ABSTRACT

Title of dissertation: ADAPTIVE ANALYSIS AND PROCESSING OF STRUCTURED MULTILINGUAL DOCUMENTS

Huanfeng Ma, Doctor of Philosophy, 2006

Dissertation directed by: Professor Rama Chellappa
Dr. David S. Doermann
Electrical and Computer Engineering Department

Digital document processing is becoming popular for applications to office and library automation, bank and postal services, publishing houses and communication management. In recent years, the demand for tools capable of searching written and spoken sources of multilingual information has increased tremendously, where the bilingual dictionary is one of the important resource to provide the required information. Processing and analysis of bilingual dictionaries brought up the challenges of dealing with many different scripts, some of which are unknown to the designer.

A framework is presented to adaptively analyze and process structured multilingual documents, where adaptability is applied to every step. The proposed framework involves:

(1) General word-level script identification using Gabor filter.

(2) Font classification using the grating cell operator.

(3) General word-level style identification using Gaussian mixture model.

(4) An adaptable Hindi OCR based on generalized Hausdorff image comparison.

(5) Retargetable OCR with automatic training sample creation and its applications to different scripts.

(6) Bootstrapping entry segmentation, which segments each page into functional entries for parsing.

Experimental results working on different scripts, such as Chinese, Korean, Arabic, Devanagari, and Khmer, demonstrate that the proposed framework can save human efforts significantly by making each phase adaptive.

ADAPTIVE ANALYSIS AND PROCESSING OF STRUCTURED MULTILINGUAL DOCUMENTS

by

Huanfeng Ma

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Commmittee:

Professor Rama Chellappa, Chair/Advisor
Dr. David S. Doermann, Co-Advisor
Professor Douglas W. Oard, Dean's Representative
Professor Min Wu
Professor Carol Y. Espy-Wilson

DEDICATION

This thesis is dedicated to my parents.

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Rama Chellappa, for his patience and efforts. Whenever I need help, he was always there for suggestions and support.

I would like to thank my co-advisor, Dr. David S. Doermann, for his friendship, encouragement, and guidance during the course of my research. He has been a valuable source of ideas and stimulating discussions on both technical and non-technical topics. It has been my great fortune and honor to earn my degree under his supervision.

I would like to thank those who served as members of my Graduate Board Oral committee, Dr. Douglas W. Oard, Dr. Min Wu, and Dr. Carol Y. Espy-Wilson for their supportive role throughout my graduate student career. Their time and energy is gratefully acknowledged. Both the assistance of the faculty and staff from University of Maryland Institute for Advanced Computer Studies (UMIACS) and Electrical and Computer Engineering Department and the financial support from the UMIACS throughout the years deserves much appreciation. The support of the TIDES and BRIDGES projects is gratefully acknowledged. In particular, I would like to thank Denise Best who did all the paper work when I went for a conference or need other help.

I would like to thank all of my colleagues and friends, Xu Liu, Jian Liang, Yang Yu, Burcu Karagol-Ayan, Tandeep Sidhu, David Mihalcik, Gang Zi and Huiping Li,

whose friendship and encouragement have made my experience at the University of Maryland a memorable one which I will cherish forever. In particular, I would like to thank Yefeng Zheng for his helpful input and stimulating discussions regarding my research. Whenever I need some help, he is always the first one I will turn to ask.

Finally I would like to thank my parents and my brother and sisters back home for all that they have done for me. I would especially like to thank Xuanhua Li with her patience, understanding, and love has helped me when I was preparing this thesis. The dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

As a subfield of digital image processing, document image analysis converts document images to symbolic form for modification, storage, retrieval, reuse, and transmission. It helps the transition from bookshelves and filing cabinets to electronic files. With the above-mentioned advantages, document image processing is becoming more popular for application in office and library automation, bank and postal services, publishing houses, and communication management [100, 55, 135, 42].

Given a binarized document image, the first step of document image analysis involves the physical layout analysis where the document image is segmented into different zones based on the spatial relations (usually spacing). Then each zone is labeled as text, graphics, table, or form. Traditional document image analysis works only on the text zone, and the analysis can be extended to table and form zones. To facilitate the next-step analysis, the text zone is further partitioned into a hierarchy of physical components containing text lines, words and characters.

During the past two decades, researchers working on document image analysis have proposed several physical layout analysis algorithms. According to the literature survey done by Mao et al. [97], O'Gorman and Kasturi [103], and Jain and

Yu [55], these algorithms can be categorized into three classes: top-down, bottom-up, and hybrid. The top-down algorithms [102, 71, 133, 55, 38, 135, 18] start from the whole document image, which is iteratively decomposed into smaller components. The bottom-up algorithms [101, 7, 40, 99, 37, 36] start with the image pixel. Pixels are clustered into characters, then words, and further into text lines. The hybrid algorithms [106, 4, 80] combine the above two approaches.

1.2 Structured Documents

Many documents we see in daily life contain a repeated structure at both the physical and semantic levels, and the layout is often based on the function of their components. Analysis of this functional layout (logical layout) is useful for converting printed documents to a paperless retrievable database. Using a technical journal paper page shown in Figure 1.1 as the example, physically segmented zones can be labeled as title, author, affiliation, section title, footer, regular paragraph, and so on. Therefore, logical layout analysis often follows the physical layout analysis and assigns each text zone a label which represents the component's function. Logical layout analysis is performed based on either a set of rules [76, 84, 51, 70, 121, 127] or formal grammars [77, 50, 23, 123].

In recent years, the demand for tools capable of searching written and spoken sources of multilingual information has increased tremendously. Global organizations, the proliferation of information on the Internet, and the explosion of information available in multiple languages have made cross-language communication essential. Access to large collections of information in a language other than English

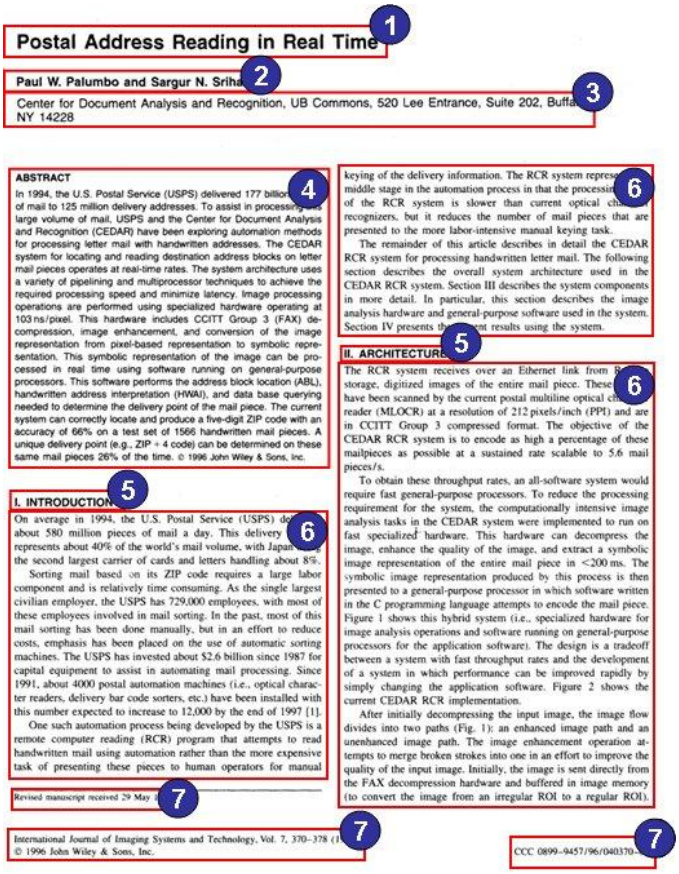


Figure 1.1: The first page of a technical paper with logical labels. (1) Title. (2) Author. (3) Affiliation. (4) Abstract. (5) Section title. (6) Regular paragraph. (7) Footer.

requires either a speaker of that language to help formulate queries and translate the retrieved documents, or an automated system for Cross-Language Information Retrieval (CLIR) and Machine Translation (MT). The former is not practical for large collections, but automated systems for CLIR and MT are evolving rapidly. CLIR systems can produce acceptable results by performing a term weight translation between the query and the document languages, thus allowing the use of well developed term-based search techniques to identify relevant documents. This strategy enables translation at the term level by building lexical resources of term-term translation pairs.

We can acquire the needed lexical resources in different ways. Resnik has shown that substantial amounts of translation-equivalent documents can be found on the World Wide Web for many languages, and a translation lexicon can be constructed from such a collection using automatic techniques for term-level alignment [110]. As the Web grows, this technique could extend to an increasingly large set of languages. Similar corpus-based techniques can also be used with printed documents [62]. Corpus-based approaches are particularly useful since the learned term-term mappings have associated translation probabilities. However, infrequent terms (which are highly valued by retrieval systems because of their selectivity) are rarely observed in training data and thus rarely included in the learned translation pairs. Hand-built translation lexicons have complimentary strengths and weaknesses; they usually lack translation probabilities, but they have better coverage of rare terms that searchers use when posing queries.

Sometimes, these translation lexicons are directly available in electronic form.

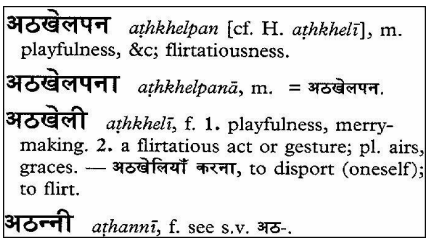
For example, the translation lexicon in an MT system could be used directly for CLIR. Available MT systems cover fewer than fifty of the world's thousands of languages, unfortunately. Online dictionaries present another possible source, but again, only a limited number of languages have such resources. When digital resources are unavailable, printed dictionaries offer a third source of term-term translations. Moreover, bilingual dictionaries often contain a wealth of supplemental information about morphology, part of speech, and examples of usage useful to CLIR and MT applications. Fortunately, bilingual dictionaries have a repeated structure which makes logical layout analysis possible. During the analysis of bilingual dictionaries, we are interested in the complete entry formed by the definition of a word and the supplemental information including the pronunciation, part of speech, example of usage and so on. Different entries are often separated spatially or semanticaly. Figure 1.2 shows a selection of bilingual dictionaries with different entry separators.

1.3 Multilingual Document Analysis

Compared with monolingual documents, the analysis of multilingual documents requires specific processing, most importantly script identification and optical character recognition (OCR). Script identification is absolutely critical to the performance of OCR. Since Latin-based characters can appear in many different documents, the OCR designers for almost all non-Latin scripts mush handle script identification. Identification is usually based on the local feature differences between two scripts, thus it requires a priori knowledge of the specific script. General script identification approaches working at page or block level were proposed by Hochberg

(a) Chinese-English dictionary

(b) German-Arabic dictionary



(c) Hindi-English dictionary

(d) French-English dictionary

Figure 1.2: Bilingual dictionaries with different separators.

et al. [46], Spitz et al. [119, 117], Waked et al. [134], and Zhu et al. [137]. These approaches assumed an entire page (or at least one block of a page) would contain the same script. This is not the case for many bilingual or multilingual documents where text with different scripts may be interlaced. So, the analysis of multilingual documents requires a general script identification approach that operates at the word level.

Another challenge the analysis of multilingual documents faces involves OCR for different languages. As the successful application of computer vision and pattern recognition, OCR for some scripts such as Latin, Cyrillic, and CJK (Chinese, Japanese and Korean) can provide high performance for scanned printed documents with high quality. Two commercial products that can recognize almost all Latin-based languages are OmniPage Pro from ScanSoft, Inc. [2] and FineReaders from

Abbyy Software House [1]. For each major world language (such as Arabic), a corresponding OCR system works for that language has been designed by researchers who are native speakers. Current technology often takes an omnicentric view, providing general solutions, which results in trade-offs between performance and application. To maintain the generality, no existing OCR system can guarantee high accuracy across a full range of documents, which makes it difficult to optimize an existing system for a specific need. The system must be retrained with respect to the specific need such as a character set or fonts and symbols. However, providing training samples for an OCR is "a high-skill, tedious, and thus often prohibitively expensive manual effort" [113]. The rapid retargeting of an existing OCR system to a specific need (fonts, symbols, and special character sets, especially for a new language) presents a big challenge in processing multilingual documents.

1.4 Problems We Want to Solve

Considering the tremendous amount of information contained in a printed dictionary, converting the text into electronic format manually is difficult. An automatic system would assist in extracting information from a printed document. An important task of automatic document processing is reading text. The automatic processing of a complex document which contains text, graphics, and/or images can be divided into three stages: (1) region extraction and text region classification using document layout analysis; (2) text line, and possibly word (glyphs separated by white space), and character segmentation; and (3) optical character recognition (OCR). Depending on document content, scanning quality, and requirements for dif-

ferent documents, an automatic text reading system usually needs to be redesigned to meet the different requirements. The redesign needs motivated us to design an adaptive algorithm for each part, bootstrapping the system to make it more generic.

Using the bilingual dictionary as an example, dictionaries are designed for easy search [32]. With a typically regular and repeating structure, "keys" offer access points for each entry. The format varies from simple word-to-phrase translation pairs to full descriptions containing parts of speech, related forms, and examples of usage. The structure analysis can help extract and organize information. Recognizing the consistency can also focus automated document analysis systems. The goal of processing the structured documents involves capturing the salient structure of the entries and labeling each element appropriately. Because of the regular structure, we typically can provide a relatively small number of training samples for each dictionary, then have the system learn the features necessary for correct entry extraction and labeling.

This research focuses on the rapid and reliable acquisition of electronic information from printed structured documents, with an ultimate goal of supporting the development of CLIR or MT systems which other translation resources lack. In most parts of the thesis, bilingual dictionaries will be used as typical representatives of the structured documents. Experimental results will also be shown for other structured documents. Given a bilingual dictionary, with one of the two languages being English, a scanner, and an operator familiar with the non-English language, a system can be trained for the new language in as little as 48 hours.

Chapter 2

Script Identification, Font Face, and Style Classification

2.1 Word-level Script Identification

2.1.1 Previous Work about Page-level Script Identification

Several techniques for determining a document's script have been proposed in the literature. Hochberg et al. [46] described a technique for identifying 13 scripts, including highly connected ones. In their algorithm, a scale-normalized cluster template was created for each script, based on frequent characters or word shapes within this script. Then scripts were classified by comparing a subset of the document's textual symbols with these templates. Spitz et al. [119, 117] initially divided scripts into Asian (Chinese, Japanese, and Korean) or Latin, based on the observation that upward concavities are distributed evenly along the vertical axis of Asian characters, but tend to appear at specific locations in Latin characters. Furthermore, discrimination among Asian scripts was based on character density. Waked et al. [134, 120] presented a set of statistical techniques for script identification. Based on connected component analysis and the horizontal projection of a text line, document images were categorized into script classes (Latin, Arabic, Ideographic, or Cyrillic) and seven European language classes. Pal and Chaudhuri [105] proposed an

approach to identify English, Chinese, Arabic, Devanagari, and Bangla text line. In their approach, the features used to identify two different scripts were different. Depending on the characteristics of scripts, the features can be shape-based, statistical, or obtained from the concept of a water reservoir. Chaudhury and Sheth [21] proposed a trainable script identification scheme to separate Indic languages from English. The identification was based on a frequency domain representation of the horizontal profile of the textual blocks, Gabor filter-based features extracted from the connected components, and the frequency distribution of the aspect ration of the connected components. In recent years, texture analysis techniques have been introduced to classify different font styles and font faces. Zhu et al. [137] presented a font recognition algorithm based on global texture analysis. Gabor filters extracted the global texture features, which were used to recognize different font styles and faces. They also demonstrated the capability by identifying Chinese and English documents with different fonts. A literature survey done by Busch et al. [17] gave a brief review about using texture for script identification.

