Optical Braille Character Recognition with Support-Vector Machine Classifier

Jie Li, Xiaoguang Yan
College of Electronic and Information Engineering
Changchun University
Changchun, Jilin Province, China
lijie@ccu.edu.cn, ccxgyan@gmail.com

***Abstract*—This paper introduces a system that uses Support-Vector Machine (SVM) to recognize Braille characters from an image. Braille images is captured by the digital cameras, and pre-process the Braille images, segment the Braille images by the image processing technology and then extract the Braille features by the fixed nature of Braille. The SVM system architecture is designed, and the specific image processing approach is introduced in detail in this paper. This method is simple, convenient, and easy to operate, also able to extract dots online in real time. The experiments show that the method is effective and accurate for Braille extraction.**

Keywords-Optical Braille image recognition; Feature Extraction; Support-Vector Machine

I. INTRODUCTION

The second Chinese national survey on the disabled people showed that the number of people with visual disabilities in China reaches 12.33 million, accounting for 14.86% of the total disabled population. Meanwhile, many teachers in special schools for the visually impaired are normal persons; some of them cannot read Braille. In these schools, paper communication between the students and teachers has become a bottleneck that needs to be addressed. Translating Braille into English or other languages is a good way to enable these teachers to understand student's writing, and to teach their students.

A cost-effective solution would be to design an optical Braille recognition system that takes a picture of the Braille document, analyze the picture and translate it into other languages. While Optical Character Recognition (OCR) has been an active area of study in the field of digital image processing and machine vision, Optical Braille Recognition (OBR) has been a niche that is developing relatively slowly.

Previously, reference [1] used the position of light area around of Braille dots to locate the dots, and the grid position of dots in a Braille character to recognize Braille characters. In 2004, reference [2] proposed a robust black/white region–based Braille dot locating method, and used relative positioning of adjacent dots to determine a Braille grid. More recently, reference [3] proposed a phone-camera-based Braille recognition system in which a Java program can process the image taken by phone camera, and translate the Braille dots into Japanese. Reference [5] proposed the use of cellular neural network for Braille recognition, and obtained recognition rate of 87.9%.

While most of the previous literature solves the problem of OBR using digital signal processing techniques, we believe machine learning and pattern recognition can provide a view and solution to the problem of OBR. In this paper, we propose an OBR system that uses Support-Vector Machine (SVM) as classifier to detect Braille dots, and discuss its design, implementation, and test results.

This paper is organized as follows: Section II gives a brief introduction to the Braille system. Section III introduces our SVM-based OBR system; major modules are described in the subsections. Section IV discusses results and points out directions for future improvements.

II. THE BRAILLE SYSTEM

The Braille system, invented by Louis Blair, a blind French man in 1824, is designed to enable the blind people read and write. It uses a character set made up of different combination of raised dots. Each Braille character contains 6 points (3 rows, 2 columns). The six dots can offer $2^6 - 1 = 63$ possible combinations, of which some are omitted because they feel the same. Letters (A~Z) and numbers (0~9) can be coded using these combinations.

For single-sided Braille documents, only verso or recto is present on a page, while for double-sided Braille documents, both verso and recto dots are present on a same page, this poses a challenge to Braille recognition system because the other can cause interference to the recognition system. So designing a double-sided Braille documents recognition system is a more challenging task.

A sample double-sided document image is shown below in Figure 1.

Fig.1 Braille image capture by scanner

III. OPTICAL BRAILLE RECOGNITION USING SUPPORT-VECTOR MACHINE

In this section, we present an optical Braille character recognition (OBR) system. Figure 2 gives the architecture of the optical Braille character recognition system.

978-1-4244-7237-6/$26.00 © 2010 IEEE

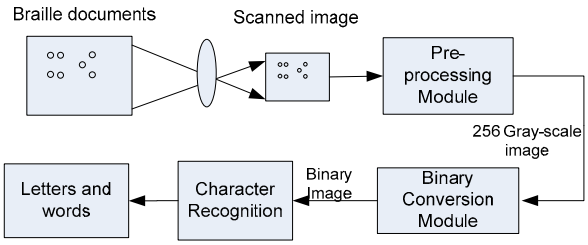

Fig.2 OBR system architecture

First, Braille documents are scanned into images using an ordinary scanner; this creates full color images of the original Braille documents.

Then images are passed through Pre-processing Module. Three tasks are performed by this module: gray-scale conversion, which converts full-color RGB image into grayscale image; brightness adjustment, which removes the possible local dark/bright area caused by non-uniform illumination; and geometric adjustment, in case the document is tilted when scanning.

