ECE 276A PR1: Stop Sign Detection

Eric Megrabov (A12177906)

Abstract—**This project uses the computer vision technique of color segmentation as well as shape statistics in order to perform a classification task on pixels that allows us to detect stop signs in a scene. I use a Single Gaussian Generative Model in order to perform this task.**

I. Introduction

Color segmentation is critical to the field of computer vision in that it is very helpful in a wide variety of detection tasks. It helps us filter out components of an image or scene that is not important to the task at hand in such a way that we are only left with the portions which will allow us to classify objects properly. In this project, our goal is to be able to detect and outline stop signs, whether it be one or many, in a given image. This is critical to many fields, such as autonomous cars, and can help further other fields indirectly via its development.
I tried two different approaches for this problem: k-ary logistic regression with nine classes, and a binary single generative Gaussian. This report will mainly cover the generative Gaussian as it yielded better results in my testing than logistic regression.

II. Problem Formulation

In this project, the input is given to us as an $MxNxC$ image, indicating the number of columns, rows, and color channels for the image. Our objective is to classify the pixels of this image into a respective color. There are K different colors, and we want to determine which of these K different colors a vector, which represents a pixel, corresponds to. The training data gathered is in the form RGB, ie red-green-blue. Each channel corresponds to one of these three colors, and the mixture of these colors forms a new color. In order to perform this classification, I model the data for red and nonred colors as a Gaussian distribution $P_{x|Y}(x|i)$ and use Bayes' decision rule under the 0-1 loss assumption in order to perform a binary classification. In other words, $g^*(x) = argmax_i P_{Y|x}(i|x)$, where i indicates red or nonred classes, and $P_{Y|x}(i|x)$ is the probability distribution for a given label. In the case of logistic regression, I chose to use nine different classes corresponding to different colors: black, brown, gray, blue, orange, red, yellow, white, and green. In this case, I would maximize $P_{Y|x}(i|x;\, w)$ where w is a set of hyperparameters. By performing gradient descent, I would be able to obtain the maximum likelihood estimate of w and thus optimize my classifier for this training task.

After each pixel is classified, I create a binary mask in which the value in the mask for a corresponding pixel location is 1 in the case that the pixel at that point is determined to be red by my classifier, and 0 otherwise. From here, I perform several steps of morphological image processing to clean up the mask and filter out elements based on region-of-interest area as well as by aspect ratio of contours.

III. Technical Approach

The project code was written using Python 3, mainly in Jupyter Notebook, as well as using MATLAB in order to perform data collection using RoiPoly.

A. Data Collection

Data collection was performed as a joint effort with two other students, Aravind Mahadevan and Abhiram Iyer. I used MATLAB's RoiPoly function in order to select regions of interest and manually label them as corresponding data. The amount of pixels collected for each color class can be seen below.

Color Class	***# Data Collected***
Black	13,946
Blue	30,000
Brown	4,646
Gray	2,810
Orange	1,920
Green	8,862
Red	88,654
White	1,947
Yellow	3,212
Total	**155,997**

All colors were collected only from data that contained stop signs by inspection; the images in the provided training set that did not have stop signs in their scene were not used. Originally, I had collected over one million blue samples due to the simplicity of doing so; the sky in most images had large chunks of blue. However, other classes had orders of magnitude less data, so I decided to cap the blue data points to 30,000. This would ensure that there is less bias towards blue. In addition, I made sure that red had the most samples because I want to be able to make as versatile of a stop sign detector as possible. However, note that I did not select every red pixel in the image; this was so that when I run testing on the images from which I collected the data, I do not overfit to them. To do this, I collected a large amount of data so that many different types of scenes would be covered (e.g. low light, shadowed, obscured, etc). Note that the collected data was in RGB format. Ultimately, I chose to use 70% of the data as a training set and the remaining 30% as a validation set.

B. Single Gaussian Generative Model

For classification of pixels into colors, I chose to use a binary classifier while using the Single Generative Gaussian via a Naïve Bayes model. This uses a generative model $p(y, X\,|w, \theta)$ with the naïve assumption that every each dimension for a sample *x* are independent when conditioned on *y*. Since this is a binary classifier, $y\epsilon\{0,1\}$, where 0 represents the red category and 1 represents the nonred category. In this case, we use 0-1 loss. In other words,

$$L[g(x), y] = \begin{cases} 1, g(x) \neq y \\ 0, g(x) = y \end{cases} \tag{1}$$

From Bayes Decision Rule, we find that the optimal decision function is

$$g^*(x) = argmax_i P_{Y|X}(i|x) \tag{2}$$

This can be more explicitly stated as 'Pick 0 (ie red) if:

$$P_{Y|X}(0|x) > P_{Y|X}(1|x) \tag{3}$$

$$\frac{P_{X|Y}(x|0)P_Y(0)}{P_X(x)} > \frac{P_{X|Y}(x|1)P_Y(1)}{P_X(x)} \tag{4}$$

$$P_{Y|X}(0|x)P_Y(0) > P_{Y|X}(1|x)P_Y(1) \tag{5}$$

(5) will be my decision rule for selecting if a given pixel is red or not red. However, I need a way to model $P_{X|Y}$ as well as P_Y. To model $P_{X|Y}$, I used a Gaussian distribution since this is mostly indicative of real-world color data. For example, in our images, the vast majority of the stop signs are bright red, with far fewer of them being light red, dark red, or obscured by shadows. As a result, this seems to be a good choice for a color model. Thus:

$$P_{Y|X}(i|x) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_i|}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i{}^{-1}(x-\mu_i)} \tag{6}$$

Where d is the dimension of the data *x* (in this case, 3, since there are R,G, and B, values for each pixel *x*).