2.1.2 Script Identification at the Word Level

The previous work cited above operate at the block or page level, which assumes the block or page contains the same script. Many bilingual or multilingual documents have text where different scripts interlace. In the English-Chinese bilingual dictionary shown in Figure 2.1, no rule identifies which part should be Chinese and which part should be English without recognizing the content. Therefore, it is impossible to combine words which belong to the same script into a whole compo-

nent before the OCR phase, and the identification must be conducted at the word level.


celebrate ['selɪbreɪt] *vt* **1** 庆祝;欢度: ~ a victory 庆祝胜利 / ~ sb.'s birthday 为某人过生日 **2** 歌颂,赞美: She was ~*d* as one of the finest dancers of her generation. 她被称颂为她这一代舞蹈演员中的佼佼者。**3** 主持,举行(仪式) —*vi* **1** 庆祝;欢度节日 **2** 主持(或举行)宗教仪式 **3**〈口〉欢乐 ‖ **celebrated** *adj* 著名的,驰名的 / **celebrator** *n* / **celebratory** ['selɪbrətərɪ] *adj*
celebration [ˌselɪ'breɪʃən] *n* **1** 庆祝;欢度 **2** 庆祝活动,庆典: hold a ~ 举行庆祝(会) **3** 歌颂,赞美 **4** 宗教仪式的举行
celebrity [sɪ'lebrətɪ] *n* **1** 著名,名声 **2** 名人,名流
celeriac [sɪ'lærɪæk] *n* 【植】块根芹

Figure 2.1: Part of an English-Chinese dictionary.

While creating an OCR system, the designer usually has knowledge of the language which he/she needs. By knowing the most significant feature that can identify that language from the Latin-based script, the designer can use those features to perform word level script identification. For example, for Chinese, the stroke complexity identifies the language in comparison to Latin-based languages, while for Hindi, the header line of Hindi language is one of its identifying features. Dhanya and Ramakrishnan [29] proposed a word level script identification approach to identify Latin and Tamil script, using spatial spread features such as character density, zonal pixel concentration, and Gabor filter responses. Except the approach proposed by Zhu et. al [137], all other script identification approaches (both at the page and word level) require sufficient knowledge about the scripts which makes them case-dependent. While the analysis of general multilingual document images can contain any unknown scripts, at the word level a general script identification

approach is required.

In our work, we perform script identification using the Gabor filter analysis of textures. The use of Gabor filters in extracting texture features of images is motivated by two factors: (1) The Gabor representation is optimal in minimizing the joint two-dimensional uncertainty in space and frequency [27]; and (2) Gabor filters can be considered orientation and scale tunable edge and line detectors, and the statistics of these micro-features in a given region characterize the underlying texture information.

2.1.3 Document Image Preprocessing

The scanning of documents inevitably introduces noise to the scanned images. Before applying the Gabor filter to extract features, we carry out the following preprocessing operations.

Deskewing

In practical situations, image skewing can result in misalignment, causing incorrect text line and word extraction. Therefore, knowing the precise skew angle of a document image is crucial for horizontal profile computation and text line extraction. To correct this, it is necessary to accurately determine the skew angle of a document image or a specific region of the image. For this purpose, a number of techniques have been presented in the literature [5, 109, 39, 107, 20, 47, 129, 85, 64, 65, 22]. Because of simplicity and efficiency, we chose the transition-count variance maximization proposed by Chen and Wang [22] to detect the skew angle and deskew

the document images.

Word Extraction

The DOCSTRUM algorithm [102] works as a bottom-up page segmentation algorithm that can handle document page images with nonManhattan layout and arbitrary skew angles. We modified the DOCSTRUM algorithm by inserting an "*make word from connected components*" operation. For some languages with many diacritics (Arabic) or with numerous accents (French), post-processing is often required to merge the separated diacritics or accents back to the core character. To merge, every small part which is not on the major text line must find and connect to the closest large component.

Word Image Replication and Scaling

Word images in different classes, even different word images in the same class, may occur in different sizes (width and height). To make the extracted features consistent, word image replication and scaling operations are applied to create a normalized image with predefined size (64×64 pixels). Features used in the following classification are extracted from same-sized images. Figure 2.2 shows word image replication and scaling examples of two different scripts (Arabic, Latin).

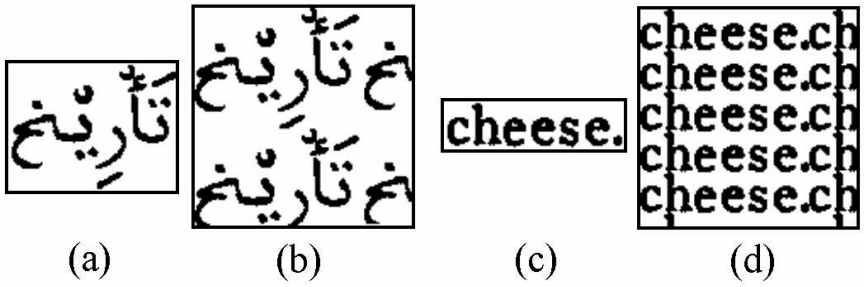


Figure 2.2: Word image replication and scaling. (a,c) The original word images. (b,d) Replicated and normalized images.

2.1.4 Extract Texture Features Using the Isotropic Gabor Filter

A pair of isotropic Gabor filters extract texture features of each class. The computational model for 2D isotropic Gabor filters are:

$$h_e(x,y) = g(x,y) \cdot cos[2\pi f(xcos\theta + ysin\theta)] \tag{2.1}$$

$$h_o(x,y) = g(x,y) \cdot sin[2\pi f(xcos\theta + ysin\theta)] \tag{2.2}$$

where h_e and h_o are the even- and odd-symmetric Gabor filters, and *g(x,y)* is an isotropic Gaussian function with form:

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{2.3}$$

The spatial frequency responses of the Gabor functions are:

$$H_e(u,v) = \frac{[H_1(u,v) + H_2(u,v)]}{2} \tag{2.4}$$

$$H_o(u,v) = \frac{[H_1(u,v) + H_2(u,v)]}{2j} \tag{2.5}$$

where $j = \sqrt{-1}$ and

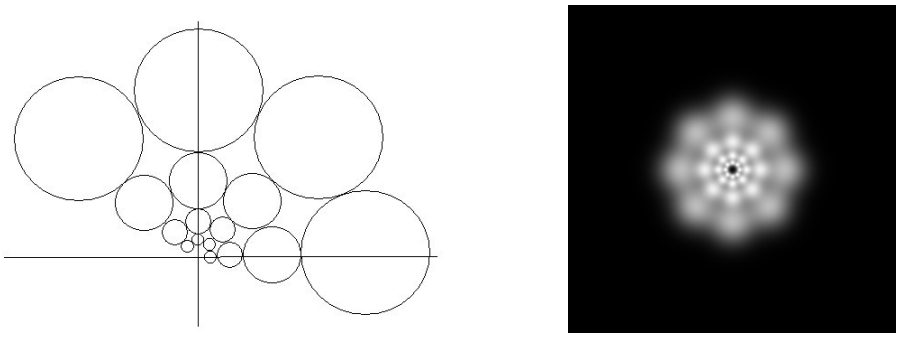
$$H_1(u,v) = exp\{-2\pi^2\sigma^2[(u - fcos\theta)^2 + (v - fsin\theta)^2]\} \tag{2.6}$$

$$H_2(u,v) = exp\{-2\pi^2\sigma^2[(u + fcos\theta)^2 + (v - fsin\theta)^2]\} \tag{2.7}$$

f, θ and σ are the spatial frequency, orientation, and space constant of the Gabor envelope.

In our case, the image size is normalized to 64×64 pixels, so four values of spatial frequency are selected: 0.04, 0.08, 0.16 and 0.32 *c/deg.* The combination of these four frequencies with four selected values of θ (0°, 45°, 90°, 135°) give a total of 16 Gabor channels.

The non-orthogonality of the Gabor wavelets implies redundant information in the filtered images. To reduce the redundancy, the filters are designed to insure the half-peak magnitude support of the filter responses in the frequency spectrum touch each other, as shown in Figure 2.3.



(a) (b)

Figure 2.3: Frequency response of Gabor filters. (a) The desired response. (b) The actual response.

2.1.5 Texture Feature Representation

The Gabor wavelet transform of an image *I(x,y)* is defined as:

$$G_{mn}(x,y) = \int\int I(s,t) g^*_{mn}(x-s, y-t) ds dt \tag{2.8}$$

where * indicates the complex conjugate.

Based on the computed mean μ_{mn} and the standard deviation σ_{mn} of the magnitude of the transform coefficients, a feature vector (with dimension 32 to represent 16 channels) is constructed as:

$$\mathbf{x} = [\mu_{00}, \sigma_{00}, \mu_{01}, \sigma_{01}, ..., \mu_{33}, \sigma_{33}] \tag{2.9}$$

where μ_{mn} and σ_{mn} are computed as:

$$\mu_{mn} = \int\int |G_{mn}(x,y)| dx dy \tag{2.10}$$

$$\sigma_{mn} = \sqrt{\int\int (|G_{mn}(x,y)| - u_{mn})^2 dx dy} \tag{2.11}$$

2.1.6 Classifiers

Four classifiers perform the classification. The first is the weighted Euclidean distance (WED) classifier, which separate the test sample based on the computed WED. Suppose the feature vector is in a d-dimensional space, and the computed mean and standard deviation feature vectors for class λ_i are $\boldsymbol{\mu}^{(i)}$, $\boldsymbol{\alpha}^{(i)}$, where $i = 1...M$ and M is the number of classes. Then for each test sample $\mathbf{x} \in \mathbf{R}^d$, the distance between this sample and each class is computed using the following formula:

$$d(\mathbf{x}, \lambda_i) = \sum_{k=1}^{d} \left| \frac{x_k - \mu_k^{(i)}}{\alpha_k^{(i)}} \right| \quad i = 1...M \tag{2.12}$$

The second classifier is the k-Nearest-Neighbor (k-NN) classifier, which extends the Nearest Neighbor classifier introduced by Cover and Hart [26] in 1967. Using the k-NN classifier, a test sample $\mathbf{x}$ is assigned the label most frequently represented among the k nearest samples. A decision is made by examining the labels of the k nearest neighbors and voting.

The support vector machines (SVMs), work as the third classifier. Introduced in the late seventies, SVMs have received increased attention recently and been applied in many fields, including handwritten digit recognition [25], object recognition [12], speaker identification [114], face detection in images [104], and text categorization [16]. The SVMs construct a "best" separating hyperplane (the maximal margin plane) in a high-dimensional feature space, which is defined by nonlinear transformations from the original feature variables. Considering the binary classification task in which we have a set of training samples $\{\mathbf{x}_i, y_i\}, \quad i = 1, ..., N, y_i \in \{-1, 1\}, \quad \mathbf{x}_i \in \mathbf{R}^d$, where y_i are labels corresponding to two classes λ_1 and λ_2, and $y_i = \pm 1$, the discriminant function is defined as:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{\Phi}(\mathbf{x}) + b \tag{2.13}$$

with the decision rule

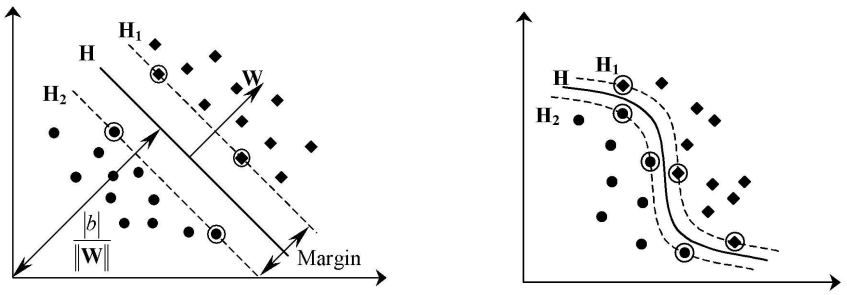
$$\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_i) + b > 0 \ \text{ for } \mathbf{x}_i \in \lambda_1 \ \text{ with } y_i = +1 \tag{2.14}$$

$$\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_i) + b < 0 \ \text{ for } \mathbf{x}_i \in \lambda_2 \ \text{ with } y_i = -1 \tag{2.15}$$

and all training points are correctly classified if

$$y_i(\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}_i) + b) > 0 \ \text{ for all } i \tag{2.16}$$

Figure 2.4(a) shows two linearly separable sets of data. Many possible hyperplanes



(a) (b)

Figure 2.4: Separating hyperplanes for two sets of data. (a) Linear separating hyperplanes; (b) Nonlinear separating hyperplanes. The separating hyperplane is $\mathbf{H} : \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + b = 0$ and two *canonical hyperplanes* are $\mathbf{H}_1 : \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + b = +1$ and $\mathbf{H}_2 : \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) + b = -1$. The circled data points (on two *canonical hyperplanes*) are support vectors.

can separate these two sets. SVMs' goal is to determine the hyperplane for which the *margin* - the distance between two parallel hyperplanes (**H1** and **H2** in Figure 2.4, which are termed the *canonical hyperplanes*) on each side of the hyperplane **H** that separates the data - is the largest. The data points that lie on the two *canonical hyperplanes* are called *support vectors* (circled in Figure 2.4).

The transformation defined by mapping function $\mathbf{\Phi}(\mathbf{x})$ in Eq. 2.13 can be linear or nonlinear, and can be applied to the separation of linearly-separable and nonlinearly-separable-only data. Figure 2.4(a) shows an example of separating hyperplanes of linearly-separable data, while the two data sets shown in Figure 2.4(b) can only be separated nonlinearly. For nonlinear SVMs, the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, which is defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}(x_i) \cdot \mathbf{\Phi}(x_j)$ can be polynomial, Gaussian, or sig-

moid. Burges [16] gave a detailed description of how to find the separating hyperplanes. We chose the SVM implementation ***SVM-light*** [57] and the polynomial kernel function for our experiments. The SVMs were trained using randomly chosen pages.

