ECE 276A: Stop-Sign Detection*

* An Exhausting Exercise

Roumen Guha
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
roumen.guha@gmail.com

***Abstract*—This paper presents a particular solution to tackle the problem of stop-sign detection. Very briefly, we implemented a color classifier via logistic regression, then imposed shape constraints upon the images. The results section contains a small subsample depicting the functionality of the algorithm.**

***Index Terms*—color classification, object detection, image segmentation**

I. Introduction

Object detection is a popular and important topic in automation. This paper focuses specifically on the problem of stop-sign detection for the problem space of autonomous driving. The problem of stop-sign detection can be broadly placed into two distinct categories: color classification and shape detection. Both problems can be complicated further: color classification becomes difficult when lighting conditions change, and shape detection becomes difficult when there is partial obstruction of the stop-sign, or when approaching the stop-sign at an angle, for example. These problems exist even with human observers, so perhaps there is an upper limit to how well any algorithm can perform. In this paper, we outline our best attempt, using logistic regression and some shape features of stop-signs, which achieves passable performance as a proof of concept.

II. Problem Formulation

In this section, we precisely define the quantities we are interested in. We also cover some necessary knowledge necessary to understand how this algorithm was implemented.

The object we are trying attempting to track is an octagon, primarily red in pixel composition, and contains non-red pixels in its center (the pixels that make up the STOP lettering). Fig. 1 is an example.

A. Color Classification

Our intuition led us to attempt building a color classifier of pixels, for which a discriminative model will suffice for this task. Our problem is then the following:

Given a class $y \in \{+1, -1\}$, find a rule to determine whether a pixel is red ($y = +1$) or not red ($y = -1$).

For a single pixel, this classification step can be written as

This project was worked on with Maria Harris, Stephen West, and Will Argus.

Fig. 1: Example of typical stop-sign. (Image 83 of provided dataset.)

$$y_* := \arg\max_y p(y|\boldsymbol{x}_*, \boldsymbol{w}^*) \tag{1}$$

where $\boldsymbol{x}_* \in \mathbb{R}^d$ is the pixel being considered for classification, and $\boldsymbol{w}^* \in \mathbb{R}^d$ is the optimized set of parameter weights. Here, d is the number of values associated with each pixel (e.g. in RGB or HSV space, $d = 3$. More on this in Section III.).

Thus, there is another optimization problem here: finding the optimal set of weights $\boldsymbol{w}^*$ (otherwise known as training):

$$\boldsymbol{w}^* := \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}|X, \boldsymbol{w}) \tag{2}$$

where $X \in \mathbb{R}^{n \times d}$ is a dataset of n pixels (typically collected from several images), with each row corresponding to a known label y in the corresponding row of $\boldsymbol{y} \in \mathbb{R}^n$.

From the previous results y_*, we can form $\boldsymbol{y}_*$ (the vector of scalar y_* for all pixels in an image X^*). This vector can be thresholded and reshaped into an image of identical dimensions as X^*, forming a binary mask. This will aid us in the shape detection step, as we will see in Section III.

B. Shape Detection

Before we get into the technical approach of shape detection, we will introduce here the concepts and reasoning behind some of our steps.

In shape detection, for any given image, we desire to output a list of lists which contain the coordinates of the bottom-left

corner and the top-right corner of rectangles that each contain a detected stop-sign.

As we will see later, one effective way to determine if our detected region of red is a stop-sign is to consider the aspect ratio of the segmented region: stop-signs have roughly equal width and height when viewed from typical viewpoints. A traditional approach to considering this aspect ratio is to attempt to fit an ellipse to the region and consider the eccentricity of the ellipse. We desire an eccentricity close to zero, as this represents a perfect circle and correlates to an aspect ratio of $1:1$. The eccentricity e of an ellipse is a constant, and can be found by

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{3}$$

where b is the semi-minor axis, and a is the semi-major axis of the ellipse.

III. Technical Approach

In this section, we discuss the algorithms and mathematical models implemented in this project.

A. Color Classification

To obtain the discriminative model, **logistic regression** was used on the pixels in RGB format.

The given dataset of 197 images was distributed in a 2:1 ratio of training set validation set, to ensure no over-fitting occurred. The images were converted into HSV colorspace and filtered for red according to the following values:

Lower Limit = [20, 150, 50]

Upper Limit = [255,255,230]

And if, in the filtered outputs, there were non-red pixels, these pixels were separated out by hand and placed in the not-red pixel dataset. Similarly, if red-pixels were segmented out by the filtering, these were separated out by hand and placed in the red pixel dataset.

Further, we subsampled the pixels in both datasets to achieve an approximately 50/50 split between them in our overall dataset. This is because, if one dataset is overrepresented, the optimization prefers reducing the error produced by misclassifying the more represented class.

