CS 6643 - Computer Vision
New York University
New York, NY 10003
(212) 998-1212

Car License Plate Detection

Shanyi Jiang (sj3358) & Sheng Tong (st4048) & Haotian Wu (hw2716)

15th May 2021

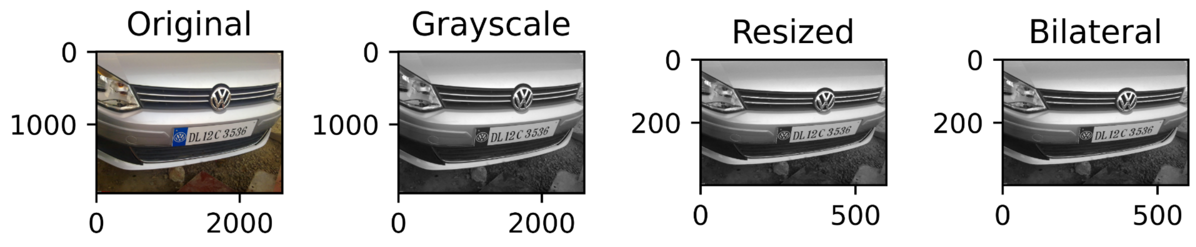
1. Introduction

The application of artificial intelligence and computer vision has greatly improved every aspect of our lives nowadays. And as an important branch, autopiloting has become more popular than any other period of time in our social development. Specifically, a reliable tool that allows machines to detect the car license from an image will be our main focus in this report. So, the goal of this project is to locate and identify car license plates from an image based on some computer vision techniques. And in this report, we will introduce methods, performance and future improvement of our program.

2. Approach

1. Image pre-processing

When we receive a raw image that may contain a car license, we convert the image into grayscale and resize the image into a 600 x 400 scale at first. Then we choose to apply bilateral filtering to remove any noise in the image.

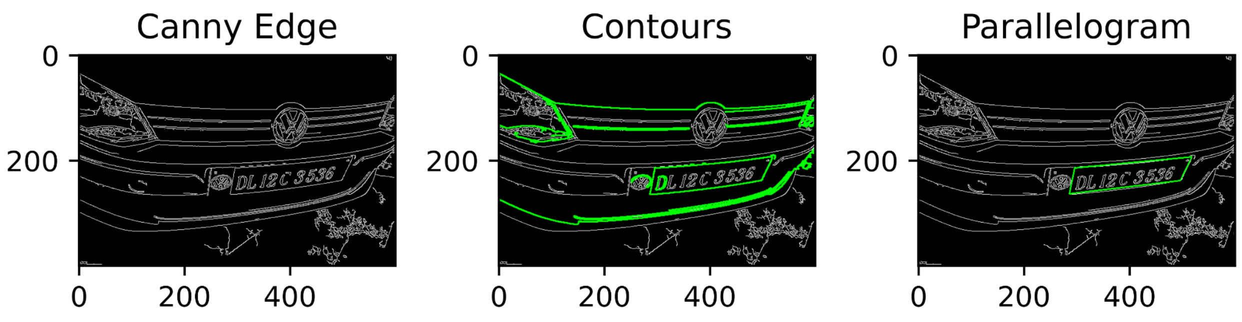


2. Car license localization

In order to localize the car license plate in the image, we convert the problem into three parts.

1. **Find the edges of the image.**
2. **Find all closed contours based on these edges.**
3. **Find parallelograms from these contours.**

By using canny edge detection, we are able to find edges of the image properly. And with the help of the cv2.findContours function, we can detect any possible contours of the image. Then based on these contours, we use the cv2.approxPolyDP function to estimate the number of corners of these contours. Since a parallelogram has 4 corners and the car license is usually the biggest parallelogram in the image, we only keep the biggest contour with 4 corners in the image to be the location of the car license.



