Research paper Business Analytics

Automation of the SET Card Game

Card Recognition Using Computer Vision

Wouter Kool
1937189

Supervisor: Mark Hoogendoorn

VU University, Amsterdam
Faculty of Sciences
Master Business Analytics
De Boelelaan 1081a
1081HV Amsterdam

December 14, 2013

Preface

This research paper has been written as part of the Masters program Business Analytics at the VU University in Amsterdam. The idea for the automation of the card game SET came up spontaneously: while playing the game, I thought that it would be cool to have an app that would be able to recognize SETs.

I decided to work out a method to achieve this and use this as a topic for the 'Research Paper Business Analytics'. The question that will be answered in this paper is: How can SETs automatically be detected from an image using Computer Vision?

Although I have implemented the resulting method in a fully operational iOS app, it has not yet been released as further enhancements should be made before it is practically useful on a larger scale. I hope I will find the time to further develop the app and to get it officially released in the App Store.

I would like to thank Mark Hoogendoorn for his suggestions on the methods used and his supervision while writing this paper.

Summary

Introduction A framework is developed for automation of the card game SET. A SET card has four properties (quantity, color, fill and shape), which each can take three values. The goal is to find special combinations of three cards, of which each individual property is equal or totally different for all three cards. The paper researches how these SETs can automatically be detected from an image using Computer Vision, dealing with three problems sequentially: the locating of cards on the image, the classification of the cards and the finding of SETs amongst the cards.

Locating cards The locating of cards can be done using two different approaches: using the card borders or using the location of individual shapes. After appropriate preprocessing steps, edge detection can be used to identify contours in the image. Filters on topological features are used to identify either card or shape contours. Results show that the approach using individual shapes is more robust, therefore the classification method is developed for the shapes resulting from this approach. After classification, shapes can be combined to identify cards.

Classification of the cards The properties shape, color and fill of an individual shape are classified individually. The shape, drawn as a thick contour, is compared to reference images to find the shape which is most similar. For both the fill and the color, numerical features are derived which are representative for the class. Upon principle components of these features different classification models are trained.

Finding of SETs An algorithm is developed that finds all SETs amongst n cards in $O(n^2)$ by iterating over all pairs and checking whether the unique third card that would complete the set is available.

Results The results, obtained from cross validation using a dataset of over 1000 shape images, are presented for the Ravensburger (RVB) and NNN Games (NNN) editions of the game. All properties, except the color for RVB, are classified (using Support Vector Machines on all principle components for the fill and color) with over 99% accuracy, resulting in a final accuracies of 95.72% (RVB) and 99.15% (NNN).

Discussion The accuracy is high, but the practical applicability is limited as all cards (≥ 12) on the table need to be correctly classified. In the practical implementation (in the form of an app) a feedback mechanism should be implemented to make small corrections.

Contents

Chapter 1

Introduction

This paper is inspired by the card game SET. The goal of the game is to find special combinations of three cards, called SETs, amongst the cards on the table. For a detailed explanation of the game we refer to section 1.1. The main question that is researched in this paper is:

- *How can SETs automatically be detected from an image using Computer Vision*

To answer this question, we separate it into three smaller questions:

- *How can SET cards be located on an image?*
  It is important to know where the cards are on the image before they can be classified.
- *How can SET cards be correctly classified from the image?*
  In order to find the SETs, it is important that all cards are correctly recognized, therefore the card should be classified correctly.
- *How can the SETs be found amongst the recognized cards?*
  For practical relevance it is important that SETs can be found amongst the recognized card in an efficient way.

We will see that the first two questions get somewhat related as we we will discuss two different approaches of locating the cards: a 'top-down' approach and a 'bottom-up' approach. Roughly, chapter 2 answers the first question, chapter 3 the second while chapter 4 connects the two questions and explains how the final digital card representation is obtained for both approaches of locating the cards. The last question is a theoretical one that is answered with the development of an algorithm, which is presented in chapter 5. The paper will compare the different localization approaches and present the results in terms of accuracy of individual features of shapes in chapter 6. Chapter 7 will describe how the developed framework was implemented as a mobile application and chapter 8 will discuss the results and the practical applicability and suggest some directions for future research.

First, this chapter gives a small introduction about the SET card game and its rules in section 1.1. In section 1.2 the relevant literature is discussed, from which it will become clear that the 'top-down' approach for localizing cards has been used before, while the 'bottom-up' approach in this paper is entirely new, just as the development of an algorithm to find SETs.

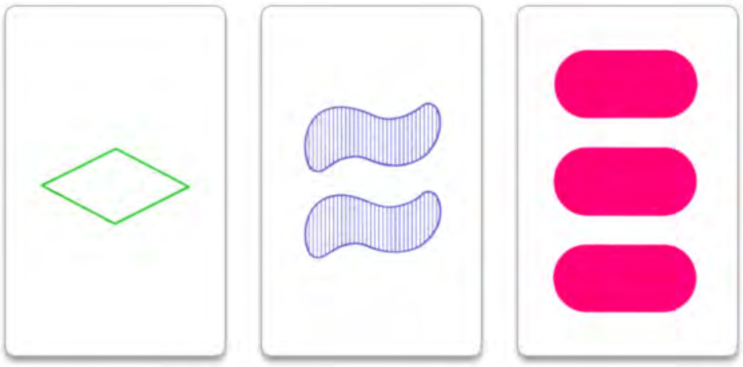


Figure 1.1: Example of a SET from the NNN Games edition.

1.1 The SET card game

SET is a card game invented by population geneticist Marsha Jean Falco in 1974. The game consists of a deck of special SET cards: special regular-sized cards that display four properties: quantity, fill, color and shape. Each property can take three values, resulting in a total of $3^4 = 81$ different cards. The original game was published by Set Enterprises in 1991, but the two most recent editions where published by Ravensburger in 2001 and 999 Games in 2009. The values each of the properties can take are listed below:

- **Quantity** - 1, 2 or 3
- **Fill** - open, half-filled or solid
- **Color** - red, purple or green
- **Shape**
  - **Ravensburger** - rectangle, tilda, oval
  - **NNN Games** - diamond, tilda, rounded rectangle

A SET is a combination of three cards such that each of the four properties is either equal for all three cards, or totally different. Figure 1.1 shows an example of a SET. When the game is played, twelve cards are laid out on a table and the players simultaneously try to find a SET. When a player finds a SET, the player takes the cards of the SET and new cards are drawn. If no SET occurs amongst the cards on the table, three additional cards are drawn. If there is still no SET, more cards are drawn in triplets, up to a maximum of 21 cards, in which case a SET always can be found[6]. The rules of SET state that new cards should be drawn if there is no SET. However, for human players, it is hard to be sure that this is the case. The framework described in this paper is used to build an application that can check whether there is a SET, or even display the SETs. For the implementation, we refer to chapter 7.

1.2 Literature review

Playing card recognition using computer vision techniques is topic widely studied in different contexts. Hollinger et al.[15] created a program that could recognize regular cards from a deck on a black table and make basic gameplay decisions for Blackjack. The system worked well but was sensitive for camera orientation and overall illumination. Cooper & Dawson-Howe[4] built a system that recognized cards on a special Blackjack table. They used the fixed layout of the table to locate the cards but developed an additional technique to locate cards that overlap in a structured manner. Zheng & Green[25] introduced a more general approach to match cards against templates, invariant of rotation and position. They use edge detection to find card contours, after which they extract the card using an affine transformation. The rank and suit are determined using character segmentation. The method proved to be computationally efficient and accurate for playing card recognition, but noise significantly influences the accuracy and the system only works for the deck of cards it was trained with. Zutis & Hoey [26] introduce a system which combines similar playing card detection techniques with analytical techniques to detect people who count cards while playing Blackjack at a casino.

Most researched methods use a 'top down' approach in which contours of individual cards (or multiple cards that overlap in a structured manner) are found, but little research has been done on the recognition of cards that overlap randomly. We will not go into details about the possibilities in normal playing card recognition (although there are sufficient thanks to the redundancy in the representation of suit and color), but we will discuss this for the special case of SET cards. For SET cards, there is no redundancy (as each shape needs to be visible to determine the quantity), but the white parts near the border are irrelevant and may therefore overlap without loss of information. Due to the nature of the game, in which the cards are laid out in a raster together, it is likely that some overlapping occurs, so this is relevant. This paper will discuss both the traditional 'top down' approach and a special 'bottom up' approach that can deal with small overlaps of cards.

On the topic of SET card recognition, no research has been done, although there are studies that are relevant. For example, Niblack et al.[19] performed a study focusing on image retrieval by content, using color, texture and shape. These are all features relevant to SET cards and classification of the cards can be regarded as retrieval of the most similar card from a database.

On the topic of the SET card game itself, a mathematical analysis has been done by Davis et al.[6]. They compare SET cards to vectors in the four-dimensional space and derive interesting probabilities about SETs, which they use to prove that any combination of 21 cards will contain at least one SET. Also besides the scientific literature, the SET card game is a much discussed topic on the internet for its interesting mathematical aspects. For example, Warne wrote a blog[1] on the probabilities of finding no SETs at any point during the game, found using simulation. The algorithm for finding SETs itself has not been formalized before.

[1]H. Warne, *SET Probabilities Revisited*, `http://henrikwarne.com/2011/09/30/set-probabilities-revisited/`, 2011-09-30.

Chapter 2

Locating cards

This chapter discusses how the cards can be located in the image. This is the first step in the process of recognizing and classifying the cards in the image. In the two sections two different approaches will be discussed:

- **'Top-down' approach** - the locating of cards using card borders, section 2.1.
- **'Bottom-up' approach** - the locating of cards using the location of individual shapes, section 2.2.

The results of the comparison of the two approaches are discussed in section 6.1.

2.1 'Top-down' approach

The 'top-down' approach consists of locating cards using card borders and is a method that has been used in previous studies. For a review of these studies, we refer to section 1.2. The method consists of three steps:

1. Finding contours of cards
2. Locating card corners
3. Extracting card image

These steps are explained in the following subsections.

2.1.1 Finding contours of cards

To locate the cards in the images, it is important to recognize the borders of the card. For this, we use the Edge Detection algorithm developed by Canny[3] on the grayscale image[1]. To reduce the finding of edges corresponding to noise, we apply a Gaussian blur before the Edge Detection. The result is a binary image with white edges and black surfaces.

To locate individual cards we have to identify contours in the image. This we do using a border following algorithm described by Suzuki[23]. It converts the binary image to vectors of coordinates representing contours in the image. These vectors are obtained as a tree that represents the topological structure, so internal contours are children of their parent contour.

