Color Segmentation for Stop-Sign detection

Siddarth Meenakshi Sundaram
Department of Mechanical and Aerospace Engineering
UC San Diego
A53299801
simeenak@ucsd.edu

***Abstract*—The aim of this project is to do color segmentation (red). I use logistic regression machine learning model to train the hand-labelled images to classify red and not red, and then implement a stop sign detector to detect stop sign in the images given. I use various high-level features such as number of sides of the polygon and shape statistics such as Bounding box area to image area ratio and width to height ratios to improve detection. The effectiveness of the methods used are observed by experimentation, discussed and the results are presented.**

I. Introduction

Color Image Segmentation partition the image into distinct regions of similar pixels based on pixel property. It is the high level image description in terms of objects, scenes and features. The success of image analysis depends on segmentation reliability and hence forms an important part of many applications in the real-world. The applications range from biomedical imaging to self-driving cars. Hence, properly segmenting images by color is a very important task forming the core of above applications and has real-life consequences. So, care must be taken to be as accurate as possible in segmenting the image by color.

This project assumes that objects are stop-signs are colored red distinctively, and that red color matters and ignores other red colors while labelling. I therefore discards other red color while labelling.

In this project, the first objective is to detect red color in the images. This is done by hand-labelling the images one by one as red and not red. The labelled images are stored and are used to train the model to detect red.

Further, the next objective is to detect stop-sign in the images as accurately as possible. The weights from the trained models are used to validate the test set. The quality of the trained model is checked based the testing benchmarks such as loss, accuracy and similarity score.

One application of this whole project can be seen in the application of self-driving car. Detection of stop-sign is small but very vital part of self-driving.

Section II goes through the problem formulation which includes information about the problem, the training data, the models used and the testing data obtained. Section III describes my approach to color segmentation such as using a logistic regression model to train the labelled data in various color spaces (RGB, HSV) and also my approach to stop-sign detection using various shape statistics and high level features. Section IV contains my training results, validation results, and discussion on what worked, what did not, and why. This section also contains some information on how to further improve this work.

II. Problem Formulation

The problem can be subclassified into two parts - Color Masking Image and Detection the stop-sign.

A. *Color segmentation - Masking image*

The first part of the problem requires hand-labelled images with red and not red as input to train the model.

Image is the input. Each pixel is a 3D vector: x = (R,G,B) or (H,S,V). The image is then normalised and then flattened. Training: Given data D such that

$$D = f(x_i, y_i) \tag{1}$$

of pixel values $x_i \in R^3$ labeled with colors(by me) y_i, we optimize the probabilistic model $p(y \mid x)$ using logistic regression.

Training is done using logisitic regression which is a discriminative model. Optimize the equation 1 in such a way that w_* produces good results on a test set $D = f(x_*, y_*)$

$$\text{Training} \quad \boldsymbol{\omega}_{MLE} := \arg\max_{\omega} \; p(\mathbf{y}|X, \boldsymbol{\omega}). \tag{2}$$

The output of the exercise given an image(x_*) is one that detects all the red(y_*) in the given validation or test set i.e. a trained color classification model.

$$\mathbf{x}_* \xrightarrow{\text{Logistic Regression}} y_* \in \arg\max_{y} \; p\left(y|\mathbf{x}_*, \boldsymbol{\omega}_{MLE}\right) \tag{3}$$

B. *Stop-Sign detection*

The second part of the problem uses the the segmented red color to further detect the stop-sign. The input in this case is output of the first function, that is y_*, the detection output (the final output) is in terms of the co-ordinates of the bounding box. The problem of detection can be solved in multiple ways and I have experimented using various shape statistics such as

$$\frac{\text{Width of B.B}}{\text{Height of B.B}} \text{ ratio } \& \frac{\text{B.B area}}{\text{Image Area}} \text{ ratio} \tag{4}$$

,where B.B is the Bounding box. The final ouput needs to be B.B co-ordinates in the form of ($x_{bottomleft}$, $y_{bottomleft}$, $x_{topright}$, $y_{topright}$)

III. Technical Approach

I approach the problem to detect red color as a classification problem. The parts of the technical approach for this project are:
Part A : Setting up virtual environment
Part B : Labelling images
Part C : Splitting Dataset
Part D : Supervised learning
Part E : Getting Bounding Box
Part F : Testing

The detailed approach is as follows :

A. Setting up Virtual Environment

The first step in this project is to set up a virtual environment with all the required packages. I have created a venv Sens_env in pycharm and activated it using the command **activate.bat** and installed the packages required with the following command :

pip install -r requirements.txt

After all the packages such as opencv-python, matplotlib, numpy, scikit-image has been installed, the venv is now ready to use.