For the mean, μ_i, we use the Maximum likelihood estimate, obtained by taking the derivative with respect to the mean of the log-likelihood of the data under the assumption that each sample 1…n is independent. We also obtain the maximum likelihood covariance estimate in a similar fashion by taking the derivative with respect to the covariance of the log-likelihood of the data and setting both to zero in order to maximize the likelihood. As a result, we obtain:

$$\mu_i = \frac{1}{n}\sum_{j=1}^{n} x_j^{(i)} \tag{7}$$

$$\Sigma_i = \frac{1}{n}\sum_{j=1}^{n} (x_j^{(i)} - \mu_i)(x_j^{(i)} - \mu_i)^T \tag{8}$$

Since I collected data from every image in my training set that contained a stop sign, I was unable to find an exact way of estimating my prior probability. However, what I did instead was I looked through the images manually and saw that there was approximately five times more nonred pixels than red pixels, so I assigned $P_Y(0) = 0.2$ and $P_Y(1) = 0.8$, which worked well for the purposes of image segmentation. Had I collected every single possible pixel, I would have approximated the prior probabilities by counting the number of reds, counting the total number of nonreds, and dividing each by the total number of pixels in the training set. This is for future iterations of this project.

My error calculation is as follows:

$$E = \frac{false\ positives + false\ negatives}{total\ samples} * 100 \tag{9}$$

In order to perform pixel classification, I tried several strategies. Initially, when reading in a test image using cv2.imread, the loaded matrix is in the form BGR rather than RGB. As a result, I chose to immediately convert this image to RGB format since I performed data collection in RGB format. Then, I created a grayscale mask in which each pixel location would be assigned as 1 if the naïve Bayesian classifier predicted red, and it would be assigned as 0 otherwise. Checking the results of my mask qualitatively, I saw that this did not yield stellar results because the red-green-blue colorspace is not very resistant to lighting changes.

I also tried normalizing all values of the data for both training and validation between -1 and 1, hoping that doing so might improve the accuracy of my mask. Due to the nature of it already being modeled by a Gaussian, this hurt my results further. I explored other normalizing possibilities, such as normalizing in different ranges, especially for different colorspaces. For example, YCrCb has a different normalizing range for Y [16, 235] compared to its other two channels [0, 255]. However, this did not improve results. Seeing this drop in accuracy helped me decide that normalization is not a strategy that should be pursued further.

Instead, I chose to experiment with a few other different colorspaces. HSV (Hue, Saturation, Value), YCrCb (luminance, redness, blueness), and LAB are much more color invariant. With just RGB, I was getting a small amount of false negatives in which only bits and pieces of the stop signs were being detected. I was also getting minor false positives, where other pixels that were not red were still being classified as such. Interestingly, my RGB accuracy was the highest of all colorspaces, but in real-world testing, LAB seemed marginally better. In addition, since my priors were only a qualitative estimate, I varied them repeatedly to tune into the ones that provide the best results. At a 50/50 split, there were far too many false positives for red, and at 5/95 (5% red, 95% nonred) split, there were too many false negatives. Ultimately, the approach I chose was to use the LAB colorspace with no normalization, as well as a 20% red, 80% nonred prior probability split. In addition to the Single Gaussian Generative Model, I also chose to use k-ary logistic regression, which I do not pursue as my ultimate model to due to convergence issues as well as low accuracies. If you would like to read about this approach, please look at section D.

C. *Stop Sign Detection*

Under the assumption that the segmentation color masks built using the classifier I implemented work as intended, I could use them to finally try finding bounding boxes on where stop signs should be. For the sake of the autograder, I resize images to be 70% of their original size if either the width or height of the input image contains more than 2,000 pixels to obtain the segmentation color mask. Without this, the autograder fails to run. Given the mask, I process it with a median blur with a kernel size of 5. This removes the majority of noise and artifacting in the image. It was slightly more effective than using a Gaussian blur.

Following this, I use the concept of morphology in image processing in order to further clean up any imperfections in the mask. Dilation typically fills in small holes within regions in an image, while erosion gets rid of islands and narrow bridges. Opening performs an erosion followed by a dilation, thus getting rid of small artifacts yet still preserving the sizes of other items in the image. On the other hand, closing performs dilation followed by erosion, which fills out small holes while again preserving size. I explored a vast variety of combinations of all six of these different operations. Ultimately, I chose to to a closing with a kernel of size 5 to fill in the "STOP" part of the stop sign to begin. I followed this with further combinations of dilation and erosion in order to remove small objects in the image that I do not want to classify as part of a stop sign. This strategy worked well, but still did not always cause a perfect filter, which is reasonable for real-world data.

To obtain and examine each foreground chunk that remained after the image processing, I used cv2.findContours. This provided me with a set of contours around which I placed bounding boxes in my first iteration of attempting to find the stop signs. This yielded a very large amount of false positives, as any small red areas such as tail lights of cars as well as misclassified pixels, would now be bounded by a bounding box indicating that they are stop signs. I chose to do further filtering of the image; one strategy I employed was the check to see if the area of the largest contour in the contour set was five times larger than the rest of the contours. If any of the contours were five times smaller than the rest of the contours, I removed them from the set of contours to be considered. Next, I checked the aspect ratio of each contour. If a photo is taken at an angle of the stop sign, it tends to be skewed and thus looks elongated vertically. However, it is very difficult to take a photo of a stop sign that causes it to be elongated horizontally. As a result of this observation, I filtered out all images that have a contour that has a width that was even slightly larger than its height because wide stop signs are very rare. In addition, I put a much milder restriction on vertically-elongated stop signs. My bounding boxes use row, col format for the sake of displaying the images properly locally. However, I do perform a conversion to the x-y coordinate plane for the autograder to process the results correctly.