The last classifier is the Gaussian Mixture Model (GMM), which models the probability density function of a feature vector, $\mathbf{x}$, by the weighted combination of M multi-variate Gaussian densities:

$$p(\mathbf{x}|\Lambda) = \sum_{i=1}^{M} p_i g_i(\mathbf{x})$$

where the weight (mixing parameter) p_i corresponds to the prior probability that feature $\mathbf{x}$ was generated by component i, and satisfies $\sum_{i=1}^{M} p_i = 1$. Each component λ_i is represented by a Gaussian mixture model $\lambda_i = N(p_i, \boldsymbol{\mu}_i, \Sigma_i)$ whose probability density can be described as:

$$g_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right)$$

where $\boldsymbol{\mu}_i$ and Σ_i are the mean vector and covariance matrix of Gaussian mixture component i respectively, and d is the dimension of the input feature vector. So, the Gaussian mixture is completely specified by the mean vectors, covariance matrices, and mixture weights of all components and can be represented by

$$\Lambda = \{\lambda_i = N(p_i, \boldsymbol{\mu}_i, \Sigma_i)\} \; i = 1...M$$

The probability that an observed input vector $\mathbf{x}$ belongs to the class $\lambda_i = N(p_i, \boldsymbol{\mu}_i, \Sigma_i)$

is given, in terms of density, by

$$p(\lambda_i|\mathbf{x}) = \frac{p(\mathbf{x}|\lambda_i)p(\lambda_i)}{p(\mathbf{x}|\Lambda)} = p_i \frac{g_i(\mathbf{x})}{\sum_{j=1}^{M} p_j g_j(\mathbf{x})} \tag{2.17}$$

For script identification, component M is the number of different scripts. So for bilingual documents and 16 channels of Gabor filter features, we have $M = 2$ and $d = 32$. Given N training samples $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, using standard techniques, the initial Gaussian mixture model represented by $(p_i, \boldsymbol{\mu}_i, \Sigma_i)$ is estimated from the training samples as:

$$\widehat{p}_i = \frac{1}{N} \sum_{n=1}^{N} \widehat{p}(\lambda_i|\mathbf{x}_n) = \frac{N_i}{N} \tag{2.18}$$

$$\widehat{\boldsymbol{\mu}}_i = \frac{\sum_{n=1}^{N} \widehat{p}(\lambda_i|\mathbf{x}_n)\mathbf{x}_n}{\sum_{n=1}^{N} \widehat{p}(\lambda_i|\mathbf{x}_n)} = \frac{1}{N_i} \sum_{k=1}^{N_i} \mathbf{x}_k^{(i)} \tag{2.19}$$

$$\widehat{\Sigma}_i = \frac{\sum_{n=1}^{N} \widehat{p}(\lambda_i|\mathbf{x}_n)(\mathbf{x}_n - \widehat{\boldsymbol{\mu}}_i)(\mathbf{x}_n - \widehat{\boldsymbol{\mu}}_i)^T}{\sum_{n=1}^{N} \widehat{p}(\lambda_i|\mathbf{x}_n)} = \frac{1}{N_i} \sum_{k=1}^{N_i} (\mathbf{x}_k^{(i)} - \widehat{\boldsymbol{\mu}}_i)(\mathbf{x}_k^{(i)} - \widehat{\boldsymbol{\mu}}_i)^T \tag{2.20}$$

In Eqs. 2.18, 2.19 and 2.20, $1 \leq i \leq M$ and N_i is the number of samples which belong to class λ_i. Considering the fact the different distributions of script components on separate pages, the estimated models are refined iteratively via the maximum-likelihood detection. At each iteration, the decision for each observation $\mathbf{x}$ (test sample) is:

$$p(\lambda_1|\mathbf{x}) \underset{\lambda_2}{\overset{\lambda_1}{\gtrless}} p(\lambda_2|\mathbf{x}) \tag{2.21}$$

Substituting Eq. 2.17 into the above equation, then computing the likelihood of both sides, we come to the following maximum likelihood decision rule:

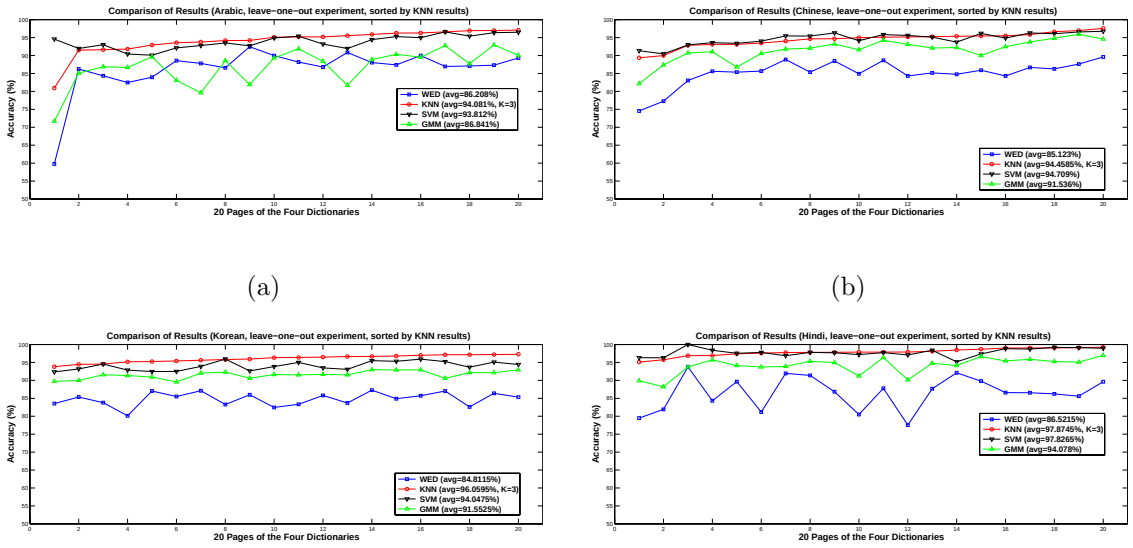
$$(\mathbf{x}-\widehat{\boldsymbol{\mu}}_2)^T\widehat{\Sigma}_2^{-1}(\mathbf{x}-\widehat{\boldsymbol{\mu}}_2)-(\mathbf{x}-\widehat{\boldsymbol{\mu}}_1)^T\widehat{\Sigma}_1^{-1}(\mathbf{x}-\widehat{\boldsymbol{\mu}}_1)\underset{\lambda_2}{\overset{\lambda_1}{\gtrless}}\ln\left(|\widehat{\Sigma}_1|\right)-\ln\left(|\widehat{\Sigma}_2|\right)+\ln\widehat{p}_2-\ln\widehat{p}_1 \tag{2.22}$$

2.1.7 Experimental Results

The proposed approaches were applied to 20 randomly chosen pages of four different bilingual dictionaries: Arabic-English, Korean-English, Hindi-English, and Chinese-English dictionary. Based on these pages, we carried out the following two experiments.

Leave-one-out

This experiment tests how the individual classifier affects the performance for limited data. For each of the four dictionaries, we partition the 20 pages into 19 training pages and 1 test page. The process repeats a total of 20 times, and Figure 2.5 shows the accuracy across all partitions. According to the experimental results, k-NN classifier provides the best performace, the identification accuracy for four dictionaries is betwee 94% and 98%. As the second best classifier for script identification, SVMs provide comparable performance for some scripts. The results of four dictionaries demonstrate the effectiveness of the texture features for word-level script identification.



(a) (b)

(c) (d)

Figure 2.5: Comparison of script identification results using four classifiers for the leave-one-out experiment. (a) Arabic-Latin. (b) Chinese-Latin. (c) Korean-Latin. (d) Hindi-Latin.

Use-one-training

In this experiment, a single page from the 20 pages is selected as the training set. The trained system is then applied to all other pages, and the average accuracy is recorded. Compared with the first experiment, these results show how a smaller (and more realistic) training set affects the performance. Figure 2.6 shows the results of this experiment. According to the experimental results, with limited amount of training data, we are still able to get reasonable performance although with an accuracy 5% drop down. k-NN and SVMs are still the two best classifiers under this restriction. For the Hidi-Latin identification result, GMM classifier has a significant accuracy drop for one page. The reason for such a low accuracy is that the randomly

chosen page used for training only contains several words. Considering the situation that the feature used for classification is a 32-dimensional vector, such a small set of training data is obviously insufficient to precisely estimate a Gaussian model with high dimension.

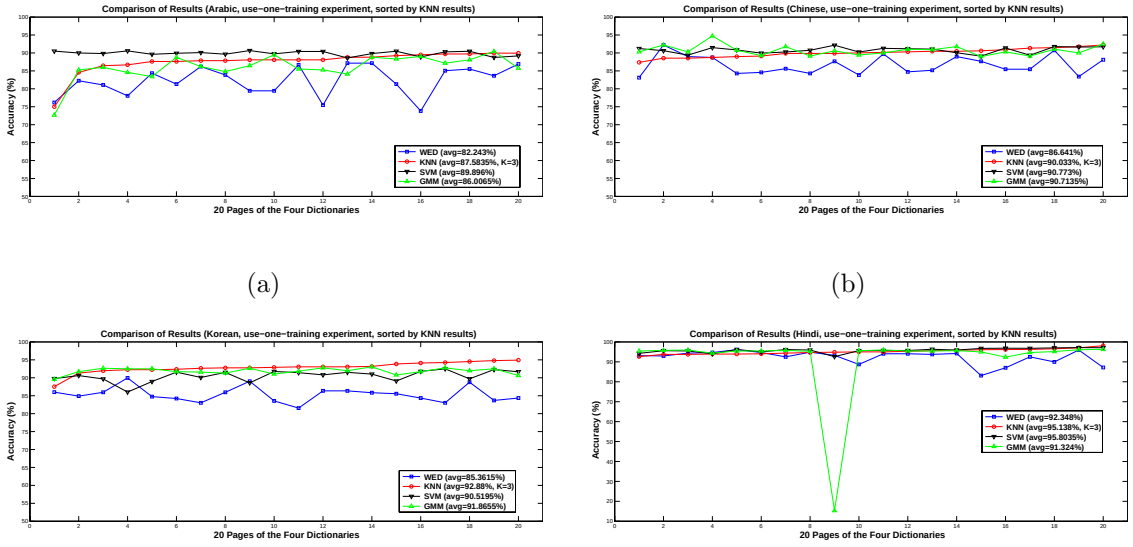


Figure 2.6: Comparison of script identification results using four classifiers for the use-one-training experiment. (a) Arabic-Latin. (b) Chinese-Latin. (c) Korean-Latin. (d) Hindi-Latin.

Classification Based on Small Size of Training Set

Since the system requires the capability to rapidly extract information from printed documents, large amounts of data are not always available. With the constraint of small training samples, the Bagging and a modified bootstrap technique are applied to improve performance.

The bootstrap technique was first proposed by Efron [34] in 1979 and fully

described in [35]. Initially proposed to evaluate the standard error of an estimation under constraint of small size of samples, the technique has been successful in improving the accuracy of certain classifiers. One of the most popular classification algorithms that applied the bootstrap algorithm is the **bagging** (**b**ootstrap **agg**regat**ing**) algorithm proposed by Breiman [14]. Define a deterministic inducer as a mapping from a training set to a classifier, the bagging algorithm is shown as follows:

Input: training set S, inducer I, integer T (number of bootstrap samples)

1. for $i = 1$ to T {
2. $\quad S' =$ bootstrap sample from S (i.i.d. sample with replacement)
3. $\quad C_i = I(S')$
4. }
5. $C^*(x) = argmax_{y \in Y} \sum_{i:C_i(x)=y} 1$ (the most often predicted label y)

Output: classifier C^*

Whether bagging will improve accuracy depends on the training procedure's stability, improvement will occur for unstable procedures where small changes in the training set can result in a large change in classifier. In his study of unstability [15], Breiman pointed out that the k-Nearest Neighbor method is stable, which means bagging may not improve the performance. In their paper [41], Hamamoto et al. analyze four different procedures to generate new bootstrap samples, which differ only in the computation of weights and drawing the first sample. Different from the traditional bootstrap samples, each generated bootstrap sample is a weighted combination of the original samples. Let $X_N = \{x_1, x_2, ..., x_N\}$ be a set of original

training samples for one class, a bootstrap sample set $X_N^B = \left\{x_1^b, x_2^b, ..., x_N^b\right\}$ with size N is generated from the original set X_N. The procedure that we applied to create the desired bootstrap set is:

---

1. Randomly select one sample x_{r_0} from X_N
2. Find the k closest samples $x_{r_1}, x_{r_2}, ..., x_{r_k}$ to x_{r_0}
3. Compute a bootstrap sample $x_1^b = \sum_{j=0}^{k} w_j x_{r_j}$
   where w_j is a weight which is given by:
   $$w_j = \frac{\Delta_j}{\sum_{c=0}^{k} \Delta_c}, \quad 0 \leq j \leq k$$
   where $\Delta_j$is chosen from a uniform distribution on [0,1] and $\sum_{j=0}^{k} w_j = 1$
4. Repeat until all N samples are generated

Our classifiers based on small size of training samples are designed by applying two steps of bootstrapping. Give N training samples, first we generate $N_B (N_B = N \times B)$ bootstrap samples by iterating the above procedure B times. Then, the bagging algorithm (with replication T) is applied to the combination of the N samples and N_B bootstrap samples. The procedure was applied to all the designed classifiers except the K-Nearest-Neighbor classifier, and the results are shown in Table 2.1.

Table 2.1: Comparison of modified bagging results ($N = 200,\ N_B = 20 \times 200,\ T = 10$) with the "use-one-training" results. (The first row shows the "use-one-training" results, and the second row shows the modified bagging results.)

Arabic			Chinese			Korean			Hindi		
WED	SVM	GMM	WED	SVM	GMM	WED	SVM	GMM	WED	SVM	GMM
82.24	89.90	86.01	86.64	90.77	90.71	85.36	90.52	91.87	92.35	95.80	91.32
81.26	86.65	86.26	89.75	90.81	90.13	77.59	90.98	91.53	94.70	95.87	88.28

The experimental results show the capability of Gabor filters to capture the features of different scripts and the effectiveness of these four classifiers to identify scripts at the word level. Considering the small number of 400 training samples in the modified bagging algorithm (one page can contain around 1000 words), the results in Table 2.1 show the modified bagging algorithm can improve the classification performance. However, we are trying to provide a general script identification approach. The identification is a sequential process which includes three main phases: document image preprocessing, word segmentation, and script identification. Any factor in the previous phase has influence on the final identification result. By manually examining the results, we found the following factors could affect the identification results: (i) word segmentation and font face; (ii) image quality; and (iii) single-character word. So, removing or reducing the effect of these factors can absolutely improve the performance. As the post-processing step, classifier combination can also be applied to improve the performance [53].

2.2 Font Face Classification

In document analysis, font classification presents an important challenge. With the availability of an accurate font classifier, an OCR system can be designed as a combination of multiple single-font recognizers, improving the performance [59, 8].

2.2.1 Previous Work

Unlike font classification approaches based on typographical features extracted by means of local attribute analysis [69, 24, 116, 139, 115], the approach proposed by

Zhu et al. [137, 122] performed the font (including face and style) identification based on the global texture features extracted with an isotropic Gabor filter bank. This approach achieved high accuracy for font style identification of computer-generated digital binary images. However, because of the difference between computer-generated images and scanned document images, the same approach only gave relatively acceptable performance for scanned document images. The performance decreased significantly, especially when the brightness and contrast vary significantly. According to this paper's analysis, one reason for the decrease involves the approach's ability to capture global font attributes but its lesser aptitude in distinguishing finer typographical attributes. Also, the classifier is a simple weighted Euclidean distance (WED) classifier, which is not effective in a 32-dimensional feature space.

Considering that they require no local analysis and may be easily tuned to work on different scripts and different fonts, font identification algorithms using global texture features are promising. In order to improve classification performance, we employed a new type of texture operator called a grating cell operator to extract the texture features and replaced the WED classifier with a back propagation neural network (BPNN) classifier. The grating cell texture operator is also based on the Gabor function but is constructed differently from the isotropic Gabor filter bank operator.

This new operator was inspired by the function of a recently discovered orientation-selective neuron type in areas V1 and V2 of the visual cortex of monkeys, called the grating cell [131, 132]. Similar to other orientation-selective neurons, grating cells respond vigorously to a grating of bars of appropriate orientation, position,

and periodicity. But unlike other orientation-selective cells, grating cells respond weakly or do not respond at all to single bars. As an operator to extract texture features, Petkov and Kruizinga [108, 78] show that the grating cell operator is very competitive in texture detection and segmentation. The following section describes the construction of this operator. Detailed presentation is available in Petkov and Kruizinga's paper [108, 78].

2.2.2 Grating Cell Operator

The grating cell operator is based on a 2D Gabor filter bank. The following family of 2D Gabor functions model the spatial summation properties of a simple cell:

$$g_{\lambda,\theta,\varphi}(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cdot cos\left(2\pi\frac{x'}{\lambda} + \varphi\right) \tag{2.23}$$

where,

$$x' = x \cdot cos\theta - y \cdot sin\theta \tag{2.24}$$

$$y' = x \cdot sin\theta + y \cdot cos\theta \tag{2.25}$$

In Eqs. 2.23, 2.24, and 2.25, the standard deviation σ of the Gaussian envelope determines the size of the receptive field. The receptive field ellipse is determined by the parameter γ, which is called *the spatial aspect ratio*. The value of γ varies in a limited range (0.23, 0.92) [58], and in our experiments γ has value 0.5. Similar to the isotropic Gabor filter, the parameter λ determines the preferred spatial frequency $(1/\lambda)$ of the receptive field function. The ratio σ/λ determines the spatial frequency bandwidth of a linear filter. We chose $\sigma/\lambda = 0.56$, as suggested in Petkov and

Kruizinga's paper [78, 108]. The parameter θ specifies the orientation, and the parameter φ ($\varphi \in (-\pi, \pi]$) is a phase offset and determines the symmetry of the function $g(x, y)$. Choosing three values (1.56, 3.12, 6.24) for λ and eight values (0°, 22.5°, 45°, 67.5°, 90°, 112.5°, 135°, 157.5°) for θ, Figure 2.7 shows the spatial frequency response of this Gabor filter bank.

