Solving Captchas with an OCR system

Sebastian Agethen, D01944015
Lin Sheng-Ching, R99222030
Jeroen Dhondt, A01922201

June 18, 2013

Contents

1 Introduction

2 Implementation

We first introduce the structure of our system. In the following sections we then present each step in greater detail.

2.1 General structure

Our OCR system runs in two stages, a training phase and a live phase. In the training phase, each our our features is evaluated on a set of 36 Bitmaps describing the characters we can recognize. Currently, this encompasses the 26 capital characters of the English alphabet as well as the digits 0 to 9. In the second phase, called live phase, we extract segments from an input file and then compute all features for each segment. Finally, training data and live data are passed together with a weight vector to a clustering method, which finds either the closest or the k-closest matching results.

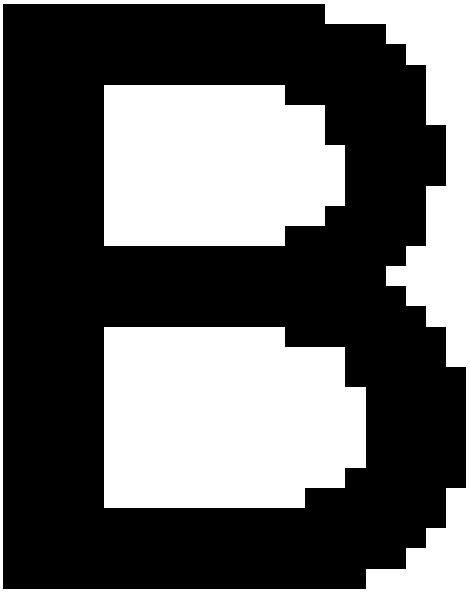
2.2 Preprocessing

2.2.1 Quantization

- Set a threshold t
- $v = \sqrt{r^2 + g^2 + b^2}$
- If $v < t$, set the pixel $p = 0$; if $v \geq t$, set the pixel $p = 1$.

2.2.2 Skeletonization

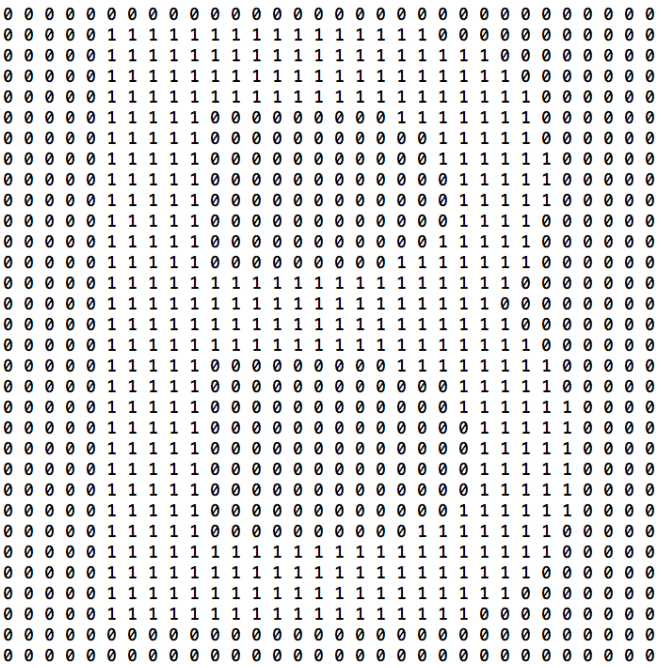
- Use the result of quantization
- Repeatedly use the conditional mark pattern and unconditional mark pattern until the nothing is changed
- Use the bridge operation to recover the connectivity