D. *Logistic Regression*

Before switching to the Naïve Single Generative Gaussian model, I had originally tried to use a k-ary logistic regression discriminative model. Here, k was between 0 and 8, where each color was represented by a respective value of k, and *y* is an element of one these classes. I use gradient descent to update randomized (between -1 and 1) weight parameters W by the maximum likelihood estimate.

$$p(y|X,W) = \prod_{i=1}^{n} e_{y_i}^{T} \frac{exp^{\mathrm{W}x_i}}{\mathbf{1}^{T} exp^{\mathrm{W}x_i}} \qquad (10)$$

Where $e_{y_i}^{T}$ is the jth standard basis vector (ie one-hot-encoding) and the term to the right of it is define as the softmax function *s*. For the softmax function, it was necessary to use a different form for softmax due to numerical conditioning:

$$softmax(z) = softmax(z - max_i \mathrm{z}) \qquad (11)$$

Because the above probability (10) is not in closed form, we must perform gradient descent using the following update equation in order to optimize our hyperparameter, *W*.

$$W_{MLE}^{(t+1)} = W_{MLE}^{(t)} + \alpha \sum_{i=1}^{n} (e_{y_i}^{T} - s(W_{MLE}^{(t)} x_i)) x_i^{T} \quad (12)$$

I used an update parameter parameter *alpha*, as seen above as a step size for gradient descent. In addition, I used a set of nine minibatches so that my algorithm has improved accuracy in fewer iterations. This modification to batch gradient descent consists of going through *j* iterations, but also splitting up the training set into a set of smaller batches. The update step occurs after each minibatch is processed rather than after the entire batch. This allows for significantly more updates and thus a stronger, faster gradient descent. Each time all the minibatches are processed, the training set is reshuffled, the minibatches are obtained again from the training set, and the process continues. I also calculate and save the accuracy that the *W* matrix for the weights forms each time that the whole training set has been processed. Therefore, in the results section, you can see the accuracies on each iteration in plot form.

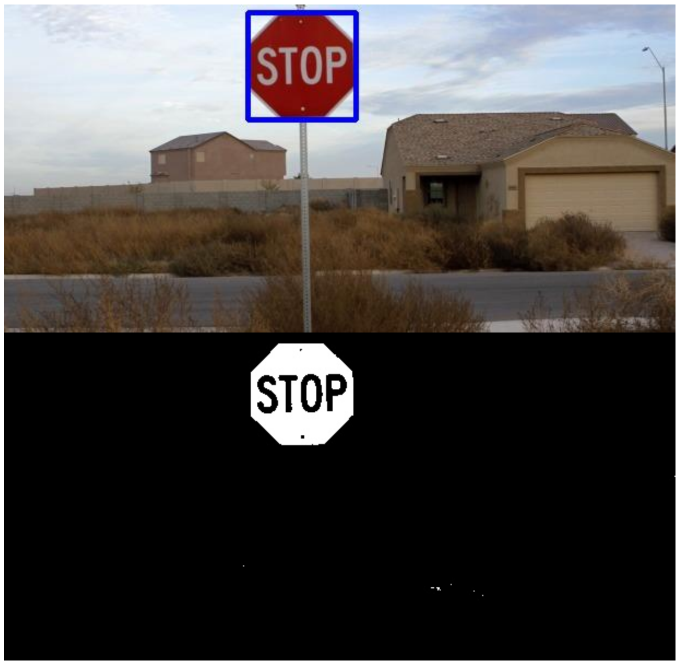
Unlike the Bayes classifier, normalization helped significantly. Before normalization, it was far worse (40%-50%). This is likely due to the fact that I did not make the same Gaussian assumption with my logistic regression trainer as I did with the Naïve Bayes one. In addition, I used a step size of *alpha=0.001* because at larger values, the gradient descent did not converge to the maximize likelihood estimate of the weights. After examining my plots as well as running some test cases, I chose to not pursue this method any longer due to poor accuracy, slow rate of convergence, and generally subpar performance, potentially due to algorithm implementation.

IV. Results

Here, I will discuss the outcome of my segmentation color masks as well as my bounding boxes.

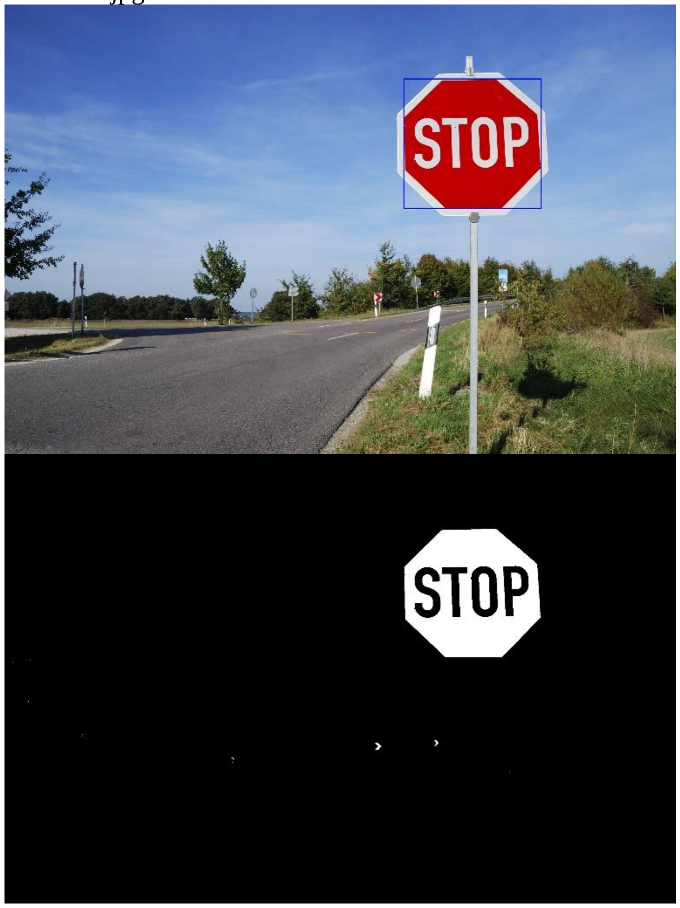
A. *Bounding Box and Image Segmentation*