[1]The grayscale image is actually the Y-component of the YCrCb representation, see section 3.4.1

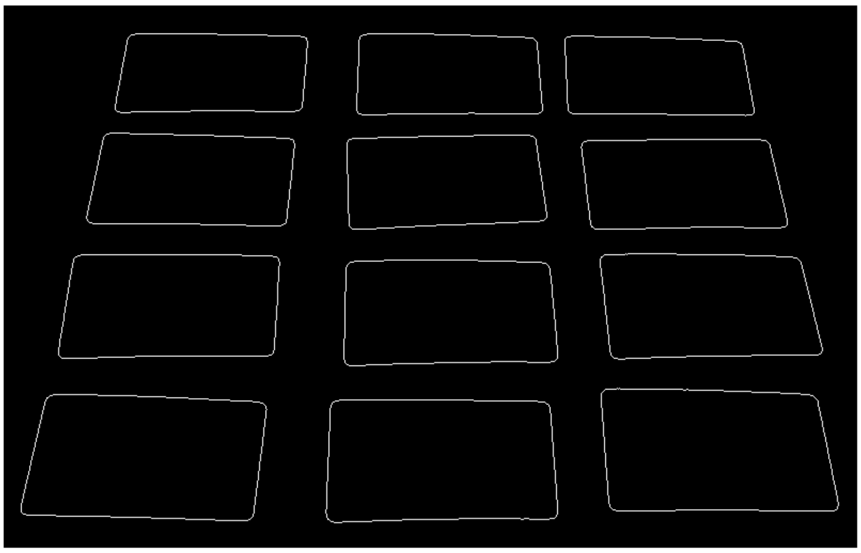


Figure 2.1: Outer contours of an image of a SET card game.

We retrieve only the top level contours as these comply to the edges of the cards. Some contours might not correspond to cards: these contours are filtered in the next step when it is not possible to extract four corner points. For a more extensive and visualized description of the contour detection we refer to section 2.2.1. An example of resulting outer contours is in figure 2.1.

2.1.2 Locating card corners

For the next step, which consist of locating the corner points of the contours that correspond to cards, we have different options:

- Approximating the contour by a polygon using the Douglas-Peucker algorithm[7]. If this results in an approximation of the contour by four points, it is most likely a card. To gain more certainty, the perpendicularity and length-ratio (with some tolerance) of the adjacent edges should be checked. The card has rounded corners, so the tolerance should be sufficiently large to prevent finding more than one point for each corner. However, this tolerance also has as result that the points found do not exactly coincide with the card corners.

- Usage of a Probabilistic Hough Transform[18] to find line segments for each contour. Using the Hough Transforms, each pixel 'casts a vote' for each of the lines (given by an angle and an intersect) it lies on. The lines with the most votes are detected. From each line the segments are found by grouping the pixels on the line. Using this method, if a contour corresponds to a card, theoretically two longer and two shorter line segments should be found. In practice, a card might be slightly bent in which case multiple line segments are found for each of the four edges. We implemented a small angular tolerance

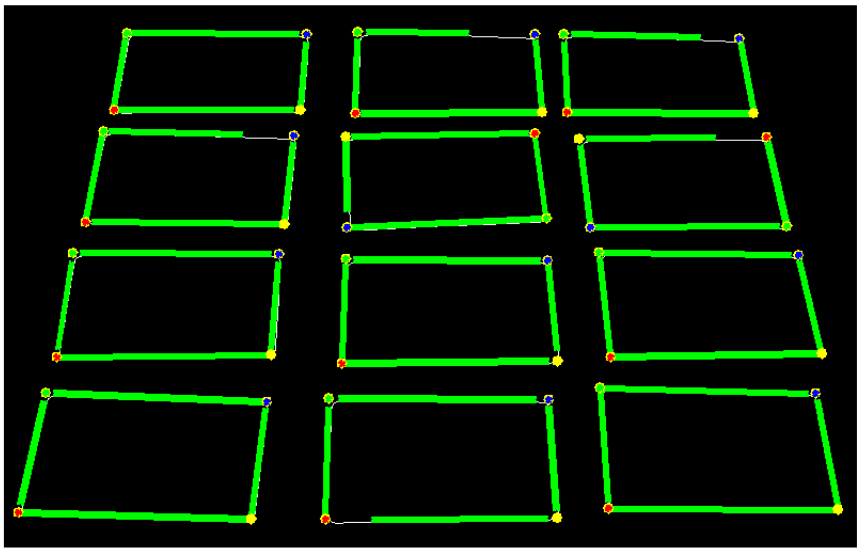


Figure 2.2: Line segments detected by the Probabilistic Hough Transform (indicated as green lines with small blue dots at the endpoints) and the corner points detected. Red, green, blue and yellow inner colors for the corner points correspond to top-left, top-right, bottom-right and bottom-left of the card respectively.

to connect these line segments. Figure 2.2 shows the detected line segments in figure 2.1.

- We can obtain the minimum rotated bounding rectangle of the contour using the method developed by Freeman and Shapria[9]. From the angle and size the corner points can be inferred immediately, but the corner points of the bounding rectangle do not correspond to the corner points of the card if the picture contains some perspective distortion.

2.1.3 Extracting card image

For extracting the card image we need the corner points of the card in order to apply a perspective transformation. Especially, we need to know which corner is the top-left, top-right, bottom-right and bottom-left (clockwise ordering) corner. For this, we implement the following algorithm inspired by Nash[2] to sort the corners clockwise:

1. Sort the corner points by ascending y-coordinate
2. If the x-coordinate of the second corner is smaller than the x-coordinate of the first, swap these two

[2] *Automatic perspective correction for quadrilateral objects*, `http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects/`, 2013-01-12.

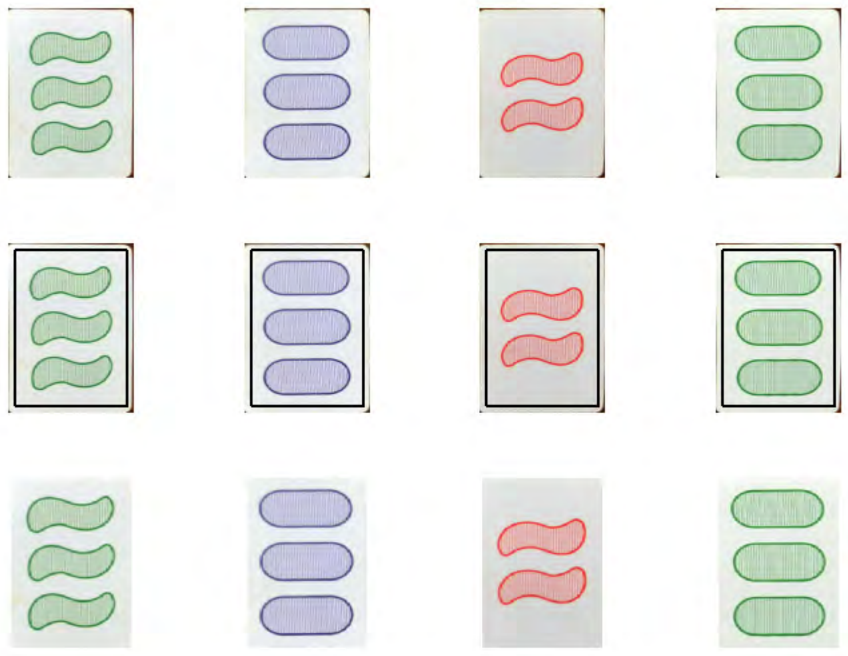


Figure 2.3: Example of the cropping of cards. Top row shows the images after they are extracted using the corners, the middle row shows the cropping area and the bottom row shows the resulting card image.

3. If the x-coordinate of the fourth corner is *larger* than the x-coordinate of the third, swap these two
4. If the average distance from corner 1 to 4 and corner 2 to 3 is smaller than the average distance of corner 1 to 2 and corner 3 to 4, the card is oriented sideways. Rotate the corners by moving the fourth corner to the front.

Figure 2.2 shows the corners that are found using this algorithm. Next a perspective transformation can be applied to extract the card image. As the card has rounded corners, it may be that we also extract some pixels of the surface behind the card. This is removed by cropping the card with 10% from the borders. This value has been found by experiments and serves our needs: the margin from the shapes on the cards to the border is sufficient, so this cropping will preserve the shapes but remove the noise. The result is a white image with only shapes, as can be seen in figure 2.3.

2.2 'Bottom-up' approach

The alternative way of locating the cards, the 'bottom-up' approach, consists of first locating the individual shapes, after which similar shapes can be combined to cards, based on their mutual distances and matching properties. This section describes the locating of cards using the location of individual shapes, which consists of the following steps:

1. Finding contours in the image
2. Filtering contours on topological features
3. Finding the group of shape contours
4. Extracting shape images
5. Combining shapes to cards

The first four steps are explained in the following subsections. The last step is related to the classification of the shapes, as the classes are used to determine which shapes belong to the same card. Therefore, this is discussed after the classification of the shapes in chapter 4. The method is visualized with a step by step example in figure 2.4.

2.2.1 Finding contours in the image

To locate the shapes, we have to find the contours that correspond to shapes. These contours are not outer contours with respect to the total image, so we have to retrieve contours at all topological levels of the image in order to make sure we find the contours of the shapes. This means that we find not only contours of the shapes, but also of the cards themselves and contours resulting from noise or contours within the shapes, which occur if the shape is not solid. After the contours are found, we will filter the relevant contours as described in section 2.2.2 and 2.2.3.

Consider the example in figure 2.4, image (a). Again we do some preprocessing by applying a Gaussian blur (b), after which we apply the Canny Edge Detection algorithm[3] (c), just as we would do for the locating of the cards. Experimental results with the threshold parameter show that a higher threshold will reduce the finding of edges that correspond to noise, but also will negatively affect the detection of edges that correspond to the shapes. For practical use, it is important that all shapes are recognized, so a low threshold should be taken (experiments show that the value 100 will most of the times find edges of all shapes). Under some lighting conditions, using the Canny Edge Detection will result in contours with small gaps. We use a border following algorithm (see section 2.1.1) on the binary image, so such a gap would result in the shape not being recognized as a single contour. To fix this, we preprocess the edge image by applying a Gaussian Blur with a 3 pixel radius and a threshold (d), which is equivalent to ‘drawing the edges’ with a pen of radius 3. This way, all contours with a mutual distance less than 2 pixels become connected. Under normal circumstances and image resolution (640×480 and 12 cards filling up at least half of the image) the distance between shapes on the cards is sufficiently large to guarantee contours of different shapes can be distinguished (this is clearly observable in image (d)). Finally, applying the edge detection algorithm results in a set of contours which are drawn in image (e).