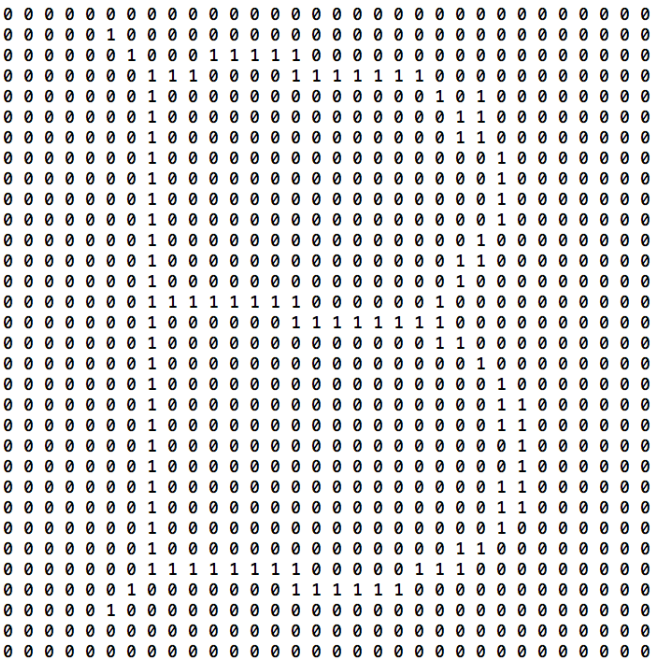
(a) Original B

(b) Quantised B

Figure 1: Original B v.s. Quantised B



(a) Quantised B



(b) Skeletonized B

Figure 2: Qunatized B v.s. Skeletonized B

2.2.3 Boundary extraction

- Scan the pixel from left to right for each row.
- When the gradient changes strictly, set the row number as the ceiling of the sentence.
- After finding the ceiling, keep scanning the pixels. If we find a totally white row, set the row number - 1 as the bottom of the sentence.
- Scan the pixels of each column from the ceiling to the bottom.
- When the gradient changes strictly, set the column number as the left boundary of the sentence.
- After finding the left boundary, scan the pixels from the rightmost pixel to the left, from ceiling to the bottom. As we find a strict gradient of the column, set it as the right boundary.
- Keep doing the steps 1 6 until all the input file is scanned.
- For each sentence, set the left boundary as the left boundary of the first chacter and scan the pixel from ceiling to the bottom, starting from the left.
- If there is a column totally white, set the column number -1 as the right boundary of the chacter.
- Keep scanning until a strict gradient occurs. Set the column number as the left boundary of the next chacter.
- Keep scanning until another strict gradient column is found and the right boundary of the chacter is set.
- Repeat step 9 11 until the sentence is totally scanned and scan the next sentence.

2.2.4 Orientation

The followings are the derivation of the orientation angle of an object: Define the (m,n)th spatial moments

$$M(m,n) = \frac{1}{J^n K^m} \sum_{j=1}^{J} \sum_{k=1}^{K} (x_k)^m (y_j)^n F(j,k) \quad (1)$$

where $x_k = k - \frac{1}{2}, y_j = J + \frac{1}{2} - j$ and

$$U(m,n) = \frac{1}{J^n K^m} \sum_{j=1}^{J} \sum_{k=1}^{K} \left(x_k - \frac{M(1,0)}{M(0,0)} \right)^m \left(y_j - \frac{M(0,1)}{M(0,0)} \right)^n F(j,k) \quad (2)$$

where $x_k = k - \frac{1}{2}, y_j = J + \frac{1}{2} - j$ and then find the maximum value of eigen value

$$E^T U E = \Lambda \quad (3)$$

$$E = \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \quad (4)$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad (5)$$

By calculation,

$$\lambda_1 = \frac{1}{2} [U(2,0) + U(0,2)] + \frac{1}{2} \left[U(2,0)^2 + U(0,2)^2 - 2U(2,0)U(0,2) + 4U(1,1)^2 \right]^{1/2} \quad (6)$$

$$\lambda_2 = \frac{1}{2} [U(2,0) + U(0,2)] - \frac{1}{2} \left[U(2,0)^2 + U(0,2)^2 - 2U(2,0)U(0,2) + 4U(1,1)^2 \right]^{1/2} \quad (7)$$

$$\rightarrow \quad \theta = \arctan \left[\frac{\lambda_M - U(0,2)}{U(1,1)} \right] \quad (8)$$

2.3 Features

In the following we present each feature we used in detail.

2.3.1 Geometrical features

Area Calculate the total number of pixels

Weight center

$$F_c = \frac{\sum F(i,j) x_{i,j}}{\sum x_{i,j}} \quad (9)$$

Average distance from weight center

$$d = \sqrt{\frac{\sum_{i,j} (x_{i,j} - F_c)}{\sum_{i,j} x_{i,j}}} \quad (10)$$

Maximum distance from weight center

$$d_M = MAX(\sqrt{x_{i,j} - F_c}) \quad (11)$$

2.3.2 Bays, Lakes and the Euler number

Each character posseses a property called the *Euler number*, which we define as the number of *Connected Components* minus the number of *Lakes*.

In a bitmap that consists of only two colors (e.g., black-and-white), we define a *Lake* to be a connected area which does not reach the bitmap borders and is colored with the background color (e.g., white). A related term, the *Bay* is defined to be a connected area of background color that does touch the bitmap borders. The last term is that of a *Connected Component*. For our purposes it will describe any connected area of non-background color (e.g., black). An example for each term is given in Figures 3 a) to d) with the capital character 'A' .

We compute these features in the following fashion: We sequentially scan the given image for white (Bays and Lakes) or black (Connected Components) pixels. Once such a pixel is found, we then execute a coloring algorithm that recursively visits neighbors of the same color and then colors each visited pixel in a third color (e.g., a greytone). If the coloring algorithm is run on a white pixel, we also determine whether we reached the border of the bitmap during execution by setting a flag `reachedBorder`. Furthermore, after each instance we increase a counter variable for the corresponding type, e.g., `numberOfLakes`.