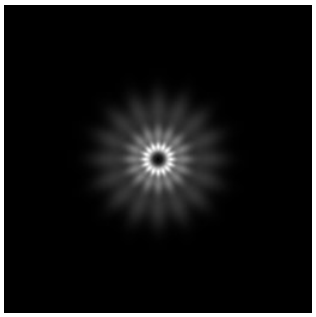


Figure 2.7: Spatial-frequency responses of the Gabor filter used for the grating cell operator.

Using the above parameterization, the response $s_{\lambda,\theta,\varphi}(x, y)$ of a simple cell modeled by a receptive field function $g_{\lambda,\theta,\varphi}(x, y)$ to an input image $I(x, y)$ can be computed as follows.

$$s_{\lambda,\theta,\varphi}(x,y) = \begin{cases} 0 & \text{if } a_\lambda(x,y) = 0 \\ \chi\left(\frac{\frac{r_{\lambda,\theta,\varphi}(x,y)}{a_\lambda(x,y)}R}{\frac{r_{\lambda,\theta,\varphi}(x,y)}{a_\lambda(x,y)}+C}\right) & \text{otherwise} \end{cases}$$

where $\chi(z) = 0$ for $z < 0$, $\chi(z) = z$ for $z \geq 0$, R is the maximum response level and C is the semisaturation constant. $r_{\lambda,\theta,\varphi}(x, y)$ and $a_\lambda(x, y)$ are computed as follows:

$$r_{\lambda,\theta,\varphi}(x,y) = \int\int I(s,t) g_{\lambda,\theta,\varphi}(x-s, y-t) ds dt$$

$$a_\lambda(x,y) = \frac{1}{\sqrt{2\pi}\sigma} \int\int I(s,t) exp\left(-\frac{(x-s)^2 + \gamma^2(y-t)^2}{2\sigma^2}\right) ds dt$$

A quantity $q_{\lambda,\theta}(x,y)$, called the activity of a grating subunit with preferred orientation θ and preferred grating periodicity λ is computed as follows:

$$q_{\lambda,\theta}(x,y) = \begin{cases} 1 & \text{if } \forall n,\ M_{\lambda,\theta,n}(x,y) \geq \rho \mathcal{M}_{\lambda,\theta}(x,y) \\ 0 & \text{if } \exists n,\ M_{\lambda,\theta,n}(x,y) < \rho \mathcal{M}_{\lambda,\theta}(x,y) \end{cases} \tag{2.26}$$

where ρ is a threshold parameter with a value smaller than but approaching one. The auxiliary quantities $M_{\lambda,\theta,n}(x,y)$ and $\mathcal{M}_{\lambda,\theta}(x,y)$ are computed as follows:

$$M_{\lambda,\theta,n}(x,y) = max\{s_{\lambda,\theta,\varphi_n}(x',y')\}$$

with (x',y') satisfying the following conditions:

$$\begin{cases} n(\lambda/2)cos\theta \leq (x'-x) < (n+1)(\lambda/2)cos\theta \\ n(\lambda/2)sin\theta \leq (y'-y) < (n+1)(\lambda/2)sin\theta \end{cases}$$

and φ_n takes values as follows:

$$\varphi_n = \begin{cases} 0 & n = -3,-1,1 \\ \pi & n = -2,0,2 \end{cases}$$

$$\mathcal{M}_{\lambda,\theta}(x,y) = max\{M_{\lambda,\theta,n}(x,y) | n = -3,...,2\}$$

In the next stage, as the final output of the grating cell texture operator, the response $W_{\lambda,\theta}$ is computed as:

$$W_{\lambda,\theta}(x,y) = \frac{1}{\sqrt{2\pi}\sigma} \int\int (q_{\lambda,\theta}(s,t) + q_{\lambda,\theta+\pi}(s,t)) \cdot exp\left(-\frac{(x-s)^2+(y-t)^2}{2(\beta\sigma)^2}\right) dsdt \tag{2.27}$$

where β takes a value 5.

2.2.3 Parameter Selection

We use an image containing four Cyrillic fonts to select the parameter values for the grating cell operator. Because of the parameter γ in expression 2.23 has a value 0.5, different from the isotropic Gabor filter bank operator, we chose eight directions for the grating cell operator, to cover all possible feature orientations. Only three spatial wavelengths (1.56, 3.12, 6.24) are chosen for the grating cell operator (Figure 2.7 shows the spatial-frequency response of this filter bank). Figure 2.8 shows no significant difference between the segmentation result using four spatial wavelengths and the result using three spatial wavelengths. Petkov and Kruizinga [108, 78] suggested the value of ρ in expression 2.26 should be a value smaller than but near one (such as 0.9). However, we found a value in range $[0.4, 0.7]$ is more appropriate for the analysis of document images. So in our experiments, the value of ρ is chosen 0.5.

(a) (b) (c) (d)

Figure 2.8: Segmentation result of four Cyrillic fonts using the K-means clustering algorithm. (a) Image with four different fonts. (b) Ground truth. (c) Segmentation result using three spatial frequencies for the grating cell operator. (d) Segmentation result using four spatial frequencies for the grating cell operator.

Although we fixed the chosen spatial wavelengths for the texture operator based on the experimental results, the values of wavelengths can be set dynamically, which could be decided by the resolution of document images and the average size of words in the images.

2.2.4 Experimental Results

With two classification techniques applied, we compare the grating cell operator with the isotropic Gabor filter operator. The first classifier used for the font classification is a simple weighted Euclidean distance (WED), which is described in Section 2.1.6 and Eq. 2.12. The second classifier is a three-layer fully-connected feed-forward neural network classifier (BPNN), using a back propagation weight tuning method. BPNN's details can be found in Duda et al.'s book [33]. For this classifier, the input layer has the same amount of units as the dimension of the input feature vector, and the output layer has the same amount of units as the font classes. The hidden layer always contains the same amount of units as the input layer. So, the BPNN classifier constructed for the isotropic Gabor filter features has 32 input units and 32 hidden units; while the BPNN classifier constructed for the grating cell features has 48 input units and 48 hidden units.

The proposed approach was applied to the scanned documents of three different scripts (Latin, Greek and Cyrillic), where each script has five common fonts Arial(AR), Century Gothic(CG), Comic Sans MS(CSM), Courier New(CN), and Times New Roman(TNR). Classification results are as follows:

Font Classification Within Script

This experiment tests the font classification for the same script. Table 2.2 shows the results for three scripts.

Table 2.2: Font classification results using different texture operators and different classifiers (**IGF**: Isotropic Gabor Filter, **GC**: Grating Cell).

Script	**Latin**				**Greek**				**Cyrillic**			
Operator	**IGF**		**GC**		**IGF**		**GC**		**IGF**		**GC**	
Classifier	**WED**	**BPNN**	**WED**	**BPNN**	**WED**	**BPNN**	**WED**	**BPNN**	**WED**	**BPNN**	**WED**	**BPNN**
AR	93.10	100.00	93.10	100.00	70.59	100.0	58.82	100.0	76.84	71.58	60.00	84.21
CG	87.50	90.62	93.75	93.75	96.67	100.0	96.67	100.0	66.67	100.00	100.00	100.00
CSM	93.54	100.00	93.54	93.54	36.67	93.33	66.67	96.67	83.19	98.32	73.95	98.17
CN	94.00	94.00	94.00	94.00	76.32	86.84	84.21	97.37	36.25	90.00	67.5	94.25
TNR	96.88	100.00	96.88	100.00	96.67	96.67	96.67	96.67	29.46	84.50	53.49	90.31

OCR Performance Comparison

As we mentioned in Section 2.2, as a preprocessor of the OCR, font classification can help improve performance. Using a recognizer based on the Zernike moment [68], we evaluated the accuracy of OCR with five fonts. The results are displayed in Table 2.3. The recognizer with no font classification is trained using the mixed training set from five fonts, and the final classifier is a Nearest-Neighbor classifier. The OCR with font classifier has five recognizers built for five fonts respectively, and the final classifier is also a Nearest-Neighbor classifier. The two OCR systems have the same training set. The results in Table 2.3 show that font classification can improve the OCR performance.

Table 2.3: OCR results for Latin, Greek, and Cyrillic scripts with and without font classification (where "WTFC" means "without font classification" and "WFC" means "with font classification").

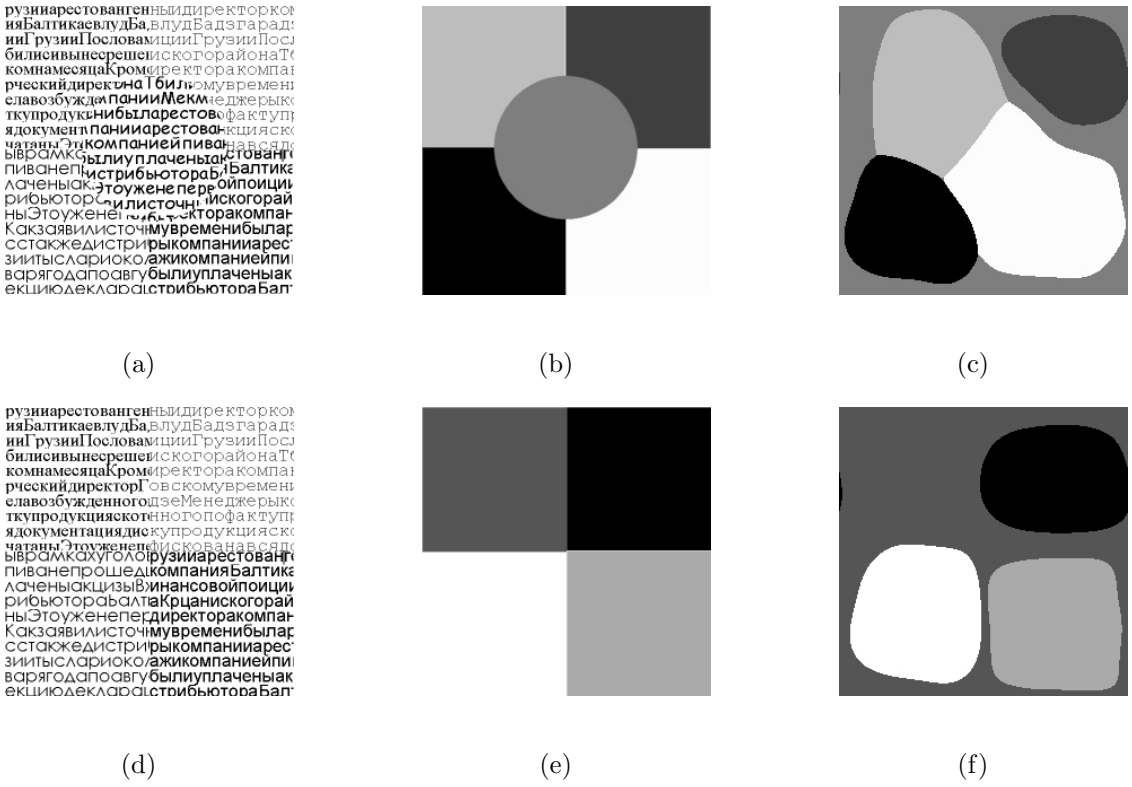
Script	**Latin**		**Greek**		**Cyrillic**	
Font	**WTFC**	**WFC**	**WTFC**	**WFC**	**WTFC**	**WFC**
AR	98.13%	98.28%	98.37%	98.48%	91.41%	93.94%
CG	95.36%	96.17%	94.86%	94.92%	79.25%	89.34%
CSM	96.63%	96.63%	95.59%	96.63%	85.89%	87.22%
CN	95.31%	96.67%	93.85%	93.91%	97.48%	98.35%
TNR	96.33%	96.37%	94.19%	94.22%	91.32%	93.97%

2.2.5 Result Analysis

The results in Table 2.2 show the grating cell operator can accurately capture texture features of different fonts. According to Petkov and Kruizinga's experiments [108, 78], the grating cell operator detects only texture and does not respond to other image attributes, while the traditional Gabor filter operator responds to both texture and other image attributes, such as edges. Because of the characteristics of document images, character strokes of different fonts often create a series of gratings with different patterns, which make the grating cell operator more effective to extract these pattern features.

In Section 2.2 we mentioned that a simple classifier (such as WED) cannot provide good performance for scanned documents, and when two fonts have close texture features, a relatively complex classifier is required. Figures 2.9(a) and 2.9(c) show images of five Cyrillic fonts and the segmentation result using the K-means

clustering algorithm; only four can be correctly segmented. However, after removing the representative of Comic Sans MS (Figure 2.9(d)) and using the same parameters to extract the features, the remaining four parts can be correctly segmented using the same K-means clustering algorithm (Figure 2.9(f)). So, the feature similarity of different fonts caused the incorrect segmentation. Comparing this result with Table 2.2, we can conclude the BPNN classifier improved the classification accuracy.



(a) (b) (c)

(d) (e) (f)

Figure 2.9: Results of font segmentation for different numbers of Cyrillic fonts using the K-means clustering algorithm. (a) Image with five different Cyrillic fonts. (b) Ground truth for five fonts. (c) Segmentation result for five fonts. (d) Image with four different Cyrillic fonts. (e) Ground truth for four fonts. (f) Segmentation result for four fonts.

2.3 Adaptive Word-level Font Style Identification

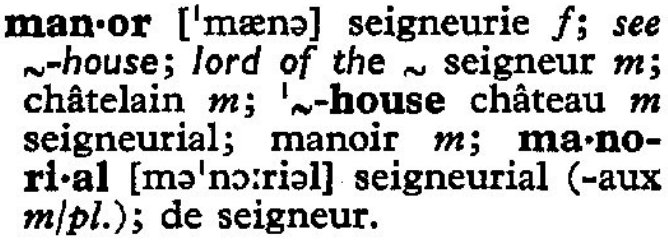


Figure 2.10: An entry from an English-French dictionary.

In a structured document, word font styles often provide additional implicit information along with the content. For example, Figure 2.10 shows an entry from an English-French dictionary with a complicated structure. We notice the headword and derived words are in bold, while the translation and pronunciation are normal, and the cross-reference, example, and gender of French words are italicized. Accurate OCR alone cannot parse such documents; style information is also necessary. In addition to providing recognized text, current commercial OCR software often provides font style information. While italic words are easily detected in most documents, boldness may differ significantly from the original printing either because of the original page's quality or because the scanner did not reproduce the original quality accurately. Figure 2.11 shows three words extracted from the same scanned dictionary using the same scanner configuration. Without comparing other words on the same page, it seems easy to identify the first word as bold and the other two as normal, where, in fact, all three words are normal. So, identifying word styles (especially boldness) from scanned documents correctly is not a trivial task, where

(a) (b) (c)

Figure 2.11: Normal words taken from different pages.

boldness must be detected as a relative measurement with respect to other words on the same page.

2.3.1 Previous Work

In earlier work on word style identification, Bloomberg [13] proposed a multi-resolution morphological approach, where features that distinguish styles were extracted and projected to the lowest reliable resolution. Properties at the lower level were then returned to the higher resolution image to make a selection mask which identifies words with different styles. This approach was successful for identification of italic words but met limited success for bold words. Manna et al. [95] used the tangent distance as a classification function in a nearest neighbor approach, and a TD-Neuron discriminant model discriminated between two similar classes for font-style detection. Kavallieratou et al. [66] presented a slant removal algorithm based on the vertical projection profile of a word image and the Wigner-Ville distribution. This approach can identify italicized words. Vinciarelli et al. [130] presented an algorithm to detect the slant based on the number of vertical strokes. This approach can also detect italicized words. In recent years, texture analysis using wavelets was introduced to classify different font-styles. Zhu et al. [137] presented a font recognition algorithm based on global textures extracted using Gabor filters. They also

showed this approach's capability to identify text blocks with different font-styles.