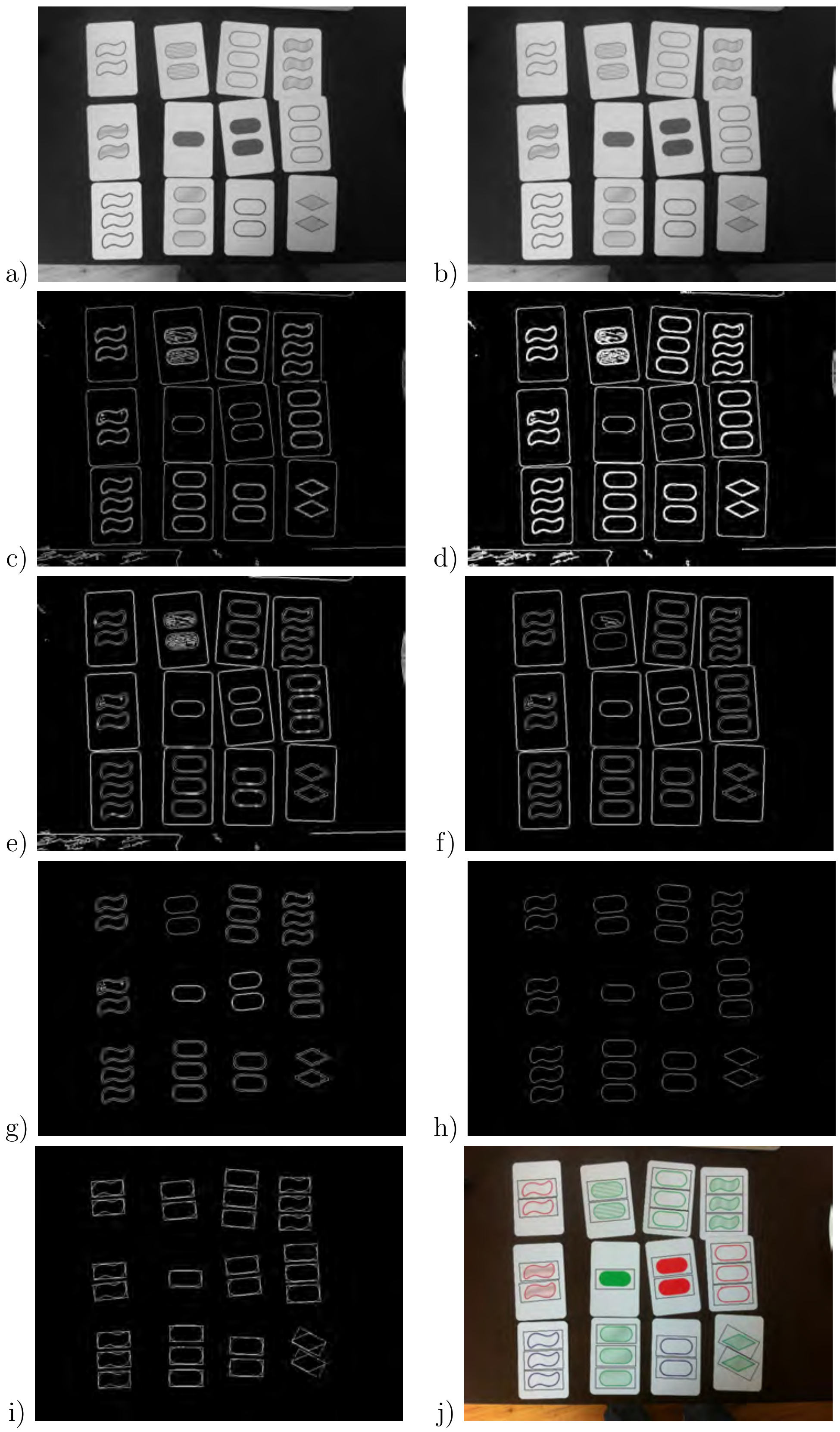


Figure 2.4: Steps of the extraction of the shapes

2.2.2 Filtering contours on topological features

After the preprocessing we have an edge image that contains all shapes as connected components, but also contains a lot of noisy contours and the contours of the cards. Again we apply the algorithm as described in section 2.1.1 to obtain a list of the contours, but this time not only the outer contours (we omit the hierarchy) as we want to find the shapes within the cards. For a background with a texture, the number of contours can get very large. Therefore, if the number of contours exceeds 150, we first apply a cost efficient filter based on contour length/perimeter (roughly estimated by the number of points a contour consist of) to take only the 150 lengthiest contours. Using the theoretical upper bound of 21 cards (see section 1.1), we have at most 21 contours of cards, 21×3 contours[3] of shapes and at most 66 lengthy noisy contours we are safe to assume the contours corresponding to shapes are contained within this subset. Next we apply a set of filters based on some basic properties of the contour to further limit the set of potential shape contours. Summarizing, we only keep the contours that satisfy all of the following conditions:

- The contour is one of the 150 lengthiest contours.
- The length/width ratio of the contours minimum rotated bounding rectangle[9] is between 1.4 and 3.5.
- The ratio of the inner area of the contour, as calculated by Greens Theorem[12], to the area of the minimum rotated bounding rectangle is at least 0.5.
- The area of the contour is at least 0.05% of the image area (measured in square pixels).

The filters are inspired by the topological features of the shapes. The shapes have a length/width ratio of approximately 2.5, but the ratio of the bounding rectangle varies with the perspective and the shape (see image (i), especially the diamonds have a lower ratio). The filter on the ratio of the inner area to the bounding rectangle area makes sure we filter out noisy contours as these often have no surface at all (if in the edge detection we get a line which does not embrace an area) or only small, while the SET shapes surfaces cover large parts of their bounding rectangles (again, this can be observed in (i)). The absolute (with respect to the image area) contour area filter filters out very small contours of just a few pixels, which are not filtered by the other filters. The resulting contours after this filter are drawn in (f).

2.2.3 Finding the group of shape contours

After the filtering on topological features we obtain a set of contours which have a high potential to correspond to a shape, but also contours corresponding to cards match these criteria (see image (f)). We cannot filter these latter contours by an

[3]This upper bound could be made more tight than 21×3 as there does not exist a combination of 21 cards with three shapes with no SET because the effective dimensionality of those cards is 3 rather than 4[6].

absolute criterium on the area, as this is dependent on the distance from card to camera and the image perspective: in some cases the size of a shape from a smaller distance may get very near the size of a card from a longer distance. However, for a single image that is taken in a reasonable perspective, there is always a clear distinction between the area of a card and a shape. Using the fact that the total number of shapes in the image is at least equal to, but most likely larger than the number of cards we should find the largest group of contours with approximately equal area. This we do using the following steps:

1. Sort the contours by descending inner area (as calculated by Greens Theorem[12]).
2. Calculate the ratios of two consecutive areas after sorting, the ratios lower than 0.8 correspond to boundaries of approximately equal area groups.
3. Find the largest group (largest distance between two boundaries). In case of a tie, the latter (smaller area) group is taken.

If there is some perspective in the image, it can be that there is a big difference in the areas of the largest (closest to the camera) and smallest (furthest away from the camera) image. However, the shapes are sorted by area and the threshold on the ratio only applies to two consecutive shapes, so within the entire group the difference can be larger.

Taking the smaller group in case of a tie ensures shapes are found rather than cards in the rare situation that all cards only have one shape. Image (g) shows the group of contours that is found. As can be seen in (g), we still have contours corresponding to the inner edges of open, and in some cases half-filled, shapes. To remove these contours, we apply the contour finding algorithm once more on (g), but we retrieve only outer contours (h). One could implement more efficient ways to achieve the same, but the computational costs of this approach are acceptable while the implementation is simple.

2.2.4 Extracting shape images

After this step we have filtered the contours down to the contours corresponding to shapes. For each shape, we use the angle of the minimum rotated bounding rectangle (visualized in (i)) to apply an affine transformation to the image[4], after which the shape can be extracted as an upright rectangular sub image of the size of the bounding rectangle, with a margin of 10%. This margin is to ensure we have some white space around the shape, which is used by the classification algorithm (see chapter 3) for the fill and color to normalize the image. Image (j) marks the images that will be extracted: each image will be resized to 100×50 to ensure uniformity in the input to the classification models. The shape will be matched using the contour itself, see section 3.2.

[4]Actually, the affine transformation is only applied to a small region of interest, which is the upright bounding rectangle of the rotated bounding rectangle. The area of this region is only a fraction of the image area, so this is much more efficient.

Chapter 3

Classifying shape images

This chapter describes the classification of a single shape. A shape is characterized by its color, fill and shape. Although one could think of methods which classify a single shape at once (with $3^3 = 27$ classes), this paper focusses on the individual classification of the three properties, which improves the accuracy as each model is specifically developed to classify a single feature. First, in section 3.1, it is explained how the brand of the the card deck is recognized. This depends on the method for classification of the shape, which will be discussed in section 3.2. Sections 3.3 and 3.4 describe the models that are used to classify the fill and color. Both of these models use supervised learning on a set of descriptive features for each entry in the training set (section 6.2). The models used for classification of the fill and color are trained separately for the different brands, as the color and fills are slightly different for these brands. Input to these methods are the descriptive features calculated from the shape image[1], while the shape is classified using the contour itself.

3.1 Recognition of the brand

For practical use of the framework (see chapter 7) it is important that it works with the various editions of the game as introduced in section 1.1. The brand NNN Games (NNN) has the most recent edition with the shapes diamond, rounded rectangle and tilda, but Ravensburger (RVB) used a rectangle, an oval and a tilda. Therefore, the set of shapes that may be recognized depends on the brand of the SET card deck used. Also, the classification models for the color and the fill differ for the two brands, as the cards differ. As we do not have information about the brand, we have to detect this. Therefore, for each found shape, we find the minimum error (as will be defined in section 3.2) by finding the best matching shape for both NNN and RVB. Averaging, per brand, these errors over all the shapes in the image, we can select the brand which has overall the smallest error, and select this as the brand. In the rare case that there are only tildas in the image, the distinction is very difficult. The potential incorrect classification of the brand could result in a less accurate classification of the fill and color, but fortunately this scenario is very unlikely: a draw of 12 cards from a deck results in twelve cards with the shape tilda with a probability of $\frac{27}{81} \cdot \frac{26}{80} \cdot \ldots \cdot \frac{16}{70} = \frac{27!/15!}{81!/69!} \approx 2.47 \cdot 10^{-7}$.

[1]The shape image is extracted as described in section 2.2.4, so this assumes use of the 'bottom-up' approach. This is the preferred approach, see section 6.1. If the 'top-down' approach is used, an additional step has to be implemented to extract the individual shapes from the cards.

3.2 Classifying the shape

For the classification of the shapes there are two options: matching using image moments of the contour or using absolute image difference of a binary image. The following two subsections will explain each of the methods.

3.2.1 Classification using image moments

Shapes of a contour can be characterized using the seven image moments as defined by Hu[16]. Using these moments a measure for the similarity can be defined as in formula 3.1[2].

$$I(A,B) = \sum_{i=1}^{7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right| \tag{3.1}$$

In formula 3.1, $m_i^A = sign(h_i^A) \cdot \log h_i^A$ and $m_i^B = sign(h_i^B) \cdot \log h_i^B$ where h_i^A and h_i^B are the Hu-moments of shape A and B respectively.

