Recognising Playing Cards in a Hand

India Leeming
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg
gusleein@student.gu.se

Abstract

In this paper, a unique system is implemented for the recognition of cards in a hand, in order to enable a computer to play popular games such as Bridge. Previous work is discussed and the implementation process described, along with the results of the recognition, which yield 87.5% accuracy. Finally, some future improvements are suggested, as well as a potential application for the system.

1 Introduction

Image recognition by computers is a task which requires the computer to pick up on the relevant information in a given image and use it in a way which benefits the user. In the case of playing card recognition, this amounts to the system identifying the card and then taking the correct action based on the fact that it has seen that particular card. Several systems have been created for playing different card games, including Poker, Blackjack and 24 (see **Previous Work** for details) but, so far, these games tend to perform image recognition on cards laid on a table or presented individually, rather than in a hand. This makes the image recognition much more straightforward but means that games such as Bridge and Whist, where cards are analysed as a hand, have not been automated. For this paper, a system was implemented which could recognise each of the cards in a hand using OpenCV and Python.

2 Previous Work

Martins et al. (2011) performed card recognition on cards in a Poker setup, where they were set out on the table. The authors did not engineer the images by keeping the background simple or by positioning the camera at a particular angle; rather, their goal was to optimise recognition in an uncontrolled environment. They managed to get a rank identification accuracy of 94.4%, despite not using colour information to distinguish between the suits: a feature which will be employed in the system implemented for this paper.

Nandi (2013) wrote a system which would recognise four cards laid out in a 2x2 formation, in order to teach a computer to play the game 24. He wrote the code using Python and OpenCV, which will also be used for this paper. The result of this implementation was a fully-functional system which could play the game 24 well, particularly as computers are able to compute mathematical problems so efficiently. The code may be found online[1] and was used as a starting-point for this implementation.

Tamrazian & Phuong (2016) implemented an Android smartphone application which would take an image of the scene at a Blackjack table and return a list of the optimal decisions based on Basic Strategy. Unlike previous methods, the authors allowed cards to overlap, providing that the following four criteria were satisfied:

> *(1) One top corner of a card must be exposed;*
>
> *(2) the top left rank must not be occluded;*
>
> *(3) the image must be taken directly above the cards;*
>
> *(4) individual cards may not be rotated more than 50 degrees relative to the vertical axis.* (Tamrazian & Phuong, 2016)

The aim of this paper is to take this idea further, by performing recognition on pictures of hands of cards, with only the top left corner of all but one of these visible.

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

3 Implementation

3.1 The Training Images

Each image that the system was trained on showed the corner of a playing card, including its value and suit. For example, the picture of the Two of Hearts was as follows:

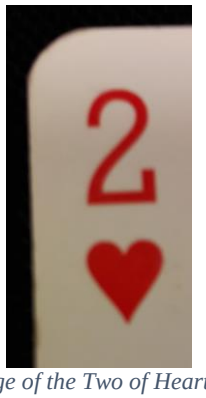

Figure 1: The image of the Two of Hearts that was used to build the system

This was done to help avoid distractors such as the two hearts in the middle of the card, one the right way up and the other upside-down. When presented with a hand, the system would only be able to use the information on the corner of the card for identification, so concentrating on that information from the outset seemed the best course of action. The card boundaries, then, would follow from the presence of the discriminatory corner markings.

3.2 Recognising Suits

In order to match a card with its suit (Spades, Hearts, Diamonds or Clubs), a function called *suit_match* is used. First of all, it decides the colour of the card; that is, it determines whether the card is red or black. The image is read and converted back to RGB from the automatic BGR that OpenCV reads it as, so that the true colours are used. The average colour of the image is computed, the idea being that images with red symbols will have a more similar average colour with each other than with those with black symbols. The average colour is then used to classify the suit as red or black and the result returned.

The suits are defined by a function called *save_suits*, which operates on the image in Figure 2.

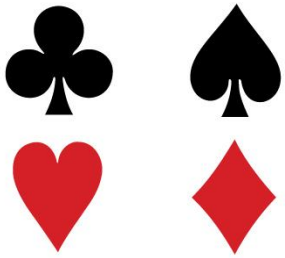

Figure 2: The four suit symbols

The contours of the shapes are retrieved from the image, saved and then returned, for use in the remainder of the *suit_match* function. The colour information dictionary is consulted in order to find whether the image being analysed is red or black and its most prominent contour is retrieved. If the colour of the card is red, the options are restricted to Hearts and Diamonds, whereas if it is black, the options can only be Spades or Clubs. The contour of the image is then matched to its two potential options in order to see which it matched best with. The suit with the smallest difference between the model contour and the image contour is deemed to be the suit on the card.

3.3 Recognising Values

The process for recognising the value on the card – that is, its value from Two to Ace – is very similar to that used for determining its suit. The function *determine_no* starts by saving the contours of the value symbols from the input images of the cards. The corresponding contour is then compared to the saved symbols and the closest match picked as the value of the card. There are four instances of each value's contour in the saved symbols, one for each of the four suits, which increases the chance of the system choosing the correct value.