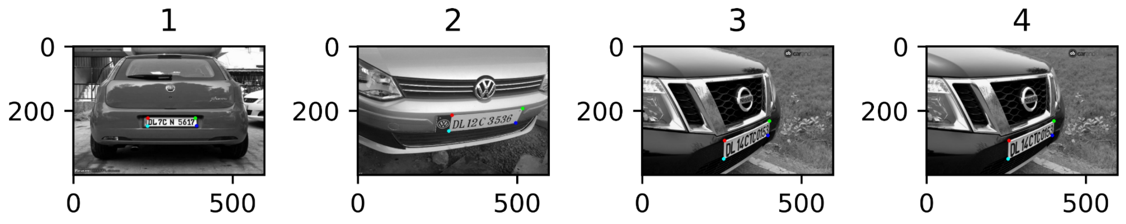
3. Car license rectification

Once we have the locations of 4 corners of the car license, we are able to transform the car license in the image into a rectangle. However, there are two major problems we have encountered during this process.

1. **Unsorted corner points.**

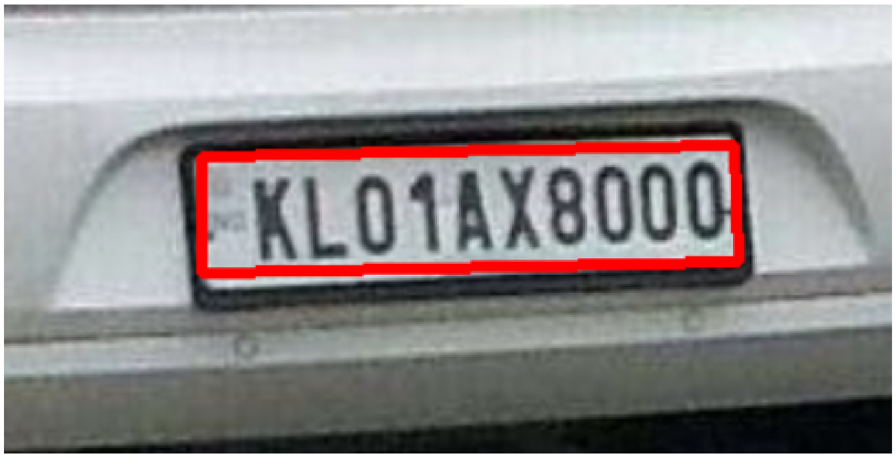
   4 points returned by cv2.approxPolyDP are not necessarily sorted. In order to have a successful transformation, we need to know which point is the top left corner, which point is the bottom left corner of the car license, etc. According to many observations, we noticed that since the car license is a long rectangle, it is

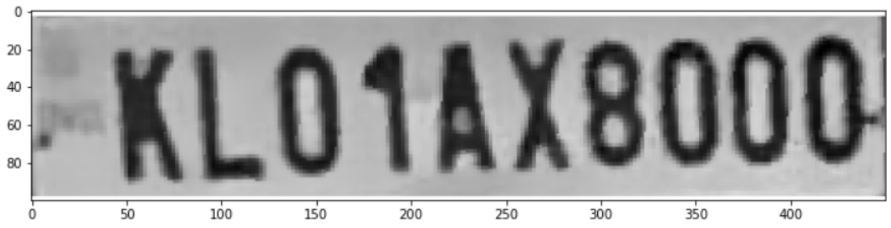
easy to tell which two points belong to the left group and which two points belong to the right group based on their x-Axis distances. Then we can identify top corner points and bottom corner points from each group via checking their y-Axis distances. This method turned out to be very reliable after applying it to many cases.