trainset1/1jpg



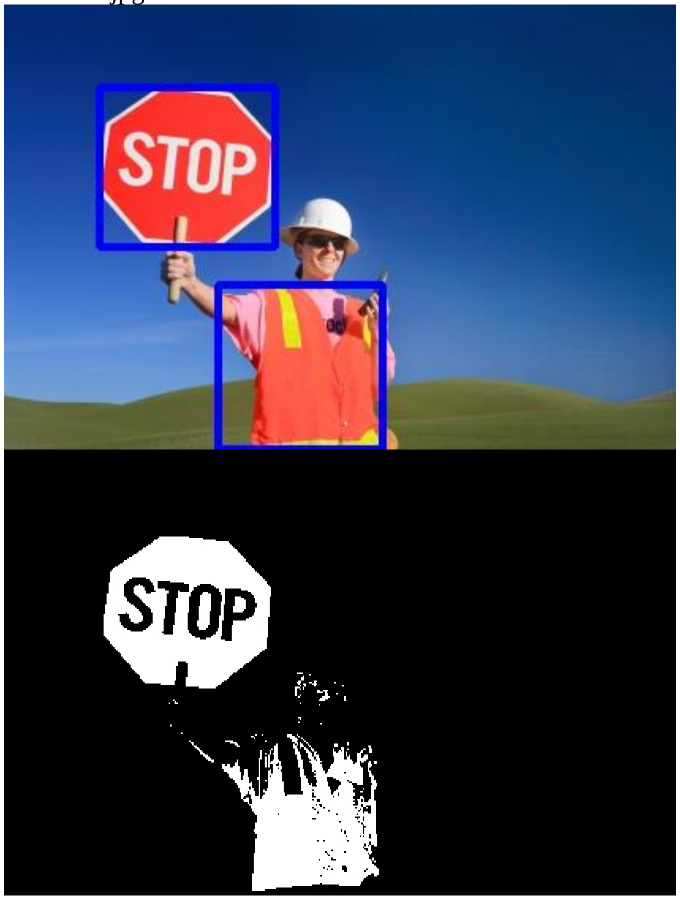
In this image, it is clear that my classifier works very well. This is a very basic example where the red pixels are strongly separated from the rest of the colors in the image. As a result, there are very little left over artifacts after morphological image processing. In addition, the aspect ratio of the stop sign is perfectly square. This is a good sanity check to make sure that the fundamentals of my classifier were implemented properly.

trainset/2.jpg



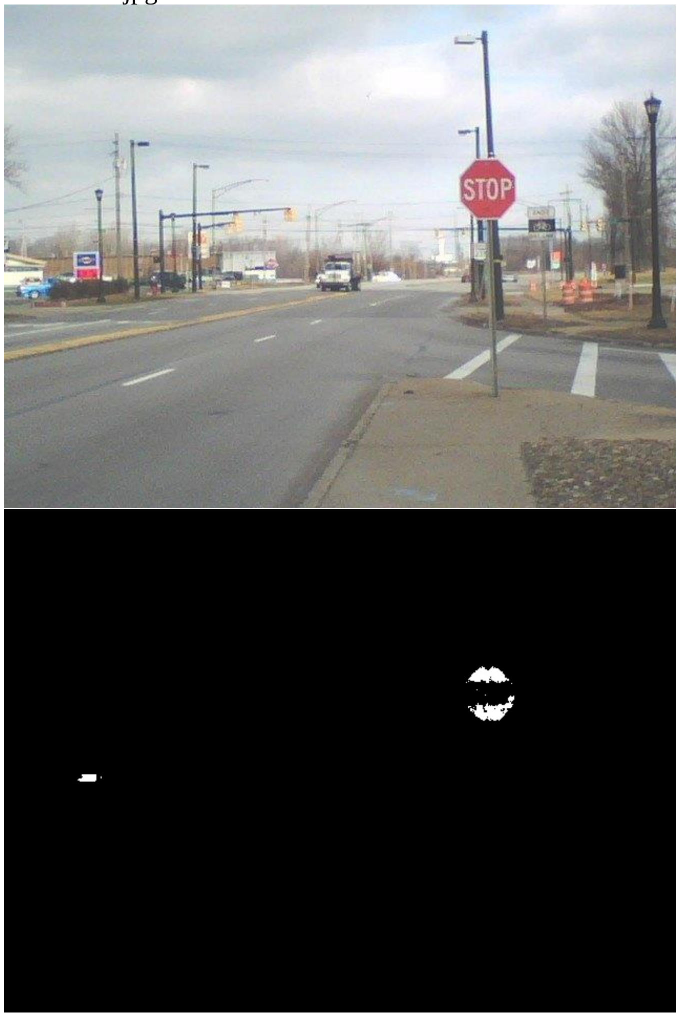
Here, we can see some specific implementation details of my algorithm upon deeper examination. First of all, note how the bounding box is *only* around the the red portions of the stop sign; it does not include the outer white edge. This is because our classifier only uses red for detection. In addition,there are some red arrows pointing right in the background of the image. When I was not performing the area check, these were actually getting classified as stop signs.

