Texture-Based Approach for Text Detection in Images Using Support Vector Machines and Continuously Adaptive Mean Shift Algorithm

Kwang In Kim, Keechul Jung, and Jin Hyung Kim

Abstract—The current paper presents a novel texture-based method for detecting texts in images. A support vector machine (SVM) is used to analyze the textural properties of texts. No external texture feature extraction module is used; rather, the intensities of the raw pixels that make up the textural pattern are fed directly to the SVM, which works well even in high-dimensional spaces. Next, text regions are identified by applying a continuously adaptive mean shift algorithm (CAMSHIFT) to the results of the texture analysis. The combination of CAMSHIFT and SVMs produces both robust and efficient text detection, as time-consuming texture analyses for less relevant pixels are restricted, leaving only a small part of the input image to be texture-analyzed.

Index Terms—Text detection, image indexing, texture analysis, support vector machine, CAMSHIFT.

---

1 INTRODUCTION

TEXTS in images include useful information for the automatic annotation, indexing, and structuring of images [31]. Text detection is the process of detecting and locating those regions that contain texts from a given image and is the first step in obtaining textual information. However, text variations related to size, style, orientation, and alignment, as well as low contrast and complex backgrounds make the problem of automatic text detection extremely challenging.

In previous literature, text detection methods are typically either connected component (CC)-based or texture-based. CC-based methods segment an image into a set of CCs, group small CCs to successively larger ones, then classify the final CCs as either text or background by analyzing their geometrical characteristics [1], [2], [3]. While they are efficient at extracting texts, CC-based methods run into difficulties when the text is noisy, degraded, multicolored, textured, or touching itself or other graphical objects, which often occurs in digital images [4]. Texture-based methods are based on the observation that texts in images have distinct textural properties that can be used to discriminate them from the background. In [5], Jain and Zhong use a neural network (NN) to discriminate between text, graphics, and halftones in document images. Zhong et al. [6] analyze local spatial variations in a gray-scale image and locate regions with a high variance as texts. They also combine a texture-based method with a CC-based method. Li et al. [4] extract the wavelet features of small windows in images, then classify them using NNs. Texture-based methods are known to perform well even with noisy, degraded, textured, or complex texts and backgrounds, however, they are usually time consuming as texture classification is inherently computationally dense.

The current paper proposes a novel texture-based text detection method. The following components are crucial to the success of

---

- *K.I. Kim and J.H. Kim are with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Taejon, 305-701, Korea. E-mail: {kimki, jkim}@ai.kaist.ac.kr.*
- *K. Jung is with the Institute of Information, Communication, and Technology, Sungkyunkwan University, Chunchun-dong, Jangan-gu, Suwon, Kyunggi-do 440-746, Korea. E-mail: jungkeechul@naver.com.*

texture-based text detection: 1) construction of texture classifiers that can discriminate between textures associated with different classes (text and nontext) and 2) construction of text region (or *chip* as referred to in [7]) generation module that can operate on the classification results obtained from 1). In the proposed method, support vector machines (SVMs) are used as the texture classifier due to their robustness even in the absence of a rich set of training examples. The previous success of SVMs in texture classification [8] and other related problems [9], [10] also provided further motivation to use SVMs as the classifiers for identifying text regions. In addition, since SVMs work well even in high-dimensional spaces, no external feature extractor is required to reduce the dimensionality of the texture pattern, thereby eliminating the need for a time-consuming feature extraction stage. In fact, SVMs can efficiently extract features within their own architecture using kernel functions (Section 2.1).

As a result of texture classification, a text score image is generated where each pixel represents the possibility of the corresponding pixel in the input image being part of a text region. Then, the text chips within the text score image are identified by applying the continuously adaptive mean shift algorithm (CAMSHIFT), the effectiveness of which has already been demonstrated in face detection [11]. This combination of CAMSHIFT and SVMs produces both robust and efficient text detection through restricting the texture classification of less relevant pixels. Consequently, only a small part of the input image needs to be texture-analyzed to detect the text regions.

2 SYSTEM

The proposed method poses text detection as a texture classification problem where problem-specific knowledge is available prior to classification. Since this knowledge (the number and type of texture classes) is often available in the form of example patterns, the classification can be supervised. An SVM as a trainable classifier is adopted for the texture classification. Specifically, the system uses a small window to scan the input image, classifies the pixel located at the center of the window into *text* or *nontext* by analyzing its textural properties using an SVM, and then applies CAMSHIFT to the results of the texture classification to obtain text chips. To facilitate the detection of various text sizes, a pyramid of images is generated from the original image by gradually changing the resolution at each level. The number of levels was empirically determined to be three. The extracted text chips are hypothesized at each level and then fused to the original scale. Fig. 1 summarizes the text detection process.

The remainder of this section is organized as follows: Section 2.1 reveals the basic theory and some useful properties of SVMs. Section 2.2 then describes the use of an SVM for texture classification, while Section 2.3 outlines the chip generation process using CAMSHIFT. Finally, Section 2.4 discusses the fusion of chips at different scales and positions.

