Using Naïve Bayes Model for color segmentation and bin detection

Hongbo Qian
Department of Electrical and Computer Engineering
University of California, San Diego
hoqian@ucsd.edu

***Abstract*—This paper presented approaches using Naïve Bayes Model for color segmentation and bin detection. In this work, First train on the provided labeled color dataset, and test the results through the validation set. Due to the abundant color data provided, a very high accuracy rate can be obtained in the end. After that, bin detection is performed in a similar way. The difference is that you need to extract color data from the training set yourself, and finally select the detected bin box through opencv, and the final accuracy rate reaches about ninety percent.**

***Index Terms*—Naïve Bayes, Machine Learning, Color segmentation, object detection**

I. Introduction

In recent years, technology related to image recognition has developed very rapidly, whether it is applied to the field of unmanned vehicles or screen unlocking, it has greatly facilitated our lives. In the field of image recognition, a very important branch is to recognize objects by color. This paper focuses on identifying the bins whose color is universal by matching the contour extraction of objects in the image. Although this paper only takes the bin as the research object, it can be extended to many aspects of life, especially for the recognition of objects with relatively fixed colors. For example, the identification of the number of products on the factory assembly line, etc.. At the same time, this technology can also cooperate with other image recognition technologies to improve the recognition accuracy.

An important point in this project is how to build a mathematical model based on a known color dataset, and obtain a set of model parameters through the training of the dataset. The obtained model parameters are used to verify the results of the test set. The Naïve Bayes classifier used in this article is very widely used in the field of machine learning. The classifier model assigns problem instances with class labels represented by eigenvalues, and the class labels are taken from a finite set. It is not a single algorithm for training such a classifier, but a series of algorithms based on the same principle. All model parameters can be estimated from the relative frequencies of the training set. A common method is maximum likelihood estimation(MLE) of probabilities.

Using the above model for the detection of bin can get very good results, because the color of bin can basically not have a particularly big difference. If we want to further improve the accuracy of the results, you can change the color space, such as converting RGB space to YUV space, etc., which can reduce the impact of sunlight on the surface of the object and the color of the surface of the object. After solving these problems, we can get a good detection result.

II. Problem Formulation

Given a training set of images containing bins in blue, bins in yellow or other objects like sky or swimming pool, the problem is to build a color classifier that could distinguish the bin in blue with any other colors. For the sake of simplicity, in this article, there are not many types of colors, only bin blue and not bin blue. If we want to get better results, we will add other colors for training. The accuracy rate should be further improved. The classes are therefore defined to be bin blue and not bin blue, for which y = 1 and y = 1, respectively. The parameters can be calculated using the following formulas during the training phase.

$$y = \arg\min_{\omega} p(y, X|\omega) \tag{1}$$

Each pixel can be classified for a discriminative maximum likelihood classifier with the following equation in the testing period.

$$y = \arg\min_{y} p(y, X^*|\omega^*) \tag{2}$$

In the above equation, ω is a vector, the dimension of which is 3 as in the RGB, YUV, and HSV color spaces, containing the parameter weights optimized for best performance. And X in the equation represents the dataset examples.

Naïve Bayes classifier uses a generative model for discrete labels and assumes that, when conditioned on y_i, the dimensions of an example x_{il} for $l = 1, ..., d$ are independent. According to the conditional probability formula, we can write the probability as follows.

$$p(y, X|\omega, \theta) = p(y|\theta)p(X|y, \omega) \tag{3}$$

According to the assumption that the conditional probabilities are independent of each other, we can further derive it.

$$p(y, X|\omega, \theta) = \prod_{i=1}^{d} p(y_i|\theta) \prod_{l=1}^{d} p(x_i|y, \omega) \tag{4}$$

The two probabilities in the above formula can be further sorted out to obtain the following results.

$$p(y_i|\theta) = \theta_k^{1y_i=k} \tag{5}$$

The Gaussian distribution is also written in mathematical form and formula form, and the specific results are as follows.

$$p(x_i|y_i = k, \omega) = \phi(x_{il}; \mu_{kl}, \sigma_{kl}^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{6}$$

The following formula can be obtained.

$$p(y_i, X|\omega, \theta) = \theta_k^{1y_i=k} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{7}$$

In this article, we only need to judge whether the probability of being blue is high or the probability of not being blue is high. Our prediction is the one with the greater probability.

$$\theta_+^{+*} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} > \theta_-^{-*} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{8}$$

The above is the part of the mathematical model used in this article. In order to facilitate the calculation, we can also take the logarithm of the above formula, which will not affect the final result.