B. Labelling images

As all the given images are in the same directory, I list and import them using **os.listdir**. I then convert the imported images to float and normalize it. I, then open them one by one to hand-label them. Hand-labelling is done using roipoly(), which is used to specify the polygon of interest and is similar in function to the matlab command of the same name. roipoly() is called 3 times because there are maximum of three stop signs in a image give. In case there is only one region of interest, I select the stop sign once and click once and exit the roipoly().

Each of the hand-labelled data is stored separately. After all the region of interest have been marked. The mask for each stop sign is created separately and then combined as one mask.

C. Splitting Dataset

I define a function to split dataset for training and validation. Initially, I split the dataset into 2 parts - One containing stop-signs and other not containing them. Then I split the first part of the split dataset into 82 images for training and 15 for validation. I split the second part of the dataset into 85 for training and 15 for validation. The splitting is done randomly for an even distribution. After the dataset is split wholly into training and validation, I move on to Supervised learning.

D. Supervised learning

I implement Logistic Regression, a discriminative model to train the data in this case. The regression model has weights and bias matrices, and the output is obtained using simple matrix operations. The equation of logistic regression is:

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

The label-generating pdf for red and not red is:

$$\begin{aligned} P(Y=1/X=x) &= \sigma(w^T x) \\ P(Y=0/X=x) &= 1-\sigma(w^T x) \end{aligned} \tag{6}$$

Combine the two cases to obtain the final model - Joint probability density function for the distribution of y in terms of the example x and the parameters ω :

$$P(Y=y/X=x) = \sigma(w^T x)^y [1-\sigma(w^T x)]^{1-y} \tag{7}$$

Taking log, we get

$$\sum_{i=1}^{n} y^i log(\sigma(w^T x^i) + (1-y^i) log(1-\sigma(w^T x^i)) \tag{8}$$

This is used to find the optimal ω_{MLE}. The overall parameteric learning ω_{MLE} in equation 3 is calculated as:

$$w_{MLE} = \text{arg max log P(y/x, w)} \tag{9}$$

from equation 8 and updated as:

$$w_{MLE}^{t+1} = w_{MLE}^{t} + \alpha \sum_{i=1}^{n} [y^i - \sigma(w^T x^i) x^i] \tag{10}$$

The above update equation is equivalent to:

$$w = w + \alpha * \nabla(x, y, w) \tag{11}$$

The above update equation is gradient descent used in the code with ∇ replaced by gradient_diff function. The final output is classified based on the following equations:

$$\begin{aligned} y^* &= +1, (x^*)^T w^* >= 0 \\ y^* &= 0, (x^*)^T w^* < 0 \end{aligned} \tag{12}$$

Logistic Regression	
Parameters	Values
w_initial	random
α (learning rate)	0.01
eps	1e-4
# of epochs	50

TABLE I: Logistic Regression Parameters

For the logistic regression model as defined, I have used the parameters in I to train.

E. Getting Bounding Box

Drawing the bounding box and getting the co-ordinates is the second part of this project. After I trained the model, I used this trained weights to create mask for the testing image. For the mask_img, I then find the bounding box using the following technique:

I have used morphological operation of dilation to make the stop-sign more visible and fill small holes in objects, thus removing them from being wrongly identified as a stop-sign. I have used a Kernel size of 3 x 3 to produce dilation.

For the next step, that is obtaining the bounding box, I have used findContours function in CV_RETR_TREE mode to find the contours and the hierarchy of the contours. Using CV_RETR_TREE mode is useful because the stop-sign polygon has child contours in the form of S, T, O, P whereas in other cases such as traffic lights, car tail lights and red buildings do not have child contours. This helps in weeding out unnecessary bounding boxes.

For finding the STOP sign, I use two shape statistics. One is using the ratio of width of the bounding box to height of the bounding box which I call the eccentricity of the box. I kept the range of this between 0.8 to 1.2.

F. Testing

I have uploaded the stop-sign-detector.py along with my final_weight.npy to the autograder in gradescope for checking performance in the testing dataset. I got the final correct results to be 11 correct out of 15.

IV. Results

I have plotted the loss versus the epoch graph.