2.1 Support Vector Machines for Texture Classification

SVMs have been recently introduced as a method for pattern classification and nonlinear regression. Given a set of labeled training examples $(\mathbf{x}_i, y_i) \in \mathbf{R}^N \times \{\pm 1\}$, $i = 1, \ldots, l$, an SVM constructs a linear classifier in a feature space F, which is nonlinearly related to the input space via a map $\Phi : \mathbf{R}^N \to F$. The classifier is then identified as a *canonical hyperplane* in F that correctly separates the largest fraction of data points, while maximizing the margin between the nearest examples called *support vectors* (SVs), and is represented in the input space as:
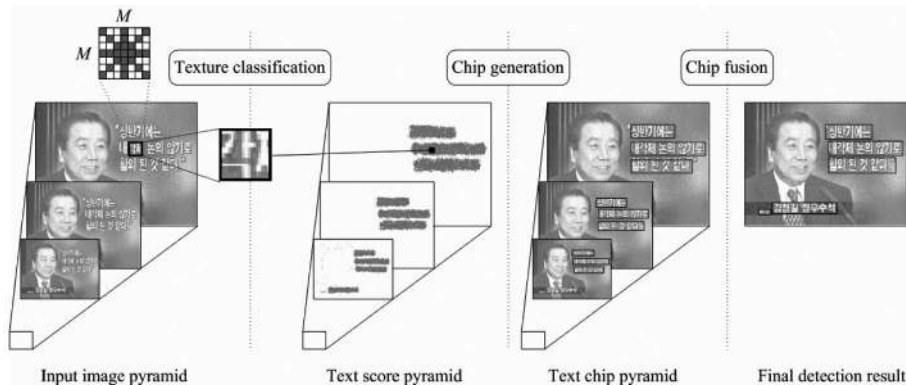
Fig. 1. Top-level process of text detection system.

$$f(\mathbf{x}) = \sum_{i=1}^{l*} y_i \alpha_i \Phi(\mathbf{x}_i^*) \cdot \Phi(\mathbf{x}) + b,$$

where $\{\mathbf{x}_i^*\}_{i=1}^{l*}$ are the SVs. The coefficients α_i and b are determined by solving a large-scale quadratic programming problem [12].

The appeal of SVMs lies in their strong connection to the underlying statistical learning theory. According to the structural risk minimization principle [12], a function that can classify training data accurately and which belongs to a set of functions with the lowest capacity (particularly in the *VC-dimension*) will generalize best, regardless of the dimensionality of the input space. In the case of a canonical hyperplane, minimizing the VC-dimension corresponds to maximizing the margin. As a result, for many applications, SVMs have been shown to provide a better generalization performance than traditional techniques, including NNs and radial basis function networks [13], [14].

For computational efficiency, the mapping Φ is often implicitly performed using a kernel function k defined as: $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$. Then, by selecting the proper kernels, k, various mappings (or feature extractions), Φ, can be indirectly induced [16]. One feature extraction is achieved by taking the p-order correlations between the entries x_i of an input vector $\mathbf{x}$. If $\mathbf{x}$ represents a texture pattern of just pixel values, this amounts to mapping the input space into the space of pth order products (monomials) of the input pixels. It should be noted that these features cannot be extracted by simply computing all the correlations since the required computation is prohibitive when p is not small ($p > 2$): For N-dimensional input patterns, the dimensionality of the feature space F is $(N + p - 1)!/p!(N - 1)!$.

However, the introduction of a polynomial kernel facilitates work in this feature space as the polynomial kernel with degree p ($k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$) corresponds to the dot product of the feature vectors extracted by the monomial feature extractor Φ_p [16]:

$$(\Phi_p(\mathbf{x}) \cdot \Phi_p(\mathbf{y})) = \sum_{i_1,\ldots,i_p=1}^{N} x_{i_1} \ldots x_{i_p} y_{i_1} \ldots y_{i_p} = \left(\sum_{i=1}^{N} x_i y_i\right)^p = (\mathbf{x} \cdot \mathbf{y})^p.$$

2.2 Data Representation and Classification

One of the simplest ways to characterize the variability in a texture pattern is by noting the gray level values of the raw pixels. This set of gray values then becomes the feature set on which the classification is based. An important advantage to this approach is the speed with which the patterns can be processed as the features do not need to be calculated. Furthermore, this approach releases the system developer from the laborious feature design task. Yet, the main disadvantage is the large size of the feature vector. Accordingly, a classifier is required to generalize the

texture patterns in a high-dimensional space. In this case, SVMs are a proper candidate as they can generalize well even in high-dimensional spaces. Furthermore, SVMs incorporate feature extractors and can use their nonlinear mapped input patterns as features for classification (see previous subsection). Therefore, it would appear to be more advantageous to allow an SVM to extract features directly from the pixels rather than forcing it to base the features on a user-defined feature set.

The input to the SVM texture classifier comes from an $M \times M$ window in the input image (Fig. 1). However, instead of using all the pixels in the window, only pixels corresponding to a configuration for autoregressive features (shaded pixels within the window in Fig. 1) are fed to the SVM. This reduces the size of the feature vector (from M^2 to $4M - 3$) and results in an improved generalization performance and classification speed. The sign of the SVM output then represents the class of the central pixel in the input window. For training, $+1$ was assigned to the text class and -1 to the nontext class. As such, if the SVM output for an input pattern is positive, it is classified as text.

When using a learning-from-examples approach, it is desirable to make the training set as large as possible to attain a comprehensive sampling of the input space. However, when considering real-world limitations, the size has to be moderate. Therefore, the problem is how to build a comprehensive, yet tractable database. For text patterns, a collection was made of all the text-containing images available. However, collecting nontext patterns was more difficult as practically any image can serve as a valid training example. From among these patterns, a "representative" set of nontext patterns was selected. A *bootstrap* method recommended by Sung and Poggio [17] was adopted for this. The idea is that some of the nontext training patterns are collected during training rather than before training: A partially trained SVM is applied to images that do not contain texts, then patterns with a positive output are added to the training set as nontext patterns. This process iterates until no more patterns are added to the training set.

However, a training set constructed by a bootstrap, which is moderate (about 100,000 in preliminary experiments) for training conventional learning systems, such as NNs, is often too large for training SVMs: When the number of training patterns is l, the training of an SVM requires an $O(l^2)$-sized memory, which then grows prohibitively in proportion to l (when $l > 10,000$ the memory grows to more than hundred gigabytes). Although several methods have already been developed to reduce the time and memory complexity for training a large-scale SVM [18], they barely provide practical facilities for training sets larger than 100,000.

