Illinois Institute of Technology

CS512 Computer Vision

Final Project Report

Ai Jiang(A20384274) and Li Xu(A20300818)

November 20th, 2017

1 Problem Statement

In this project, we implement an algorithm to detect and recognize playing cards. One deck of modern playing cards contains 54 different cards with different ranks and suits. To simplify our problem, two joker cards are not considered. The algorithm is achieved by edge detection, perspective transformation and template matching. The orientation of the playing card is an important factor in recognition and affects the result. To make our algorithm more robust, rotational and angle invariant feature is added to our algorithm. Our work is inspired by [3] and further implementation and improvement is done by ourselves. We implemented rotational invariant mentioned in paper [2], and achieved angle invariant by perspective transformation.

Following the instruction above, our playing card recognition task will divide into two major parts: Detect Card and Match Card.

2 Applied Methods

In this section, we will introduce the principle of our main methods in our work.

2.1 Method for Card Detection

We conclude the major steps to implement card detection by several steps below.

1. **Find contours and detect the card contour:** Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. We use binary images for obtaining a better accuracy. So before finding contours, we apply threshold on our original image. For our model, we only interested in the card outer edge contour, which is the largest contour line from all the detected contours.

2. **Get the right position of card:** Our baseline method performing transformation in this part use a four point perspective transform to obtain a top-down view of an image, which provide four reference points that are ordered as defined. We also need to determine the dimensions of our new warped image. We determine the width (the largest distance between the bottom-right and bottom-left x-coordinates or the top-right and top-left x-coordinates), and the height (the maximum distance between the top-right and bottom-right y-coordinates or the top-left and bottom-left y-coordinates.) of the image. These consistent defined ordering representation will allow us to obtain the top-down view of the image by using perspective transform function.
   For the purpose of achieving rotation invariance, we modified this baseline method to create an array of corrected listing points based on different situations of captured card. If the card is 'diamond' oriented, a different condition has to be used to identify which point is top left, top right, bottom left, and bottom right. More specifically, if furthest left point is higher than furthest right point, card is tilted to the left. If furthest left point is lower than furthest right point, card is tilted to the right.
   After getting the correct ROI, we perform perspective transformation and got our corrected top-down image that could be stored for future use.

2.2 Method for Card Matching

In our work, we applied a multi-scaled template matching method for matching the corrected preprocessed top-down card image with our collected training card image set, which contains all templates for training our model.
Our baseline method for matching a test card image is calculating the absolute difference between test card image and templates, then matching the test image card with the template with minimum difference. For a better matching result, we applied a standard templet matching that are sensitive to the dimensions of templates and the dimensions of the test card image.Template matching is translation invariant, in order to increase the robustness, we further implement a multi-scale template matching method. We first loop over the test card image at multiple scales and apply template matching. The key idea of our method is to keep track of the matching with the largest correlation coefficient (along with the x, y-coordinates of the region with the largest correlation coefficient). We looping over all scales of the test card image and matching with one template, we take the label of the template with the largest correlation coefficient matching region on the test card image after matching all templates in our training set.

3 Implementation & Results

In this section, we will introduce the implementation details of our work. Our program are running on Mac OX, using camera to take card pictures for training and test data sets. We use a deck of playing cards as our target pattern. To run our program, you may need to first install a support module *imutils*. The overview of our work are shown in Fig. 1.

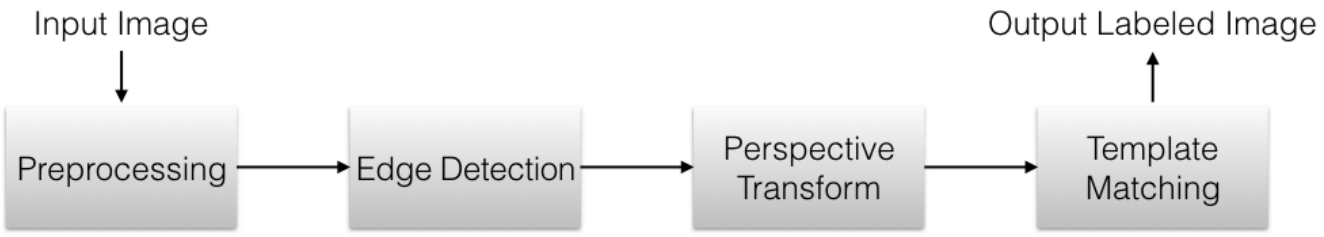

Figure 1: Overview of Playing Card Recognition

3.1 Data Collection

Fig. 2. shows the card image we collected for training and test our model.