2.3.2 Proposed Approach

We present a new approach to identify bold and italic words within scanned documents. Assuming the availability of OCR results, the approach adapts to different documents, different font faces, and different image qualities. The approach consists of two steps: (1) semi-supervised training which performs exhaustive feature selection used for classification; and (2) testing performs the style classification based on selected features. Both steps divide further into the "initialization" and "iteration" steps shown in Figure 2.12. We describe this approach in the following subsections.

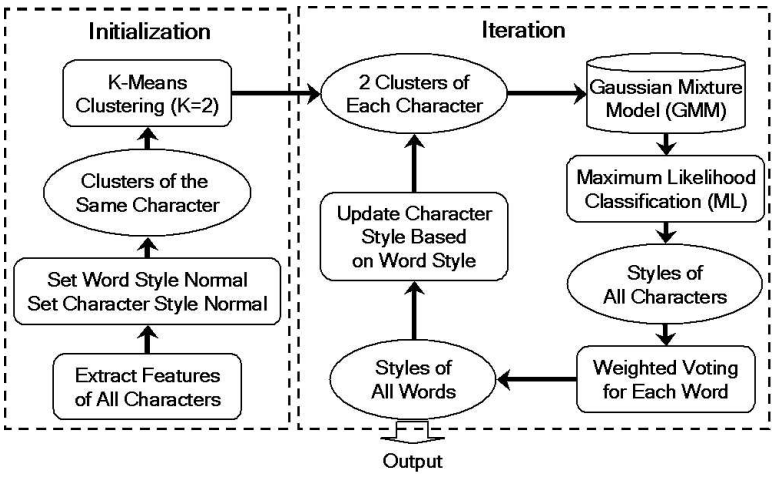


Figure 2.12: The flow chart of the approach.

2.3.3 Feature Extraction

For each character, the following features are extracted: (1) stroke width (SW); (2) foreground density (FD); (3) character aspect ratio (CSR); (4) vertical skeleton pixel ratio (VSPR); (5) possible slant angle (PS); and (6) nine-zone foreground density (NZFD). The first three features are straightforward, so we focus on the last three features as follows:

Vertical skeleton pixel ratio (VSPR): We first skeletonize the original character

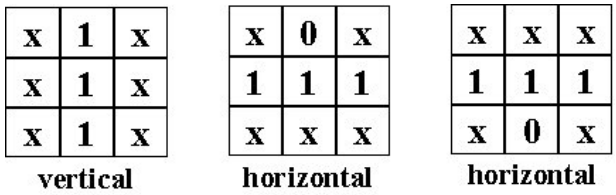


Figure 2.13: The cases of vertical and horizontal skeleton pixels.

image by the simple "hit-and-miss" transform described in [54]. For each foreground pixel in the skeleton image, the stroke's orientation is decided based on its 3×3 neighbors. Figure 2.13 shows the vertical and horizontal stroke cases, where "1" is the foreground pixel, "0" is the background pixel and "x" represents the pixel we don't care. The VSPR is defined as the fraction of vertical stroke pixels over the total number of foreground pixels in the skeleton image.

Possible slant angle (PSA): Similar to [130], for each column k of the character image, we compute the histogram using the following formula:

$$H(k) = \frac{h(k)}{\Delta y(k)}$$

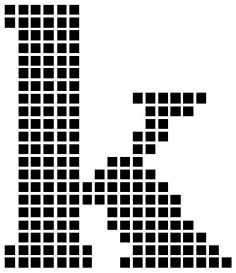


Figure 2.14: Example of continuous vertical stroke ratio (ratio=0.5).

where $h(k)$ is the vertical projection of column k, $\Delta y(k)$ is the difference between the maximum and the minimum y coordinates of column k. The angle with the maximum number of value 1 in $H(k)$ is considered the normal style, and the possible slant angle is computed by rotating the character image by a small angle in a predefined range. Figure 2.14 shows an example with continuous vertical stroke ratio 0.5.

Nine-zone foreground density (NZFD): Each character box is divided into 9 (3×3) zones (Figure 2.15). For each zone, we compute the foreground pixel density and position it into a nine-dimensional vector.

1	2	3
4	5	6
7	8	9

Figure 2.15: Nine zones of a character.

2.3.4 Feature Selection

For each character, the extracted features are placed into a 14-dimensional feature vector for training and testing. According to [56], feature selection can reduce not only the cost of recognition but can also provide a better classification accuracy due to finite sample size effects. In our work, an exhaustive feature selection procedure guarantees the optimal feature subset. For each training page, only the number of words of each style is given, and the evaluation of different feature subsets is based only on the number of words, not on the actual word styles. It is, therefore, a semi-supervised training. The classification procedure will be described in subsection 2.3.5. We address how to evaluate the performance of a feature subset as follows. Suppose the total number of words in one training page is $N^{(g)}$, and the number of normal, bold, and italic words is $N_n^{(g)}$, $N_b^{(g)}$, $N_i^{(g)}$ respectively. After classification based on one subset of features, the classified numbers of normal, bold, and italic words are $N_n^{(c)}$, $N_b^{(c)}$, and $N_i^{(c)}$. For a word k with M characters, we first compute the style consistency, defined as: $C(k) = M_c/M$, where M_c is the number of characters whose style is the same as the word style. The performance for the feature subset $\mathbf{S}$ is defined as:

$$J(\mathbf{S}) = \sum_{k=1}^{N^{(g)}} C(k) - \sum_{j=n,b,i} \left| N_j^{(g)} - N_j^{(c)} \right|$$

Let $\mathbf{F}$ be the set of all features, the optimal feature subset $\mathbf{S}_{opt}$ is selected as:

$$\mathbf{S}_{opt} = \underset{\mathbf{S} \subset \mathbf{F}}{argmax}\, J(\mathbf{S})$$

Compared with the standard training method and tested using different data sets, we claim the above performance evaluation provides the optimal feature subset in

most cases because the classes are continuous in feature space and labels are not random.

2.3.5 Classification

Considering the fact that italic words are more apparent visually than bold ones, the classifier is designed as a tree classifier with two levels. At the first level, words are classified into italic and non-italic words, and, at the second level, non-italic words are classified into bold and normal words. A Gaussian Mixture Model

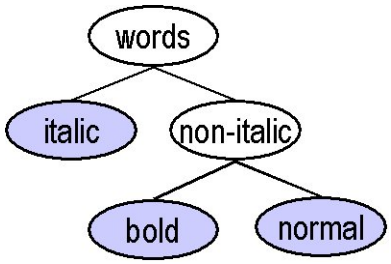


Figure 2.16: The decision tree for style identification.

(GMM) (described in Section 2.1.6) is constructed for each cluster of characters with the same code on one page. For each set of the characters with the same code, we collect their feature vectors, then perform the K-Means clustering ($K = 2$). The K-Means clustering result estimates all the parameters of the GMM using standard techniques (see [11]).

The initial Gaussian mixture model represented by $(p_i, \boldsymbol{\mu}_i, \Sigma_i)$ is estimated from the training samples as:

$$\widehat{p}_i = \frac{N_i}{N} \qquad \widehat{\boldsymbol{\mu}}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} \mathbf{x}_k^{(i)} \tag{2.28}$$

$$\widehat{\Sigma}_i = \frac{1}{N_i}\sum_{k=1}^{N_i}(\mathbf{x}_k^{(i)} - \widehat{\boldsymbol{\mu}}_i)(\mathbf{x}_k^{(i)} - \widehat{\boldsymbol{\mu}}_i)^T \tag{2.29}$$

N_i is the number of samples which belong to class λ_i.

Using Maximum Likelihood detection, the decision rule for a character with feature $\mathbf{x}$ is:

$$L(\mathbf{x}) \underset{\lambda_2}{\overset{\lambda_1}{\gtrless}} TH \tag{2.30}$$

where

$$L(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_2)^T\Sigma_2^{-1}(\mathbf{x} - \boldsymbol{\mu}_2) - (\mathbf{x} - \boldsymbol{\mu}_1)^T\Sigma_1^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) \tag{2.31}$$

$$TH = \ln\frac{|\Sigma_1|}{|\Sigma_2|} + 2\ln\frac{p_2}{p_1} \tag{2.32}$$

The confidence of decision for vector $\mathbf{x}$ is defined as:

$$\mathit{Conf}(\mathbf{x}) = 1.0 - e^{-\left|\frac{L(\mathbf{X})-TH}{TH}\right|} \tag{2.33}$$

It is easy to demonstrate that $\mathit{Conf}(\mathbf{x})$ has the response shown in Figure 2.17. After

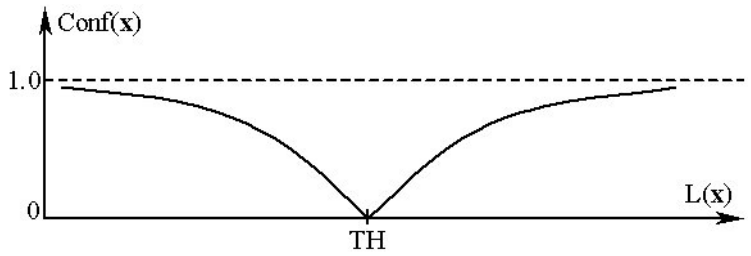


Figure 2.17: Confidence response.

obtaining the style of character and the decision confidence, a weighted majority voting approach determines the word style. For example, for the boldness of a word W with M characters, we compute the weighted voting score as follows:

$$S(W) = \sum_{k=1}^{M} Conf(\mathbf{x}_k)[I(k\ is\ normal) - I(k\ is\ bold)]$$

where $\mathbf{x}_k$ is the feature vector of the kth character, $I(\cdot)$ is the indicator function which only takes values 1 or 0 depending on whether the logical expression inside is true or false. The word style of word W is determined based on the following rule:

$$S(W) \underset{bold}{\overset{normal}{\gtrless}} 0$$

In the iterative procedure, once a word's style is determined, the styles of characters are set the same as the word style, then the updated character information help update the GMM in the next iteration.

2.3.6 Experimental Results

Using one page for training, we applied the proposed GMM approach to four bilingual dictionaries and compared the results with ScanSoft's SDK 2000 by randomly choosing 20 pages from each dictionary. Figure 2.18 shows the comparison. From the results, we can see the proposed approach performs better for both bold and italic style detection. For some documents, the improvement can reach 50%.

2.3.7 Robustness Analysis

The proposed approach depends on OCR performance. To evaluate the robustness with respect to OCR results, we generate four images from four pages of texts. Using one image as the training sample to select features, the approach was applied to 16 sets of images which were degraded with different levels of noise. The results are shown in Figure 2.19, where the x-axis is the accuracy of OCR. The result shows that this approach is more robust for boldness detection than for italic

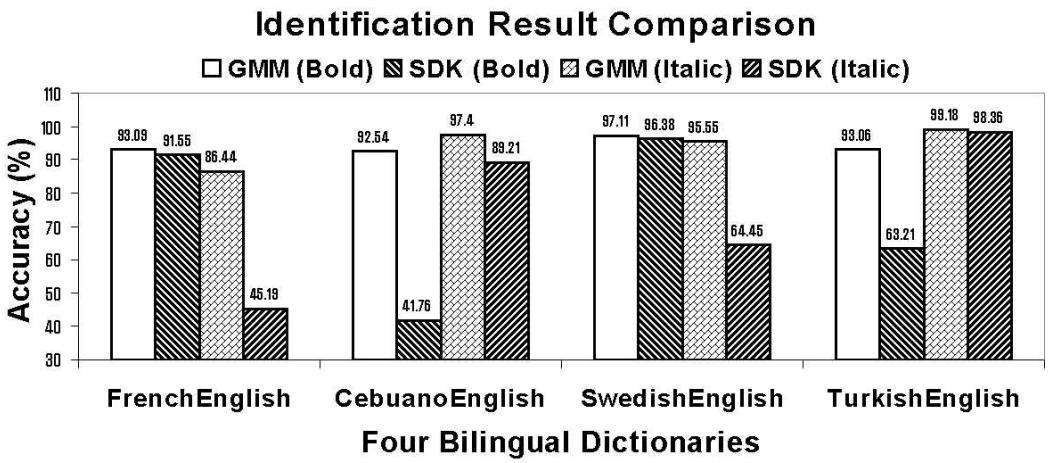


Figure 2.18: Result comparison for four bilingual dictionaries.

style detection. The detection rate for boldness is always higher than 90% while the false alarm rate is higher than 10% when the OCR accuracy is lower than 65%. For italic style detection, the detection rate is higher than 90% when the OCR accuracy is higher than 87%. By examing the degraded images, we found there are many broken strokes for images with low qualities, which affect not only the character segmentation, but also the features for style detection. For SDK, the detection of boldness totally failed, and the italic style detection is at the same level as the proposed approach when the image has a high quality. The work was published in [89]

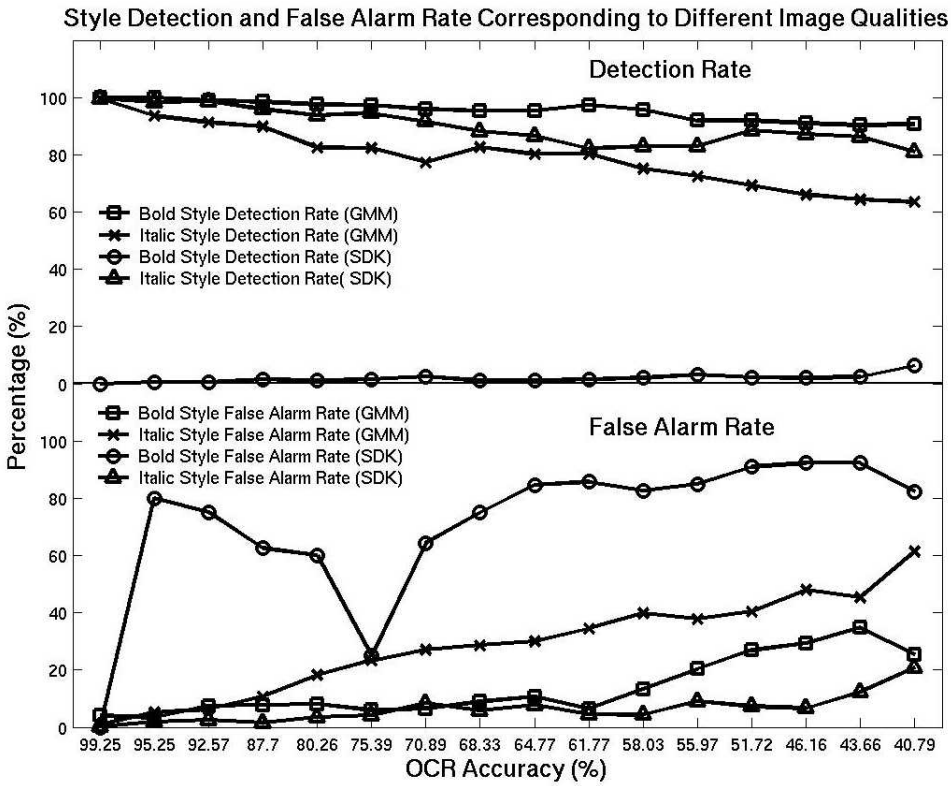


Figure 2.19: Performance corresponding to different image quality.

Chapter 3

Adaptable Optical Character Recognition

3.1 Introduction

Digital document processing is gaining popularity for application int office and library automation, bank and postal services, publishing houses, and communication management. An important task of automatic document processing involves reading text. The automatically processing of the text components of a complex document which contains text, graphics, and/or images can be divided into three stages: (1) region extraction and text region classification using document layout analysis; (2) text line, and possibly word (glyphs separated by white space), and character segmentation; and (3) optical character recognition (OCR). Typically, the OCR classifier stage needs to be redesigned for each new script, while the other stages are easier to port. OCR technology for some scripts like Latin and Chinese is fairly mature and commercial OCR systems are available with accuracy higher than 98%, including *OmniPage Pro* from ScanSoft, *FineReader* from ABBYY for Latin and Cyrillic scripts, and *THOCR* from Tsinghua University for Chinese.