The pre-processed image is then passed through a Binary Conversion Module. This module uses a window to crop from the image a sub image the size of a Braille dot, and uses SVM to determine whether the cropped image (referred to as sub image hereafter) contains a dot. By sliding the window and combining the output of SVM classifier for each step, the input image is converted into a binary image, with 0s' representing the background, and 1s' representing the presence of Braille dot.

The binary image is then sent to Character Recognition Module, where it is translated into English letters using a simple searching algorithm and a look-up table.

3.1 Braille Image Acquisition

We used a BenQ s6000 scanner at 600 dpi to scan the Braille documents. At resolution of 600 dpi, each Braille dot is about the size of 50x50 pixels. Figure 1 shows part of the scanned Braille image.

3.2 Braille Image Pre-processing

Preprocessing module assumes three tasks: full-color to grayscale image conversion, brightness adjustment, and geometric correction.

3.2.1 Full-color to Grayscale Conversion

The conversion algorithm from full-color image to grayscale image is not unique, and each algorithm has its own applications. We used a simple computation:

$$Y = 0.299R + 0.599G + 0.112B \tag{1}$$

Where Y is the grayscale value; R, G, and B are the red, green and blue component, respectively, of the full-color image.

3.2.2 Image Geometry Correction

The image acquired by scanner may be rotated or translated. Geometric correction aims to eliminate such effects and bring all scanned Braille document image to a common reference point.

We used a simple technique to solve this issue: we manually draw a 15x10mm rectangle using black ink pen at a fixed position on each page of Braille document.

By detecting the 15x10mm rectangle in the scanned image and estimating the position and orientation of the rectangle, we can transform the scanned image to a common reference point. Rotation can be simply calculated using the following equation:

$$x_2 = \cos(\theta)(x_1 - x_0) - \sin(\theta)(y_1 - y_0) + x_0 \tag{2}$$

$$y_2 = \sin(\theta)(x_1 - x_0) + \cos(\theta)(y_1 - y_0) + y_0 \tag{3}$$

Where (x_0, y_0) is the coordinates of the center of rotation (in the input image), (x_1, y_1) and (x_2, y_2) are the original and new coordinate of the pixel respectively, and θ is the angle of rotation with clockwise rotations having positive angles.

It is possible that the new coordinate is not integer. We used a heuristic of calculating the intensity level at integer pixel position based on average of 4 nearest non-integer values.

3.2.3 Image Brightness Adjustment

Local dark or bright areas can occur, due to non-uniform illumination of scanner, or part of the surface of the document being uneven. Left untreated, these areas can cause classification errors in later stage.

We used a simple technique to adjust the brightness: partition the image into small rectangular blocks (we used 200X200 pixels block size). For each block, a mean is calculated, and subtracted from the block. The rationale behind this method is that the mean of a block is an optimal estimation of the intensity of the background, and hence, by subtracting the mean from each block, we subtract the background from the image, leaving only the dots and possible noise data.

It should be noted that using this simple technique may cause intensity discontinuity for pixels lying on the boundary of two adjacent blocks. A better approach would be to use a bilinear interpolation to smooth out the boundary pixels, at the price of increased computational cost.

3.3 Grayscale to Binary Image Conversion Using SVM

The job of Binary Conversion Module is to take in a grayscale image and convert it to binary image, with the 1s' in the image denoting Braille dots, and 0s' the background.

For this purpose, we use a sliding window (the size of 60x70 pixels, slightly larger than a Braille dot the size of 50x50 pixels), to crop a sub image from the whole image. We then pass the grayscale data (for each sub image size of 60x70 pixels, we have 4200 data points) to an SVM, which determines whether this sub image contains a dot or not. Figure 3 illustrates this process.

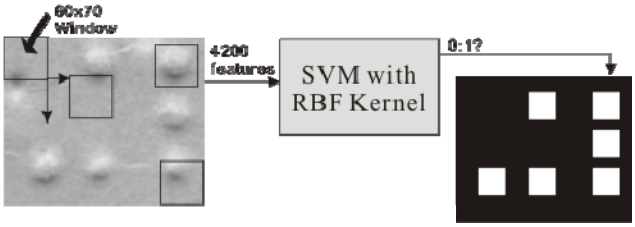

Fig. 3 Grayscale to binary conversion using SVM

Support-Vector Machine (SVM) is a max-margin classifier, first introduced in COLT-92 by Boser, Guyon & Vapnik. Since introduction, it has been used extensively in pattern recognition and machine vision. Reference [4] provides a good tutorial on SVM.