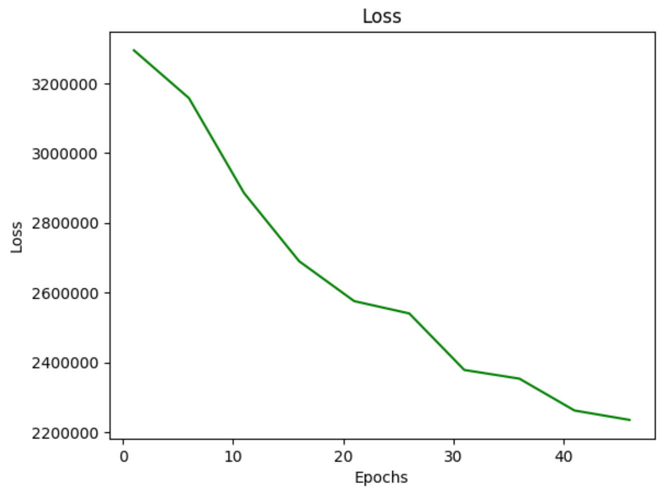

Fig. 1: Loss vs Epoch

We can see from the plot that the loss decreases with the epoch count. So, we can infer that the logistic model has been trained.

Case 1: We can see from Figure 2 that the bounding box has been formed perfectly and the mask confirms this. So the Example 1 in figure is a successful case.

Case 2: We can see from Figure 4 that there is a stop sign clearly, but it is not detected. Hence, this is a failure case.

Case 3: In case of Figure 6, there is no stop sign but a stop sign is detected, hence this is a failure case.

Case 4: In case of figure 8, there is no stop sign and none are detected, hence this is a successful case.

Case 1 is a True positive case.

Case 2 is False negative case.

Case 3 is False positive case.

Case 4 is True Negative case.

Also we can gather from the autograder results that only stop-signs in 11/15 images are correctly detected. This may