This measure is invariant to orientation and scale, as the moments themselves are invariant to this[16]. To classify the contour, we take the average distance to a set of reference shapes for each of the three possible classes of shape. The matched class is the one which results in lowest average distance.

3.2.2 Classification using absolute image difference

A more natural way of comparing shapes is simply comparing the images that represent the shape. For this, a computationally efficient method is taking the sum over the absolute values of the differences in pixel intensities of corresponding pixels in two images[2]. First, we need a clear representation of the shape as an image, which will not differ so much under small perspective transformations and different conditions. We need to transform the image to a standard size (we take 100×50 pixels) so it can be compared on the pixel level to a reference image. We find the clear representation of the shape using the following steps:

1. Draw the contour in a black image of the original image size.

2. Blur the image using a Gaussian blur with radius $\frac{1}{8}$-th of the height of the shape (the height of the bounding rectangle), and apply a threshold (with threshold value 0 so that any value that is not black becomes white).

3. Apply an affine transformation (using the angle of the minimum rotated bounding rectangle of the shape) on the resulting binary image to rotate the contour horizontally. The affine transformation rotates the contour, but it does not correct for a perspective distortion. However, if the perspective is reasonable, the shape is still sufficiently recognizable (see section 6.1).

[2]This formula is implemented by the OpenCV framework, see `http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html`. For more about the implementation, see chapter 7.

4. Extract the upright rectangle with the size of the minimum rotated bounding rectangle and a margin of 20% (for the thick line).

5. Resize the rectangle to 100×50 pixels.

The resulting image is a 100×50 binary image of the shape contour drawn with a thickness of about $\frac{50}{8}$-pixels (see figure 3.1 (a)). Because of this thick drawing, there will always be some overlap with an image of the same shape, even if it is different. Therefore, we can classify the shape by comparing the image to a reference image of each of the possible shapes. As both the extracted image and the reference image are binary, they can be regarded as a matrix with 0's and 1's, and calculating the sum of the absolute differences is therefore equivalent to counting the number of pixels that differ. This is implemented as the sum of the elements of the matrix resulting from a binary xor operation on the two matrices representing the image. If we normalize this value by dividing it by the number of pixels (5000), we get a percentage which indicates the difference or error between the images. The reference image (class) which results in minimum error defines the class that is predicted. Figure 3.1 provides an example: the unclassified shape (a) is compared to reference shapes (b), (c) and (d). The differences are represented by white pixels: clearly reference shape (c) gives the least differences and therefore the lowest error.

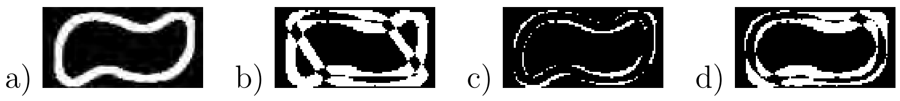


Figure 3.1: A shape to be classified compared to reference images of three different shapes

Classification of a rotated contour

The problem of classification of a shape using absolute image difference, is that the shapes need to be in the same orientation. For most shapes, this is not a problem as the shape is point-symmetric (i.e. invariant under a half rotation) and the minimum rotated bounding rectangle is unique. However, for the diamond shape of the NNN Games edition, and (to a less extent) the oval shape of RVB have two possibilities for the minimum rotated bounding rectangle. The extraction of the shape images using the two possibilities results in a 'mirrored' version of the shape (horizontally or vertically; this is equivalent thanks to the point symmetry). Figure 3.2 shows an example. The result is that comparison of the images would give a high error for equal shapes if one is mirrored. As it is hard to detect which of the two bounding boxes is extracted, so the simplest solution is to compare both the extracted image and its 'mirrored' version. The error for each shape would be the minimum of the two errors. A more computationally efficient implementation of this is to keep a mirrored version of the reference image in memory, and to compare each new image to these two. This way, no runtime mirroring of images is required.

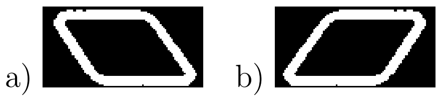


Figure 3.2: The NNN Games diamond shape and its mirrored version, each of which is extracted with equal likelihood using the bounding rectangle.

3.3 Classifying the fill

Recognition of the fill is done independently of the other properties. The dependency of the fill on the color is removed by converting the image to a grayscale image, or equivalent, taking the Y-component of the YCrCb image representation (see section 3.4.1). Any differences in grayscale representation of the different colors is removed by normalization as a preprocessing step. The dependency of the fill on the image shape is removed by selecting a region of interest within the image, which is inside the shape, independent of the shape. This region of interest (ROI) is the rectangle of 32×14 pixels in the center of the 100×50 pixels shape image, as this region is practically always fully within the shape.

3.3.1 Preprocessing the image

To be independent of color and lighting conditions we want to normalize the image. That is, considering a range of 0 - 255 of the pixel luminance, a pixel should have a value (close to) 255 if it represents a white part of the card, and a value (close to) 0 if it represents a colored part. As the entire shape image (not only the region of interest) always contains some colored (darker) areas (at least the outline of the shape) and some white areas (outside of the shape), we can easily find the range of the pixel intensities. We use a robust estimator for the range: the 1% and 99%-quantiles. We normalize using this values, truncating values outside of this range. That is, if *min* and *max* represent the two quantiles we use formula (3.2) to map the pixel intensities, represented by a matrix M^{gray}.

$$m_{ij}^{norm} = max(0, min(255, 255 \cdot \frac{m_{ij}^{gray} - min}{max - min})) \tag{3.2}$$

Although we need the full image to calculate *min* and *max*, the transformation only needs to be done on the ROI of 32×14 pixels in the center.

3.3.2 Feature calculation

The next step is to calculate from this preprocessed ROI a set of features which is representative for the fill of the shape. We will try to select those features that are the most representative and build a classification model upon these using a training set. We propose the following set of candidate features:

- **Mean luminance** - the mean of the pixel intensities.

- **Quantiles of the luminance distribution** - the 5%, 25%, 75% and 95%-quantiles of the distribution of the pixel intensities.
- **Mean absolute Laplacian** - the mean of the values obtained from applying the Laplacian operator on the matrix with pixel intensities.
- **Mean absolute difference to blurred image** - the mean of the absolute differences in pixel intensities between the image and a heavily blurred version of the image.

Mean luminance

The mean luminance (3.3) is a natural feature for classification of the fill as in the normalized image open/white should correspond to a high luminance, solid/black to a low luminance, and half-filled (on average) to a value in between.

$$lum = \frac{1}{nrows \cdot ncols} \sum_{i=1}^{nrows} \sum_{j=1}^{ncols} m_{ij}^{norm} \tag{3.3}$$

Quantiles of the luminance distribution

The quantiles of the luminance distribution provide additional data about the fill. The combination of lower and higher quantiles gives information about the range of the luminance values: if the lower quantiles have low values but the higher quantiles higher values, there is a big range in luminance and the shape is most likely half-filled. If all the quantiles have either low or high values the shape should be solid or open respectively.

Mean absolute Laplacian

The mean absolute Laplacian (3.4) is a feature that represents the amount of texture in the image. It is the mean of the values of the matrix obtained from applying the Laplacian operator: the sum of the derivatives in horizontal and vertical direction. If we have a texture, which is the case when the shape is half-filled, then there are many pixels on the edge between white and black pixels, which thus have a high derivative value. Therefore this value should be high for the shapes which are half-filled and low for the open or solid shapes. Clearly the latter two cannot be distinguished by this feature, but it could be a good combination with other features (such as the mean luminance) which clearly separate solid and open shapes but have a vaguer distinction with those that are half-filled. The feature is scaled by a factor 5 (found by observing the range of this feature on the training data) so that its magnitude is comparable to the other features, which have a value from 0 to 255 (8-bit encoding) as they originate from the luminance distribution.

$$lap = \frac{5}{(nrows-2)(ncols-2)} \sum_{i=2}^{nrows-1} \sum_{j=2}^{ncols-1} |m_{i,j-1}^{norm} + m_{i,j+1}^{norm} + m_{i-1,j}^{norm} + m_{i+1,j}^{norm} - 4m_{i,j}^{norm}| \tag{3.4}$$

Mean absolute difference to blurred image

Another way to quantify the amount of texture in the image is based on the simple observation that texture is removed by blurring the image using a Gaussian blur. Therefore, if an image contains a texture and this image is compared to a blurred version of it, one would expect differences, while if the original picture would contain no texture the blurred image would be the same. Taking the mean of the absolute differences between the matrices M^{norm} and M^{blur} (obtained by applying a Gaussian blur with Kernel size 5×5 on M^{norm}), we find the feature value (3.5). This feature is scaled by a factor 16 for the same reason the Mean absolute Laplacian was scaled (section 3.3.2).

$$bluration = \frac{16}{nrows \cdot ncols} \sum_{i=1}^{nrows} \sum_{j=1}^{ncols} |m_{ij}^{norm} + m_{ij}^{blur}|. \tag{3.5}$$

Actually, this measure is similar to the Mean absolute Laplacian as it is also based on a difference between a combination of neighbor pixels and the value of the pixel itself. For a kernel size of 3×3 with a Gaussian blur that only uses the four neighbor pixel intensities, the value would be linearly correlated to the Mean absolute Laplacian. With the 5×5 kernel, this feature provides some more robust texture information.

3.3.3 Feature set reduction

The set of features calculated consist of seven features, while the problem of determining the fill is a 3-class classification problem. That means a classification model should divide the 7-dimensional feature space into three regions that correspond to the three possible fills. All feature value functions are ‘well behaved’, that is, we would not expect big differences in the value of the feature for images that are similar with respect to that feature. Therefore, we would expect the regions corresponding to the three classes to be connected components in the feature space (we will see later that this is more or less the case in figure 3.3 and 3.4). For a 3-class classification problem a high dimensionality of the feature space may cause overfitting towards the training data, as with such a high dimensionality it is more likely that a ‘good separation’ is found that is incorrect simply because the size of the training data is not sufficiently large to fill the high dimensional feature space. Previous research from Janecek et al.[17] has shown that a reduction of the feature space can indeed improve the accuracy of the classification model. Two possible techniques are feature subset selection or dimensionality reduction. Examples of feature subset selection methods are filters (based on statistical measures or information gain) or wrapper methods that search the space of feature subsets evaluating the accuracy for each subset (Janecek et al.[17]). Dimensionality reduction can be implemented using Principle Component Analysis (PCA)[24] and the results are more accurate[17] as PCA takes into account all features of the original feature set to calculate a new feature set, rather than selecting a subset. Figures 3.3 (NNN) and 3.4 (RVB) show the first three principle components calculated from the features of the training set for each sample, where the different markers represent the different fills. From these figures, it can already be seen that the set of solid shapes is perfectly separable from the set of open and half-filled shapes. The open and half-filled shapes are not perfectly separable using only three principle components, so this may become a problem for training the classification model (section 3.3.4). However, using more principle components and thus more information, there might be a clearer boundary between the two classes.