2. Inappropriate transformation due to inaccurate point detection.

However, the locations of 4 corners of the car license returned by cv2 function approxPolyDP( ) are not perfect, sometimes they cannot exactly cover the whole license plate, then after the transformation, then the license plate will not be successfully rectified:

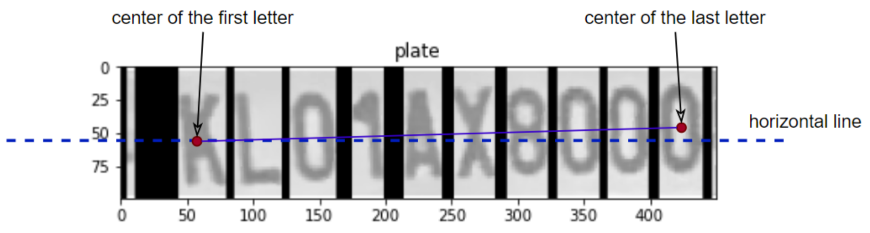


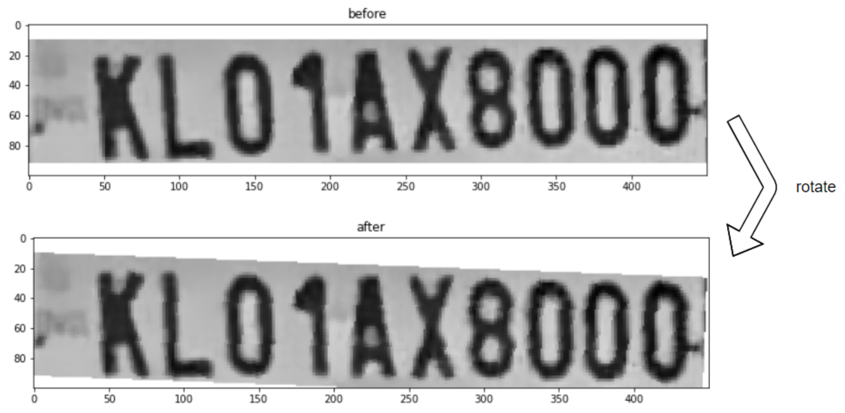


Here is an example of unsatisfying license plate detection. After we transform the area of the license plate into a rectangle, we expect the letters on the license plate to be in upright positions. But with inaccurate corner points, our transformation will cause unexpected rotation or shearing to the actual license plate area.

Because letter recognition using the template matching method is sensitive to the angle of the letter, if the license plate can not be adjusted to an upright position, there will be many mismatches between similar letters. So, we must perform a secondary rectification.

Assuming that only unexpected rotation occurs due to inaccurate license plate locating, we can perform an extra rotation to bring the license area back to an upright position. And because all letters share the same height on the same plate, we can find the angle we need to rotate by finding the slope of the line that links the centers of the first and the last letter. We can perform a primary letter cut to find the location of the first and the last letter (we will introduce how to perform letter cut in the next few pages).





Finding the slope of the line is same as finding tan α where α is the angle between the line and the horizontal line. We can then compute sin α and cos α for performing rotation for the secondary rectification.

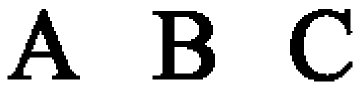
Because the rotate angle for the secondary rectification is an approximation, we have set a parameter to adjust the power of secondary rectification. With this additive rotation, we are able to cut letters out in relative upright positions from the license plate, which makes our template matching possible.

4. Letter and number template matching

1. Binary image processing & letter cut

After we rectified the license plate, we now need to collect templates for each alphabet and number

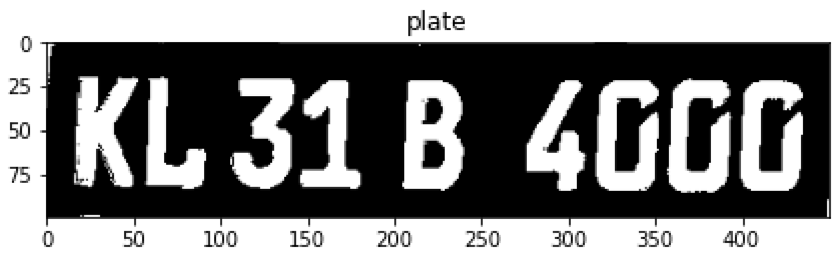
We first found out that Indian license plates are using Times Roman and Arabic. So, we opened up a Word document and typed A to Z and 0 to 9 using Times Roman, and saved them as our first-try templates. Below are some examples:

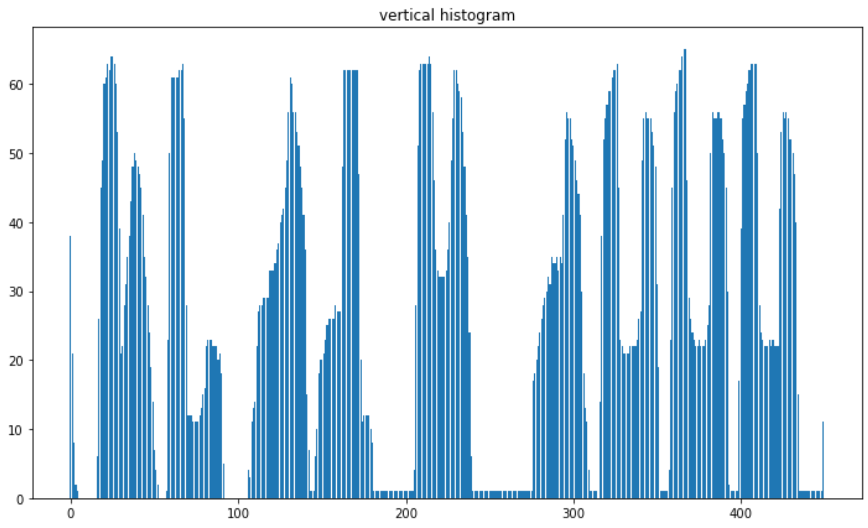


However, we found out that Indian license plates are not exactly using Times Roman and Arabic. So we decided to use the license plates that are rectified in the previous step to get our templates. And we would collect several templates for each alphabet letter and number.

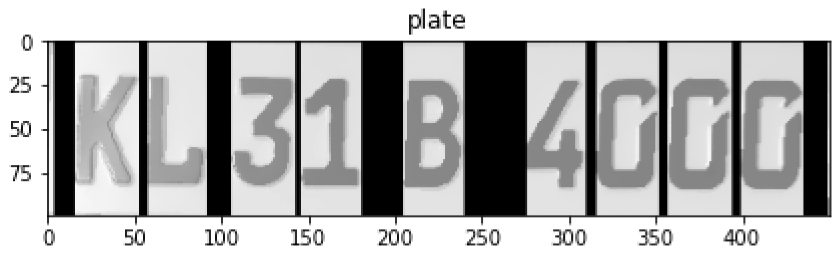
To detect each alphabet letter and number, we will use a “vertical histogram” to detect the change in vertical sum of intensities of our rectified plate. Below is an example:

Firstly, we set up a threshold to turn the plate into a binary image, so we can get rid of the influence of unnecessary pixels, then we will calculate the vertical sum of intensities of this image, which will form a “vertical histogram”.



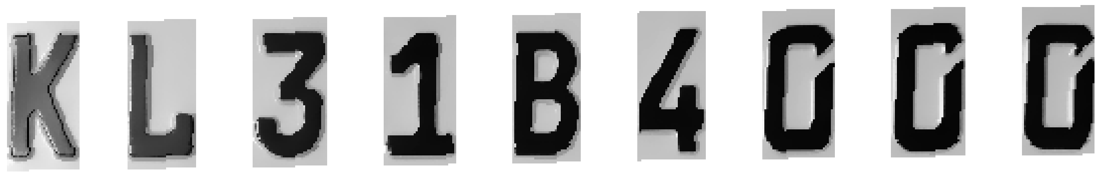


Because a license plate usually can be separated as foreground letters and background color (usually in white), and those letters will not intersect with each other. So assume that the license plate is in an upright position, after making the plate a binary image, there will be apparent black gaps between letters, and in the vertical histogram we can find evident peaks and valleys. And we can make use of this feature to crop all the letters out. We can draw vertical black lines on the original license plates as cuts based on the vertical histogram by finding a vertical sum threshold, this threshold can not be 0 because there are still some noise points on the binary image, it can not be the average of the histogram either because it will create many mistake cuts (for example, cut letter 'H' in half). Through experiments, we find out that finding a threshold that covers about 25% of columns of the image is the best choice. And this can be easily implemented using numpy functions.



