Computer Vision & Object Recognition: Playing Card Detection

Brad Gray, The University of Texas at Austin, Software Engineering
Jon Reynolds, The University of Texas at Austin, Computer Engineering

February 2015

Introduction

Using a combination of original code, code provided in the matlab libraries and documentation, as well as code provided in class, we developed a program in Matlab to identify captured images of playing cards. Our algorithm operates in the three main stages: segmentation of the symbols on the card, isolation of the individual symbols to extract features, and card recognition based on these features. Feature sets were built with less than half of the provided set of training cards, yet testing proved successful on all 32 training images. Additionally, our program successfully identified the provided 32 test images.

Methods

Segmentation: Detect the Card Region

The first stage of our program is centered around segregating the important symbols on the playing card, like numbers and suits, from the background and other minor features on card. We achieve this through a varying degree of thresholding, RGB normalization, primary color detection, and binary imaging.

Through trial and error, we found that the black and red cards reacted differently in each stage of our recognition code. For instance, RGB normalization works wonderfully on red cards, guaranteeing almost perfect thresholding, whereas black cards become almost impossible to threshold once normalized. As a result, we determined that identifying the color of the card was an important task early in our algorithm so that we could apply appropriate imaging techniques. To dictate the color of the card, we create a binary image from the red spectrum of the original image, then count the number of bits that are are "1" in the binary image. The tally represents the number of pixels in the image that are mostly red. If this count is high enough, we consider the card to be red; otherwise, the card is black.

If the card is red, then RGB normalization is applied. This process eliminates variation in intensity across the photograph in all color spectrums, reducing the chance of false thresholding. If the card is black, we eliminate small artifacts in the image using morphological opening and amplify the difference between the dark card features and the light background by adding the background to original image.

Next, our code creates a binary image using segmentation. First, the original picture is converted to grayscale. The value of each pixel is then mapped to a histogram. A threshold is calculated based on the shape of the histogram and the color of the card. Using the Matlab function im2bw() and our threshold, our image is converted to binary format, with pixel data equal to 1 belonging to an object and 0 belonging to the background. The resulting image contains a primarily black image with white objects throughout that are shaped like the symbols on the card.

Isolation: Describe the Card Features

Once we have produced a binary image, we isolate the white objects from the black background, make inferences about the card from their positions, and extract information from their shapes and compositions.

The objects are easily isolated with the use of the Matlab bwlabel function, which provides an array in which the objects are identified by unique integers. Using this array and the function called regionprops, we draw bounding boxes around each individual object. Each box's edges are tightly bound to its companion object and its vertical and horizontal edges are parallel to the images vertical and horizontal edges, respectively. Our code uses the bounding box in identifying the suit of the card, which will be expounded upon later.

Following the boxing of objects, our algorithm counts the number of meaningful symbols in the binary image. The particular playing cards used in this lab have N + 4 symbols each, where N is the number of the card. Our algorithm counts the number of total symbols and subtracts four, thereby determining the number of the card without using classification techniques. When initially using this approach, we faced issues with counting false symbols (i.e. dark shadows) that couldn't be eliminated with thresholding and RGB normalization. To solve this problem, we wrote a function called getSymbolCount, which counts the symbols but discriminates against objects in the binary image that are too far away from the majority of objects (i.e. shadows in the corners). Firstly, the method calculates the mean centroid of the objects, which represents the weighted center of the symbols. Next, this function finds the distance between each object's centroid and the average centroid. If the center of the object is too far from the average centroid, it is considered some kind of erroneous artifact near the edge of the image that thresholding was incapable of removing. Therefore, it is not included in the count of symbol objects.

The next step in recognizing the card suit is determining which objects are suits and which are not. Using a similar discrimination approach as counting the important symbols, our method classifyCard considers the the distance of an object from the average centroid and the compactness of every object in the binary image. If a given object is too far from the average centroid, it is considered erroneous and is not classified. If the compactness is too large, then the symbol is not a suit (all of which have a compactness under 3) and is not classified. However, if the symbol is both within a logical distance from the average centroid and compact enough, then it is classified using getSuit, which will be explained further. Every suit is classified and the most common classification is returned as the actual suit classification of the card. By using the mode classification, our method ensures that one or two incorrect classifications do do not throw off the entire shape recognition objective.