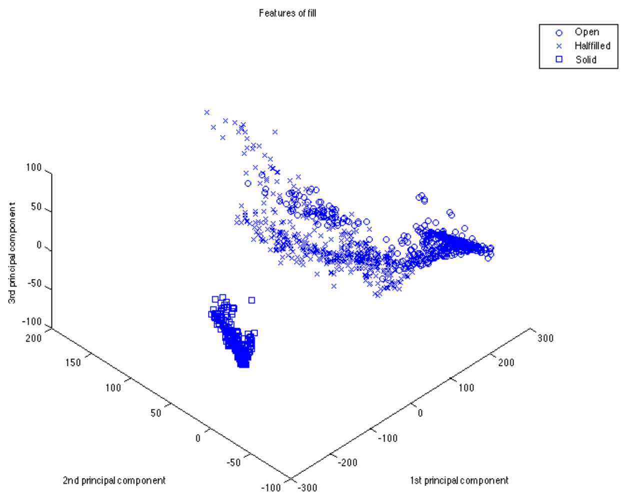


Figure 3.3: The first three principle components of the fill features for NNN.

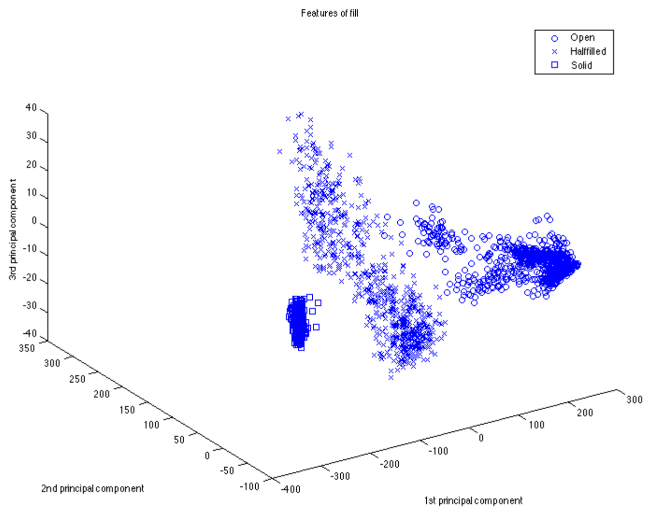


Figure 3.4: The first three principle components of the fill features for RVB.

3.3.4 Training the classification model

Upon the set of features that remains after the feature set reduction we will train a classification model. In this paper, we will consider the following models:

- **Support Vector Machines**[14] - find separation of training samples by maximizing the distance to the boundaries. Support vectors are the vectors on those boundaries.
- **K-Nearest Neighbor**[5] - predict the class of the feature vector in the training set that has minimum Eucledian distance to the feature vector to classify.
- **Normal Bayes Classifier**[11] - predict the class that is most likely considering the (Guassian) distributions of the features.
- **Gradient Boosted Trees**[10] - an iterative scheme is used to Boost decision trees to create a powerful multi classification model.

The selection of these models is based on the idea that feature vectors representing the fill and the color can be observed as realizations from three multidimensional probability distributions, one for each of the classes. The imaginary 'perfect shape image' would have a feature vector consisting of the means of each of the feature distributions. Feature vectors of the images fill up the multidimensional feature space and the target of a classification model is to create boundaries of regions in this space which correspond to the different classes, or equivalently, to find the distributions of the features such that the class with maximum likelihood can be predicted (which implicitly also defines boundaries in the space). The Normal Bayes Classifier uses the additional assumption that all features are normally distributed. Although we could validate this using statistical analysis, we think the assumption is justified if the model performs well. The performance, as estimated by the accuracy, will be discussed in chapter 6.

3.4 Classifying the color

The framework for classification of the color is similar to that used for classification of the fill. The preprocessing consists of converting to the best color space and selecting the relevant pixels. In this case, we will not select a region but a set of pixels (the pixels of interest, or POI).

3.4.1 Preprocessing the image

Preprocessing of the image for color classification consist of selecting the best color space to represent the image. The standard RGB color space is not very good for this purpose as color representations vary highly under different lighting conditions. A good color space for classification consists of individual components for the 'actual' color and darkness. Possible alternatives for three-dimensional color spaces are:

- **HSV**: The hue (H) is an angle which indicates the 'color direction', the saturation (S) gives the intensity of the color and the value (V) the darkness. This is a more intuitive representation of color and usage of the color space can lead to better results when using for image segmentation[22].
- **YCrCb**: Widely used color space that separates the luminance (Y) and two chrominance components (Cr and Cb).
- **YES**: Alternative color space to separate luminance (Y) and chrominance components (E and S)[20].

Experiments show that usage of the HSV color space causes new problems as the main color component (H) is circular rather than linear. That means that the values of 0 and 360 degrees are equivalent, so it is not possible to calculate a meaningful mean by taking the average of the values. This problem can be overcome using circular statistics, but then still problems arise as the calculated mean itself is also circular. Classification models often use Euclidian distances internally, which are, in some cases, incorrect for circular values.

In contrast to the HSV color space, the YCrCb and YES color spaces use two components for the color representation, which can be represented in a plane that does not wrap around. As the two components can be considered as coordinates in the plane, Eucledian distances are valid and therefore these color spaces are better used to calculate features as input to classification models. From their definitions the it is clear that both spaces are similar (with two chrominance components and one for luminance), therefore we will only consider the more common YCrCb space in this paper.

3.4.2 Selecting pixels of interest

The images of the shapes contain many white pixels, especially if the shape is open. As these white pixels provide no information on the color of the shape, it is important to disregard them and select only those pixels that do contain color before calculating the features. We use the simple observation that when the image is in grayscale, the contour of the shape (which is colored in the normal representation) will always be darker than the white background. Therefore, we extract some of the darkest pixels as the pixels that contain the color. This we do by sorting the pixels in ascending order by their luminance (Y) level and selecting a range amongst the first sorted pixels. The range of pixels that should be selected is determined by manual experiments. A larger set of pixels naturally contains more information, but also might reduce the accuracy of the features when irrelevant pixels are included. Especially in the case of an open shape the number of relevant pixels is very low: there is only a thin contour on the large white surface. Moreover, in some images of open shapes, the darkest pixels in the center of the contour are registered very dark by the camera as they are contrasted against the white background. This means that the color is hard to distinguish from these pixels, while at the edges of the contour, there is some diffusion with the white background, resulting in lighter pixels in which the color can be better observed. Therefore, it makes sense not to select the overall most dark pixels, but a range of pixels just after those. For the results in this paper, we used the range from 21 to 60 darkest pixels. This range might look very small to provide accurate values, but remember that it is taken from the sorted values and therefore already contains much information[3].

3.4.3 Feature calculation

Also for classification of the color we will try to create a set features that is representative upon which a classification model can be built. The set of candidate features is:

- **Mean Cr and Cb** - the means of the Cr and Cb values of the pixels of interest.
- **Quantiles of the Cr and Cb distribution** - the 5%, 25%, 75% and 95%-quantiles of the distributions of the Cr and Cb values of the pixels of interest.
- **Lightness** - the 95%-quantile of the distribution of the Y values of the *full* image.

Mean Cr and Cb

The mean Cr and Cb are the main estimators for the location coordinates of the color within the CrCb plane.

Quantiles of the Cr and Cb distribution

The quantiles of the Cr and Cb distribution may provide additional information about the color. In some cases the mean might not be sufficiently distinctive for the color while the quantiles represent the distribution of the color, which might improve accuracy.

Lightness

The lightness is a robust estimator for the light intensity as the 95%-quantile of the luminance corresponds to a white pixel for all of the fills as each image contains over 5% white pixels. This value provides no information about the color itself, but about the lighting conditions which might influence the Cr and Cb values. For example, under heavy lighting conditions a red color might appear as purple, but can still be correctly classified if this is taken into account by the model. Also, when it is darker, colors are harder to distinguish and thus get closer to each other in the color space. This may cause the effect we can observe in figure 3.5: the points are closer together for lower values of the second principal component.

[3]Another example in which much information is contained in a small range of values is the median (which is a range of only a single value): this is also a single value but representative as it is taken from a sorted sequence.

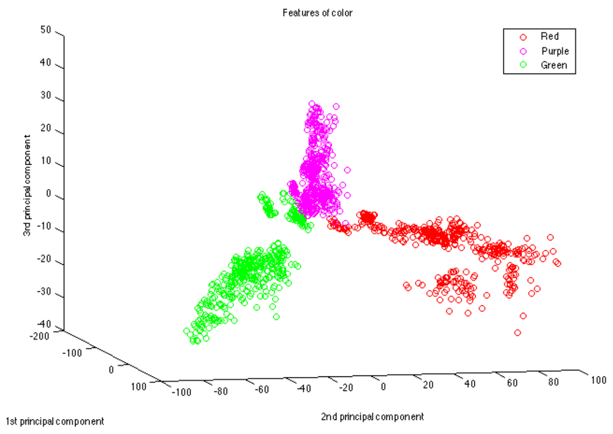


Figure 3.5: The first three principle components of the color features for NNN.

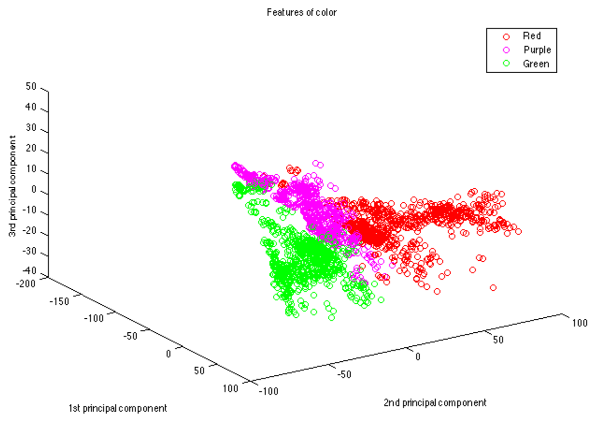


Figure 3.6: The first three principle components of the color features for RVB.

3.4.4 Feature set reduction and model training

Just as we did for the fill, we will try to find a set of features that is most representative for the color using Principle Component Analysis. The plots of the first three principle components for these features are in figure 3.5 (NNN) and figure 3.6 (RVB). As with the fill, we clearly see regions corresponding to classes, but the classes are not very clearly separable. Next we will train the models described in section 3.3.4 exactly as we did for the fill, but with the principle components of the features for the color.

Chapter 4

Extracting card metadata

When we have used the 'top-down' approach, we have to combine the classifications of the shapes of a card. If all shapes are classified equally, we may assume that the card is of the corresponding class. However, if the two or three shapes have been classified differently, we must conclude that we are unable to classify the card. The other option is to define a classification of the card based on the different classifications of the shapes on the card, but this is undesirable for reasons that will be discussed with the 'bottom-up' approach.