As such, the current approach used a moderately sized subset of the training set rather than the whole training set, where a set of "important" or "relevant" patterns was selected first to utilize the

1. Create initial training set that includes complete set of text patterns and partial set of non-text patterns.
2. Train SVM using training patterns.
3. Replace training set with complete text pattern set plus non-text SV set.
4. Exhaustively scan image that contains no text using SVM.
5. Add 10% random selection of patterns where corresponding SVM decisions are text class to training set.
   Iterate steps 2-5 until no more images are available
6. Train SVM on finalized training set.

Fig. 2. SVM training process for texture classification.

Fig. 3. Examples of training images: (a) text images, (b) nontext images, (c) images for bootstrap, and (d) classification results of (c).

information contained in the original training set. While, generally, this is not an easy task, it is well-known that SVs are the only informative patterns for constructing an SVM [12]. Furthermore, the number of SVs compared to all the training patterns is usually extremely small. Therefore, only SVs were collected as the training set for the construction of the classifier. However, the problem here is that SVs are only obtained after the training of the SVM (on the whole set) is finished. Therefore, as an alternative, the SVs were iteratively gathered during the bootstrap: At each iteration t, nontext training data $\tau(t)$ for training an SVM $\sigma(t)$ was obtained by gathering the nontext SVs identified during training $\sigma(t-1)$ along with newly found negative patterns $\mu(t-1)$ at $t-1$, instead of $\tau(t-1)$ plus $\mu(t-1)$. Although the collections of SVs obtained by training SVMs using several partial sets are not necessarily exactly the same as the SV set obtained from an SVM trained using the entire set, they still have a similar effect on the construction of the classifier because they are very informative patterns, despite being partial. Accordingly, if certain patterns have to be selected, selecting sets of SVs may be a proper choice. Similar ideas have already been successfully applied in the *virtual SV* method [30] and a training set decomposition technique referred to as *chunking* [33]. After finishing the bootstrap, a new SVM is trained using the finalized training set, which is then used for classification. The training process is summarized in Fig. 2.

Fig. 3 shows some of the images used in training: Figs. 3a and 3b are examples of text and nontext images, respectively, from which the initial training patterns were collected. Fig. 3c shows some of the nontext training images used in the bootstrap and Fig. 3d shows examples of misclassifications of Fig. 3c (marked with black pixels) where nontext training patterns were tagged. Fig. 4 is an example of texture classification performed by a trained SVM.

2.3 Chip Generation

As shown in Fig. 4b, the classification results obtained from SVMs are not directly suitable as text detection results for future processing, such as OCR, because of the presence of noise and spurious regions. The classification stage only transforms an input image into a representation from which the desired regions (or chips) can be easily extracted.

For chip generation, a technique is borrowed from well-developed face detection methodologies. CAMSHIFT was originally developed by Bradski [11] to detect and track faces in a video stream. As a modification of the mean shift algorithm that climbs the gradient of a probability distribution to find the dominant mode [22], CAMSHIFT locates faces by seeking the modes of the *flesh probability distribution*.[1] The proposed strategy is simply replacing this flesh probability distribution with a text score distribution $\{y_{i,j}\}_{i,j=1,\ldots,IW,IH}$ (IW: image width, IH: image height), where the entry $y_{i,j}$ represents the probability of a pixel $x_{i,j}$ in the original image $\{x_{i,j}\}_{i,j=1,\ldots,IW,IH}$ being part of a text region, and operating CAMSHIFT on $\{y_{i,j}\}_{i,j=1,\ldots,IW,IH}$. Then, each $y_{i,j}$ is obtained by applying an SVM to the input $x_{i,j}$. Although the SVM output is not a probability (it is not even bounded), when it is scaled into an interval $[0, 1]$ using a sigmoidal activation function, the detection results with CAMSHIFT are acceptable. Henceforth, for convenience, these scaled SVM outputs for the pixels within a selected window W will be referred to as the "probability distribution within W."

However, for the purpose of text detection, mode seeking (or the center position of the window) is insufficient. There are many variations to text appearances, for example, font, perspective, orientation, etc., therefore, the shape of a text region is variable. As such, not only the location but also the shape needs to be calculated to be useful as a text detection result. Since it is not easy to estimate every possible shape variation, many simplifications have been made in previous text detection systems. For example, [1] and [6] assume that all text regions are rectangular, while [2], [3], and [4] further assume that texts are horizontally aligned. This is reasonable

1. Actually, it is not a probability distribution because its entries do not total 1. However, this is not generally a problem with the objective of peak (mode) detection.
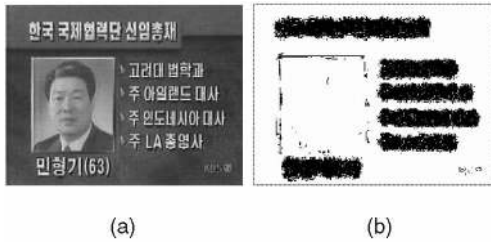
Fig. 4. Example of texture classification: (a) input image (640×480) and (b) classification result.

as most texts appearing in media are mainly in a horizontal direction [2]. Accordingly, the proposed method detects texts as horizontal rectangular shapes: CAMSHIFT initially sets a rectangular search window. During the search process, the location and size of the window are continuously adapted to fit the contents of window (or probability distribution within the window). Then, the finalized search window is output as the detection result. The parameters controlled in each iteration t are the position $x(t)$, $y(t)$, width $w(t)$, and height $h(t)$ of the search window.