For each suit symbol, as determined above, we extract a multitude of features. In addition to calculating an object's compactness and moments, we also extract 200 histogram of oriented gradient (HOG) features. Extracting more than 200 HOG features presents problems as not all encountered objects provide an equal number of features.

By restricting our feature extraction to objects with other appropriate features, however, we're effectively able to prevent encountering objects with fewer than 200 features.

Recognition: Identify the Card Based on Its Features
Once the features are extracted from the region, the features are fed into a multiclass support vector machine (SVM). This SVM was trained on a number of observations using the Matlab fitcecoc function, with each observation containing 200 features. Varying amounts of features were tested with SVMs, ranging from as few as 100 to as many as 500. Ultimately, training the SVM with too many features proved problematic if a feature extraction from a region provided fewer features than the SVM required. At the same time, using fewer features provided a greater opportunity for SVM misprediction. Thus, a compromise of 200 features was used.

Using the determined color of the card, the features are fed to one of two separate svms, with one svm for red cards and one svm for black cards. For red cards, the svm was trained with 5 observations of a heart symbol and 6 observations of a diamond symbol. Likewise, the svm for black cards was trained with 5 observations of a club symbol and 6 observations of a spade symbol. The trained SVM data was saved to prevent the need for continual retraining, and the program uses this data when given new objects to predict the suit symbol.

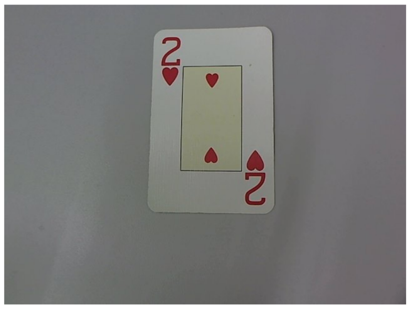
To determine the suit of the entire card, each region is put through the applicable SVM, and a count of all detected suit symbols is maintained. After all regions are scanned, the suit of the entire card is assumed to coincide with the most commonly detected suit.

With the color, number and suit of the card determined, the program can accurately identify the original playing card.

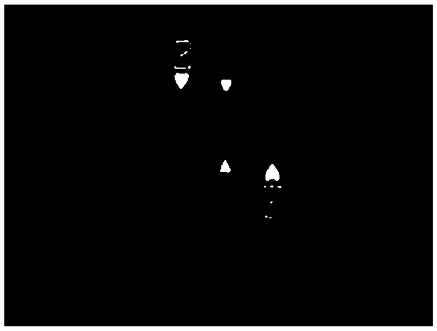
Results

Our program achieved 100% accuracy in determining the overall color, number, and suit of each card in a test file of 32 images after training on a file of 32 images. However, the SVM classifier did not recognize the correct suit for every suit symbol in a given binary image. In this case, suit recognition works 92% of the time.

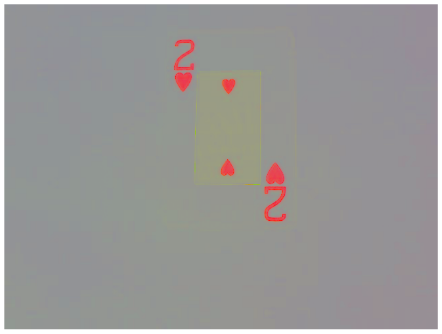
Below is an example of a test image which contains a card on a white background of varying illumination.

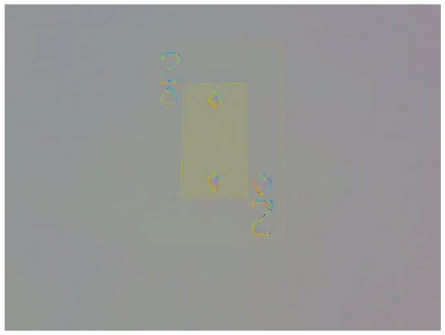


The first step of our program, segmentation, resulted in a binary image from which objects can be extracted. The results from this phase varied based on lighting conditions and color of the card. As previously explained, we specialized the segmentation techniques based on the color of the card. The binary image generated from the red spectrum of an image had white objects in it when the card was red, whereas the binary image was completely void of objects if the card was black, such as:

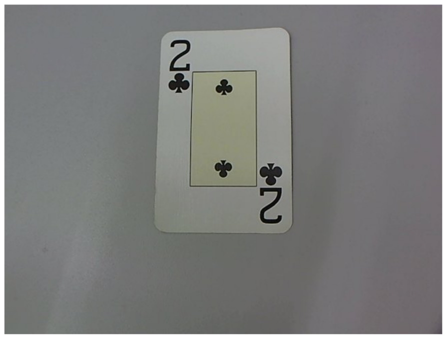


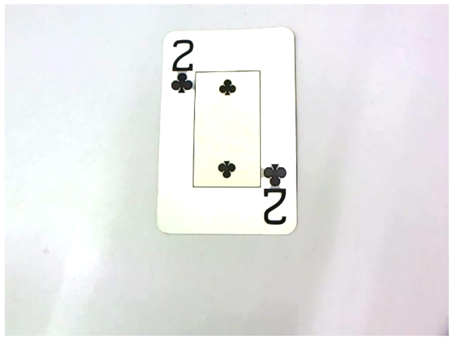
If the card was red, we eliminated variation in the image use RGB normalization. To show why we only normalized the red cards, here are two images. One is normalization of a red card, one of a black card.



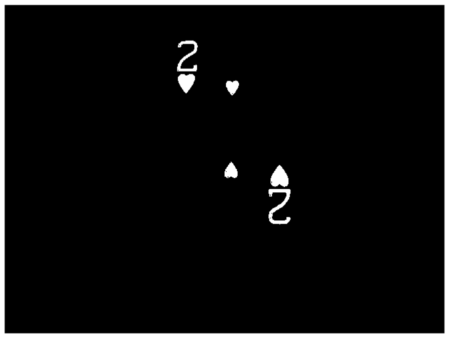


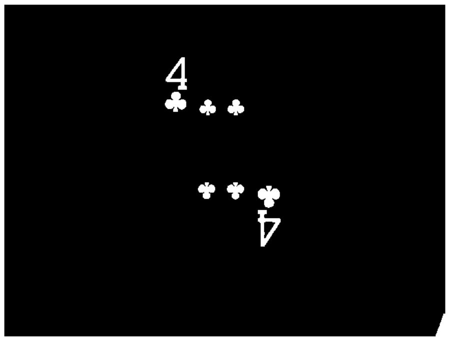
As one can see, the black objects in the image were replaced with iridescent edges and gray fillings. This means that the thresholding could not find an cutoff for objects versus the background. As a result, the binary image produced from a threshold of a black card was not usable for the purposes of shape recognition. Instead, we turned to morphological opening and background duplicating for black cards. This removed random background artifacts and increased the color difference between symbols and the background. The next two images show the improvement on a black card made by these two steps.





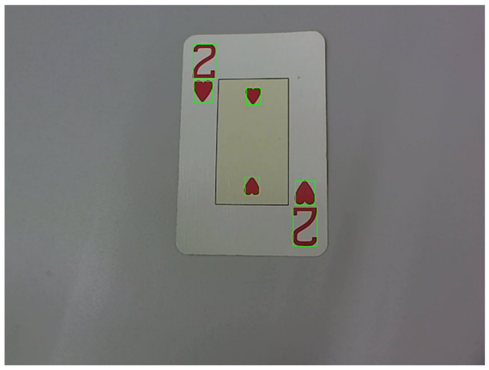
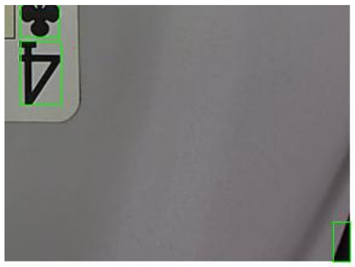
After enhancing the image based on card color, we extract a binary image using thresholding. Very rarely, this technique included false artifacts near the edges caused by unusually dark shows in the images. It's important to note that these falsities only occurred on black cards because RGB normalization couldn't be used to eliminate shadows. The picture below on the left is the binary image of a successfully thresholded red card image. The picture below on the right is the binary image of a black card that included an artifact in the bottom right corner (the artifact manifests as a cut corner on of the image).



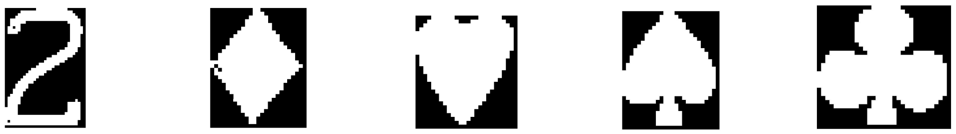


We calculated the card number by counting the number of objects that were close enough to the average centroid of all the objects in the binary image. We found that objects centered within 2.5 times the average object distance to the average centroid were symbols on the card. Subtracting four from the number of approved objects produced the correct card number with 100% accuracy.