Although commercial systems are available for Latin, Cyrillic, far east, and many middle eastern languages, such systems for Indic scripts, as well as many low density languages, are still in the research and development stage. Sometimes technical challenges present the difficulty, but more often it is due to a lack of a

commercial market. Nevertheless, a real need exists for OCR in these languages.

The DARPA TIDES program supports a project at the University of Maryland that focuses, in part, on acquiring resources rapidly from printed resources such as bilingual dictionaries. With the large number of languages globally, obtaining OCR system of every language is unrealistic. Instead, we need to be able to retarget OCR systems to deal with specific tasks, including new languages or new scripts.

During the "Surprise Language" task for TIDES that focused on Hindi, we faced the challenge of rapidly acquiring Hindi OCR capabilities. Since no feasible commercial OCR system was available, we wanted to develop one rapidly. In this chapter, we present a Devanagari (Hindi) OCR system using generalized Hausdorff image comparison that was developed and trained in less than a month. Trained using character samples extracted from different documents, the OCR system can be easily adapted to perform Devanagari OCR of other fonts. Details of a complete system for segmenting, parsing, and tagging bilingual dictionaries can be found in [93].

3.1.1 Background

Devanagari, an alphabetic script, is used in a number of Indic languages, including Sanskrit, Hindi, and Marathi, and many other Indic languages use close variants of this script. Although Sanskrit is an ancient language no longer spoken, written material still exists. Hindi has descended directly from Sanskrit through Prakrit and Apabhramsha, and has been influenced and enriched by Dravidian, Turkish, Farsi, Arabic, Portugese, and English. The third most common language

after Chinese and English, approximately 500 million people speak and write Hindi. Thus, research on Devanagari script, mainly int Hindi, attracts interest. In the rest of this chapter, Hindi, the language, and Devanagari, the script, are interchangable.

Unlike English and other Latin script languages, Hindi has few, commercial OCR renders. Chaudhuri and Pal proposed a Devanagari OCR system that is being marketed as a custom solution, but is not yet available commercially. The basic components of the system, were described in the literature [19, 20]. After word and character segmentation, a feature-based tree classifier recognized the basic characters. Error detection and correction using a dictionary search brought the recognition accuracy of the OCR to 91.25% at the word level and 97.18% at the character level on clean images. In his Ph.D. thesis [9], Bansal designed a Devanagari text recognition system by integrating knowledge sources. Character features such as horizontal zero crossings, moments, aspect ratios, pixel density in nine-zones, number, and position of vertex points, with structural descriptions of characters were used to classify characters and perform recognition. Based on dictionary search, the accuracy after correction averaged about 87% at the character level for scanned document images.

Both of the OCR systems mentioned above need vast amounts of training data with ground truth to achieve acceptable levels of performance. Data collection and ground-truthing is time consuming and labor intensive. Even so, before feeding a new Hindi font to the OCR, the system must be retrained to obtain reasonable accuracy. In our application, we benefit by needing only a small number of fonts for any given dictionary. We propose an approach to build a Hindi OCR quickly. Character

segmentation is similar to the approach proposed in [9] with minor changes, and the recognition is based on the generalized Hausdorff image comparison, which is like a template matching method but overcomes some traditional disadvantages. This OCR does not need to be trained using a large number of training samples, and easily adapts to different types of documents.

3.1.2 System Design

Our Hindi OCR, designed to work on pure Devanagari, or bilingual and multilingual document images with one script being Devanagari, is shown in Figure 3.1. The system contains three functional components: (1) document image pre-processing including denoising and deskewing; (2) segmentation and script identification at the word level; and (3) a classifier. In the following sections, we describe the word level script identification and focus on the design of the Hindi classifier.

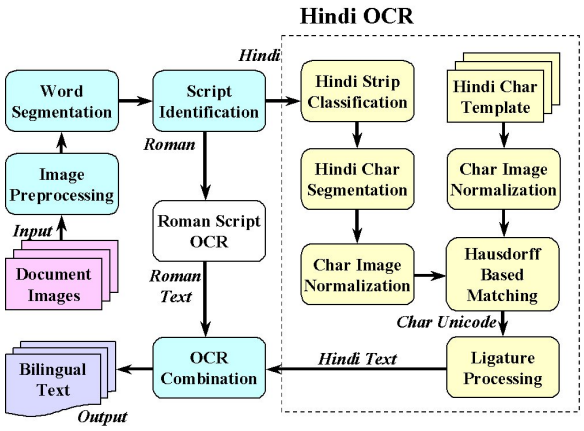


Figure 3.1: System architecture.

First, the system scans pages of Hindi text at 300 or 400 DPI. Images are first pre-processed with denoising and deskewing [52, 47]. An implementation of DOC-

TRUM [102] segment the pre-processed images into zones, text lines, and words. Then, components of the page are segmented further into entries based on the functional features of documents using the approach described in [87]. Figure 3.2 shows segmented dictionary entries. Script identification identifies the segmented word images as Devanagari script or Latin script words (including symbols neither Latin nor Devanagari). The identified Latin script word images can be processed by a commercial English OCR, while the Hindi word images are segmented into characters, then, all character images are fed into a classifier to perform recognition. After post-processing, the output of the Hindi OCR combines with the OCR output of the Latin script to provide a complete result. The details of the approach and results are described in the following sections.

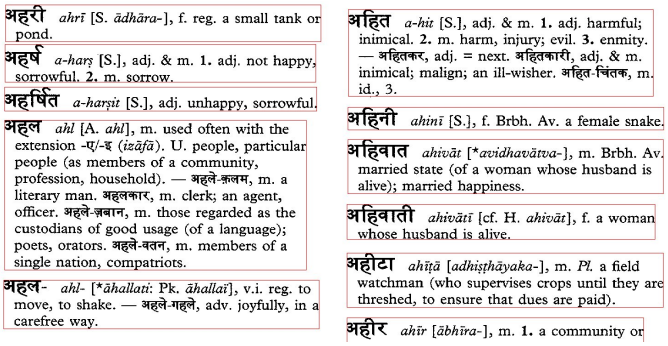


Figure 3.2: Segmented entries of the Hindi-English dictionary.

3.2 Technical Approach

3.2.1 Devanagari Script Identification

Before describing which features can identify Devanagari script words within bilingual or multilingual document images, we examine the appearance of Devanagari script. Typically, regular Hindi words can be divided into three strips: top, core, and bottom. For the Hindi word अकुलीन , Figure 3.3 shows the five-character-word image, in which three strips are illustrated. The header line always separates the top strip and core strip but no corresponding feature separates the bottom strip from core strip. The top strip contains the upper modifiers, and the bottom strip contains the lower modifiers. In a Hindi word, the top and bottom strips are not always necessary, which depend on the appearance of upper and lower modifiers.

Figure 3.3: Three strips of a Hindi word.

In [88] and the previous chapter, we proposed an approach to identify scripts at the word level based on the texture features extracted using Gabor filters. The performance of the approach can be improved by applying Supported Vector Machines (SVMs), as found in [90]. The approaches in [88, 90], identify scripts under the assumption that the operator knows nothing about non-Latin script, and the only feature used for script identification is extracted in 16 Gabor channels. For a specific script, some non-texture features can improve the performance when known

by the system. The Hindi header line presents such a powerful feature that it identifies Hindi words in bilingual or multilingual document images. For each segmented word with width W and height H, we compute the horizontal projection (denoted by HP) of this word, then find the maximum value HP_{max} and the position PS_{max} of the maximum value. A word could be identified as a Hindi word if and only if:

$$HP_{max} > 0.8W$$

$$PS_{max} > 0.5H$$

In some documents, a single Latin character, such as ***E***, ***e***, ***R***, ***T***, ***t***, ***I***, ***P***, ***D***, ***F***, ***l***, ***Z***, ***z***, or ***B*** in a specific font face may also satisfy the above two criteria. So, these misidentification cases must be handled to improve performance. In a regular document, all these characters except ***I*** seldom appear as a single character, and ***I*** is usually much narrower than a single Hindi character. These misidentifications are removed by setting a word width threshold that depends on font size and resolution of the document image. Figure 3.4 shows the performance comparison of two script identification approaches, *SVM* and the above-mentioned *Script-Oriented* approach, on 20 randomly chosen pages. The results demonstrate both approaches work effectively with an average accuracy higher than 93%, and the latter is much higher with an average accuracy of 98.94%.

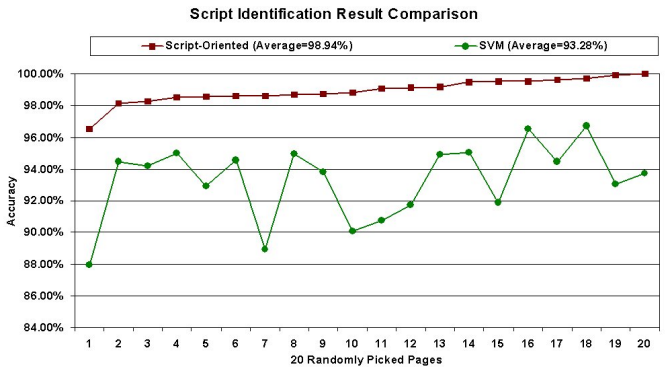


Figure 3.4: Accuracy comparison of two script identification approaches (random pages, sorted by script-oriented accuracy).

3.2.2 Character Segmentation

Devanagari Script Overview

Devanagari contains 11 vowels (shown in the first row of Table 3.1) and about 33 consonants (shown in Table 3.2). Each vowel except अ corresponds to a modifier symbol as shown in Table 3.1's second row. In Hindi, when consonants combined, a consonant with a vertical bar may appear as a *half-form*. Except for the characters क and फ, the half forms of consonants are the left part of original consonants with the vertical bar and the part to the right of the bar removed. Table 3.3 shows the half consonants, where the order of characters corresponds to the order in Table 3.2. Table 3.4 gives examples of combinations of half-consonants with other consonants, which are not always left-right structured. Sometimes the combination orients from top-down, or becomes a new character. Examples of special combinations are shown in Table 3.5. In addition to these special combinations, Table 3.6 lists some special

Hindi symbols. The list of special combinations is far from complete, while all cases need to be handled. In subsection 3.2.3, we address dealing with special cases with an operator's feedback.

Table 3.1: Vowels and corresponding modifiers.

Vowels:	अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ
Modifiers:		ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ

Table 3.2: Hindi consonants.

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
ठ	ड	ढ	ण	त	थ	द	ध	न	प	फ
ब	भ	म	य	र	ल	व	श	ष	स	ह

Table 3.3: Half forms of Hindi consonants with a vertical bar.

क्	ख्	ग्	घ्		च्		ज्	झ्	ञ्	
			ण्	त्	थ्		द्	न्	प्	फ्
ब्	भ्	म्	य्		ल्	व्	श्	ष्	स्	

Hindi Character Segmentation

Taking the segmentation of the Hindi word स्वीकृति as an example, the procedure to segment a Hindi word into characters (including core characters, and upper and lower modifiers) is illustrated in Figure 3.5. The numbered arrows in Figure 3.5 represent the segmentation steps, and the characters with solid boundary are the final segmentation results. The procedure to perform character segmentation can be described as follows:

Table 3.4: Examples of combination of Hindi half-consonants and consonants.

क क्क	क ल क्ल	घ न घ्न	च ञ च्ञ	ञ च ञ्च	त न त्न	प त प्त	प ल प्ल
ब व ब्व	भ न भ्न	म ल म्ल	ल ल ल्ल	श न श्न	श ब श्ब	श ल श्ल	स न स्न

Table 3.5: Examples of special combination of Hindi half-consonants and consonants.

क ष क्ष	ज ञ ज्ञ	ट ट ट्ट	ट ठ ट्ठ	त र त्र	द द द्द
द ध द्ध	द व द्व	द व र द्व	श र श्र	द भ द्भ	द य द्य

- *Step 1:* Locate the header line and separate the core-bottom strip (containing the core strip and bottom strip) from the top strip (containing the header line and the upper modifiers).
- *Step 2:* Separate the core strip and the bottom strip, and extract the lower modifiers.
- *Step 3:* Separate the core strip into characters which may contain conjunct/shadow characters.
- *Step 4:* Segment the conjunct/shadow characters into single characters.
- *Step 5:* Remove the header line from the top strip and extract the upper modifiers.
- *Step 6:* Put the header line back to the segmented core characters.

The details for each step are described below. We denote the width of the Hindi word bounding box as W, the height as H, and the coordinates of the left-top

Table 3.6: Special Hindi symbols.

क़	ख़	ग़	ज़	फ़	ड़	ढ़	ं	ः	:	।	ऽ	ॅ

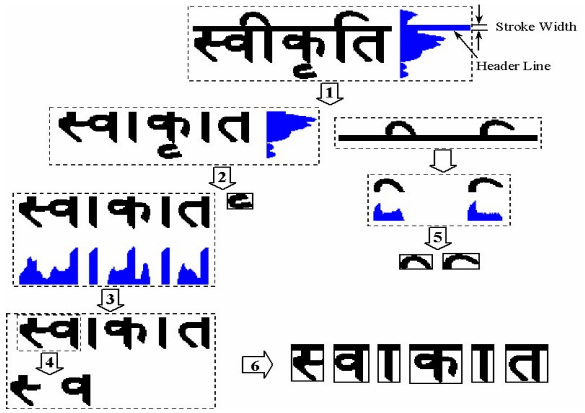


Figure 3.5: The procedure of Hindi character segmentation.

corner are set as (0,0).

Step 1: Separate the top strip and the core-bottom strip. The separation of the top strip and the core-bottom strip is based on the header line's location. For each word, we compute the horizontal projection (HP) and find the row (with Y-coordinate y) having the maximum value of HP. This is the candidate of the header line position. A header line candidate can be the real header line if: $y \leq 0.4H$. If this condition is not satisfied, then set the HP value of this row to zero and search the row again with the maximum value until a real header line position is located. The maximum HP value is marked as HP_{max}, and the position of the header line is marked as *hPosition*. Setting *hPosition* as the center, traverse the adjacent ten HP values at each side of *hPosition* and find the continuous rows whose HP values are all greater than $0.8HP_{max}$. The number of these continuous rows is the word's

stroke width, which is marked as *StrokeWidth* and important for the post-processing. *hPosition* is updated as the first row's Y-coordinates of the header line. The header line separates the Hindi word into the top strip, including the header line, and the core-bottom strip. Step 1 of Figure 3.5 demonstrates this procedure.

Step 2: Separate the core strip and the bottom strip. This procedure is shown briefly in step 2 of Figure 3.5. Denoting the width and height of the core-bottom strip obtained in the last step as W_{cb} and H_{cb}, and using the Hindi word खुशनुमा containing two lower modifiers as an example, the detailed procedure is shown in Figure 3.6, by dividing it into the following steps:

(1) Compute the vertical projection VP_{cb} of the core-bottom strip (Figure 3.6(a)).

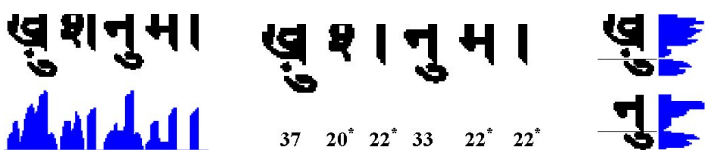
(2) The columns with no black pixels separate the Hindi word into several character candidates, which may contain conjunct/shadow characters or even incorrectly segmented characters (Figure 3.6(b)).

(3) Find the maximum height of these characters and denote it H_{max}. The separated characters divide into three groups. The first group contains all characters with height greater than $0.8H_{max}$, the second group has characters whose height is between $0.8H_{max}$ and $0.64H_{max}$, and the remaining characters make the third group. The group with the maximum number contains normal characters without the lower modifiers, and the maximum height of members in this group is set as a threshold hTh. If $(H_{cb} - hTh) \geq H_{cb}/4$, the word contains at least one lower modifier (Figure 3.6(b)).

(4) The horizontal projection HP_{cb} is computed for each separated character with

a lower modifier.