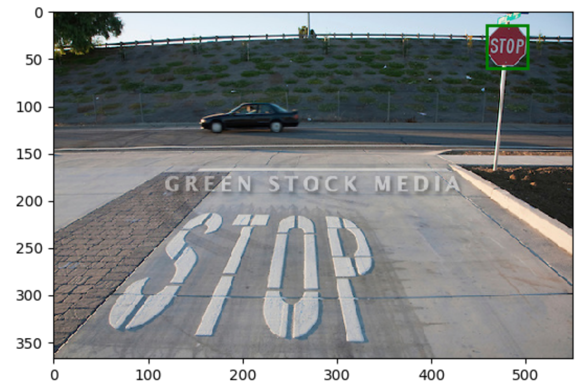

Fig. 2: Example 1

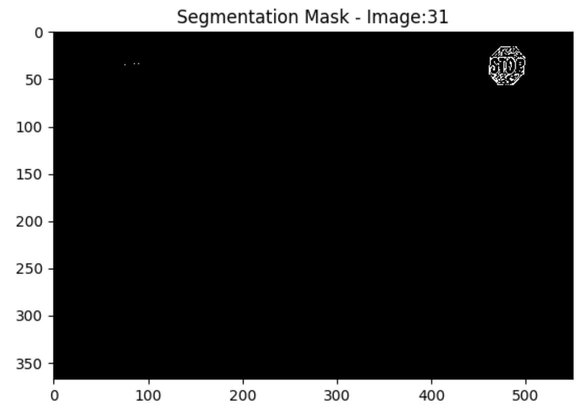

Fig. 3: Mask of Example 1

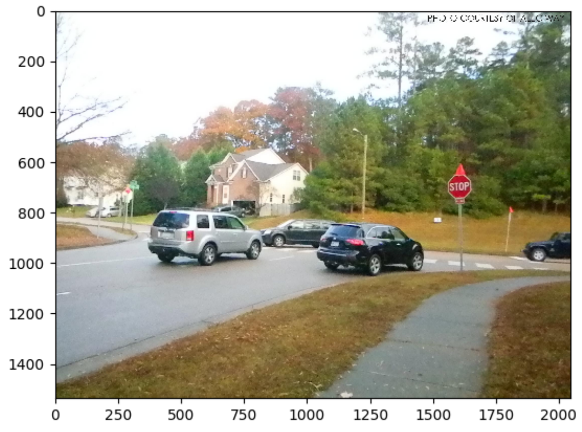

Fig. 4: Example 2

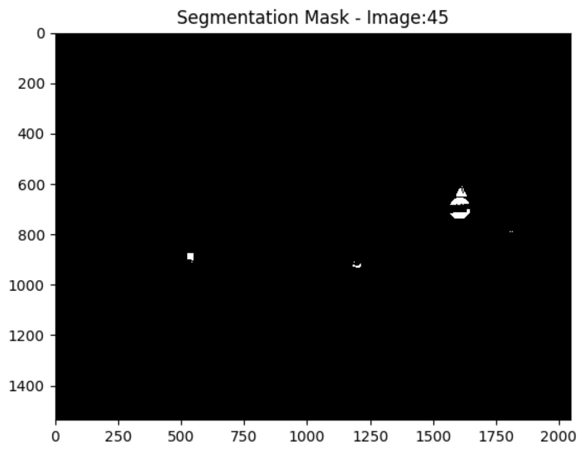

Fig. 5: Mask of Example 2

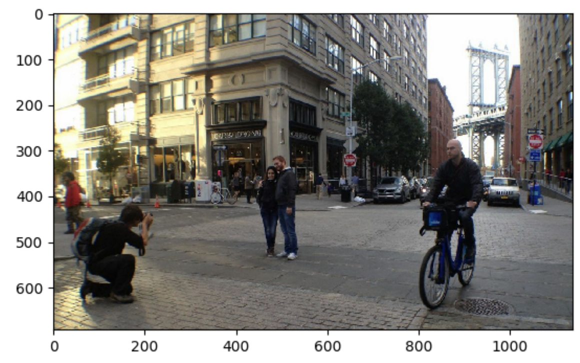

Fig. 8: Example 4

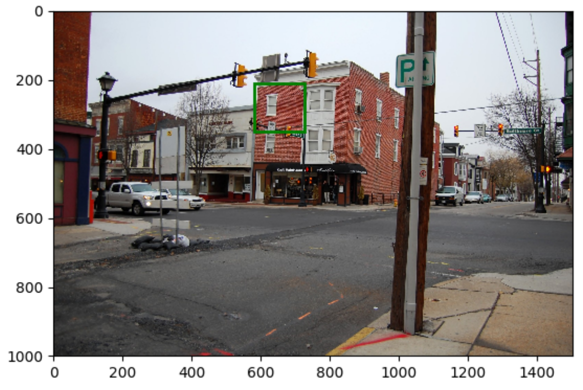

Fig. 6: Example 3

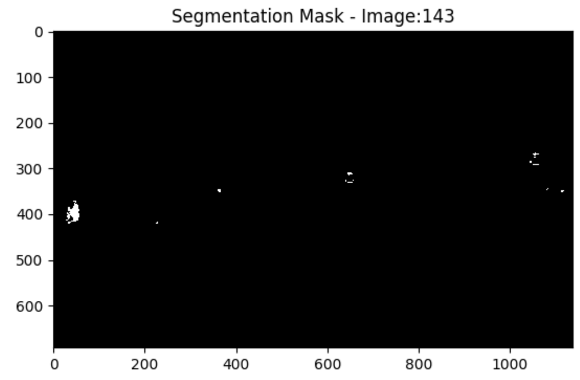

Fig. 9: Mask of Example 4

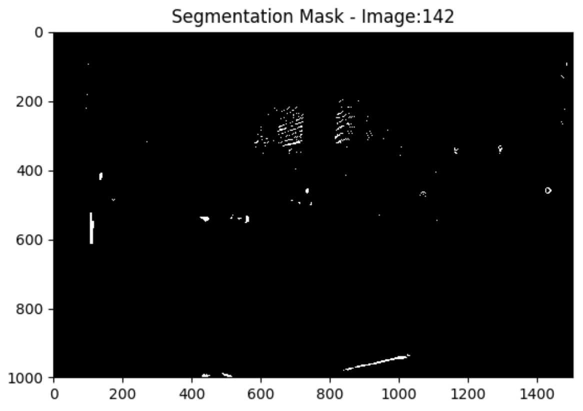

Fig. 7: Mask of Example 3

be because the training dataset is very small with about 200 images with only 97 of them containing stop signs. Also the training model might have overfitted for the same reasons.

To further improve upon this work, the training can be done on a larger dataset than the one used here. Also, to better detect stop signs in various lighting, we can use HSV color space or YUV color space which might give better results. To further improve the mask, we can try to implement various other supervised learning methods such as Gaussian model or Gaussian Mixture generative model.

ACKNOWLEDGEMENTS

I wish to acknowledge here that I collaborated with the following students while working on the project.

- Anirudh Swaminathan
- Shakeel Ahamed
- Venkatesh Prasad

Our discussion was purely about the outline of the project. We also hand-labeled the images together, and created the dataset together.