trainset/6.jpg



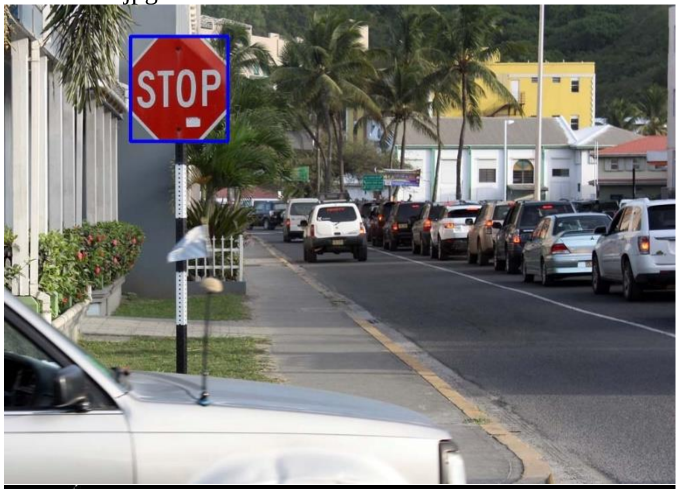
This one of the first failed cases in the training set. Clearly, it identifies the stop sign well. However, it unfortunately also detects the workers' shirt as a stop sign, too. This is due to the fact the stop sign and the shirt are almost exactly the same color. This highlights an issue with my shape statistics; if two objects are both red and similar size as well as aspect ratio, they will both be treated as stop signs.

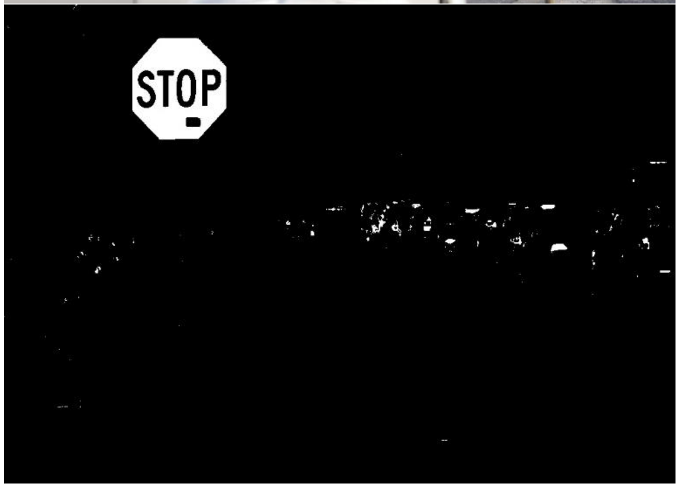
trainset/13.jpg



Here, we see an issue with stop sign detection. The image is relatively low resolution, and as a result, my classifier has a hard time detecting red pixels throughout the middle section of the stop sign. Therefore, it splits the stop sign in half, creates two bounding boxes (upper and lower), which then get filtered by my aspect ratio checker. So, I am left with no bounding boxes. This can potentially be fixed with better morphology that connects the broken sections together.

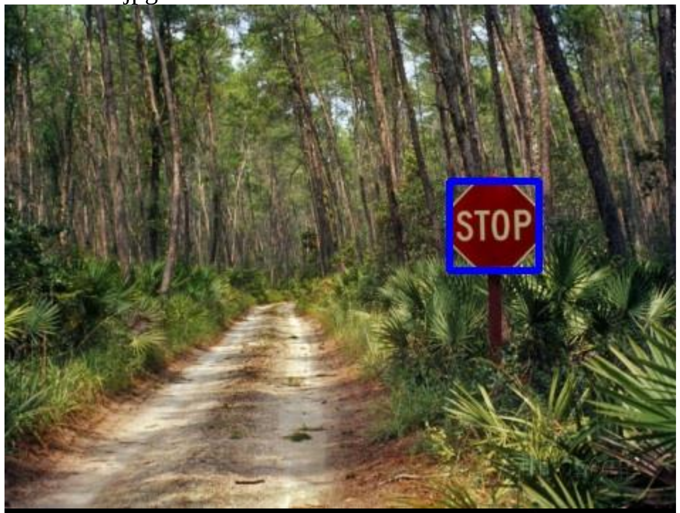
trainset/23.jpg

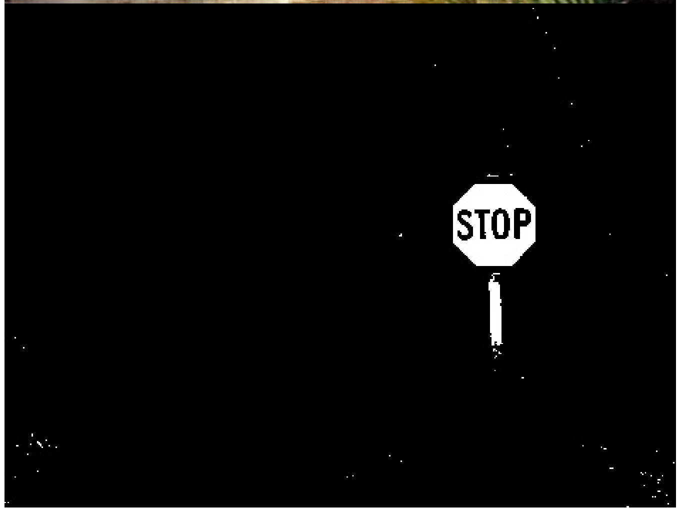




Here is an example of successful classification despite there being many other red pixels in the image due to car taillights. This works well due to the morphological image processing performed before bounding box placement.