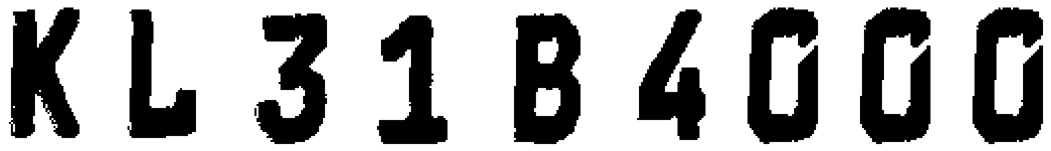
After separating the plate, it is easier for us both in getting the template, and also easier for us to do template matching.

And we get the following:

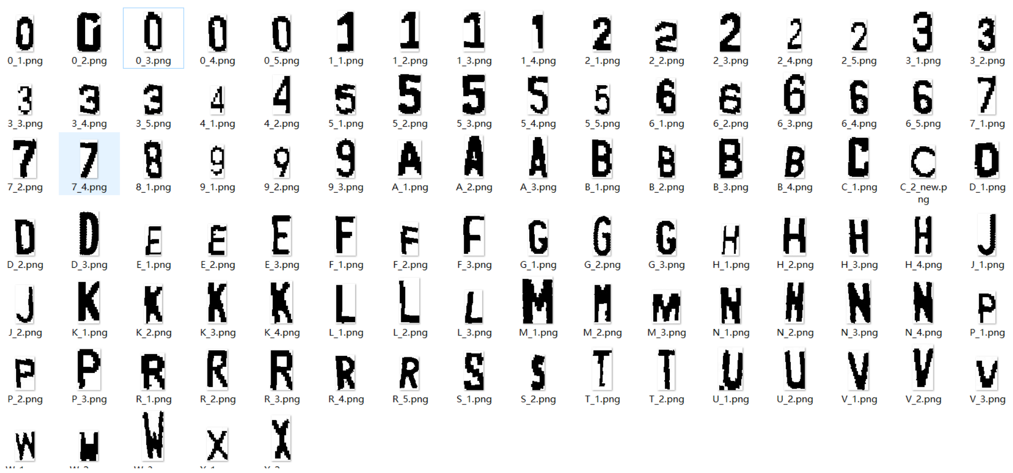


To get a good template, we still need to do several steps. We needed to get rid of the top and bottom blank space by setting different parameters for each alphabet, and then set the background to 0, and the alphabets to 255.

Below are the final templates:



And we have a list of templates:

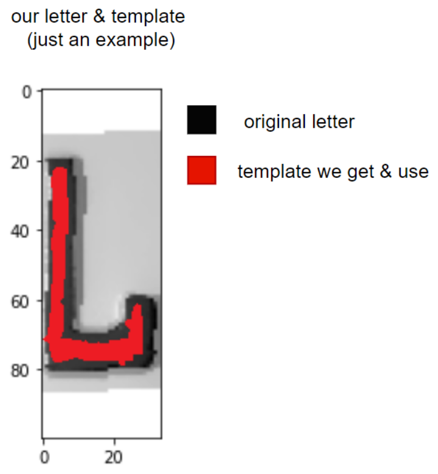


2. Template matching

After cutting all the letters into pieces in the license plate and getting all templates that we need, we will perform template matching to find out which letter one piece actually is, we will match one piece of letter with all our templates and find the letter with the highest matching score. And there are several special features for our template matching process:

1.background value of the piece and the template

Usually when performing template matching, we will set the value of background as 0 for the image we want to match and set the value of background as -1 for templates, so the matching score will decrease if there are foreground pixels of the image that match with the background pixels of templates. However, in our project, we collect our templates using our own methods and have manually edited them, which makes our templates thinner than the letters in our dataset, which means we will have a lot of such kinds of score penalties. This problem might cause unexpected mistakes because the template of the biggest letter might get a higher score than the correct letter just because it has fewer background pixels.