(5) In the HP_{cb} obtained in the last step, set hTh as the center. Traverse the adjacent five values at each side of hTh. The row with the minimum HP_{cb} value is the boundary that segments the core-bottom strip character into the core character and the lower modifier (Figure 3.6(c)).



(a) (b) (c)

Figure 3.6: Extraction of lower modifiers from the core-bottom strip. (a) The core-bottom strip and its vertical projection. (b) Separated characters based on the vertical projection, the number under each character is its height, and numbers with '*' are used to compute the threshold $hTh = 22$. Note: The second character is segmented incorrectly into two characters, but it doesn't affect the final result. (c) Two characters with lower modifiers and their horizontal projections, where the two straight lines denote the separation positions.

Step 3: Separate the core strip into characters. In this step, the core strip is decomposed into characters. The conjunct/shadow characters, which need further segmentation, will be determined in this step as well. We borrow the definition of shadow character from Bansal and Sinha [10]. A character is under the shadow of another character if they do not physically touch but cannot be separated merely by drawing a vertical line. In their paper, Bansal and Sinha proposed an approach

based on the statistical information (such as average width, minimum and maximum width) of characters on the text line. The approach obviously can not model our case, because in the bilingual documents, Hindi words and English words are usually interlaced. It is impossible to obtain one Hindi text line that contains words of the same size. Therefore, we separate a Hindi word into characters and determine conjunct/shadow characters based on the statistical information obtained from the current Hindi word. Before extracting the information, the Hindi modifier "ा" has a much smaller width than the regular characters after removing the header line. So, this character cannot be applied in the computation of statistical information of character width. Fortunately, this character is easily located based on the obtained stroke width from the first step. The separation of the core strip and the determination of conjunct/shadow characters are shown in step 3 of Figure 3.5, where one conjunct character is located. Taking the segmentation of another Hindi word, इस्तेमाल , as an example, Figure 3.7 details the procedure, which can be describe as:

(1) Using the method in step 2, separate the core strip into characters based on the vertical projection (Figure 3.7(b));

(2) For each separated character, if the width is smaller than $2StrokeWidth$, consider this as ा and remove it.

(3) Find the minimum width of remaining characters and denote this as W_{min};

(4) For each remaining character, if the width is greater than $1.5W_{min}$, remove the character as it may be a conjunct/shadow character that can affect the statistical information significantly (Figure 3.7(c));

(5) After removing the too-narrow and too-wide characters, compute the average width of the remaining characters and denote it as W_{avg};

(6) Traverse all the separated characters, each character with width wider than $1.2W_{avg}$ is considered a conjunct/shadow character which needs further segmentation (Figure 3.7(d)) ;

(a) (b) (c) (d)

Figure 3.7: Conjunct/shadow character determination. (a) Original word image (located header line provides *StrokeWidth*=6). (b) Five characters separated based on the vertical projection, with width 26, 51, 28, 7, 32, respectively. (c) Three characters used to compute the average width, with width 26, 28, 32, respectively, where $W_{min} = 26$ and $W_{avg} = 28.7$. (d) Detected conjunct character (with width 51).

Step 4: Segmentation of the conjunct/shadow character. The segmentation of a conjunct character is complicated by the different characteristics of conjunct and shadow characters. They are described as follows:

Segmentation of the Conjunct Character: To segment the conjunct character is to find the segmentation column from the right and the left sides of the word image, then determine the final segmentation position by comparing the two columns. After examining all the consonants, we found the following four observations:

(1) In each conjunct character, the right part is a full consonant wider than the left part, and the left part is always a half consonant.

(2) For each consonant that can be combined with a half consonant to create a conjunct character, after removing the header line, the vertical bar, and the part to the right of the vertical bar (if there is a vertical bar), the horizontal projection of the remaining part always connects without any discontinuity.

(3) Neither of a conjunct character's two parts can be too short.

(4) The pixel strength in the touching column of the two characters is usually less than that of other columns.

So, the segmentation algorithm contains three steps, based on the above observations. In the first step, segmentation column C1 is located by examining the right part of the conjunct character image (based on observations (1), (2) and (3)). Then, segmentation column C2 is located by examining the left part of the conjunct character image (based on observations (1), (3) and (4)). In the last step, the final segmentation column C is determined by comparing C1 and C2.

We use the computation of the *collapsed horizontal projection* (*CHP*) as defined by Bansal and Sinha [9, 10] to detect the continuity of an inscribed image. *Collapsed horizontal projection* can be described as: for each row of the inscribed image, if one foreground pixel can be found, then set the projection of this row 1; otherwise set the projection of this row 0. The operations in the three steps are as follows.

Locate the segmentation column C1: Illustrated in Figure 3.8, the procedure to locate the segmentation column C1 can be described as

(1) Check if a vertical bar exists in the right part of the image by computing the vertical projection of the right half of the conjunct image.

(2) If a vertical bar is present, remove the bar and the image to the right of the bar from the conjunct image. Figure 3.8(b) shows the new image.

(3) Suppose the right boundary of this new image is *nRight*, then the initial C1 is set at one stroke width to the left of *nRight*. Inscribing the right part of the image between C1 and *nRight*, the *CHP* is computed.

(4) If the *CHP* has no discontinuity and the inscribed image is higher than *H/3*, C1 is the segmentation column and stop the searching procedure.

(5) Otherwise, shift C1 one column left and repeat the above computation until a C1 that satisfies the above criteria is found.

This searching procedure is detailed in Figure 3.8(c), where the first two inscribed images have discontinuity, and the final C1 is shown in the last image.

Locate the segmentation column C2: As we mentioned in subsection 3.2.2, most half consonants have no vertical bar, but two consonants, क and फ , whose half forms also have a vertical bar. The appearance of the vertical bar can affect the final segmentation column result. So, before setting the initial segmentation column C2, we need to check the occurrence of the vertical bar in the left half of the conjunct image using the vertical projection. If no vertical bar is found in the left part of the

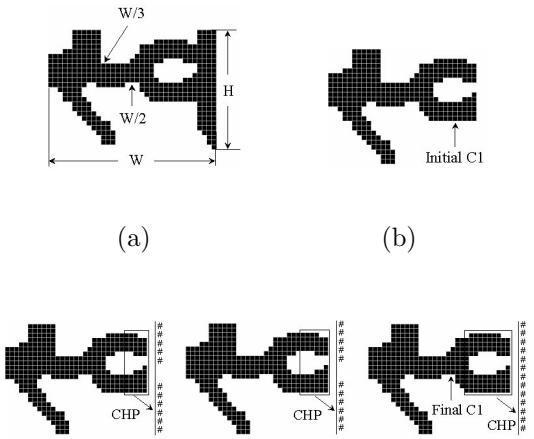


Figure 3.8: Segmentation of the conjunct Hindi character (to find C1). (a) The conjunct character image. (b) The remaining character image with vertical bar removed. (c) Steps to search for C1.

image, the initial C2 is set at *W/3*, where W is the width of the conjunct image. Searching of the segmentation column C2 is described as:

(1) Suppose the left boundary of the conjunct image is *nLeft*, the height of the inscribed image between *nLeft* and *W/3* is computed.

(2) If the computed height is less than *H/3*, then C2 shifts one column right. If the inscribed image is higher than *H/3*, then C2 shifts one column right, but only if the pixel strength of the new column is not greater than the present column. The pixel strength of the column is defined as the number of black pixels.

(3) Iterate the above steps until locate a segmentation column C2 that meets the requirement.

This procedure is shown in Figure 3.9, where the inscribed image is always higher than *H/3*, and the strength for the next column is shown in all three subfigures.

If the left part of the image has a vertical bar, suppose the right column of the bar has location *bRight*, then set the initial segmentation column C2 as *bRight+1*. Since the height of the inscribed image between *nLeft* and *C2* is higher than *H/3*, C2 shifts one column right only if the pixel strength of the new column is not greater than the present column. Figure 3.10 shows one example.

Considering the observation (1), C2 must be smaller than *W/2*, which adds another stop condition for the determination of C2.

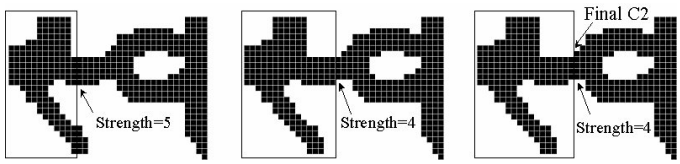


Figure 3.9: Segmentation of a conjunct Hindi character (to find C2).

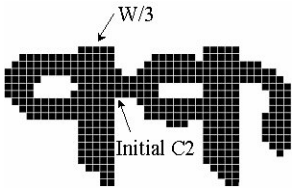


Figure 3.10: Example of conjunct Hindi character with two vertical bars.

Locate the segmentation column C by comparing C1 and C2: If an actual conjunct character is detected, the found segmentation columns C1 and C2 should be close. Considering the stop conditions of the searching iterations of C1 and C2, C1 cannot be less than C2 for a real conjunct character. So, segmentation column

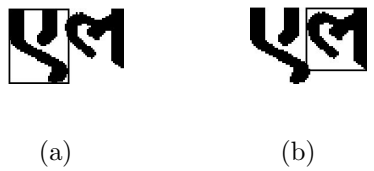
C is decided using the following three situations:

(a) C1 is less than C2: the detected character is not a real conjunct character, so no further segmentation is needed.

(b) C1 is greater than C2, and C1 and C2 are close: If the difference between C1 and C2 is less than the stroke width, then the segmentation column C is set as the average of C1 and C2.

(c) C1 is one or more stroke width larger than C2: The segmentation column C is set as the column that is one stroke width left of C1, and only the right part will be extracted. The remaining left part will be considered as a new conjunct character image with further segmentation needed.

Segmentation of the Shadow Character: The detection of a shadow character is straightforward. First, we find the character's left-most pixel. Then, the connected component starting from this pixel is detected, and the bounding box of this connected component is computed. If the right value of the bounding box is less than the right value of the original character image, then the character is considered a shadow character requiring further segmentation. The segmentation of a shadow character is shown clearly in Figure 3.11. Not many shadow cases exist in Hindi words. Usually in the shadow character image, the right character can be represented as a single connected component. So, the shadow character segmentation starts at the right side. First, we find the image's right-most black pixel, then use this pixel to find the connected component using 8-neighbor tracing. The connected component is considered the right character and separated from the original image. The left character is the remaining part with the detected connected component



(a) (b)

(c) (d)

Figure 3.11: Segmentation of the shadow character. (a) Determination of the shadow character. (b) Bounding box of the connected component (the right character). (c) Bounding box of the left character. (d) Segmented characters.

removed.

The above mentioned segmentation can be extended to the segmentation of the shadow upper modifiers (the lower modifiers usually do not have shadow situations). Three examples of shadowed upper modifiers are shown in Figure 3.12.

Figure 3.12: Examples of shadow upper modifiers.

Step 5: Extract the upper modifiers. The extraction of upper modifiers from the top strip (show in step 5 in Figure 3.5) is simple and straightforward. The header line is removed from the top strip first, then the vertical projection of the remaining strip is computed. The upper modifier's boundary is located based on the column without any black pixels. In some special cases, two upper modifiers may touch each other, which are separated as a single upper modifier. Further segmentation of the upper modifiers will be handled as a special case, described in subsection 3.2.3.

Step 6: Replace the header line in the segmented characters. A straightforward step, the header line is replaced to each segmented core character for recognition in the next step.

In these operations, some constants are defined. Part of these constants depend on the natural characteristics of the Hindi character, such as the factors 0.4, 0.8, 0.64, 1.2, 1/4, 1/3, and 1/2, which are usually fixed for different fonts or different sizes of Hindi words. Some constants (such as the 5 and 10 when traversing the projection profile) depend on size of Hindi words. We chose these constants based on experimental results as they can be fixed when the font has the standard size used in regular documents, or they can be changed based on a new size.

3.2.3 Recognition

In a typical OCR system, feature extraction is probably the most important step to achieve high performance of character recognition. Devijver and Kittler [28] defined feature extraction as the problem of "extracting from the raw data the information which is most relevant for classification purposes, in the sense of minimizing the within-class pattern variability while enhancing the between-class pattern variability." Considering many variations of the same character exist, good features for character recognition should be invariant to transformation, such as scale and rotation. Some feature extraction methods work on grayscale images, while others work on binarized images, vector images (thinned skeletons), or outer symbol contour. Trier et al. presented an overview of feature extraction methods for recognition of segmented (isolated) characters [126]. OCR approaches can be classified into

template matching, transforming, zoning or moment based.

Our Hindi character recognition approach is based on the Hausdorff image comparison. Huttenlocher et al. [49] proposed efficient algorithms to compute Hausdorff distance, comparing the resemblance between two binary images with the assumption that only a translation appears between two images. We applied the generalized Hausdorff image comparison (GHIC) to determine the resemblance of one segmented character (a point set) to another character (a template point set), by examining the fraction of points in one set that lie close to points in the other set, and vice versa. Two parameters determine the degree of resemblance of two point sets: (1) the maximum distance at which points can be separated and still be considered near; and (2) what fraction of the points in one set are at most this distance away from points of the other set. Hausdorff-based distance measures differ from the correspondence-based matching techniques in that no pairing of points in the two sets are compared [49]. Often in matching and recognition problems, the two images undergo a geometric transformation in the matching process. We are concerned with finding the transformations of one image (character image) that produce good matches to the other image (template image). In the following subsections, we give a brief introduction of generalized Hausdorff image comparison.

Generalized Hausdorff Image Comparison (GHIC)

Given two sets of points $A = \{a_1, ..., a_m\}$ and $B = \{b_1, ..., b_n\}$, the Hausdorff distance is defined as:

$$H(A, B) = max(h(A, B), h(B, A))$$

where $h(A,B) = max_{a \in A} min_{b \in B} ||a-b||$. The function $h(A,B)$ is called the "directed Hausdorff distance" from A to B (this function is not symmetric and thus not a true distance). It identifies the point $a \in A$ furthest from any point of B, and measures the distance from a to its nearest neighbor in B. So the Hausdorff distance, $H(A,B)$, measures the degree of mismatch between two image point sets, as it reflects the distance of the point of A furthest from any point of B and vice versa.

The Hausdorff distance is very sensitive to even a single *outlying* point of A or B. For example, consider $A = B \cup x$, where the point x is a large distance D from any point of A. In this case, $H(A,B) = D$, which is determined solely by the point x. Considering the fact that scanned images are often noisy and of different quality, directed Hausdorff distance cannot provide a satisfying match between two character images. Therefore, when performing recognition, rather than using $H(A,B)$, a generalization of the Hausdorff distance (which does not obey the metric properties on A and B, but does obey them on specific subsets of A and B) is used. This generalized Hausdorff measure takes the k^{th} ranked distance rather than the maximum, or the largest ranked one:

$$h_k(A,B) = k^{th}_{a \in A} min_{b \in B} ||a-b||$$

where k^{th} denotes the k^{th} ranked value (or equivalently the quantile of m values). For example, when k=m, then k^{th} is max. When k=$m/2$, then the median of the m individual point distances determines the overall distance. Therefore, this measure generalizes the directed Hausdorff measure by replacing the maximum with a quantile.

For character recognition, we are interested in using the Hausdorff distance to measure the similarity of one image bitmap I (the character image) with some "model" bitmaps M (character templates or samples), under the assumption that only translation transformation can exist between two matched bitmaps. In other words, we seek all translations $t \in R^2$ such that $h_k(M + t, I) \leq \delta$. The parameter k tells us how many of the model points should be near image points to classify a given translation as a potential matching instance of the model (i.e., we allow m-k of the m model points to be outliers). The parameter δ tells us how close each non-outlying model point must be to some image points. In our work, the parameter k depends on the scanning image quality, while parameter δ is determined based on the shape variability of same characters in different context.

