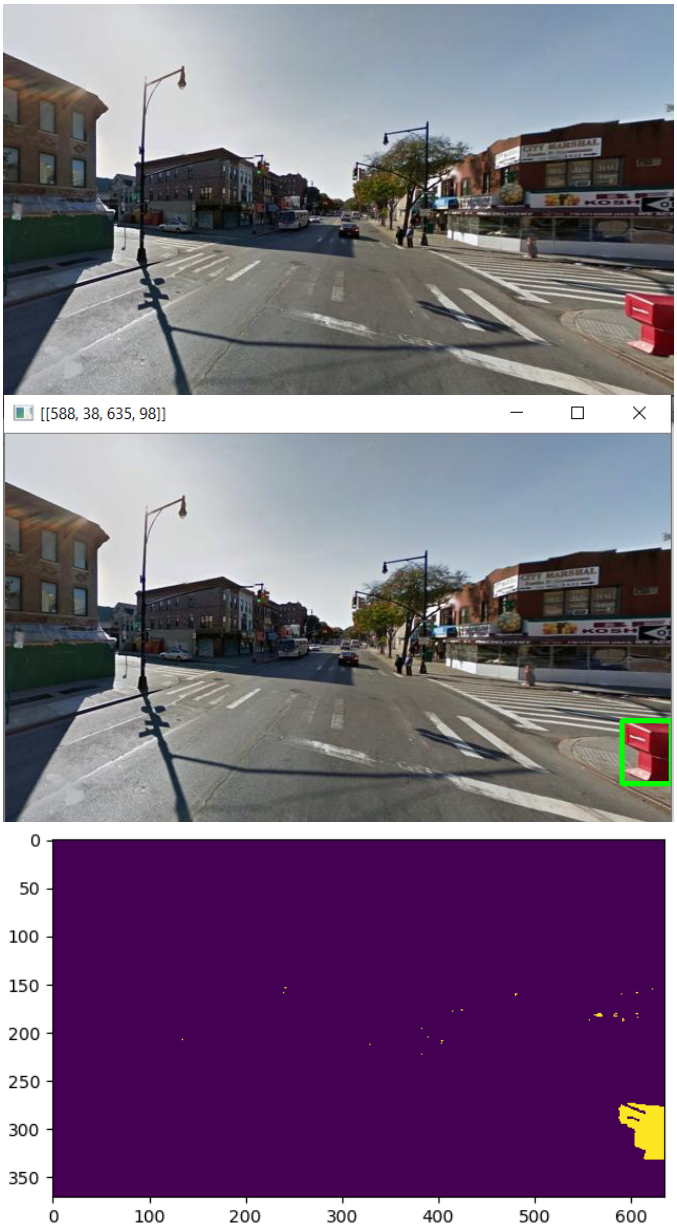


Fig. 18. Image number 134 output and its segmentation Mask. This image has the same problem as figure 17 i.e. classifier works perfectly, but the contour formed has area , no of sides that are the range we set for α, β and γ and h/w to detect stop signs, hence the algo detects a stop sign but in reality there are none