When we have used the 'bottom-up' approach, that is we located and classified individual shapes, we will group the shapes together to find the cards. This we do by a labeling algorithm: as long as there is an unlabeled shape, we label this shape with the next card index and we recursively find neighboring shapes which we label with the same index. A neighboring shape is a shape which has the same color, fill and shape as the current shape and is within a maximum Euclidean distance (from center to center) from the current shape. This maximum distance is determined as the average width of the rotated bounding rectangles of all shapes. Observing the cards it can be seen that two shapes on the same card will always be within this distance[1], while it is less likely (though not impossible) that a shape on a neighboring card is within this distance. To be safe a check on color, fill and shape is implemented, which means that shapes are only regarded as being on the same card if they are classified as the same shape, fill and color. The result of this implementation is that when one of the shapes on a card is classified incorrectly, it will result in two separate cards being recognized. One could think of implementing this in an alternative way in which neighboring cards with different classifications in just one of the properties could be corrected as they are most likely a single card with one of both classes being correct (this is the same case as with the 'bottom-up' approach). We choose not to do this for three reasons:

1. It enlarges the risk of matching shapes as a single card while they actually belong to two different cards.

2. The method will not improve accuracy for cards with a single shape so it is actually not an improvement on the classification itself.

3. The implementation used for Multi-class SVM does not provide a numeric value such as a score or confidence so it is not possible to predict the class

[1]The width of the NNN shape diamond as measured by the width its bounding rectangle is a bit smaller, but taking the average over all shapes still results in an 'average width' such that centers of shapes belonging to the same card will be within this distance to each other.

> with the most confidence or to calculate average scores for each class. In the case there are three shapes a majority vote could be used, but if there are two shapes or three shapes with all different values, this results in a tie. Predicting a class arbitrarily is simply guessing and therefore undesired.

However, it could be an interesting topic for further research to modify the prediction models such that a measure for the certainty is obtained and to use this for better performance. For instance, one could think of using three binary classification models rather than a single multi-class model, which, in case of SVM, dó provide a numeric value that can be used as a measure for the certainty.

After the groups have been formed, each group corresponds to a card of which the fourth property, the quantity, simply is the number of shapes in the group. The first three properties can be taken from any shape arbitrarily, as they are all equal. The location of the card, represented by its center, is irrelevant for the finding of the SETs but might be use for visual indication of cards. It can by found by taking the average of the centers of the bounding rectangles of the shapes.

Chapter 5

Finding of SETs

The actual finding of the sets is a problem of complexity $O(n^2)$, with n the number of cards. To see this, we use the representation of a SET card as a four-dimensional point. Then a SET is equivalent to a set of 3 points that are collinear in the four-dimensional space[6]. This means that each pair of two cards forms a SET with only one of the other cards. Using this we can find all sets in a collection of cards by systematically going through all the $\binom{n}{2}$ pairs and checking whether the complementing card is also in the collection. If we build an index of the cards we can do this check in $O(1)$ time. Therefore the algorithm will run in $O(n^2)$ time. To prevent finding the three permutations of the same set, we will only add a set to the set collection if the cards are in natural order, that is when the third card has an index larger than the first two.

Algorithm 1 formalizes the method. The method takes as input the integer n and vectors qty, $color$, $fill$ and $shape$, each of length n, in which entry i has a value of 1, 2 or 3 that represents the class of the quantity, color, fill and shape of card i respectively. The first part builds a four dimensional lookup table that has a value of 0 if the card is not available, or the index (1 to n) of the card that has the four properties corresponding to the indices of the table. Then the second part iterates over all pairs of cards and checks whether the third card is available. If so, it checks if the index of this third card is larger than the index of the second card because otherwise this set has been found before. Finding the third card is done per feature: if the feature of the two cards in the pair is equal, the third card should have that feature as well. Otherwise, the three cards should have three different feature values of 1, 2 and 3, so the third value is always found by taking 6 minus the first two values.

Algorithm 1 Finding SETs

```
procedure FINDSETS($qty, color, fill, shape, n$)
    for $i \leftarrow 1, 3$ do
        for $j \leftarrow 1, 3$ do
            for $k \leftarrow 1, 3$ do
                for $l \leftarrow 1, 3$ do
                    $cards[i][j][k][l] \leftarrow 0$          ▷ Initialize card lookup table
                end for
            end for
        end for
    end for
    for $i \leftarrow 1, n$ do
        $cards[qty(i)][color(i)][fill(i)][shape(i)] \leftarrow i$
    end for
    nSets = 0;
    for $i \leftarrow 1, n-2$ do
        for $j \leftarrow i+1, n-1$ do
            $qty3 \leftarrow getThirdFeature(qty(i), qty(j))$
            $color3 \leftarrow getThirdFeature(color(i), color(j))$
            $fill3 \leftarrow getThirdFeature(fill(i), fill(j))$
            $shape3 \leftarrow getThirdFeature(shape(i), shape(j))$
            $matchedCard \leftarrow cards[qty3][color3][fill3][shape3]$
            if $matchedCard > j$ then    ▷ If matchedCard is zero there is no set, if $0 < matchedCard < j$ this set has been found already.
                $nSets \leftarrow nSets + 1$
                $sets[nSets][1] \leftarrow i$
                $sets[nSets][2] \leftarrow j$
                $sets[nSets][3] \leftarrow matchedCard$
            end if
        end for
    end for
    return sets
end procedure
procedure GETTHIRDFEATURE($feature1, feature2$)
    if feature1 = feature2 then
        return feature1
    else
        return 6 - feature1 - feature2
    end if
end procedure
```

Chapter 6

Results

This chapter discusses the results. First, the different approaches for localization of the cards will be compared in section 6.1. The rest of the chapter covers the results in terms of accuracy of the classification of individual shapes. Section 6.2 will discuss the training set and section 6.3 the experiment setup. The next three sections will present the results for the individual properties shape, fill and color respectively. Section 6.7 presents the accuracy of the combined classification method.

6.1 Comparing the card localization approaches

The 'top-down' approach is the method that has been studied and used in similar applications. It has the advantage that a perspective transform is used to extract the card image, so it corrects (to a certain extent) perspective distortion, which may improve classification accuracy of the shape. However, the approach has the major disadvantage is that it does not work when cards overlap. It is hard to design an experiment setup in which the implication of this is tested as it is hard to simulate 'the laying down of cards', but in practice this happens 'a lot', so the results of this approach are bad. Fortunately, the second method (the 'bottom-up' approach) is able to successfully detect the shapes in the image. The extraction of shapes using affine transformations based on bounding rectangles does not correct for perspective, but if the angle is within a reasonable range (from perpendicular to the surface up to 45 degrees), the shapes are still sufficiently recognizable. Contrasting to the first method, the second method cán handle cards that overlap, as long as they do not cover parts of shapes of other cards. Such a big overlap is rarely the case, although small overlaps occur frequently, so the advantage of 'bottom-up' approach is beneficial. As we saw in section 4, using the 'bottom-up' approach may result in cards being recognized as two cards. Although this is a bad result, it is not a reason to favor the 'top-down' approach is this approach, although it would recognize just one card, would not be able to classify it correctly. Therefore the final implementation (chapter 7) uses the 'bottom-up' approach.

6.2 The training set

We started with the generation of a training set of over 1000 images of shapes for each of the two editions. For this, pictures of the entire card deck of 81 cards have been taken, 9 pictures of 9 cards at a time, under 7 different lighting conditions and angles. The pictures were taken by daylight, by direct lumination and in the

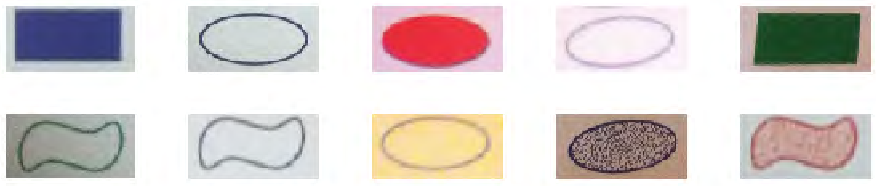


Figure 6.1: Random sample of ten shapes from the RVB dataset.

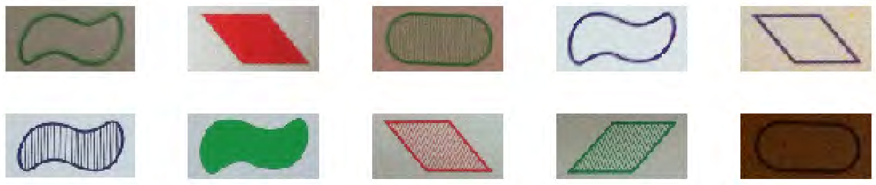


Figure 6.2: Random sample of ten shapes from the NNN dataset.

direct shadow of an artificial light, all under multiple different angles. From each of the pictures, the individual shapes have been extracted using the method in section 2.2. The images were stored with their color, fill and shape: for each of the $3^3 = 27$ combinations the resulting dataset contained about 50 images of 100×50 pixels. After a first implementation it turned out that the results were unsatisfactory in practice, contrasting the high accuracy values that were found using crossvalidation. Therefore a method has been implemented to correct predictions during actual use of the program, which resulted in new images being added to the dataset continuously. The final dataset that was used to find the results in this paper contained 2266 images for Ravensburger and 1299 for NNN Games. Figure 6.1 and 6.2 give an example of ten shapes from the RVB and NNN dataset respectively.

6.3 Experiment setup

We designed an experimental setup in which the model was tested using five fold cross validation. Therefore we randomly divided the dataset in five subsets of approximately equal size. We implemented an iterative scheme in which in each of the five iterations we trained the model on the union of four subsets, which we called the training set, and applied the trained model on the 5th set, the test set. As the problem is a classification problem, a prediction of the model is either correct or incorrect, but there is no scale. Therefore, we introduce a measure of accuracy which is simply the percentage of correctly classified instances. To find the best model and feature set, we runned this scheme on each of the four models for different numbers of principle components, to determine the optimal combination of features with a model. All results are based on the same division of the training set into folds, for best comparison.

6.4 Shape

The accuracy of the classification of the shape as it would be in the whole framework cannot be measured exactly using the training set. This is because the method uses directly the found contours, which result from various filters over the entire image, instead of the individual images of the shapes. However, we can get a good impression of the accuracy if we are able to find the contour in the image. This we do by applying a slightly modified methodology, in which we take only outer contours (as we have no cards) within the picture. The largest one is expected to represent the shape. The results are very good: for the RVB dataset the accuracy is 99.74% and for NNN 99.92%.