This algorithm visits each pixel at most twice, since the coloring algorithm only visits pixels that have not been colored yet, and therefore requires linear time $O(n)$.

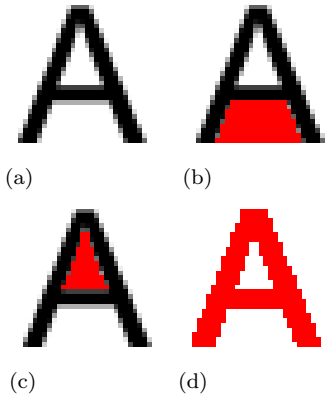


Figure 3: a) Original character, b) 'A' has a single bay (colored red), c) 'A' has one lake, d) 'A' has one Connected Component. Thereby 'A' has Euler number 0

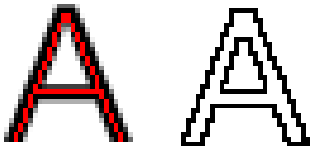


Figure 4: a) In Red: Skeleton of 'A' having one horizontal line, b) Boundary Extraction: Two horizontal lines

2.3.3 Line and Circle Components

Another important property of lating characters are the number of straight lines and the number of circles within a *Connected Component*. For an example, please refer to Figure 4: The character 'A' is 0 vertical lines, 1 horizontal line and 0 circles. Note that we currently do not count diagonal lines. However, we reserve this feature for our Future Work.

We use a very simple method to count straight lines: We first search a pixel that is not of background color. Once it is found, we then visit the next pixel in one direction recursively and thereby determine the length of a possible line. For example, to compute horizontal lines and given that the current pixel is located at (x,y), we visit (x+1,y), (x+2,y) and so on until a different color is encountered. We then count the line

Figure 5: Sampling process: Left: Perform Boundary Extraction, Center: Choose a fixed number of random samples, Right: Compute log-distances between samples

if its length meets a predetermined threshold.

A major issue with this algorithm arises when lines have a thickness of more than one pixel: In that case, for varying sizes of characters, varying number of lines would be found! It is therefore imperative to first do preprocessing. We suggest to use either *Skeletonization* or *Boundary Extraction*. When using *Skeletonization*, each line of a character is shrunk to a thickness of 1, enabling easy parsing. When using *Boundary Extraction* however, two lines are (usually) created. In our implementation, we have used *Boundary extraction*.

We can also count the number of circles in a character. To do this, we find a *Lake* (see Sec. 2.3.2) and determine the following formula:

$$C_0 = \frac{4\pi A_0}{P_0^2} \tag{12}$$

where A is the area of the Lake and P is the *Perimeter*, i.e., the number of neighboring pixels.

2.3.4 Shape Context

We include a method proposed by Belongie et al. [1] as a feature: Shape Descriptors. A Shape Descriptor is a single value that describes how similar two shapes are, where a value is 0 when two shapes are identical, and > 0 otherwise.

The authors construct this value by sampling a picture, computing the distances between samples and

then creating a histogram for each sample. When comparing two shapes, the Chi-Square distance

$$C_S = \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)}$$

is computed for each pair of histograms of shape A (g in the formula above) with histograms of shape B (h). This would result in n^2 values! The suggested solution is to find a bipartite matching between samples of shape A and B and then compute the sum of all matched chi-square distances.

2.4 Decision mechanism

Once every feature has been evaluated on all segments, we need to decide which character a segment resembles. In our system, we use a simple clustering algorithm.

2.4.1 Weight vector

An important point to notice is that above features are not normalized. The *Euler number* of a character may range between -1 and 1, while the total black area of a segment may easily have 300 pixels for letter of font size 32. Normalizing all values would improve results greatly. However, another problem occurs: In reality, the Euler number distinguishes a character far more than the area. We would like to **weigh** the Euler number higher than the area. To achieve this, we must therefore include a weight-vector.

2.4.2 Clustering

Once this is done, the clustering algorithm is very straightforward: For each segment a vector is created, where each dimension of the vector resembles another feature. Each component of a vector is furthermore multiplied by the corresponding entry in the weight-vector.

We then choose the vectors found in our training phase to be the cluster centroids. For all other vectors, namely those found in the live phase, we compute the distances to all cluster centroids and then assign it to the closest centroid. The advantage of this method over, e.g., a decision tree is that we can also support k-next-neighbor request, which is useful for post-processing. As we currently do not do any post-processing, we keep this matter for our Future Work.

3 Experimental Results

4 Conclusion

5 Division of Labor

References

[1] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *In NIPS*, pages 831–837, 2000.