To make the model more expressive, we appended a column of ones to our matrix of pixels X, to use as a 'bias' or 'slack' variable. This makes it so each pixel contains 4 values, and so $d = 4$.

Therefore, Eq. 2 can be rewritten as

$$\boldsymbol{w}^* := \arg\max_{\boldsymbol{w}} \prod_{i=1}^{n} \sigma(y_i \boldsymbol{x_i}^t \boldsymbol{w}) \tag{4}$$

where $\boldsymbol{x_i}^t \in \mathbb{R}^{1\times d}$ is the i^{th} pixel, which occupies the i^{th} row of X, and $y_i \in \{+1, -1\}$ is the corresponding label of the i^{th} pixel, and $\sigma(\cdot)$ is the sigmoid function, defined as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{5}$$

Since the data here is very large in sample size ($n \geq 5000000$), to find $\boldsymbol{w}^*$ we opt to use gradient descent to iterate through the following equation until the difference between $\|\boldsymbol{w^{(t+1)}} - \boldsymbol{w^{(t)}}\|_2$ becomes arbitrarily small (i.e. when $\boldsymbol{w}$ converges to a value):

$$\boldsymbol{w^{(t+1)}} = \boldsymbol{w^{(t)}} + \alpha * \sum_{i=1}^{n} y_i \boldsymbol{x_i}^t (1 - \sigma(y_i \boldsymbol{x_i}^t \boldsymbol{w})) \tag{6}$$

where α represents the learning rate, and is found experimentally. We can then set $\boldsymbol{w^*} := \boldsymbol{w^{(t+1)}}$ to form the decision boundary for the classification of a single pixel:

$$y_* = \begin{cases} +1, & \boldsymbol{x_i}^t \boldsymbol{w^*} \geq 0 \text{ (red)} \\ -1, & \boldsymbol{x_i}^t \boldsymbol{w^*} < 0 \text{ (not red)} \end{cases} \tag{7}$$

B. Shape Detection

In broad strokes, our shape detection algorithm works like this:

1) Obtain the binary mask by calling the segment_img() function.
2) Blur the image slightly using a Gaussian blur.
3) Erode the image slightly to connect the top-half and bottom-half of the stop-sign, to reduce the possibility of having two disconnected halves of a stop-sign.
4) Obtain the two largest contours in the image using cv2.findContours()
5) For each contour, use cv2.approxPolyDP() to return points along the approximated polynomial.
    a) Continue only if there are at least 8 points in this list of points.
        i) Fit an ellipse to these list of points using cv2.fitEllipse()
        ii) Using the axes property of this ellipse, continue only if the eccentricity is less than a pre-specified maximum value OR if the polynomial has less than 12 points
            A) Obtain the coordinates of the bounding box from cv2.boundingRect()
            B) Continue only if this bounding box has an area greater than a pre-specified minimum value
            C) Append the coordinates of this box to our list of boxes

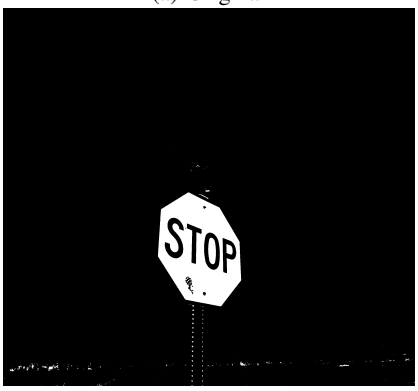
IV. Results

A. Discussion

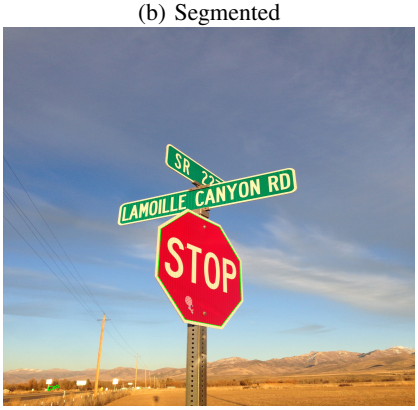
As we can see from the images attached, the algorithm outlined here performs well on the training set and the validation set, but fails completely on the testing set (of which I only have access to a single image).

It seems that the failures stem from the color classification step: the lighting conditions of Fig. 6 likely result in my color classifier missing the color red in the image altogether; indeed, our algorithm picked the tail reflector of the truck and the

(a) Original



(b) Segmented



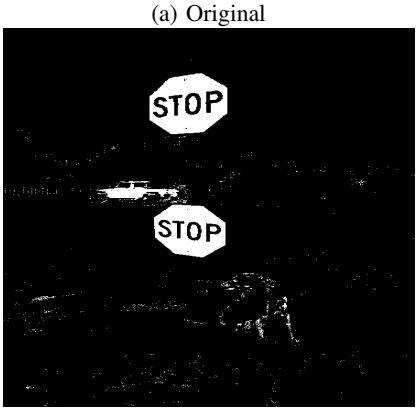
(c) Bounded: [[919, 687, 1416, 1672]]

Fig. 2: Training image 52.