III. Technical Approach

In this part, we mainly introduce how to use Python and packages such as OpenCV to convert mathematical models into codes and train known datasets.

A. Color Segmentation

In order to generalize the segmentation model, I use three classes including red, green and blue. The data in the training set has thousands of colors for each color. We associate the color with the label.

$$\begin{cases} Red = & 1 \\ Green = & 2 \\ Blue = & 3 \end{cases} \tag{9}$$

First, extract the provided color data set. Since they are all solid color images, there is no need to do more processing on the image. Just select a pixel and use its RGB value as our data. The next step in processing the dataset is to calculate the parameters of the data, including the prior probabilities, means, and covariance. Their specific calculation formula is as follows.

$$\theta = \frac{\sharp\ examples\ that\ are\ blue/not\ blue}{total\ \sharp\ of\ examples} \tag{10}$$

$$\mu = \frac{sum(X)}{N} \tag{11}$$

$$\sigma^2 = \frac{sum(x-\mu)(x-\mu)^T}{N} \tag{12}$$

The resulting mean and covariance result is as follows:

μ			
color channel	red(**R**)	blue(**B**)	green(**G**)
Red	0.75250609	0.34808562	0.34891229
Greed	0.35060917	0.73551489	0.32949353
Blue	0.34735903	0.33111351	0.73526495

$$\sigma_r^2 = \begin{bmatrix} 0.0370867 & 0.01844078 & 0.01863285 \\ 0.01844078 & 0.06201456 & 0.00858164 \\ 0.01863285 & 0.00858164 & 0.06206846 \end{bmatrix} \tag{13}$$

$$\sigma_g^2 = \begin{bmatrix} 0.05578115 & 0.01765327 & 0.00873955 \\ 0.01765327 & 0.03481496 & 0.0170234 \\ 0.00873955 & 0.0170234 & 0.05606864 \end{bmatrix} \tag{14}$$

$$\sigma_b^2 = \begin{bmatrix} 0.05458538 & 0.00855282 & 0.0171735 \\ 0.00855282 & 0.05688308 & 0.01830849 \\ 0.0171735 & 0.01830849 & 0.0357719 \end{bmatrix} \tag{15}$$

After calculating the above parameters, the values of the training, validation, and test results are substituted into Equation 7, taking the largest probability as the final result. Compare the predicted result with a known label to calculate the prediction probability.

B. Bin Detection

Bin detection has certain similarities with the color segementation mentioned above, but the big difference is that we don't have the bin blue data, we need to extract the data ourselves. The tool we use in this section is roipoly, which is a very effective tool in Python. It allows us to select the target pixel with the mouse and save the selected pixel. In this example, I picked 20 photos numbered 20-40 and manually picked blue and non-blue pixels in them.

Since the data read by OpenCV is in BGR form, we must convert the BGR data into RGB data. In this step, we can also directly convert it into YUV data for subsequent calculations. After obtaining the dataset, do the same as the color segmentation, calculating the mean and covariance. The results of this section are as follows.

μ			
color channel	red(**R**)	blue(**B**)	green(**G**)
Blue	36.4407839	72.2794173	166.418020
Not Blue	82.1419953	80.9434330	62.6142176

$$\sigma_b^2 = \begin{bmatrix} 1453.004481 & 1713.934563 & 833.148662 \\ 1713.934563 & 2297.152504 & 1285.90672 \\ 833.1486629 & 1285.906720 & 1493.04847 \end{bmatrix} \tag{16}$$

$$\sigma_{nb}^2 = \begin{bmatrix} 4912.367836 & 3935.912522 & 1882.473630 \\ 3935.912522 & 3745.267471 & 2125.705807 \\ 1882.473630 & 2125.705807 & 2296.945556 \end{bmatrix} \tag{17}$$

After obtaining the above parameters, we can judge the color of the pixels in the dataset. If the calculated probability that it is blue is greater, it is set to 1, and if it is not blue, it is 0. For every pixel in each image, we implement this algorithm to obtain a color segmented image. We get the binary image in which shows whether it is blue in the image. The specific result will be shown in the next part.