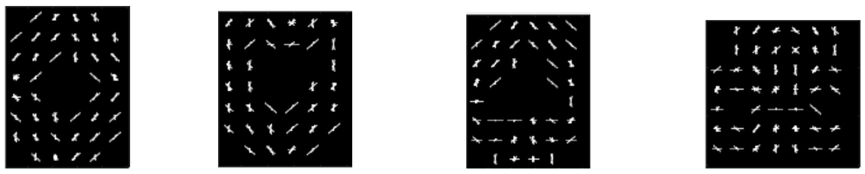
Next, we used regionprops to place a bounding box around each object in the image so as the isolate the symbols and indicate to human eyes which objects were identified in the binary image. The false symbols were bounded as well, as in the below-right image.

Once bounding boxes were formed, our algorithm sought to isolate each object in the binary image. Thanks to the array of objects produced by bwlabel and the bounding features of regionprops, our program never failed to bound an object. Here are examples of symbols that were isolated.



From here, as explained before, our algorithm determined which symbols were suits and extracted features from only them. Plotting the features obtained with the Matlab extractHOGFeatures function produced the following images for each suit symbol:



These HOG features were used to train multiclass SVMs (one for each color). When provided HOG features for a newly encountered region, the SVM could compare the features to the trained data set to predict the correct suit symbol.

We compiled the results of card classification into three confusion matrices. Each confusion matrix is constructed such that that the columns represent the true class/attribute of the card, and each row represents the class/attribute calculated by our program. This data was accumulated using a test file of 32 images, one of each suit (heart, diamond, club, spade) and number (2 .. 9) combination.

The first matrix indicates how many times each card's (overall) suit was classified correctly. Every card contributed one mark:

	Heart (true)	**Diamond (true)**	**Spade (true)**	**Club (true)**
Heart (comp.)	8	0	0	0
Diamond (comp.)	0	8	0	0
Spade (comp.)	0	0	8	0
Club (comp.)	0	0	0	8

The second matrix indicates how many times each card's suit was classified correctly. Every card contributed 4 marks:

	2 (true)	**3 (true)**	**4 (true)**	**5 (true)**	**6 (true)**	**7 (true)**	**8 (true)**	**9 (true)**
2 (comp.)	4	0	0	0	0	0	0	0
3 (comp.)	0	4	0	0	0	0	0	0
4 (comp.)	0	0	4	0	0	0	0	0
5 (comp.)	0	0	0	4	0	0	0	0
6 (comp.)	0	0	0	0	4	0	0	0
7 (comp.)	0	0	0	0	0	4	0	0
8 (comp.)	0	0	0	0	0	0	4	0
9 (comp.)	0	0	0	0	0	0	0	4

The third matrix indicates how many times a suit symbol was classified correctly. A 2 of clubs contibuted four marks, while an 8 of clubs contributed ten marks:

	Heart (true)	**Diamond (true)**	**Spade (true)**	**Club (true)**
Heart (comp.)	48	0	0	0
Diamond (comp.)	12	60	0	0
Spade (comp.)	0	0	60	7
Club (comp.)	0	0	0	53

Discussion

Overall, the program is quite successful at identifying the card by number and suit. By isolating the red spectrum from the image and using the resulting information to determine the color of the card, the task of determining the suit is reduced to classifying symbols within two classes.

To mitigate the chances that a mis-prediction of one symbol would lead to a mis-prediction of the entire card, we implemented a function to iterate through each region detected in the image. If an encountered region was near a pre-determined compactness value and average centroid, we attempted to classify the object as one of the four suits. Our function then returns the most common suit detected.

As stated above in Results, the success rate for individual suit pip detection was 92%. Using the most common suit classification for all the symbols in an image, however, increased the success rate to 100%. Additionally, the success rate for color detection remained at 100%, as did the success rate for number identification.

The SVMs used to predict a suit pip in an individual region is very successful, but this could still be improved. Currently, both black and red suit SVMs use 11 observations, and this could be expanded. From the test data, detection of both diamonds and spades was impressive, yet detection suffered with heart and club symbols. Meanwhile, each SVM was provided with one more diamond or spade observation than heart or club observation provided. This discrepancy in the number of observations could contribute to the decrease in classification accuracy in hearts and clubs.