The rationale behind SVM is that, the positive data samples (in our case, sub images containing a dot) and negative samples (sub images contains no dot) can be separated by a boundary (a decision hyperplane). The goal of SVM is to find the best boundary that separates the positive and negative samples, where 'best' means the boundary separates the negative and positive samples with the largest margin or 'distance'.

To find the decision hyperplane, we must first feed a group of positive and negative samples to SVM, which uses convex optimization or other gradient-descent based algorithms to calculate the optimal separating hyperplane from these samples. This process is called SVM training. After the optimal separating hyperplane is obtained, we can then use it to determine whether new data belongs to positive or negative group, a process called SVM testing. Stated mathematically, suppose we have

$$\{x_i, y_i\},\ i = 1,...m,\ \ y_i \in \{-1,1\},\ \ x_i \in \Re^n \qquad (4)$$

where x_i are the training samples (in our case, sub images used in training), each of the training sample is a vector of n real numbers (in our case, each sub image contains 4200 numbers representing the intensity of the pixels in the sub image), m is the number of training samples, and y_i is the group label of x_i (in our case, whether the sub image contains a dot or not).

SVM seeks to separate the data by a hyperplane using a linear combination of sample data:

$$w^T x + b = 0 \qquad (5)$$

Where x is the input vector, w is an adaptive weight vector, and b is a bias, calculated using optimization algorithm. By Lagrangian formulation, the prediction of SVM (whether a new sub image contains a dot or not) is given by

$$y(x) = \mathrm{sgn}\{\sum_{i=1}^{m} y_i \alpha_i \langle x, x_i \rangle + b\} \qquad (6)$$

Where $\langle x, x_i \rangle$ denotes the inner product of x and x_i; $\mathrm{sgn}\{z\}$ denotes the sign of z, α_i is the Lagrangian multiplier, calculated using optimization algorithm. Only some of the Lagrangian multipliers are non-zero, and the corresponding x_i s' whose multiplier are not zero are called support vectors.

In the case when data cannot be separated by a hyperplane (not linearly separable), a kernel trick is often applied by replacing $\langle x, x_i \rangle$ in (6) with a kernel function

$$y(x) = \mathrm{sgn}\{\sum_{i=1}^{m} y_i \alpha_i K(x, x_i) + b\} \qquad (7)$$

This operation maps the non linear separable data to a high or infinite dimension space, in which they can be separated linearly. The frequently used kernel functions are Radial Basis Function (RBF), and polynomial function.

For our purpose, we used 130 positive examples (containing dots), and 160 negative examples (containing background or verso dot) to train an SVM with RBF kernel. The training samples are manually cropped from one of the images and labeled. The training samples are subtracted of the mean of the image from which they are cropped, before being sent to SVM for training.

By sliding the window from left to right and top to bottom at a fixed step (we used step length of 20 pixels) then sending cropped sub image to SVM and obtain classification result, we can translate input gray-scale image into a binary image. Result of this algorithm working on Figure 1 is shown as Figure 4.

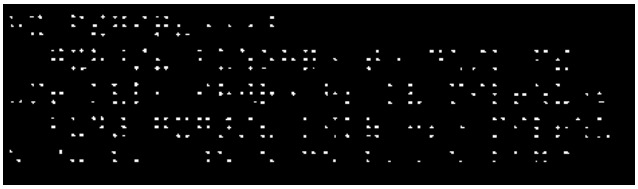

Fig. 4 Binary conversion module output

3.5 Braille Character Recognition

A Braille character is a 3 row, 2 column matrix. The left two rows of Figure 5 show a sample Braille character of English letter 'N'. For the convenience of analysis, we denote the six dots position as A-F, as shown the right 2 rows of Figure 5.

●	●	*A*	*D*
	●	*B*	*E*
●		*C*	*F*

Fig. 5 Braille sample graph

According to standard definition [9], the distance between two Braille dots in the same cell is 2.3-2.5mm, while distance between corresponding dots in adjacent characters is 6.1-7.6mm, and the distance between corresponding dots from one character directly below is 10.0-10.2mm. Given the scanner's resolution in dpi, these numbers can be easily translated into pixels. Then, using pixel value in the original image and the step size of the sliding window used in Binary Conversion Module, we can calculate the relative position of Braille dots in the binary image.