The binary image mask obtained from the previous operation contains some random noise, which may result in poor performance for the detection algorithm. Additional preprocessing operations are needed to remove the random noise. Morphological methods are good methods including erode and dilate. It is useful for removing small noise from an image and sharpens edge outlines. These operations have been encapsulated in the function. We can easily call it directly.

After removing the random noisy pixels, we now could find all the contours of bin. This operation could be done with the opencv2 function $cv2.findcontours$. It would return all the points on the contours of each instance. However, this function must input the binarized image, so the image is binarized first, and then it is entered into the function. With another opencv2 function $cv2.boundingRect$, we could get the detail of the box. The details we can get are as follows.

- 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

The area of each contour could be easily obtained from the function $cv2.contourArea$. Having obtained the area of the rectangle, we can further calculate its proportion of the entire image. The reason we want to get the area proportion and its width and height is to filter out the blue that is not belong to bins. The blue in the picture is not necessarily all bins, such as blue sky, blue swimming pool, etc. In the test, if you do not add parameter restrictions, the final detection result will have many other objects framed, which is not the result we want to get. After some parameter debugging, I finally found a more suitable constraint as follows.

- area ratio > 0.005
- $\frac{width}{height} > 1$ and $\frac{width}{height} < 2$

That's most of the bin detection, and the rest is testing with a dataset. The specific results are shown in the next section.

IV. Results and Discussion

A. Color Segmentation Results

There is a very consistent with the color data, my prediction in the train set and validation set is 1, but in the autograder's test set the result is 0.93, there are still some image calculation problems. I personally think that the reason for this problem is that there are some confusing pictures in the test set, or the training data entered by the book is not enough, so that the results have some deviations, but more than 90% is also an acceptable result.

B. Bin Detection Results

Using 10 images from training and 10 images for validation, the accuracy is given below:

$$Accuracy = \frac{17}{20} = 85\%$$

The results of this part are still lower than the color segmentation, after all, this part is more difficult. It also has to do with my test set data. From the results of the binary plot, the effect of identifying blue is still ideal, the noise is relatively small, and the data that is blue but does not meet the requirements because the aspect ratio or area ratio does not meet the requirements. The problem that remains is that some of the data on the ground that is not blue and part of the car are identified. I tried many ways to solve this problem and didn't get particularly good results. I tried changing the aspect ratio parameter, changing the area ratio parameter, and also tried several sets of collected data, and also tried to have more blue data than non-blue data, and tried to use the YUV color space. Among them, the YUV color space should be able to improve the final result to a certain extent, but due to the time relationship, it is not completed. The result of the RGB color space is more than 80 percent, which is also an acceptable result.

I think the other ways to improve the end result are:

- Use other color spaces to reduce the effect of daylight on color.
- Increase the training data, in order to shorten the training time this time, there is no particularly large selection of training data.
- Adding an unexpected color other than blue as a label should improve the final accuracy.
- Further optimization of parameters.

The results of the test are shown in the picture that follows.