So, instead, we set the value of background of templates as -1, but we also keep the value of background pixels of matching pieces to be a negative value so that we can prevent “partly matching”, for example, for letter D, template ‘1’ might get higher score than template ‘D’. We find out that setting a value as -0.2 for a piece background can get best average accuracy by experiments. By the way, to prevent background matches with background, we have set that in the cross correlation process, the product of two negative values will be zero.

We find out that these small changes dramatically improve our matching accuracy. When letters are cut and collected in upright positions, the matching process seldom makes mistakes, unless two letters are very similar (for example, 8 and B, 6 and G, 0 and D and etc. after resizing).

2. Multiple templates for single letter, choose highest score

We have about 100 templates, and each piece of letters from the plate needs to match with all of them. We compute the highest value of the cross correlation result as the matching score for one template, and we will choose the template with the highest score as the matching result.

However, template matching is slow, and we have to perform it 100 times for one single letter piece. To save computing time so that we can get results in an acceptable time, we need to down sample the matching pieces and templates. Setting a 4x down sample rate will ensure us to get the matching result for one license plate in about 60 seconds while keeping acceptable accuracy. We make the down sample a parameter so we can adjust it, usually a lower down sample rate results in more accurate results but it costs much more computing time.

3. Performance & Accuracy

The process of cutting letters out, which includes license plate finding, primary cut, secondary refinement and secondary letter cut, is very fast and can be computed in a few seconds. And as long as we can find the best parameters, we can get nearly perfect letter cuts. And because it's fast to perform this step, it's easy to find better parameters with several tries in a short time.

The process of template matching takes much longer time. When we set the down sample rate as 4x, we need about one minute to compute the result out. If we choose a lower down sample rate, the accuracy will slightly increase. But more computing time is needed. There are not many parameters that need to be adjusted, most of the time using their default values work well, so users do not need to try many times to find good parameters.

For the all 97 test images, the program can find possible contours from 54 of them. And from these 54 images, the program succeeds in finding locations of car plates in 44 of them. And from these 44 images, 40 of them are successful to crop the letters out by the program. For the template matching results, most letters and numbers have an average success rate of 65.2%. For the letters like 'B', '6', 'G', '0' and 'D', the average success rate is much lower than other letters.

4. strengths and weaknesses

Strength:

1. Suitable for any Indian License plate
2. Can detect license plates even though it's tilted
3. Have a high accuracy if we can detect a license plate in an image & find proper parameters for our functions

Weakness:

1. Cannot have any object that has the similar shape as license plate, otherwise we might not detect the license plate
2. We have some difficulty in identifying B and 8, 6 and G.
3. Not guaranteed to detect the license plate in the image
4. May need several tries to find the best parameter sets for each license plate.

5. Future works

1. Plate localization is unsatisfying, and makes results of our program unstable, we might want to seek out different methods to detect plates.

2. Our program now is only able to detect a single row of letters with known fonts. There are many situations where there are multiple rows of letters or letters with different fonts or symbols in the plates. Better methods need to be implemented to handle such situations.

3. We have made relatively great progress in cutting letters, if we use deep learning methods to recognize those letters we might get great results. Because template matching is sensitive to letters' positions and sizes while neural networks can overcome such problems. And meanwhile, a neural network might have a hard time recognizing all the letters on a license plate at a time, but it will be much easier if recognizing letters one by one. In other words, our work can help improve the performance of deep learning methods.

6. References

1. *https://www.kaggle.com/ckay16/indian-number-plate-detection*

2. *https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/*

3. *https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c*