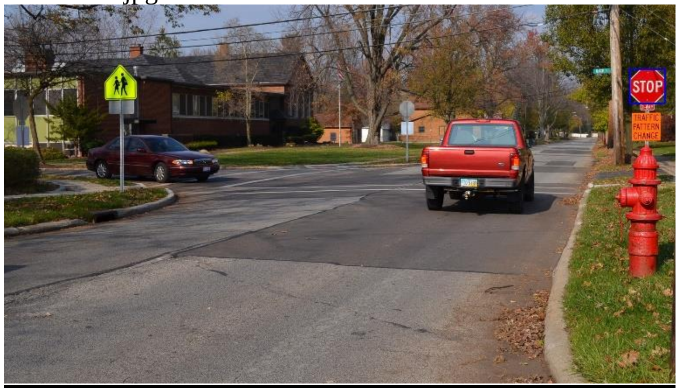
trainset/27.jpg

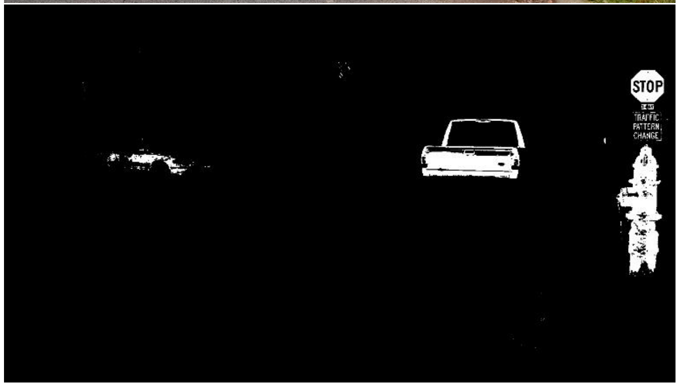




This image shows that my classifier is invariant to large changes in lighting. Despite there being several shadows and discolorations, the classifier still worked well.

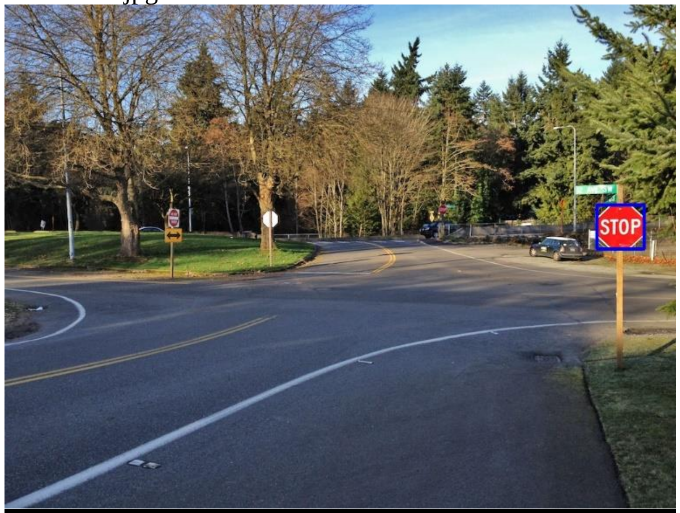
trainset/41.jpg

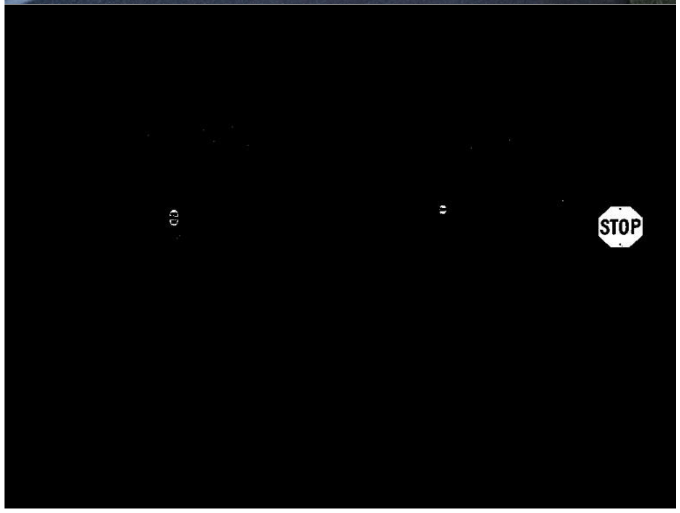




This is perhaps one of the most impressive successful classifications. If you look carefully, you can see a blue bounding box around only the stop sign. Despite there being multiple red cars and a red fire hydrant extremely close to the stop sign, the stop sign is the only one that gets detected due to strong shape statistics.