We assume geometric adjustment and classification are done properly, so the dots in the binary image should form a matrix. We determine the coordinate of the first Braille character in the image (x_s, y_s), as the minimum of coordinate of all the dots found in the image:

$$x_s = \min(x_i \mid I(i) = 1) \tag{8}$$

$$y_s = \min(y_i \mid I(i) = 1) \tag{9}$$

Where (x_i, y_i) is the coordinate of the i-th pixel in the image, and $I(i)$ is the intensity of the pixel.

With the first character's coordinate on the page determined, we can use the grid position information to determine the value of each character. For each Braille character, we sequentially check the intensity value for position A-F, if a dot is present in that position, we have a 1; otherwise we have a 0. This method translates the six dot character into a binary string of 6 digits.

Take Figure 5 for an example: starting from reference position A, we get a 1, we then jump to position B, where we have a 0, then to position C, and get a 1…, until we reach position F, where we found a 0, so the Braille character is translated into 101110. A simple look-up table can efficiently translate this binary string into a character in other languages.

If we take into account the possible slight offset to each Braille dots and the step size of sliding window used in Binary Conversion Module, it may make sense to check the pixels adjacent to position A-F as well before making a decision of whether a dot is found.

IV. IMPLEMENTATION AND RESULTS

We used Spider SVM [7], which is a MATLAB implementation of SVM. Radial Basis Function (RBF) kernel function is used with σ= 1000.

After training with 130 positive and 160 negative examples, classification error can be reduced to less than 5%.

Processing time is about 20 minutes per page, using sliding window with step size of 10 pixels. Processing time can be easily reduced by:

1. Using C implementation of SVM, instead of Matlab.
2. Using larger step.
3. Extracting only sub images of specific positions, assuming page layout and grid and dot position is known.

The advantage of using SVM as a classifier is that it seeks to maximize the margin between positive and negative examples. So the classifier is more tolerant to background noise (the presence of recto/verso dots on a double-sided page) than traditional image processing algorithms.

Although in human face recognition or pedestrian detection, SVM is frequently used as a classifier for Haar wavelet coefficients or the output of other feature extraction algorithm [10], we have chosen to pass brightness information directly to SVM for classification without applying feature extraction. Considering the fact that the structure of Braille dots are not as complicated as human face (containing many higher structures), it should be no surprise that our simplified approach can still obtain satisfactory results. However, there should still be room for further improvement of classification performance or processing time by using combination of higher features and other classifiers, such as PCA–SVM [8] or Haar wavelet features – Adaboost [6].

Yet another desirable property provided by the OBR system architecture presented in this paper is that when you find a sub image misclassified, you can extract the image and add it to the group of negative training example, and let SVM generate a new decision boundary, then use the new decision boundary to classify images. This process can be repeated until classification error can be reduced to a satisfactory level.

REFERENCES

[1] Jan Mennens, Luc van Tichelen, Guido Francois, and Jan J. Engelen. Optical Recognition of Braille Writing Using Standard Equipment IEEE TRANSACTIONS ON REHABILITATION ENGINEERING, VOL. 2, NO. 4, DECEMBER 1994, pp 207-211

[2] Antonacopoulos, A. and Bridson, D., A Robust Braille Recognition System, Lecture Notes in Computer Science, Vol.3163 (2004), pp.533-545, Springer-Verlag GmbH

[3] Zhang S and Yoshino K(2007) A braille recognition system by the mobile phone with embedded camera. ICICIC 2007: 1321-1324

[4] Christopher J. C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, v.2 n.2, p.121-167, June 1998

[5] M. Namba and Z. Zhang, Cellular Neural Network for Associative Memory and Its Application to Braille Image Recognition, 2006 International Joint Conference on Neural Networks

[6] Viola Jones, Robust real-time face detection, International Journal of Computer Vision, 2004, 57(2): 137～154

[7] http://www.kyb.tuebingen.mpg.de/bs/people/spider/index.html

[8] Chengcui Zhang; Xin Chen; Wei-bang Chen, A PCA-Based Vehicle Classification Framework, Proceedings. 22nd International Conference on Data Engineering Workshops, 2006. Pp 17-19

[9] http://ww.brailleauthority.org/sizespacingofbraille/sizespacingofbraille.pdf

[10] M. Enzweiler and D. M. Gavrila. "Monocular Pedestrian Detection: Survey and Experiments". IEEE Trans. on Pattern Analysis and Machine Intelligence, 17 Oct. 2008.