x and y can be simply computed using geometrical moments:[2]

$$x = M_{10}/M_{00} \text{ and } y = M_{01}/M_{00}, \tag{1}$$

where M_{ab} is the $(a+b)$th moment as defined by

$$M_{ab}(W) = \sum_{i,j \in W} i^a j^b y_{i,j}.$$

w and h are estimated by considering the two eigenvectors and their corresponding eigenvalues in the correlation matrix $\mathbf{R}$ of probability distribution within the window.[3] These variables can be calculated using up to second order moments as follows [11], [19], [32]:

$$w = 2\sqrt{\left((a+c) + \sqrt{b^2 + (a-c)^2}\right)/2},$$
$$h = 2\sqrt{\left((a+c) - \sqrt{b^2 + (a-c)^2}\right)/2}, \tag{2}$$

where the intermediate variables a, b, and c are

$$a = M_{20}/M_{00} - x^2, \ b = 2(M_{11}/M_{00} - xy), \text{ and } c = M_{02}/M_{00} - y^2.$$

Note that the moments involve sums of all the pixels and, thus, are robust to small element changes. Consequently, a robust estimation of the parameters is possible even with the existence of noise (misclassifications) in the window. Nonetheless, the use of (2) to estimate the window size is not directly suitable for the purpose of text detection. When the whole text region is completely contained within the search window, (2) produces exactly what is needed. However, when the actual text region is larger than the current (iteration t) window, the window size needs to be increased beyond the estimation of (2) so as to explore a potentially larger text area in the next iteration ($t + 1$). In this case, h and w are set slightly larger than that estimated by (2):

2. For notational convenience, the temporal parameter t is dropped when this does not cause confusion.

3. Since the input space is 2D, $\mathbf{R}$ is 2×2 along with the existence of two (normal) eigenvectors: The first gives the direction of the maximal scatter, while the second gives the related perpendicular (assuming that the eigenvectors are sorted in descending order of their eigenvalue size). Their corresponding eigenvalues then indicate the degrees of scatter along the direction of the corresponding eigenvectors [14].

1. Set up initial locations and sizes of search windows Ws in image. For each W, repeat Steps 2 to 4 until terminal condition is satisfied.
2. Generate text probability distribution within W using SVM.
3. Estimate parameters (location and size) of W using eqns. (1) and (3).
4. Modify W according to estimated parameters.
5. Re-estimate sizes of Ws using eqn. (2).
6. Output bounding boxes of Ws.

Fig. 5. CAMSHIFT for text detection.

$$w = 1.2*2\sqrt{\left((a+c) + \sqrt{b^2 + (a-c)^2}\right)/2},$$
$$h = 1.2*2\sqrt{\left((a+c) - \sqrt{b^2 + (a-c)^2}\right)/2}. \tag{3}$$

The new estimate enables the window to grow as long as the major content of the window is text pixels. When the iteration terminates, the final window size is then reestimated using (2).

The terminal condition for iteration is that, for each parameter, the difference between two parameters $|x(t+1)\text{-}x(t)|$, $|y(t+1)\text{-}y(t)|$, $|w(t+1)\text{-}w(t)|$, and $|h(t+1)\text{-}h(t)|$ in two consecutive iterations $t + 1$ and t is less than the predefined thresholds T_x, T_y, T_w, and T_h, respectively.

As there may be more than one text region in an image, multiple windows can be run with different initial positions and sizes. This also reduces the possibility of being stuck to local optima. Fig. 5 summarizes the operation of CAMSHIFT for text detection. It should be noted that, in the case of overlapping windows, the classification results are cached so that the classification of a particular pixel is only performed once for an entire image.

Besides robustness, an important advantage of CAMSHIFT is that it does not require all the pixels in the input image to be texture-classified: In the CAMSHIFT iteration, the parameters of the windows are computed using only the text probabilities of the pixels located within the windows. As such, instead of calculating the text probability distribution over the whole image, the distribution calculation can be restricted to a smaller image region surrounding the window of interest. Then, the texture-classifier is embedded in CAMSHIFT and only activated for those pixels located within the search windows. This results in significant computational savings when text regions do not dominate the image. Fig. 6 shows an example of text detection: A considerable number of pixels (black pixels in Fig. 6c occupying 79.2 percent of all the pixels in the image) are excluded from the texture classification for the given image.

2.4   Chip Fusion

The proposed method can make certain overlapping detections when the overlapping chips occur at different scales (between pyramid levels) or positions (within a pyramid level). In the case of overlapping chips, an examination is made as to whether they are originally a single text or multiple texts. This is performed by checking the degree of overlap between the two chips, which is measured using the size of the overlapdivided by the size of each chip.

Supposing that D_α and D_β are the areas covered by two text chips α and β, then the degree of overlap between α and β is defined as

$$\Lambda(\alpha, \beta) = \max\big(\text{size}\,(D_\alpha \cap D_\beta)/\text{size}\,(D_\alpha), \text{size}\,(D_\alpha \cap D_\beta)/\text{size}\,(D_\beta)\big),$$

where size(λ) counts the number of pixels within λ. Then, α and β are determined to be a single text if $T_o \Lambda(\alpha, \beta)$, multiple texts otherwise, where T_o is the threshold set at 0.8. Then, in the fusion process, every pair of overlapping chips is checked and those pairs
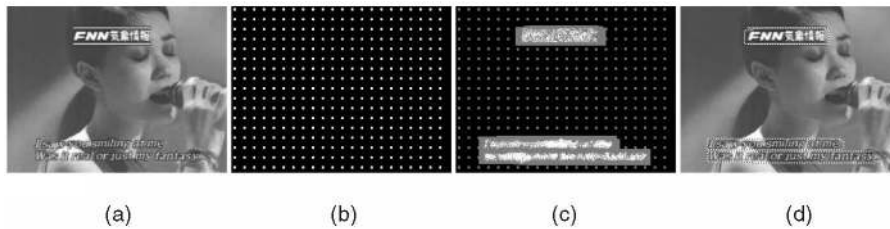
Fig. 6. Example of text detection using CAMSHIFT: (a) input image (540×400), (b) initial window configuration for CAMSHIFT iteration (5×5-sized windows located at regular intervals of $(25, 25)$), (c) texture classified region marked as white and gray levels (white: text region, gray: nontext region), and (d) final detection result.

identified as a single text are merged. After chip fusion, any small chips are eliminated as they are usually false detections. Fig. 7 shows an example of chip fusion. For better visibility, only those chips corresponding to text areas are shown for each level. Although none of the text detection results for a single pyramid level detected all the texts, the chip fusion results correctly detected all the texts.