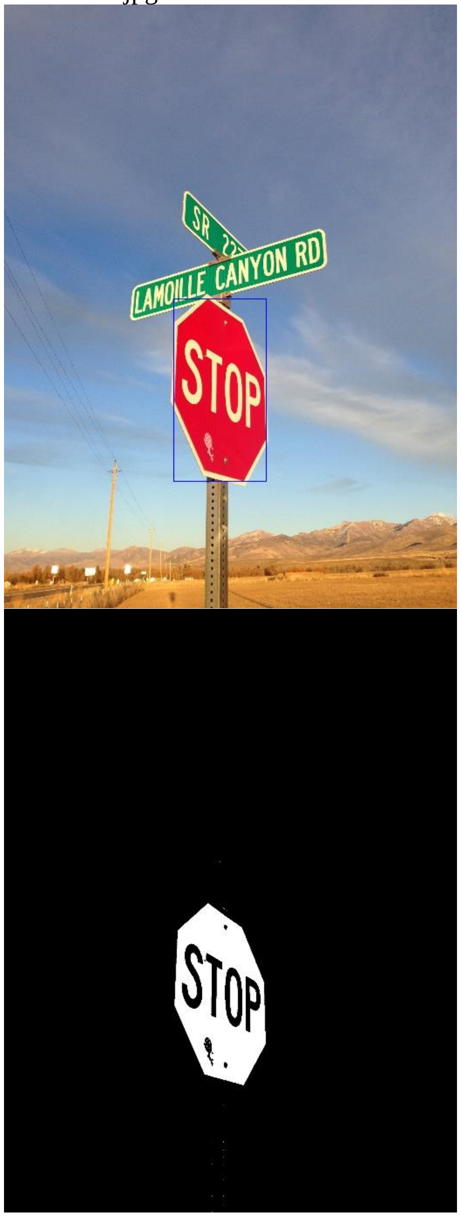
trainset/48.jpg





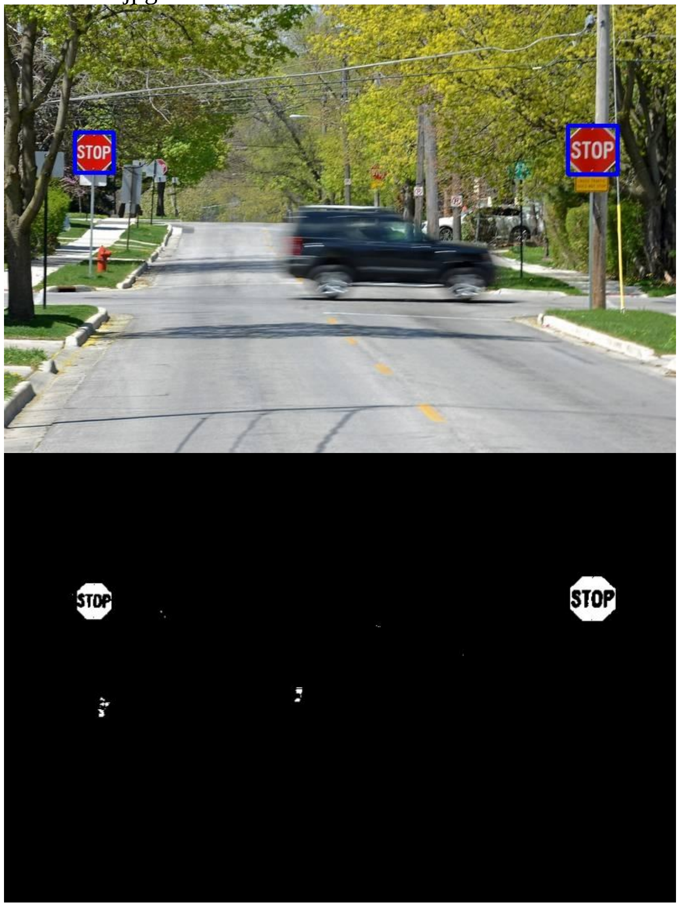
A mild failure can be seen above. My mask actually captures both stop signs in the image, but due to the implementation of my shape statistics, the background one gets filtered out because it is so much smaller than the foreground one. This was a sacrifice I had to make in order to ensure better overall classification for other stop signs in the set, but with fine tuning of my shape statistics hyperparameters, this one could be easily classified.

trainset/52.jpg



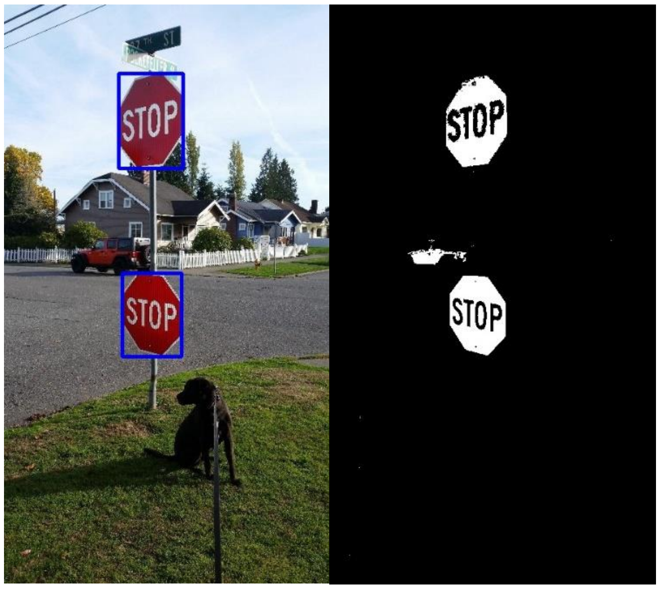
Despite the stop sign being very vertically elongated, the classifier was still able to create a proper bounding box for it because my bounding box algorithm was very loose with vertical aspect ratio requirements.

trainset/64.jpg



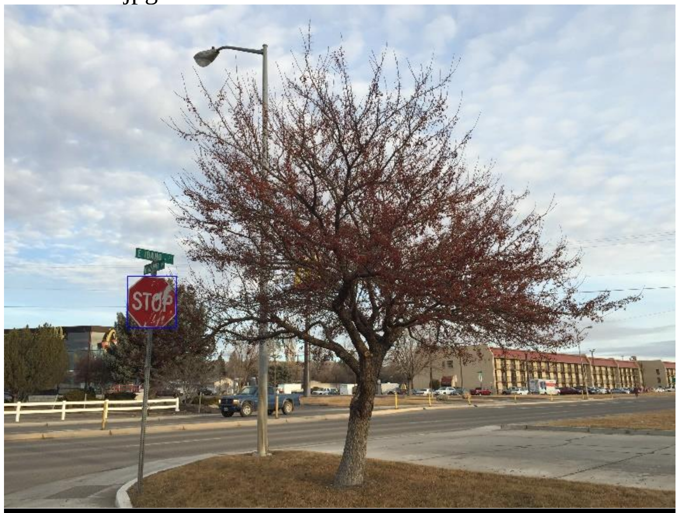
My bounding box algorithm is capable of detecting multiple stop signs in an image properly.