1. Training set: Capture 52 playing cards images and segment the left top corner of each card to extract rank and suit. Grayscale and blur these segmentation images and save them as our templates.
   At beginning we use camera to capture template images, however, result is affected by environment variables like illumination, intensity and orientation. After changing the template images captured by scanner, we get a more uniform training set.

2. Test set: To prove the robustness, we use regular camera to take card images under different illumination, intensity and orientation.

(a) Training Data (card template)

(b) Test Data (input card image)

Figure 2: Collected Data Set

3.2 Card Recognition

1. Card detection: We use $cv2.findContours$ function in OpenCV which modifies the source image. So we store our source image separately before finding contours. Besides, in OpenCV, finding contours is somewhat finding white object from black background. Thus, we capture our test card image on a black background.
   We use $cv2.getPerspective$ to transform. This function requires two arguments, $rect$, which is the list of four ROI points in the original image, and dst, which is our list of transformed points. The $cv2.getPerspective$ transform function returns M, which is the actual transformation matrix. The result contour for card image is shown in Fig. 3. After applying our rules to make the card into right position, we get the result shown in Fig. 4.

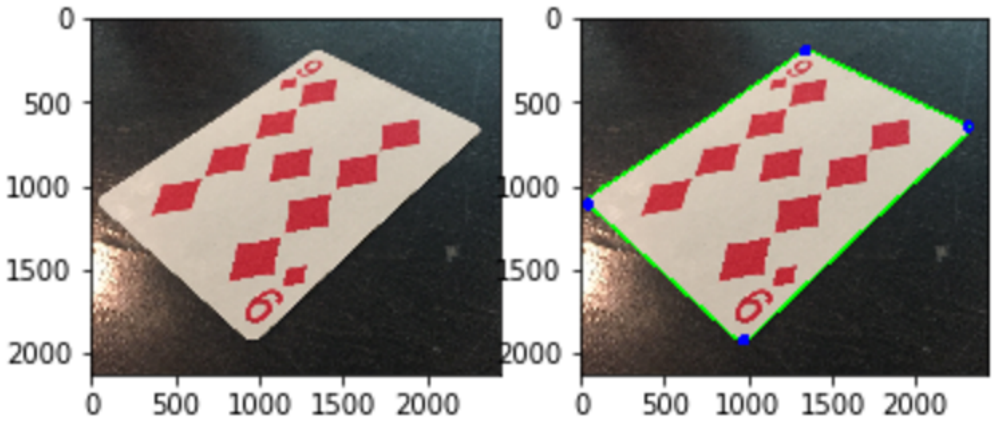

Figure 3: Find Contour

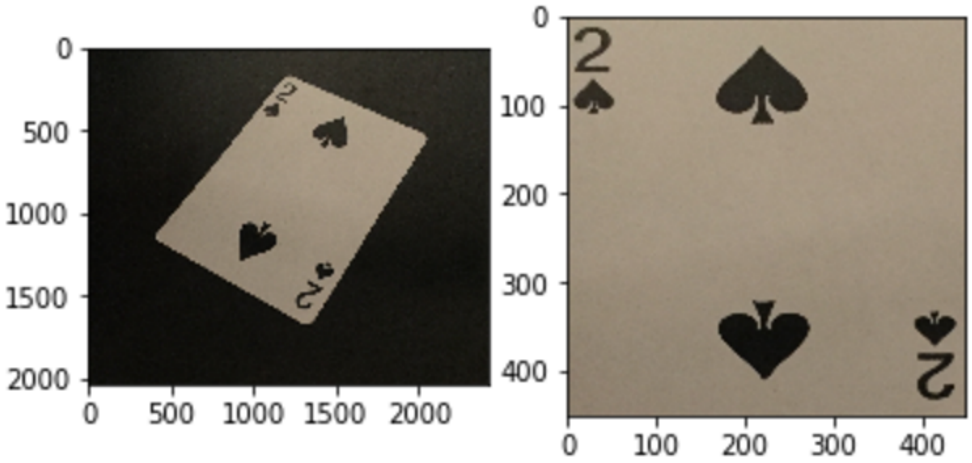

Figure 4: Find Card

2. Card matching: Template matching is a method to locate the template in a larger image. The basic idea of template matching is using template to slide through the entire input image and compare the similarity. In this part, we use OpenCV build-in function $cv2.matchTemplate()$. This function returns a grayscale image where each pixel represents the similarity between template and input image. When calling the $cv2.matchTemplate()$ function, we use $cv2.minMaxLoc()$ to find the max value of the result, which can be considered the similarity of two images. Iterate the input image with all training set images to find the best match, which is our recognition result.