3 EXPERIMENTAL RESULTS

The system was tested on three sets of images. Database I contained 400 images composed of Web images, video frames, scanned documents, etc. The images included examples of homogeneous, multicolored, textured, and imaged backgrounds where the text appearance varied in terms of color, texture, size, and style. The resolution of these images ranged from 182×256 to $1,024 \times 2,096$ and the text strings size varied from about 5×9 to 162×73. The image formats were BMP, JPEG, GIF, TIFF, etc. Database II was obtained from the Web site of the Language And Media Processing (LAMP) Laboratory, media group at the University of Maryland, College Park [34]. It contained 75 text-containing video frames. All the images were in JPEG format. Database III was comprised of three MPEG video streams obtained from the Web site of the movie content analysis (MoCA) project [25] and contained examples of texts moving on a complex background. This database was also used by Lienhart and Stuber [1].

One hundred randomly selected images from database I were used for collecting the initial samples for training the SVMs, while the others were used for testing. Nontext training examples for bootstrapping the SVMs were also collected from 86 images that contained no text and were distinct from the text image databases. For each detected text region, the proposed system drew dotted lines encompassing that region in the input image. It should be noted that the finalized text regions were not always rectangular in shape because multiple text chips were sometimes merged into a single detection.

Fig. 7. Example of chip fusion: (a) intermediate detection results for each pyramid level and (b) final detection result.

3.1 Tuning of Parameters

Prior to evaluating the overall performance of the proposed system using the database, various parameters were tuned. These included parameters for the texture classifier, i.e., input window size M and polynomial degree p, plus the CAMSHIFT parameters, i.e., initial window configuration and termination condition.

3.1.1 Evaluation of SVM Parameters

To investigate the effect of the parameter settings on the performance of the texture classifier, two experiments were run while varying one parameter at a time. The performance criterion was the classification accuracy of the SVMs for text and nontext patterns rather than the overall text detection results. For this purpose, 100 training images were divided into two different classes of 70 training images and 30 validation images from which training patterns and validation patterns were collected, respectively. The SVMs were then trained using the training patterns and tested using the validation patterns. Table 1 shows the error rate as p was varied from 1 to 6 when $M = 15$. The best results were obtained when using degree-4 and -6 polynomial kernels. However, no obvious optimum was identified. Higher degrees of p afforded a lower error rate, while a saturation point was reached when $p \geq 4$.

Although the use of small input windows can reduce the processing time, if the window size is too small, the classification errors will increase because the classifier is unable to fully utilize the textural information. Conversely, if the window is too large, this can result in unstable classifications at the texture boundary (edge of text area). Accordingly, selecting the appropriate input window size is a trade off between classification accuracy and processing time. As such, experiments were performed with different input window sizes of 3×3, 5×5, 7×7, 9×9, 11×11, 13×13, 15×15, and 19×19. The results are summarized in Table 2. While a smaller error rate tendency was observed with the larger input window sizes, except for a 19×19 window, the best results were obtained with a 13×13 window.

3.1.2 Determination of CAMSHIFT Parameters

The initial locations and sizes of the search windows are dependent on the application. An effective selection for the initial search windows should be relatively dense and large enough not to miss texts located between the windows, yet also moderately sparse and small enough to ensure fast processing. The current study found that 5×5-sized windows located at a regular interval of $(25, 25)$ were sufficient to detect the text areas.

TABLE 1
Classification Error Rates Using Varying Polynomial Degree

Polynomial degree p	1	2	3	4	5	6
Error rate (%)	10.7	5.2	5.5	5.1	5.3	5.1

TABLE 2
Error Rates with Different Window Sizes

Window Size	5×5	7×7	9×9	11×11	13×13	15×15	19×19
Error rates (%)	19.2	17.9	9.6	6.4	5.1	5.3	8.7

Variations in the threshold values T_x, T_y, T_w, and T_h (for termination condition of CAMSHIFT) did not significantly affect the detection results except when they were so large that the search process converged prematurely. Therefore, based on various experiments with the validation images, the threshold values were determined as $T_x = T_y = 3$ pixels and $T_w = T_h = 2$ pixels. Although these parameters were not carefully tuned, the results were acceptable with both the validation and test images, as described in the next subsection.

3.2 Text Detection Performance Results

For the text detection experiments, the texture classifier parameters were set according to one of the optimal results ($p = 4$, $M = 13$) identified in the previous section. Based on these parameters, the classifier was then retrained with all the training data, including the validation patterns. The systems were coded in C++ and run on a Pentium3 CPU with an 800 MHz clock speed. The time spent processing an image depended on the image size, and the number and size of the texts in the image. Most of the time was spent in the texture classification stage. For the 640×480-sized images, an average of 3.2 seconds was taken to classify all the pixels in the image. However, when the classification was restricted to just the pixels located within the search windows involved in CAMSHIFT, the entire detection process only took an average of 0.45 seconds.

Fig. 8 shows examples of input images from databases I and II along with the computed text bounding boxes: While considerable variations in contrast, intensity, and texture, even within a text string were observed, almost all the text regions were correctly located. Some text lines (Figs. 8d, 8e, and 8l) were missed because of their small size or low contrast. However, such texts are usually not important as regards the original purpose of text detection, i.e., image indexing or compression. The background of Fig. 8j was

Fig. 8. Text detection examples: (a)-(c) Web images, (d)-(h), (m)-(o) video frames obtained from various video sources covering the news, games, movies, and LAMP lab [34], and (i)-(l) scanned documents (scanned at 150 dpi).