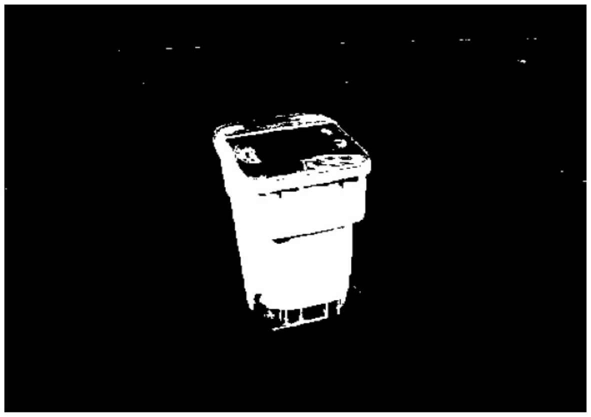

Fig. 1. binary result 1

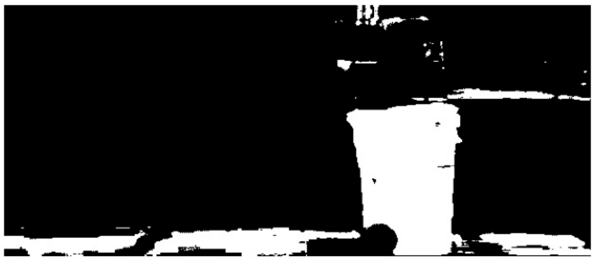

Fig. 3. binary result 3

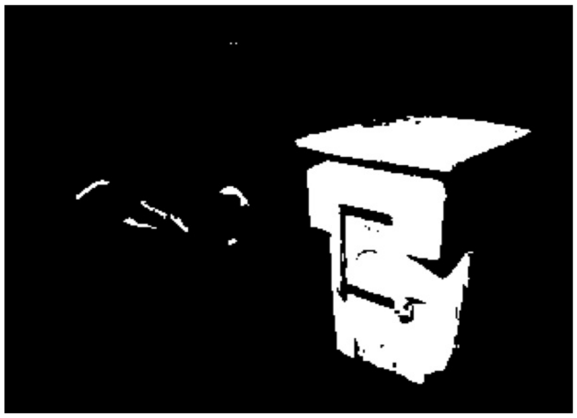

Fig. 2. binary result 2

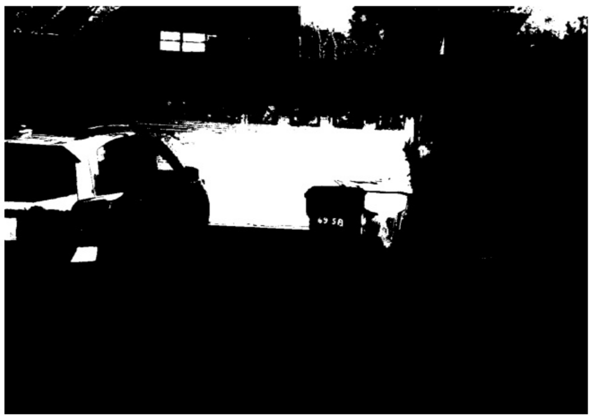

Fig. 4. binary result 4

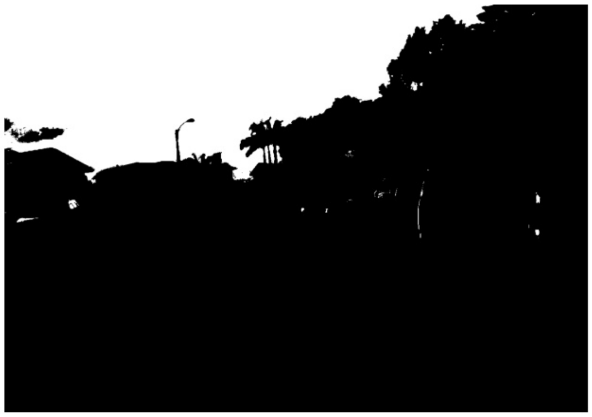
Fig. 5. binary result 5

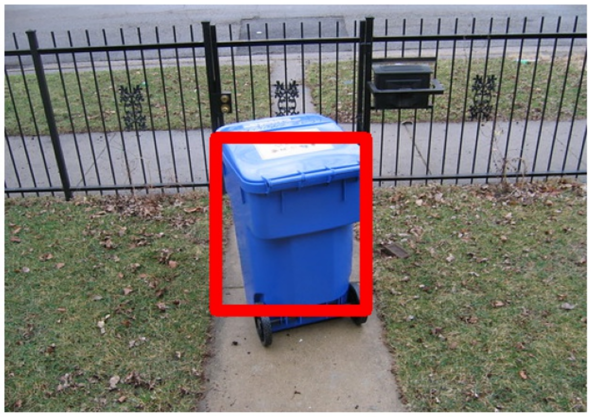