3.3 Recognition Results

Fig. 5,6,7,8,9 shown the results of our test card image. The left is the input test card image, the middle image is the detected card, the right image is the recognition results with the matching label shown on the test card image.

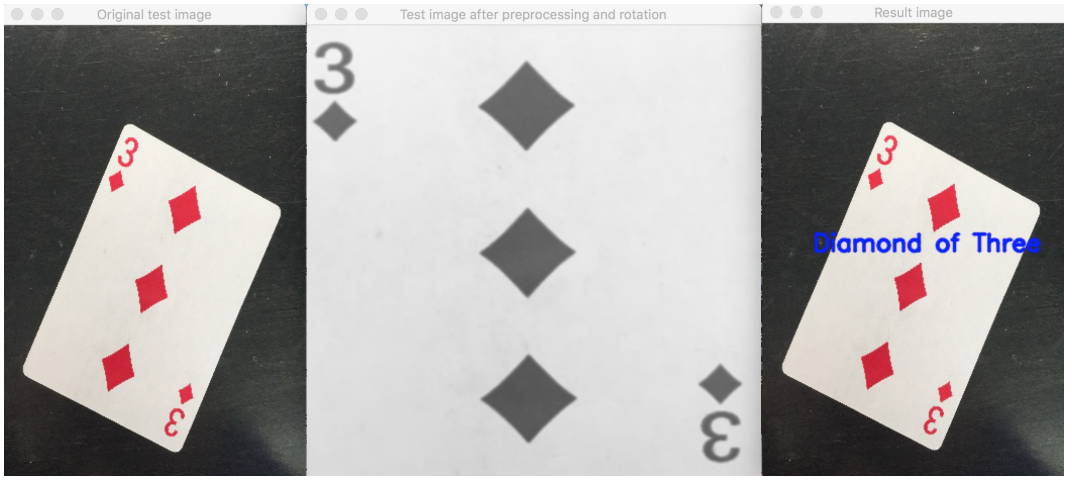

Figure 5: Test Result

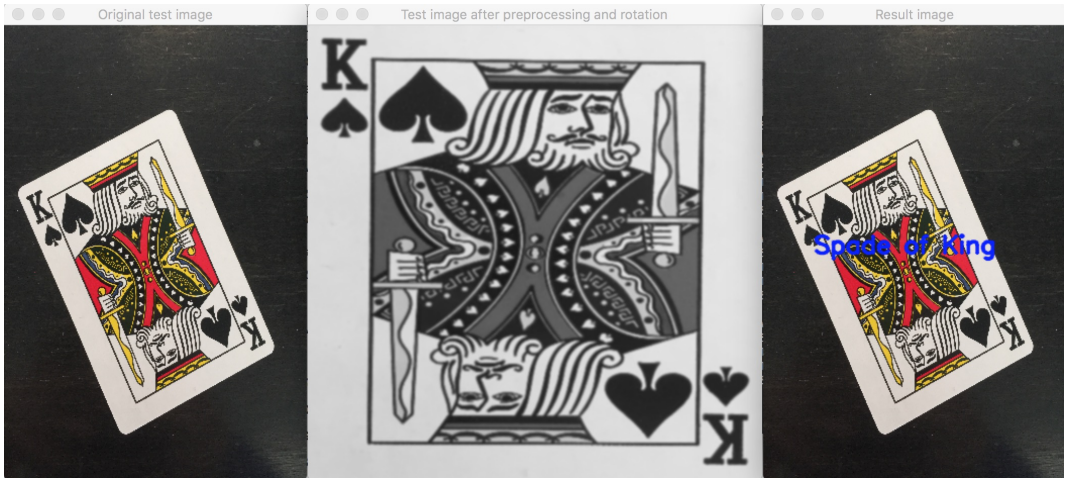

Figure 6: Test Result

Figure 7: Test Result

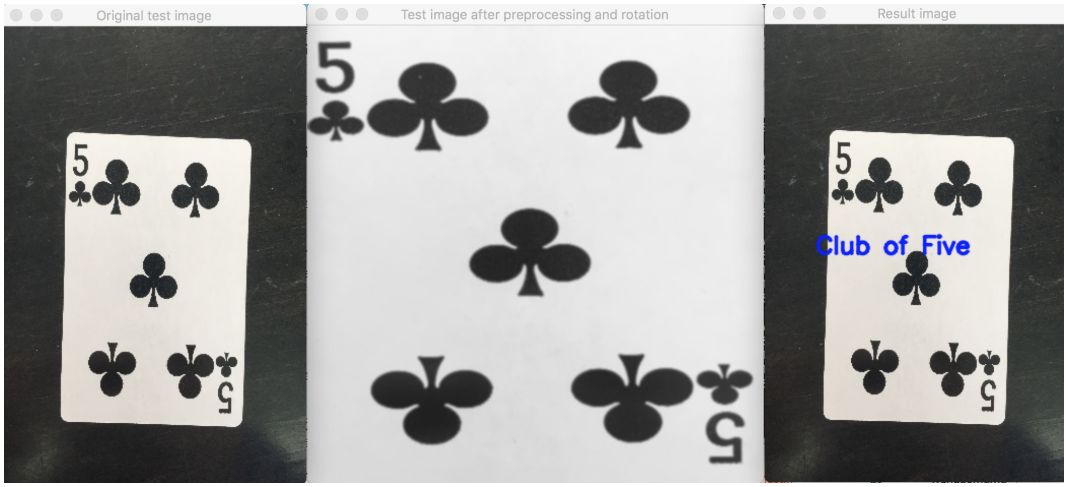

Figure 8: Test Result

4 Discussion & Future Work

Fig. 10 shows a false result that our model produced. The labeled is closer to the actual result, since '8' is similar with '3'. Currently, our model could achieve around 80% accuracy on our test card image.

Figure 9: Test Result

Future work can be done to improve the match process. Instead of using template matching, we can use Scale-Invariant Feature Transform (SIFT) and