Fig. 9. Examples of text detection from two video streams (sequence from left to right): (a) input video streams and (b) detected text regions.

filled with handwritten texts that were degraded by heavy shadow and blur. As a result, only portions (with more moderate shadow) were detected as texts. However, it is not easy even for humans to locate all the texts in this image. There were some spurious detections in Figs. 8h, 8i, 8k, and 8m. In particular, in the case of Figs. 8h, 8i, and 8m, the depicted structure and intricate designs were detected as texts because they exhibited a similar contrast and texture to text, thereby making them difficult to distinguish from text when only analyzing localized characteristics. Consequently, a more complicated system is needed to deal with such designs. It should also be noted that the texts detected in Figs. 8i, 8j, and 8l were either textured or multicolored. Such texts cannot be easily detected with CC-based methods.

Fig. 9 shows text detection examples from two video streams in database III. To facilitate a comparison with other methods, the detection results were converted into masked images where a black mask was used to represent the background region. Despite a complex moving background and variations in the text regions related to size and position, nearly all the texts were detected with very few false detections. When these results were compared with the performance of various other methods [1], [26], the proposed system was found to produce more precise boundaries and fewer false detections.

To quantitatively evaluate the performance, evaluation criteria were established for the detections produced automatically by the system. Although several previous methods [2], [27] provide results on detection accuracy, they are either restricted to specific image classes or cannot be extended to an automatic evaluation of large collections of diverse images. Accordingly, a new technique was proposed for the automatic evaluation of text detection in images. First, ground truths GTs are created by manually constructing bounding boxes around each text string in the image. Then, the outputs produced by the system As are compared with the GTs based on checking the similarity, defined as the size of the

overlap divided by the size of the union. Formally, the similarity between two detections α and β is defined as

$$\Gamma(\alpha, \beta) = \text{size}(D_\alpha \cap D_\beta)/\text{size}(D_\alpha \cup D_\beta). \qquad (5)$$

Actually, this is the Tanimoto similarity [28] between two detections when they are represented as $IW \times IH$ vectors whose elements are 1 if the corresponding pixels are contained within the text areas and 0 otherwise. As such, the output of the comparison is a binary value of $\{correct, incorrect\}$, which is determined as

$$\text{correct, if } T < \Gamma(GT, A), \text{ incorrect, otherwise,} \qquad (6)$$

where T is the threshold, which is set at 0.8.

The incorrect detections are further classified into $\{false\ detection,\ miss\}$ as defined by:

$$\text{false detection, if size}(D_A)\text{-size}(D_{GT}) > 0, \text{ miss, otherwise.}$$

There are situations when a text region can be decomposed into two or more detections $A^1, \ldots, A^S$, as exemplified in Fig. 10. In this case, the detections are simply merged into a single large detection A' ($D'_A = \bigcup_{j=1,\cdots,S} D^j_A$) containing all the A^js, then the GT is compared with A'. However, in this case, the threshold value T is lowered to 0.75 to disregard the spaces between the lines and words. Conversely, when several ground truths $GT^1, \ldots, GT^P$ correspond to a detection A, they are merged into a single large ground truth GT' ($D'_{GT} = \bigcup_{i=1,\cdots,P} D^i_{GT}$) and compared with A. To enable the automatic counting of correct and incorrect detections from this configuration, a simple merging procedure is presented as summarized in Fig. 11. The basic strategy is to merge as many detections as possible, while the resulting merged detection satisfies the detection criterion (6) or increases the similarity score (5). Although the resulting merging is not guaranteed to be optimal, the results were found to be acceptable in preliminary experiments.

Two metrics are used to summarize the detection results, as defined by:

$$\text{miss rate}(\%) = \frac{\text{\# of misses}}{\text{\# of texts}} \times 100 \text{ and}$$

$$\text{false detection rate }(\%) = \frac{\text{\# of false detections}}{\text{\# of texts}} \times 100.$$

The performance of the proposed system with databases I, II, and III was a 2.4 percent, 4.3 percent, and 1.2 percent miss rate and 71.5 percent, 53.1 percent, and 22.4 percent false detection rate,
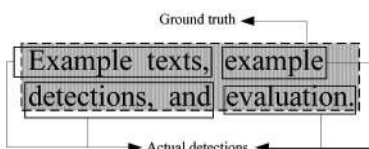

Fig. 10. Example of decomposed detection.

Do{

    1. For each ground truth GT^i, obtain a set of candidate detections $\mathbf{A} = \{A^j | \Gamma(GT^i, A^j) > 0, j$ goes for all detections$\}$.

    $t = 0$.

    $A'(0) = \arg\max_{A^k \in \mathbf{A}(t)} \left\{ \Gamma(GT^i, A^k) \right\}$

    Do{

        Create a set of merging hypotheses $\mathbf{A}'(t) = \{A^k = A^{k(a,b)} | D_A^{k(a,b)} = D_A^a \cup D_A^b,$ for all $A^a, A^b \in \mathbf{A}\}$.

        $A'(t) = \arg\max_{A^k \in \mathbf{A}'(t)} \left\{ \Gamma(GT^i, A^k) \right\}$.

        Add $A'(t)$ to $\mathbf{A}$ and remove corresponding A^a, A^b from $\mathbf{A}$.

        $t = t+1$.

    } While $((\Gamma(GT^i, A'(t-1)) \leq \Gamma(GT^i, A'(t))$ or $\Gamma(GT^i, A'(t)) > T))$.

    Store $\mathbf{A}'(t-1)$ as a new set of candidate detections.

    2. Same as step (1), except the roles of GT and A are exchanged.

} Until (no more merging occurs)

Fig. 11. Detection merging process.