6.5 Fill

Figure 6.3 shows the results for RVB. On the x-axis there are the different numbers of principle components used in the experiment, while each series corresponds to one of the four models used. Although the differences are small, SVM performs best with an accuracy between 99.60% and 99.87%, where the higher accuracy is reached with more principle components as input. Figure 6.4 shows the results for NNN. For two principle components the accuracy is rather low, but for three or more we see an increasing accuracy. Again, SVM performs best with an accuracy of 96.38% for two principle components up to 99.54% for all seven components. The differences with KNN and NBC are small, but GBT performs significantly worse. To test the significance, we apply a Chi Squared test on the contingency table with four rows corresponding to the classification methods and two columns corresponding to correct and incorrect classifications. Each cell counts the number of (in)correctly classified instances for the classification model corresponding to the row, so we have the model with fixed row marginals, equal to the size of the dataset. We test the null hypothesis that the (Bernoulli) distribution of correct and incorrect classifications is independent of the classification model. Using all principle components, the p-value is smaller than 2.2×10^{-16} for NNN and equal to 0.04839 for RVB. With a significance level of $\alpha = 5\%$ we reject the null hypothesis in both cases and conclude that the distribution of correct and incorrect classifications (and thus the accuracy) significantly depends on the classification model. For NNN this result is stronger, which complies with the figures.

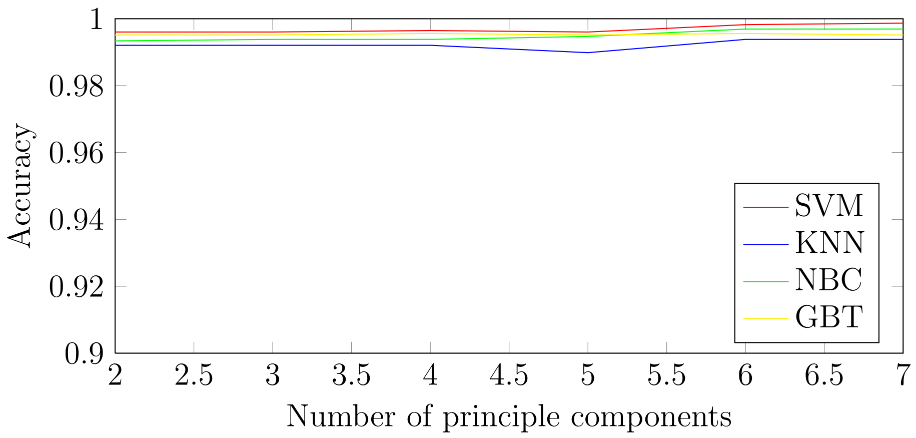


Figure 6.3: Relationship between the number of principle components and the prediction accuracy of the fill for the Ravensburger edition.

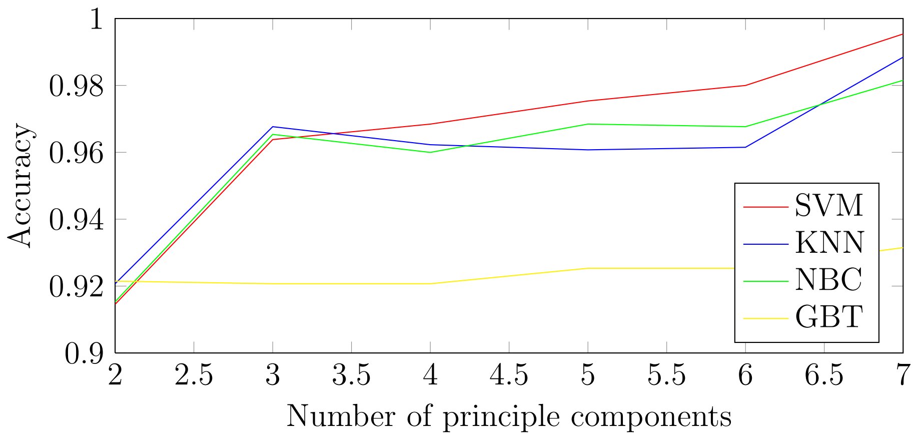


Figure 6.4: Relationship between the number of principle components and the prediction accuracy of the fill for the NNN Games edition.

6.6 Color

With the color, we found different results of accuracy than we found with the fill. For RVB, the accuracy is highest for SVM, but NBC is very close. Also we observe that using more principle components results in an increase in accuracy. The accuracy is 98.59% for SVM with all components. GBT has a lower accuracy and does not perform any better using more principle components. KNN clearly performs the worst, and seems a bit random for the different numbers of principle components. For NNN, the results are different: NBC performs better than SVM for most principle components, but using the full representation SVM performs best, although the difference with NBC is minimal. The accuracy is 99.92% for the full model with all 11 components. Again, we apply the Chi Squared test for contingency tables to test the influence of the different classification models. Using all principle components, the p-value is less than 2.2×10^{-16} for both brands, so again we conclude the influence of the classification model is significant.

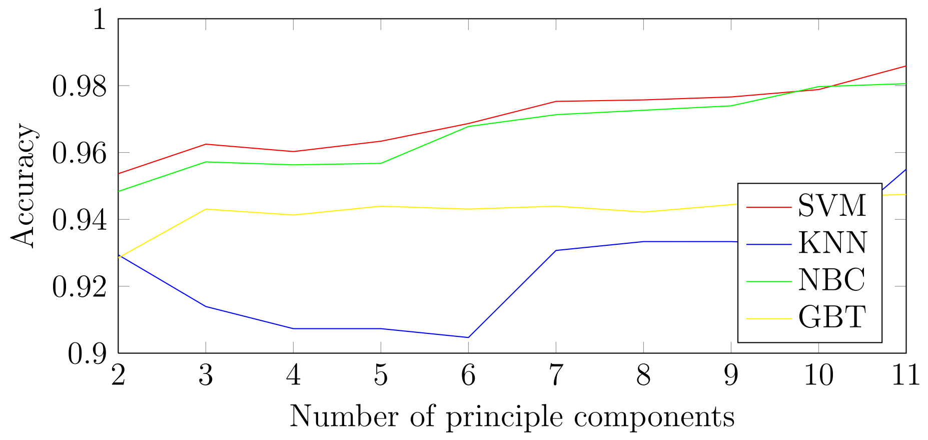


Figure 6.5: Relationship between the number of principle components and the prediction accuracy of the color for the Ravensburger edition.

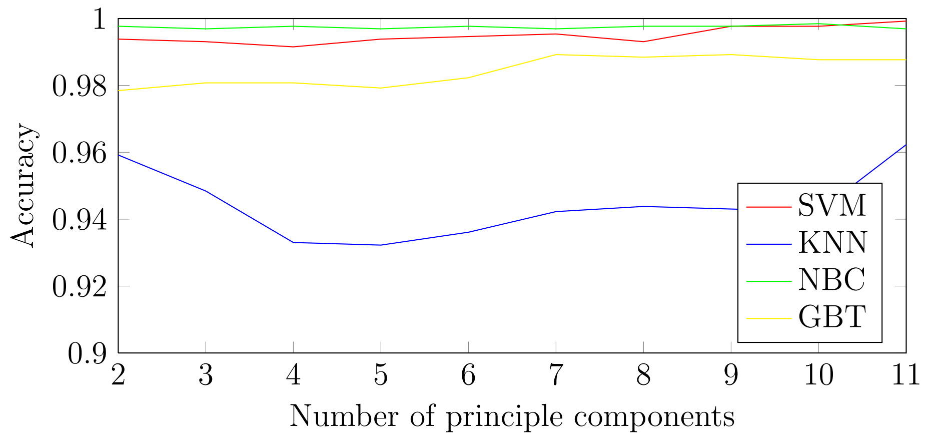


Figure 6.6: Relationship between the number of principle components and the prediction accuracy of the color for the NNN Games edition.

6.7 Combined

This section presents the combined results. Table 6.1 shows the final prediction accuracies for the color, fill and shape and all properties combined. For the color and fill the results are for the full model with SVM classification using all principle components. The column 'All' shows the percentage of instances of which all three properties were correctly classified. This can be regarded as the accuracy for the full classification problem and it is 95.7% for Ravensburger 99.2% for NNN Games. From the table it can be observed that the major culprit for the lower accuracy for Ravensburger is the color property.

Table 6.2 shows the same overall accuracy, but together with three columns with the percentages of instances that have exactly 1, 2 or 3 properties incorrectly classified. Therefore each row represents the distribution of the number of incorrectly classified properties for that edition. It can be observed that the percentage of instances that have more than one property incorrectly classified is very low. This

Edition	Color	Fill	Shape	All
Ravensburger	96.03 %	99.87 %	99.74 %	95.72 %
NNN Games	99.77 %	99.46 %	99.92 %	99.15 %

Table 6.1: The final prediction accuracies for the individual color, fill and shape and the combination of all properties.

indicates that there is no significant dependence between the classification accuracies of the different features.

Edition	All correct	1 incorrect	2 incorrect	3 incorrect
Ravensburger	95.72 %	4.19 %	0.09 %	0.00 %
NNN Games	99.15 %	0.85 %	0.00 %	0.00 %

Table 6.2: The final prediction accuracy for all properties and the percentages of instances of which 1, 2 or 3 properties were incorrectly classified.

Chapter 7

Implementation: the SET app

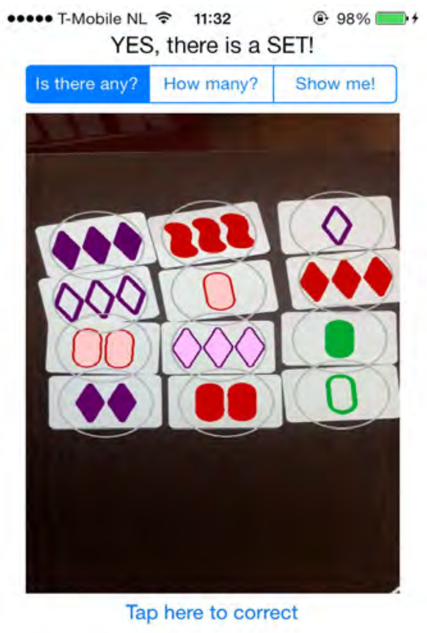
The implementation has been done using the OpenCV [1] framework, which has been connected to Matlab using mexopencv [2]. Prototyping of the application has been done in Matlab and this implementation has been used to derive the results presented in chapter 6. In a later stadium, an implementation in the form of an iOS app has been made. Therefore, the implementation has been optimized and rewritten in C++. The app consists of a one page user interface which shows a real time video stream from the camera. Tapping the screen will capture an image, which will be processed after which the desired information can be shown. The user can choose three modes:

- **Is there any?** - the app will display YES or NO depending on whether at least one SET was found or not.
- **How many?** - the app will display the number of SETs that were found (there may be overlap in the SETs but each SET is unique).
- **Show me!** - the app will display the number of SETs that were found and mark them by displaying colored circles around the cards.