Histogram of Gradients (HOG) to improve robustness. Using these feature matching techniques, rotation to the same orientation of training set is no longer needed. The input file can be improved to capture frames from a video stream to achieve a real-time playing card detection and recognition.

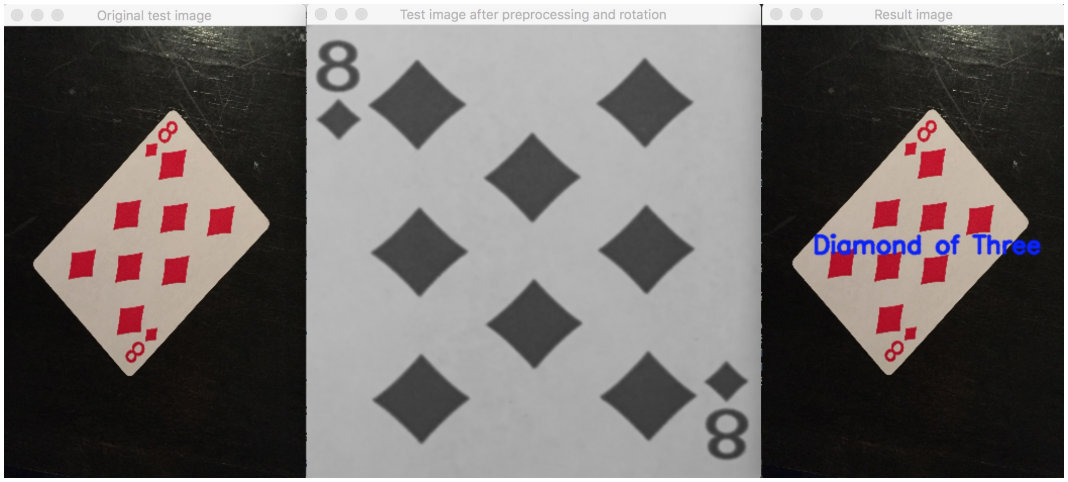

Figure 10: Test Result

5 Reference

[1] J.Pimentel, A. Bernardino, A Comparison of Methods for Detection and Recognition of Playing Cards

[2] C.Zheng, R Green. Playing Card Recognition Using Rotational Invariant Template Matching

[3] https://github.com/arnabdotorg/Playing-Card-Recognition

[4] https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

[5] P. Martins1, L. P. Reis, and L. Te'ofilo, Poker Vision: Playing Cards and Chips Identification based on Image Processing

[6] D. Brinks, H. Texas Hold'em Hand Recognition and Analysis