TABLE 3
Performances of Various Systems

	Miss rate	False detection rate	Avg. proc. time per image (sec.)
A: SVM+CAMSHIFT	2.4 %	71.5 %	0.72
B: SVM+histogram analysis	3.0 %	63.7 %	5.36
C: NN+histogram analysis	8.1 %	112.4 %	3.63
D: NN+CAMSHIFT	3.9 %	92.5 %	0.47
E: CC-based	9.1 %	75.4 %	0.05
F: wavelet+NN+CC analysis	4.8 %	101.8 %	4.57

respectively. The false detection rate can go above 100 percent because the number of nontext detections can be higher than the number of text detection. To gain a better understanding of the relevance of the results obtained using the proposed method, benchmark comparisons with other methods were carried out. A set of experiments was performed using a CC-based method and two texture-based methods with database I. The CC-based method [3] quantized the color spaces into a few prototypes and performed a CC analysis on the resulting gray-scale images. Then, the extracted CCs were verified according to their shapes. The first texture-based method [20] adopted an NN to analyze the textural properties of the texts and performed a histogram analysis to generate text chips. The second texture-based method extracted up to third order central moments as features from wavelet subband images, selected some features based on the Bayes error rate, and then classified them using an NN, following the classification scheme of [4].[4] Finally, the text chips were identified by a CC analysis.

Table 3 summarizes the performances of the different systems: A was the proposed method, B used an SVM for texture classification and histogram analysis for chip generation, D used an NN adopted from [20] for texture classification and CAMSHIFT for chip generation, and C, E, and F were the CC-based method and texture-based methods described above, respectively.

A and B exhibited similar performances and were far better than the other four methods, while A was much faster than B. C showed the highest false detection rate. However, when an NN was combined with CAMSHIFT (D), the performance was improved. E produced the highest miss rate, which mainly stemmed from a poor detection of noisy and textured texts that often occur in video images. However, preliminary experiments showed that, when the texts were rather clean and single-colored (such as texts on journal covers, Web logos, etc), E performed slightly better and was much faster than A. F exhibited a medium miss rate and rather high false detection rate with the second longest processing time.

---

4. Neither the training procedure nor the design of the NN classifier was carefully optimized, therefore, the results should only be taken as baseline comparisons to facilitate the evaluation of the proposed methods.

4 CONCLUSIONS

A texture-based method for detecting texts in images was presented. The system analyzes the textural properties of texts in images using an SVM and locates the text regions by operating CAMSHIFT on the texture classification results. The proposed method can facilitate fast text detection, even though it does not assume the type of media or the color and textural properties of texts and is relatively insensitive to image resolution. It also works well in extracting texts from complex and textured backgrounds and was found to produce a better performance than some other techniques. However, the proposed texture classifier did encounter problems classifying very small text or text with a low contrast.

REFERENCES

[1] R. Lienhart and F. Stuber, "Automatic Text Recognition in Digital Videos," *Proc. ACM Multimedia,* pp. 11-20, 1996.

[2] A.K. Jain and B. Yu, "Automatic Text Location in Images and Video Frames," *Pattern Recognition,* vol. 31, no. 12, pp. 2055-2076, 1998.

[3] E.Y. Kim, K. Jung, K.Y. Jeong, and H.J. Kim, "Automatic Text Region Extraction Using Cluster-Based Templates," *Proc. Int'l Conf. Advance in Pattern Recognition and Digital Techniques,* pp. 412-421, 2000.

[4] H. Li, D. Doermann, and O. Kia, "Automatic Text Detection and Tracking in Digital Video," *IEEE Trans. Image Processing,* vol. 9, no. 1, pp. 147-156, 2000.

[5] A.K. Jain and Y. Zhong, "Page Segmentation Using Texture Analysis," *Pattern Recognition,* vol. 29, no. 5, pp. 743-770, 1996.

[6] Y. Zhong, K. Karu, and A.K. Jain, "Locating Text in Complex Color Images," *Pattern Recognition,* vol. 28, no. 10, pp. 1523-1535, 1995.

[7] V. Wu, R. Manmatha, and E.M. Riseman, "TextFinder: An Automatic System to Detect and Recognize Text in Images," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 11, pp. 1224-1229, Nov. 1999.

[8] K.I. Kim, K. Jung, S.H. Park, and H.J. Kim, "Supervised Texture Segmentation Using Support Vector Machines," *IEE Electronics Letters,* vol. 35, no. 22, pp. 1935-1936, 1999.

[9] E. Osuna, R. Freund, and F. Girosi, "Training Support Vector Machines: An Application to Face Detection," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 130-136, 1997.

[10] V.P. Kumar and T. Poggio, "Learning-Based Approach to Real Time Tracking and Analysis of Faces," *Proc. IEEE Int'l Conf. Automatic Face and Gesture Recognition,* pp. 96-101, 2000.

[11] G.R. Bradski, "Real Time Face and Object Tracking as a Component of a Perceptual User Interface," *Proc. IEEE Workshop Applications of Computer Vision,* pp. 214-219, 1998.

[12] V. Vapnik, *The Nature of Statistical Learning Theory.* New York: Springer-Verlag, 1995.

[13] B. Schölkopf, K. Sung, C.J.C. Burges, F. Girosi, P. Niyogi, T. Pogio, and V. Vapnik, "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers," *IEEE Trans. Signal Processing,* vol. 45, no. 11, pp. 2758-2765, 1997.

[14] S. Haykin, *Neural Network—A Comprehensive Foundation.* second ed. Prentice Hall, 1999.

[15] T.M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Trans. Electronic Computers,* vol. 14, pp. 326-334, 1965.

[16] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery,* vol. 2, no. 2, pp. 1-47, 1998.

[17] K.K. Sung and T. Poggio, "Example-Based Learning for View-Based Human Face Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 1, pp. 39-51, Jan. 1998.