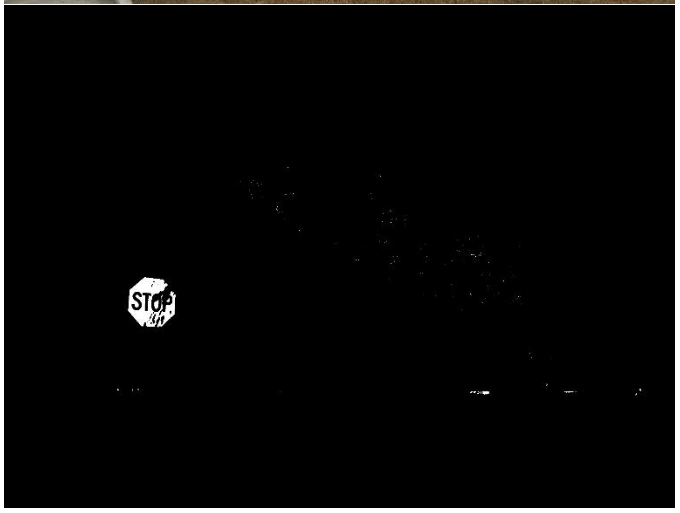
trainset/68.jpg



Another example of proper multi-stop sign detection.

trainset/99.jpg





This does sometimes work well with partially obscured stop signs, as long as it is not huge portion of the stop sign.

B. *Bayes Training/Testing*

I performed extensive qualitative testing amongst different test cases with several colorspaces. I also calculated training and validation accuracies, as can be seen below. For this, I used a 70% training, 30% validation split.

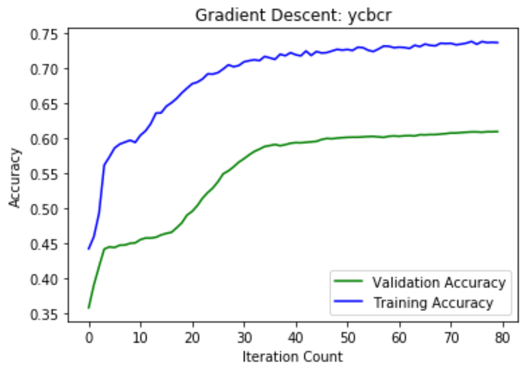
Accuracies

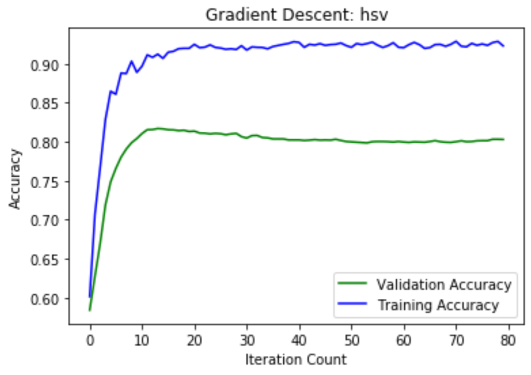
Colorspace	***Training (%)***	***Validation (%)***
RGB	96.662	96.005
HSV	89.192	87.051
YCrCb	96.771	95.904
LAB	95.600	94.532

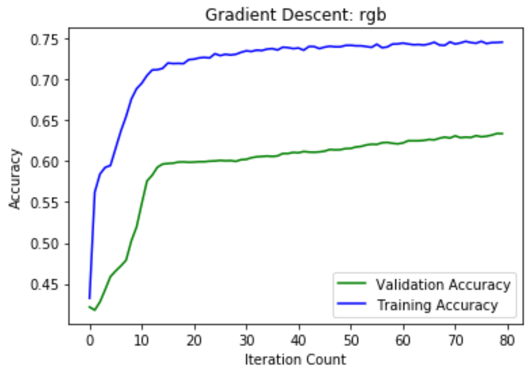
There may have been some overfitting due to stop signs having a high degree of similarity; there are several accuracies that are very close to testing accuracies. Although these accuracies indicate that YCrCb or RGB would be a better choice, I chose to go with LAB because theoretically, it would be more lighting invariant than the other two. Since they are all still quite similar, and I cannot be certain that there may have been issues with data gathering such as misclassified pixels and improper sampling, I chose to use the one I believe to be theoretically best at classifying. Real world performance was very similar to others.

C. *Logistic Regression Training/Testing*

Before I chose to switch the Gaussian model, I tried several logistic regression colorspaces and techniques. The results can be seen below. Note that all colorspaces are normalized between -1 and 1. The accuracies yielded were low quantitatively, and the qualitative masks that they produced were not up to standard, so I did not pursue the technique further.







The accuracies ranged between 60% and 80%, depending on the strategy. HSV seemed to perform best. However, the masks they produced were not very useful in terms of image classification.

D. *Conclusion*

After trying a couple of different methods, I found that a single generative Gaussian model for binary classification works better than 9-ary logistic regression for the task of pixel color classification. This might be because differentiating between two classes is easier than differentiating between 9. In the future, I would like to try implementing 9-class naïve Bayes' classifier and see whether or not the binary case would perform better. Ultimately, shape statistics are necessary for the task of stop sign detection since it involves detecting which pixels are false positives. The strategy of using color will almost always cause some cases to fail simply due to the difficulty in using shape analysis and morphology to clean up the image since it cannot be generalized to every real-world scenario with ease. Still, this method of using pixel color classification followed by shape statistics yields overall good results in detecting where stop signs are found in images and scenes.

ACKNOWLEDGMENT

I used Professor Atanasov's slides extensively throughout the use of the reports, and I also performed data collection with two colleagues, Aravind Mahadevan and Abhiram Iyer.