Fig. 7. validation result 1

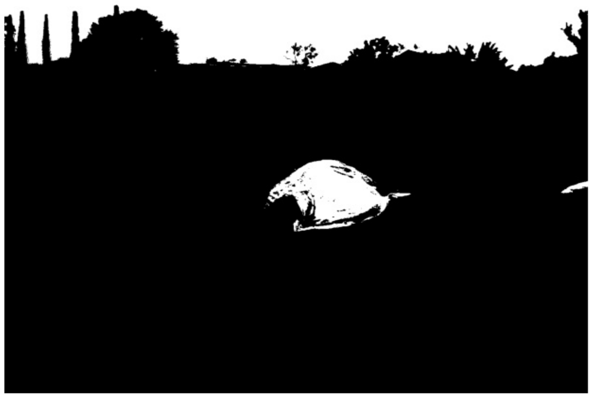
Fig. 6. binary result 6

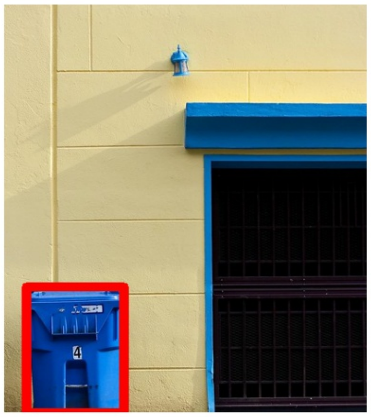
Fig. 8. validation result 2

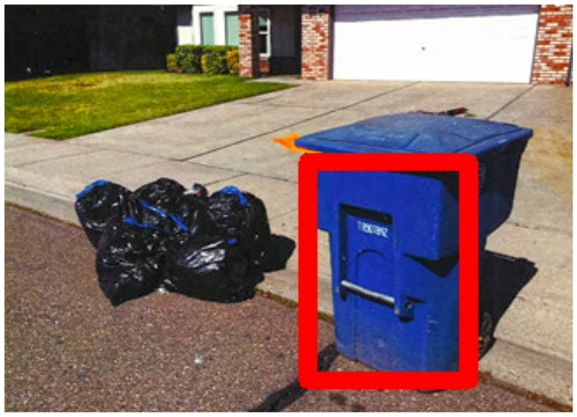
Fig. 9. validation result 3

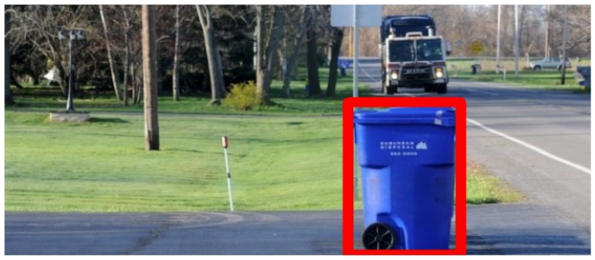
Fig. 10. validation result 4

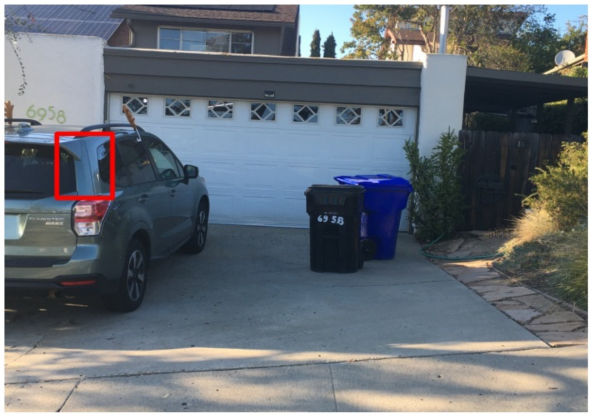

Fig. 11. validation result 5