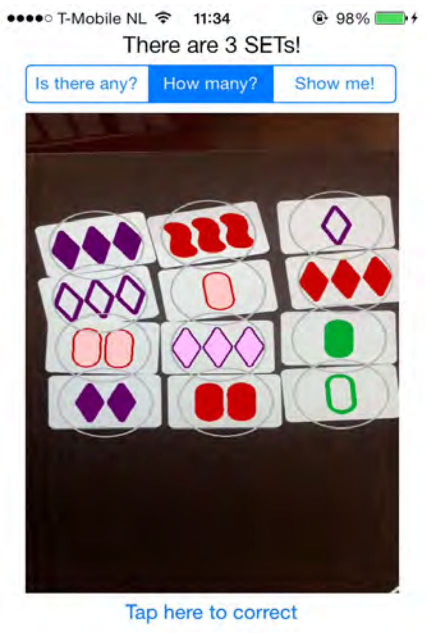
In each of the modes, the app will mark the shapes that it has recognized. This is done by drawing a thick contour in the recognized color as an overlay to the image. Depending on the recognized fill, the contour will be open, filled with a light color (which represents half filled) or filled with the same dark color as the contour. The cards will be marked with a gray circle, which has as a radius the average width of the shapes. If the user selected the option to show the SETs, the cards will not be marked with a gray circle, but with a circle in the color of the SET it is part of. If a card is part of multiple SETs, multiple circles will be drawn. The centers of these multiple circles are chosen uniformly on a small circle around the actual center of the card so that they will not coincide. Table 7.1 shows the resulting implementation of the different modes in the app.

[1]OpenCV (`http://opencv.org`) stands for Open Source Computer Vision. It is a widely used open-source framework that implements many algorithms that can be used for computer vision applications and a wide variety of classification- and regression models.

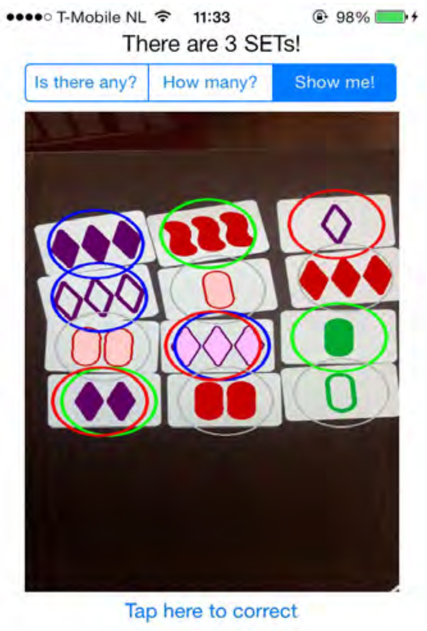
[2]mexopencv (`http://www.cs.stonybrook.edu/~kyamagu/mexopencv/`) is a software package that provides an interface to the OpenCV framework within Matlab using mex functions.



a)



b)



c)

Table 7.1: Screenshots of the SET app implementation in the three different modes.

Chapter 8

Discussion

Although the classification of individual shapes is quite accurate, we noticed that practical application of the framework is limited. Using basic math shows us that the probability of correctly classifying all cards is indeed rather low. Assuming that all the shapes are located correctly, we can estimate this probability: with 12 cards the average number of shapes is 24 so, roughly, assuming that the correct classification of the shapes is independent, the board is correctly classified with probability $0.957^{24} = 0.350$ for Ravensburger and $0.992^{24} = 0.815$ for NNN Games. The assumption of independence is highly questionable (as lighting conditions may influence classification of all shapes), but the numbers illustrate the effect of the high dimensionality of the problem. The result confirms what is observed when using the app: it happens frequently that not all shapes are correctly classified, although this is heavily dependent on lighting conditions. Nevertheless, the app can be of practical use with the implementation of a small feedback mechanism which allows the user to make a correction.

When a single card is regarded as a single instance the results are quite accurate, although in practice it is still sometimes observed that an image is misclassified while it is, to human perception, 'very clear'. This is unexpected as, with the high accuracy achieved on the test set, one would expect only misclassifications on 'border cases': images that appear very similar also to humans. This indicates that the model might still not have learned the true characteristics of each class: although it fits the training set quite well it is not representative for the entire population[1]. This means that the dataset either misses particular cases or is unbalanced, that is, contains many irrelevant samples and too few important samples. This is not represented in the resulting measures as the test set in the cross validation scheme also originates from the dataset. This is a practical limitation as it is often hard to create a dataset that is balanced and represents the entire population, while it is generally an assumption of the model that this is the case. Borovicka et al.[1] introduce several techniques which could be used to overcome parts of this problem. Using these techniques on the dataset for improvement of the model could be a topic for further research.

The recognition of the shape turned out to be an easy problem as almost all of the shapes were correctly classified by simply comparing binary contour images. The

[1]The problem that a machine learning algorithm learns something that it should not have learned occurs more often. In 1998, Neil Fraser wrote about a US military project in which a system was trained to recognize tanks from images. Although the system performed well on both the training and test set, its results were poor on new instances. It turned out that the system had actually learned to recognize the weather conditions, as all pictures with tanks were taken in bad weather and those without tanks in the sun. `https://neil.fraser.name/writing/tank/`.

accuracy of the fill prediction is quite high. Solid shapes are classified correctly in practically any case, but there still is some interference between open and half-filled shapes. The different lighting conditions can make open shapes appear as textured and the texture of a half-filled shapes varies with the quality of the picture. Further research could look at the possibilities of using actually real texture descriptors[13] rather than just features indicating the 'amount of texture' as features related to the fill. The color classification algorithm performs well when tested using cross validation, but gives unexpected results occasionally. In this direction, further research could be done on the influence of additional preprocessing, especially focusing on the color. Finlayson et al.[8] suggest a method for color image normalization, eliminating dependency on lighting geometry and illumination color. However, it should be checked if it is suitable for the sparsely colored set images of open shapes. Alternatively, Stottinger et al.[21] introduce light-invariant color interest points for image classification which could be used as features.

Bibliography

[1] T. Borovicka, M. Jirina, P. Kordik & M. Jirina, *Selecting Representative Data Sets*, Advances in Data Mining Knowledge Discovery and Applications, Associate Prof. Adem Karahoca (Ed.), ISBN: 978-953-51-0748-4, InTech, DOI: 10.5772/50787, 2012.

[2] L.G. Brown, *A survey of image registration techniques*, ACM Computing Surveys, Volume 24 Issue 4, December 1992, p. 325 - 376, 1992.

[3] J. Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume PAMI-8, Issue 6, November 1986, p. 679 - 698, 1986.

[4] W. Cooper & K. Dawson-Howe, *Automatic blackjack monitoring*, Proceedings of Irish Machine Vision Conference, p. 248 - 254, 2004.

[5] T. Cover & P. Hart, *Nearest neighbor pattern classification*, IEEE Transactions on Information Theory, Volume 13, Issue 1, January 1967, p. 21 - 27, 1967.

[6] B.L. Davis, D. Maclagan & R. Vakil, *The card game set*, The Mathematical Intelligencer, Volume 25, Issue 3, Summer 2003, p. 33 - 40, 2003.

[7] D. Douglas & T. Peucker, *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer, Volume 10, Issue 2, December 1973, p. 112 - 122, 1973.

[8] G.D. Finlayson, B. Schiele & J.L. Crowley, *Comprehensive colour image normalization*, Computer Vision-ECCV'98, p. 475 - 490, Springer Berlin Heidelberg, 1998.

[9] H. Freeman & R. Shapira, *Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve*, Communications of the ACM, Volume 18, Issue 7, July 1975, p. 409 - 413, 1975.

[10] J.H. Friedman, *Greedy Function Approximation: A Gradient Boosting Machine*, The Annals of Statistics, Volume 29, Issue 5, October 2001, p. 1189 - 1232, 2001.

[11] K. Fukunaga, *Introduction to Statistical Pattern Recognition. second ed.*, New York: Academic Press, 1990.

[12] G. Green, *An Essay on the Application of Mathematical Analysis to the Theories of Electricity and Magnetism*, author, 1828.

[13] R.M. Haralick, K. Shanmugam & I. Dinstein, *Textural Features for Image Classification*, IEEE Transactions on Systems, Man and Cybernetics, Volume SMC-3, Issue 6, November 1973, p. 610 - 621, 1973.

[14] M.A. Hearst, S.T. Dumais, E. Osman, J. Platt & B. Scholkopf, *Support vector machines*, IEEE Intelligent Systems and their Applications, Volume 13, Issue 4, July/August 1998, p. 18 - 28, 1998.

[15] G. Hollinger, N. Ward & E.C. Everbach, *Introducing Computers to Blackjack: Implementation of a Card Recognition System using Computer Vision Techniques*, Colby College, Waterville, 2003.

[16] M. Hu, *Visual pattern recognition by moment invariants*, IRE Transactions on Information Theory, Volume 8, Issue 2, February 1962, p. 179 - 187, 1962.

[17] A. Janecek, W. Gansterer, M. Demel & G. Ecker, *On the relationship between feature selection and classification accuracy*, Journal of Machine Learning Research: Workshop and Conference Proceedings 4, 2008, p. 90 - 105, 2008.

[18] J. Matas, C. Galambos & J. Kittler *Robust Detection of Lines Using the Progressive Probabilistic Hough Transform*, Computer Vision and Image Understanding, Volume 78, Issue 1, April 2000, p. 119 - 137, 2000.

[19] C.W. Niblack et al., *The QBIC project: querying images by content, using color, texture, and shape*, S&T/SPIE's Symposium on Electronic Imaging: Science and Technology. International Society for Optics and Photonics, 1993.

[20] E. Saber, A. Murat Tekalp, R. Eschbach & K. Knox, *Automatic Image Annotation Using Adaptive Color Classification*, Graphical Models and Image Processing, Volume 58, Issue 2, March 1996, p. 115 - 126, 1996.

[21] J. Stottinger, A. Hanbury, N. Sebe & T. Gevers, *Sparse Color Interest Points for Image Retrieval and Object Categorization*, IEEE Transactions on Image Processing, Volume 21, Issue 5, May 2012, p. 2681 - 2692, 2002.

[22] S. Sural *Segmentation and histogram generation using the HSV color space for image retrieval*, Proceedings International Converence on Image Processing, 2002, p. 589 - 592, 2002.

[23] S. Suzuki & A. Keiichi, *Topological structural analysis of digitized binary images by border following*, Computer Vision, Graphics, and Image Processing, Volume 30, Issue 1, April 1985, p. 32 - 46, 1985.

[24] S. Wold, K. Ebesen & P. Geladi, *Principal Component Analysis*, Chemometrics and Intelligent Laboratory Systems, Volume 2, Issues 1-3, August 1987, p. 37 - 52, 1987.

[25] C. Zheng & R. Green, *Playing Card Recognition Using Rotational Invariant Template Matching*, Proceedings of Image and Vision Computing, New Zealand, December 2007, p. 276 - 281, 2007.

[26] K. Zutis & J. Hoey, *Who's Counting? Real-Time Blackjack Monitoring for Card Counting Detection*, Lecture Notes in Computer Science, Volume 5815, 2009, p. 354 - 363, 2009.