3.4 Finding the Hand

The image used as an image of a hand of cards may be seen in Figure 3. It comprised eight cards, the four Twos and the four Sixes, and was arranged in suits, as is conventionally done at the beginning of card games where the players have hands. The order of the cards played no part in the recognition process but could be incorporated into it (see **Future Improvements**).

Figure 3: A hand of eight cards

The hand was divided into individual cards using a function called *split_hand*. This function starts by cropping the bottom of the hand off, in order to focus on the top corners. It then takes the largest contours of the image and finds pairs of contours with a short distance between them: the pairs of number and suit symbols. Next, it combines each pair and draws a rectangle around them; the result can be seen in Figure 4.

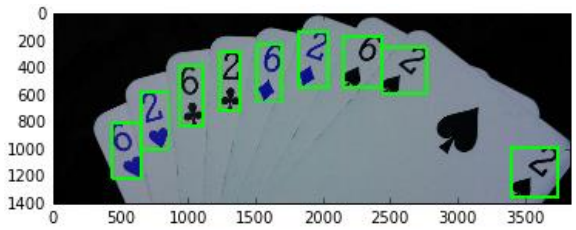

Figure 4: Cropped hand with a box around each value-suit symbol pair

Finally, the function cuts the image into the individual rectangles and returns them, ready for each one to be analysed as a separate card. The Two of Hearts is shown removed from the hand in Figure 5.

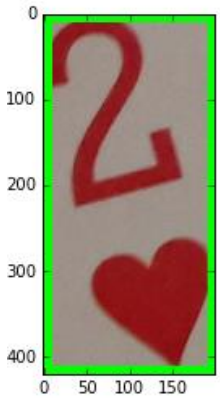

Figure 5: The Two of Hearts, cropped out of Figure 4 by the split_hand function

When the program is run, it splits the hand into cards; it takes each card and detects first its value, then its colour and suit; it decides whether it has seen the card before; and returns the result if it has not. This last step means that it avoids returning the top right-hand corner of the card on the end: the Two of Spades in this case.

4 Results

The system correctly recognises seven of the eight cards; the Two of Diamonds is incorrectly judged to be the Six, perhaps due to its tilted angle in this hand (see **Future Improvements** for a potential solution).

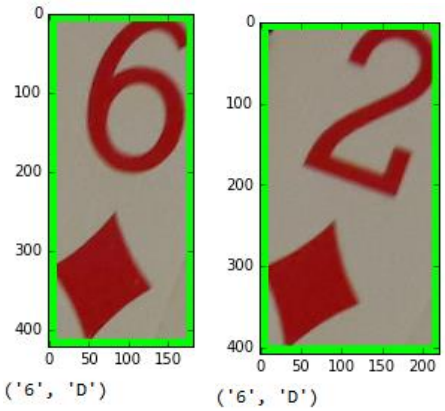

Figure 6: The Six of Diamonds, correctly detected, and the Two of Diamonds, incorrectly detected

This means that the final hand only yields seven of the eight cards; that is, 87.5% accuracy. The recognised hand is shown in Figure 7.

```
Cards Detected:
[('6', 'H'), ('6', 'S'), ('6', 'C'), ('2', 'S'),
 ('2', 'H'), ('6', 'D'), ('2', 'C')]
```

Figure 7: The hand suggested by the program

5 Future Improvements

The most natural modification will be to extend this method to the rest of the pack, enabling all of the 52 cards to be recognised. This will be a matter of storing an image of every card and ensuring that the correct contours are detected for its suit and value symbols.

As mentioned above, the order of the cards in the hand does not currently make a difference to the recognition process. However, under the assumption that the user orders the cards correctly, and if the cropped mini-cards could be ordered as they were in the original image, restrictions could be added. These would include both narrowing down the suits, once all of the previous suit had been seen, and narrowing down the values within each suit according to the previous card, which must be higher than the card currently being recognised.

In order to solve the problem with this system where the angle of a two caused it to be misrecognised as a six, the training images could include versions of each card value at various angles. This would improve the recognition of numbers at each of the positions in a hand.

6 Conclusion

Despite misrecognising one of the cards, this system shows promise for the future of card recognition. Once a whole hand of cards can be correctly recognised, it will be a simple matter for the system to make decisions based on the cards in the hand. Taking Bridge as an example, the system would be able to count the points in a hand according to the number of honour cards held and make bids based on the number of points and length of the suits. This would then be a handy tool for learners of the game to check whether they are bidding their hands correctly. Most of all, though, this system shows that automatic recognition can not only be carried out on cards set out individually but on a hand of almost entirely overlapping cards.

References

Tamrazian & Phuong, 2016. *Blackjack Assistant:Robust Card Rank Detection Using Scene-Informed Template Matching.* Published on Semantic Scholar at https://pdfs.semanticscholar.org/2_3dd/8cc6c69e9f338beebc4c5a37f868 68d78942.pdf

Martins, Reis & Te'ofilo, 2011. *Poker vision: playing cards and chips identification based on image processing.* Proceedings of the 5th Iberian conference on Pattern recognition and image analysis, pp. 436443, Springer-Verlag: Berlin, Heidelberg.

Nandi, 2013. *So I Suck At 24: Automating Card Games Using OpenCV and Python.* At http://arnab.org/blog/so-i-suck24-automating-card-games-usingopencv-and-python [accessed January 2017].