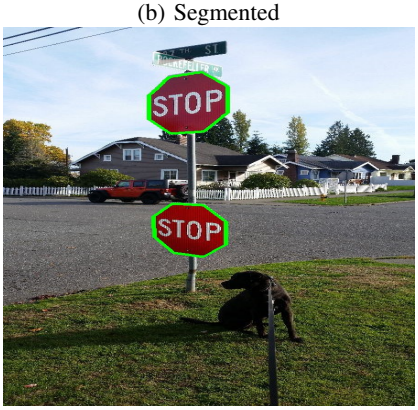
rear-wheel hub of the car in the driveway instead. However, we did see a marked improvement in the classification with an increased number of iterations! So, it's likely that this is the best our logistic classification can do, at least given the datasets as defined, because as we can see in Fig 7, the loss became relatively constant after 200 iterations.

We suspect this would be much improved using Gaussian Mixture Models instead of logistic regression, and propose this for further study.

(a) Original



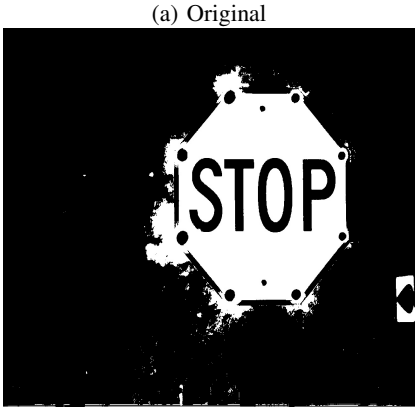
(b) Segmented



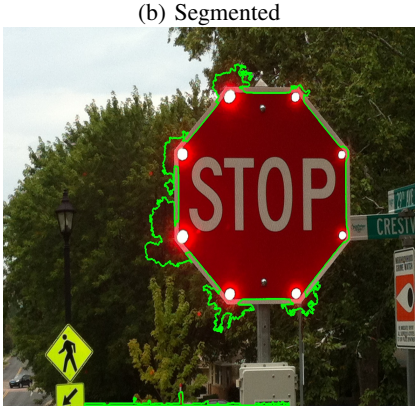
(c) Bounded: [[159, 575, 246, 705],[164, 315, 244, 427]]

Fig. 3: Training image 68

(a) Original



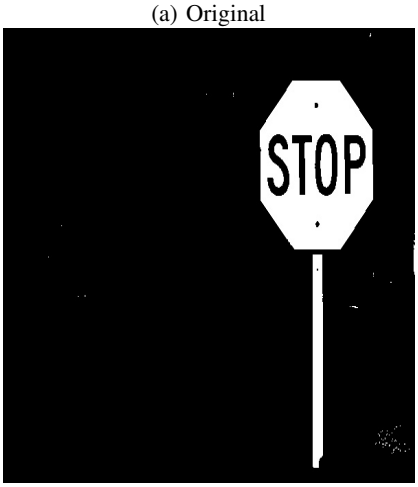
(b) Segmented



(c) Bounded: [[447, 188, 1089, 873]]

Fig. 4: Validation image 18.

(a) Original

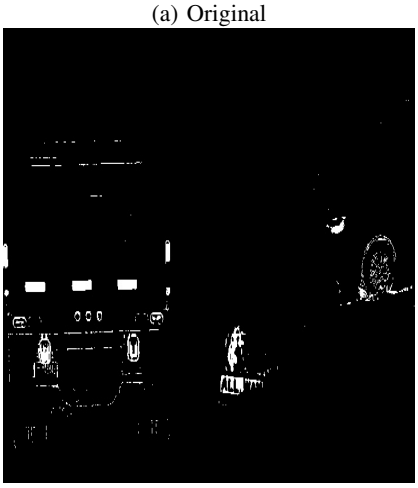


(b) Segmented

(c) Bounded: [[373, 230, 540, 398]]

Fig. 5: Validation image 89.

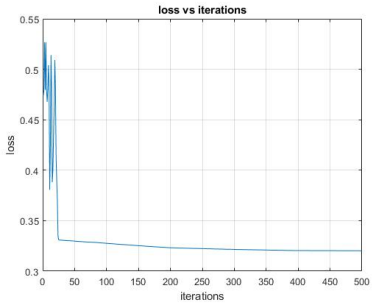
(a) Original



(b) Segmented

(c) Bounded: []

Fig. 6: Test image 5.



(a) Loss vs iterations

(b) Classification using weights after 20 iterations

(c) Classification using weights after 100 iterations

Fig. 7: Effect of more iterations