[18] T. Joachims, "Making Large-Scale SVM Learning Practical," *Advances in Kernel Methods—Support Vector Learning,* B. Schölkopf, C. Burges, and A. Smola, eds., pp. 169-184, Cambridge, Mass.: MIT Press, 1999.

[19] H. Li and D. Doermann, "A Video Text Detection System Based on Automated Training," *Proc. Int'l Conf. Pattern Recognition,* pp. 223-226, 2000.

[20] K.Y. Jeong, K. Jung, E.Y. Kim, and H.J. Kim, "Neural Network-Based Text Location for News Video Indexing," *Proc. Int'l Conf. Image Processing,* pp. 319-323, 1999.

[21] E.Y. Kim, K.I. Kim, K. Jung, and H.J. Kim, "A Video Indexing System Using Character Recognition," *Proc. IEEE Int'l Conf. Consumer Electronics,* pp. 358-359, 2000.

[22] Y. Cheng, "Mean Shift, Mode Seeking, and Clustering," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 8, pp. 790-799, Aug. 1995.

[23] D. Comaniciu, V. Ramesh, and P. Meer, "Real-Time Tracking of Non-Rigid Objects Using Mean Shift," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 142-149, 2000.

[24] D. Comaniciu and P. Meer, "Mean Shift Analysis and Applications," *Proc. IEEE Int'l Conf. Computer Vision,* pp. 1197-1203, 1999.

[25] Praktische Informatik IV, MOCA Project, http://www.informatik.uni-mannheim.de/informatik/pi4/projects/MoCA, 2003.

[26] Y. Zhong, H. Zhang, and A.K. Jain, "Automatic Caption Localization in Compressed Video," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 4, pp. 385-392, Apr. 2000.

[27] M.A. Smith and T. Kanade, "Video Skimming and Characterization through the Combination of Image and Language Understanding Techniques," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* pp. 775-781, 1997.

[28] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley-Interscience, 1973.

[29] L. Agnihotri and N. Dimitrova, "Text Detection for Video Analysis," *Proc. IEEE Workshop Content-Based Access of Image and Video Libraries,* pp. 109-113, 1999.

[30] B. Schölkopf, "Support Vector Learning," PhD thesis, Munich: Oldenbourg Verlag, 1997.

[31] F. Idris and S. Panchanathan, "Review of Image and Video Indexing Techniques," *J. Visual Comm. and Image Representation,* vol. 8, no. 2, pp. 146-166, 1997.

[32] A.K. Jain, *Fundamentals of Digital Image Processing.* Prentice Hall, 1989.

[33] E. Osuna, R. Freund, and F. Girosi, "An Improved Training Algorithm for Support Vector Machines," *Proc. IEEE Workshop Neural Network for Signal Processing,* pp. 276-285, 1997.

[34] Language and Media Processing (LAMP) Laboratory, media group, Univ. of Maryland, College Park, http://lamp.cfar.umd.edu, 2003.

Extraction of Features Using M-Band Wavelet Packet Frame and Their Neuro-Fuzzy Evaluation for Multitexture Segmentation

Mausumi Acharyya, Rajat K. De, *Member, IEEE,* and Malay K. Kundu, *Senior Member, IEEE*

Abstract—In this paper, we propose a scheme for segmentation of multitexture images. The methodology involves extraction of texture features using an overcomplete wavelet decomposition scheme called discrete M-band wavelet packet frame (DMbWPF). This is followed by the selection of important features using a neuro-fuzzy algorithm under unsupervised learning. A computationally efficient search procedure is developed for finding the optimal basis based on some maximum criterion of textural measures derived from the statistical parameters for each of the subbands. The superior discriminating capability of the extracted features for segmentation of various texture images over those obtained by several existing methods is established.

Index Terms—Texture segmentation, M-band wavelet packet frames, feature selection, fuzzy feature evaluation index, neural networks.

---------♦---------

1 INTRODUCTION

SEGMENTATION of multitexture image is an important problem in image analysis [1], [2]. In this regard, some applications of octave band wavelet decomposition scheme for texture analysis have been attempted [3], [4]. The main difficulty in octave band wavelet decomposition is that it can provide only a logarithmic frequency resolution which is not suitable for the analysis of high-frequency signals with relatively narrow bandwidth. The investigations of Chang and Kuo [5] and Laine and Fan [6] indicate that the texture features are more prevalent in the intermediate frequency band and showed promising results using wavelet packet frames [7], [8]. Therefore, the main motivation of this work is to utilize the decomposition scheme based on M-band ($M > 2$) wavelets, which, unlike the standard wavelet, provides a mixture of logarithmic and linear frequency resolution [9], [10]. The use of M-band wavelet decomposition gives rise to a large number of features, most of which are redundant for accurate and efficient recognition process. Therefore, proper selection of the appropriate features using some feature selection algorithms is necessary. Some recent attempts have been made for multiscale basis and feature selection in the framework of artificial neural networks (ANN) [11], [12]. In this paper, we propose a method of multitexture segmentation scheme based on discrete M-band wavelet packet frame analysis ($DMbWPF$) in order to extract the most significant information of a texture which often appears in the middle frequency channels. The other motivation of *frame* analysis is to achieve the translational invariance in texture analysis.

The proposed methodology has two parts: In the first part, we develop a computationally efficient and adaptive technique for finding out an appropriate tree-structured (less than the complete tree) M-band wavelet packet basis, to avoid a full decomposition. This selection of basis is based on some maximal criterion of textural measures to locate dominant information in each frequency channel (subband) and to decide whether information from a particular subband is needed or not. With this transform,

---

● *The authors are with the Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Kolkata - 700 108, India.*
E-mail: mau_ach@yahoo.co.in, {rajat, malay}@isical.ac.in.