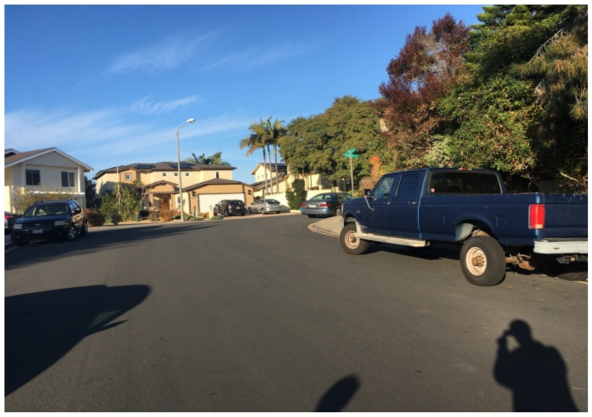
Fig. 12. validation result 6

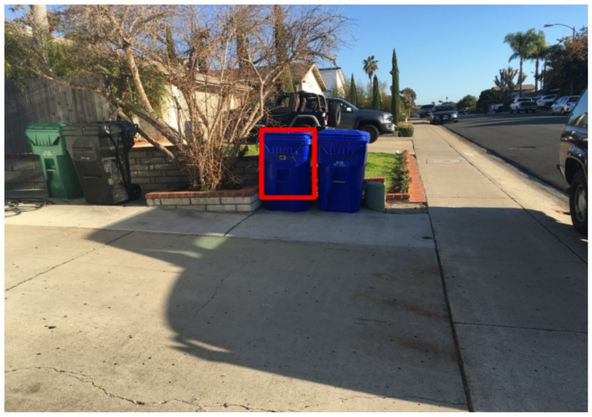

Fig. 13. validation result 7

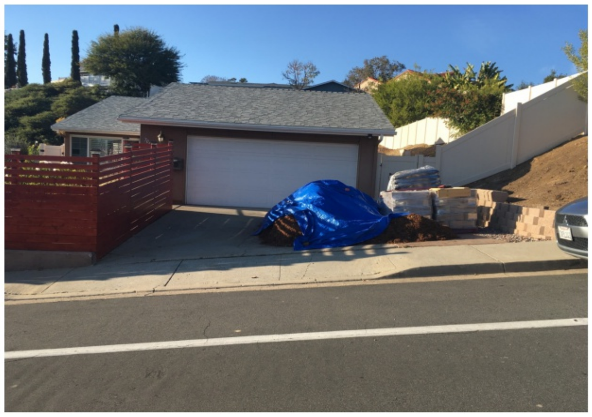

Fig. 14. validation result 8

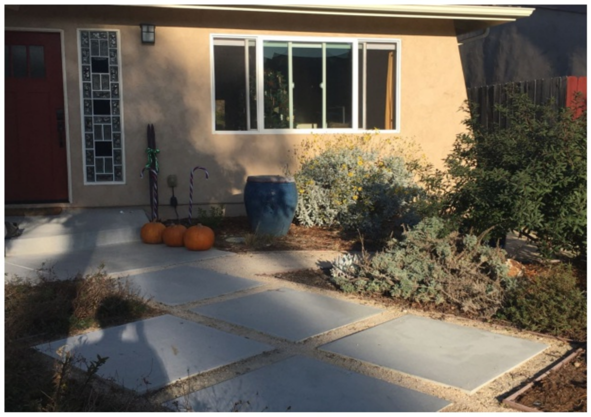

Fig. 15. validation result 9

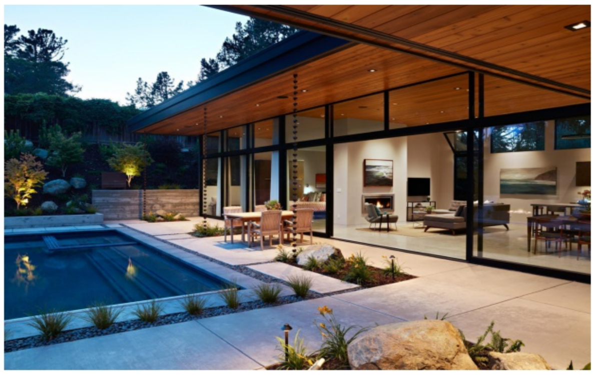

Fig. 16. validation result 10