To find each translation such that $h_k(M+t, I) \leq \delta$, we form $I' = I + C_\delta$, then compute the correlation of M with I'. For each translation t of M with respect to I', the correlation determines p, how many points of $M + t$ are superimposed with I' (the logical ***and*** of $M + t$ and I'). If the point number p of one translation is greater than or equals k (i.e. $p \geq k$), then the current model M with translation t is considered a match to I, i.e. $h_k(M+t, I) \leq \delta$ is satisfied. We refer to p/m as the Hausdorff fraction for a given translation t (at some fixed δ). The Hausdorff fraction measures the percentage of $M + t$ near (within δ of) points of I, which provides the degree of similarity of two image point sets and can provide the confidence value of recognition.

The computation of $h_k(M + t, I) \leq \delta$ alone does not necessarily find good matching of M in I, rather it finds portions of I that could contain M plus some

other points. For instance, with black as the foreground pixel, a totally black image will match any model in any translation. Thus, we must use the other direction of the generalized Hausdorff measure, $h_k(M + t, I)$, to "verify" those translations where $h_k(M + t, I) \leq \delta$. This reverse Hausdorff fraction ensures a given portion of an image's points (covered by the model array) are near points of $M + t$. It thereby prevents situations in which a totally black image may match any other images.

The details of the generalized Hausdorff image comparison can be found in [49]. In our recognition, the forward and reverse distance thresholds are specified to compare the resemblance between a character and the templates. Under the constraint of the two thresholds, more than one character may satisfy the conditions, thus the forward and reverse Hausdorff fractions can prune the character candidates. Furthermore, we also set two thresholds of these two fractions. The sum of the forward and reverse Hausdorff fraction computes the confidence (half of the sum which has value between 0.0 and 1.0) of character recognition, and the confidence value is used for follow on processing, which will be discussed in Section 3.4.

Normalization of Template and Character Images

Since we consider only the translation when computing the generalized Hausdorff image measure to perform recognition, the same character cannot be matched in different sizes. One solution to solve the scaling problem involves image normalization, i.e. the template and character images fed into the system must be normalized before computing the Hausdorff distance. We employ a simple normalization based on the long edges of images, which normalizes all core characters into

images with long edge 32 and all upper and lower modifiers into images with long edge 16, while preserving the aspect ratio. Denoting the image's dimensions as W (width) and H (height), the normalization procedure is:

$D_{max} = max(W, H)$

if *low or upper modifiers*, **then**

$Factor = 16/D_{max}$

else

$Factor = 32/D_{max}$

$NW = W \times Factor,\ NH = H \times Factor$

Resize the image into a new image with size $NW \times NH$

The constants 16 and 32 are not important in the normalization step, we can change this value into a variable, dependent on the average size of character images.

Classification of Characters

Hindi characters do not contain a vertical bar at the left. If a vertical bar is present, it either appears at the right end (End Bar) or in the middle (Middle Bar) of characters. Given the presence and position of the vertical bar and the conjunction number of the character with the header line, all core Hindi characters can be divided into the following six groups shown in Table 3.7.

The four characters ग ण श ग़ in the *Special Case* class possess an end bar. After removing the header line and computing the vertical projection, each of these four characters will decompose into two parts. The over-segmentation is handled in the next subsection. The character ए in the *No Bar* class is also special as it is the

Table 3.7: Classes of core Hindi characters.

Open Header	अ थ ध भ
One Conjunction End Bar	च ज ञ त न व ज़
More Conjunctions End Bar	ख घ झ प म य ष स ख़
Middle Bar	ऋ क फ क़ फ़
No Bar	इ उ ऊ ए ङ छ ट ठ ड ढ द र ह ड़ ढ़
Special Case	ग ण श ग़

only *No Bar* character which has more than one conjunction with the header line.

Over-segmentation Processing

In the above character segmentation procedure, a possibility of over-segmentation of characters exists, i.e. one single character could be segmented into two or more parts. We divide the over-segmentation into two types, horizontal and vertical, and handle them separately.

The over-segmentation in the vertical direction happens only to long characters, such as ए ह , and other strongly combined forms of consonants such as द्ध द्ध द्ध द्ध and ङ्ग. After being segmented into two parts, often the bottom part is rejected or incorrectly recognized during the recognition procedure. This type of over-segmentation is handled by adding the over-segmented top part into the templates and assigning them special codes. Once recognized, we know this is an over-segmented character and its following part will be replaced. For example, Fig-

ure 3.13(a) shows some of the over-segmented characters we added to our templates, and characters in Figure 3.13(b) are the original complete forms.

(a) (b)

Figure 3.13: Examples of over-segmented characters added in the template. (a) Over-segmented characters. (b) Original characters.

The handling of over-segmentation in the horizontal direction can be more complex because of the half-consonant. Taking the four *Special Case* characters, their half forms occur in the left part of the over-segmented parts. The determination of an over-segmented character depends whether the following character is character ◌ or not. If the four cases are over-segmented, then the next character must be character ◌, but this cannot be determined until the next character has been recognized. We will discuss this type of over-segmentation in the "Ligature Processing" section.

Dealing With Special Characters

It is impractical to add all special characters to the template because many of them are rarely used. In addition, adding more templates will significantly slow the recognition. We provide a simple scheme allowing the operator to add new special characters easily, meaning the recognizer can easily adapt to new cases. If the operator finds one new character, he first classifies this character (end bar, middle bar, no bar, etc), and places it into a corresponding file that includes the

template name and unicode of this character. Then, he cuts this character from the image, saves it into a TIFF file, and places the file name into the training template directory. The recognizer will automatically read the templates from the file and perform recognition. The recognition of all of the special characters listed in Table 3.5 and Table 3.6 are handled by our system.

Due to the quality of scanned document images, two or more upper modifiers may touch each other and cannot be segmented using the approach described above. Some of these touching upper modifiers are shown in Figure 3.14. These touching upper modifiers are considered as special upper modifiers added to the class. Fortunately, few cases exist, so this case can be handled easily.

Figure 3.14: Examples of touching modifiers.

3.2.4 Ligature Processing

Devanagari characters, like characters from many other scripts, combine or change shape depending on their context. A character's appearance is affected by its relation to other characters, the font used to render the character, and the application or system environment. Subsection 3.2.3 discusses these special characters.

Additionally, a few Devanagari characters may result in a change in the order of the displayed characters. This re-ordering is rare in non-Indic scripts. One such character is र, which is always displayed one consonant left of its real position. When exporting the codes of one Hindi word with this character, the codes of characters

must be reordered.

The Devanagari script is noted for its large number of consonant conjunct forms that serve as orthographical abbreviations (ligatures) of two or more adjacent letter forms. This abbreviation takes place only within a consonant cluster.

Some independent characters, such as ई, ऐ, ओ, औ, अं, ो and ौ have a top strip segmented and classified as upper modifiers. When exporting the characters' encoding, the upper modifier should be replaced into the corresponding core character to generate a correct code.

Independent characters such as आ, ओ, औ will be segmented into अ plus ा, ो, ौ. When exporting the encoding of these characters, the separated parts should also be replaced to generate a correct single code.

Our system has success with the above schemes. Since most of them are caused by the Devanagari ligature, we discuss this in "Ligature processing".

3.3 Experimental Results

3.3.1 OCR Evaluation

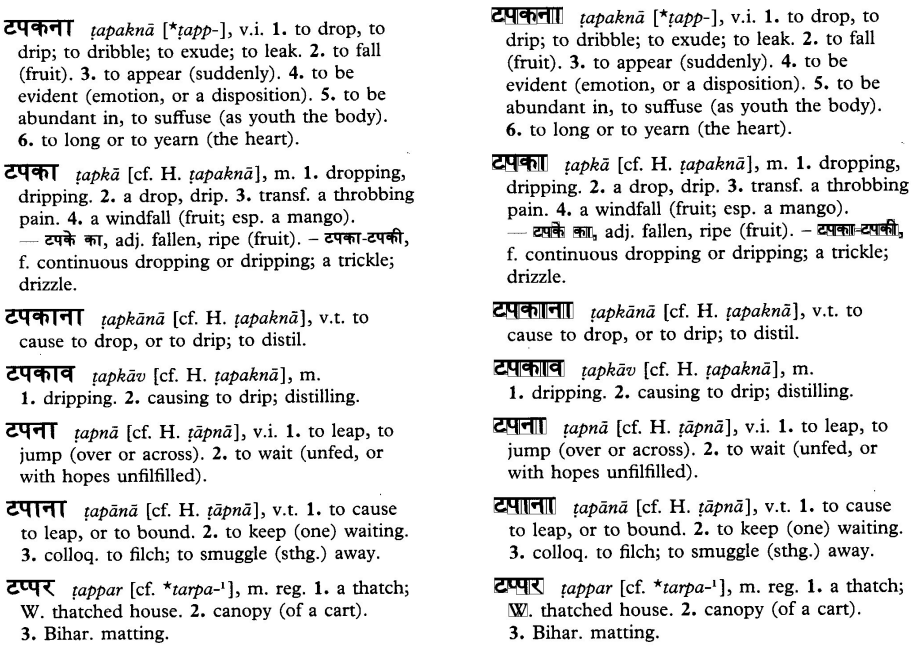
The proposed system was applied to the 1083 pages of the "Oxford Hindi-English Dictionary" [98] and to a collection of PDF-converted Hindi document images. The dictionary binding was burst and scanned at 400 DPI. The PDF-converted Hindi document images are obtained directly without the introduction of scanner noise, so they are ideal images.

Figure 3.15 shows an example of the scanned dictionary image, where (a) is

the original image, and (b) shows the identified Hindi words (with errors) and the segmented characters. The OCR result which combines the Hindi OCR and the Latin OCR is shown in Figure 3.16. To evaluate the accuracy of this OCR, we randomly chose 7 pages and counted the number of Hindi words and characters recognized. Table 3.8 demonstrates the results. From Figure 3.15, we notice the identified Hindi words can be correctly segmented into isolated characters using the proposed approach. The evaluation result displayed in Table 3.8 shows the recognition accuracy at the character level reaches 87.75%, while the accuracy at the word level nears 67%. The experiment was conducted on scanned images containing noise, and the result is the pure recognition result without any spell checking and word correcting based on dictionary search. We strongly believe, with the availability of Hindi language constraints and electronic text, correction techniques can be applied to the postprocessing stage to improve performance.

As we mentioned in subsection 3.1.1, Bansal [9] proposed a Hindi text recognition system by integrating knowledge sources. After correction, based on dictionary search, the average accuracy nears 87% at the character level for scanned document images. This result is comparable with our system's performance without any correction. The recognition accuracy of the Hindi OCR system proposed by Chaudhuri and Pal can achieve 91.25% at the word level and 97.18% at the character level. However, this accuracy was obtained on clean images with error detection and correction based on a dictionary search.

To test the effectiveness of the proposed approach working on clean images, we processed PDF-converted ideal images and evaluated them. The accuracy comes



टपकना *ṭapaknā* [**ṭapp-*], v.i. **1.** to drop, to drip; to dribble; to exude; to leak. **2.** to fall (fruit). **3.** to appear (suddenly). **4.** to be evident (emotion, or a disposition). **5.** to be abundant in, to suffuse (as youth the body). **6.** to long or to yearn (the heart).

टपका *ṭapkā* [cf. H. *ṭapaknā*], m. **1.** dropping, dripping. **2.** a drop, drip. **3.** transf. a throbbing pain. **4.** a windfall (fruit; esp. a mango). — **टपके का**, adj. fallen, ripe (fruit). – **टपका-टपकी**, f. continuous dropping or dripping; a trickle; drizzle.

टपकाना *ṭapkānā* [cf. H. *ṭapaknā*], v.t. to cause to drop, or to drip; to distil.

टपकाव *ṭapkāv* [cf. H. *ṭapaknā*], m. **1.** dripping. **2.** causing to drip; distilling.

टपना *ṭapnā* [cf. H. *ṭāpnā*], v.i. **1.** to leap, to jump (over or across). **2.** to wait (unfed, or with hopes unfilfilled).

टपाना *ṭapānā* [cf. H. *ṭāpnā*], v.t. **1.** to cause to leap, or to bound. **2.** to keep (one) waiting. **3.** colloq. to filch; to smuggle (sthg.) away.

टप्पर *ṭappar* [cf. **tarpa*-¹], m. reg. **1.** a thatch; W. thatched house. **2.** canopy (of a cart). **3.** Bihar. matting.

(a)

टपकना *ṭapaknā* [**ṭapp-*], v.i. **1.** to drop, to drip; to dribble; to exude; to leak. **2.** to fall (fruit). **3.** to appear (suddenly). **4.** to be evident (emotion, or a disposition). **5.** to be abundant in, to suffuse (as youth the body). **6.** to long or to yearn (the heart).

टपका *ṭapkā* [cf. H. *ṭapaknā*], m. **1.** dropping, dripping. **2.** a drop, drip. **3.** transf. a throbbing pain. **4.** a windfall (fruit; esp. a mango). — **टपके का**, adj. fallen, ripe (fruit). – **टपका-टपकी**, f. continuous dropping or dripping; a trickle; drizzle.

टपकाना *ṭapkānā* [cf. H. *ṭapaknā*], v.t. to cause to drop, or to drip; to distil.

टपकाव *ṭapkāv* [cf. H. *ṭapaknā*], m. **1.** dripping. **2.** causing to drip; distilling.

टपना *ṭapnā* [cf. H. *ṭāpnā*], v.i. **1.** to leap, to jump (over or across). **2.** to wait (unfed, or with hopes unfilfilled).

टपाना *ṭapānā* [cf. H. *ṭāpnā*], v.t. **1.** to cause to leap, or to bound. **2.** to keep (one) waiting. **3.** colloq. to filch; to smuggle (sthg.) away.

टप्पर *ṭappar* [cf. **tarpa*-¹], m. reg. **1.** a thatch; W. thatched house. **2.** canopy (of a cart). **3.** Bihar. matting.

(b)

Figure 3.15: One example from the bilingual dictionary. (a) Original image. (b) Identified Hindi words and character segmentation.

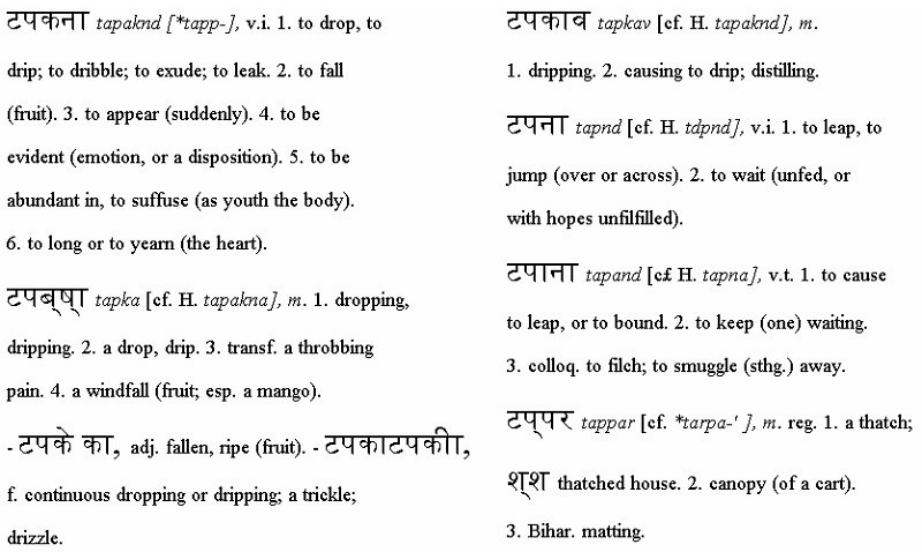


Figure 3.16: OCR results of images shown in Figure 3.15 (reconstructed by combing Hindi and Latin results).

close to 95% (with 2584 characters and 2450 correctly recognized for one page). Figure 3.17 shows part of one converted clean image and its OCR result.

3.3.2 Discussion

In examining the data, we found a number of factors contributed to the incorrect recognition including:

(1) *Incorrect word segmentation.* The word segmentation performance depends on the quality of document images. Noise may cause the incorrect under- or over-segmenting of words or merging of words with other symbols. Incorrect word segmentation can further affect the script identification result, leading to the degradation of OCR performance.

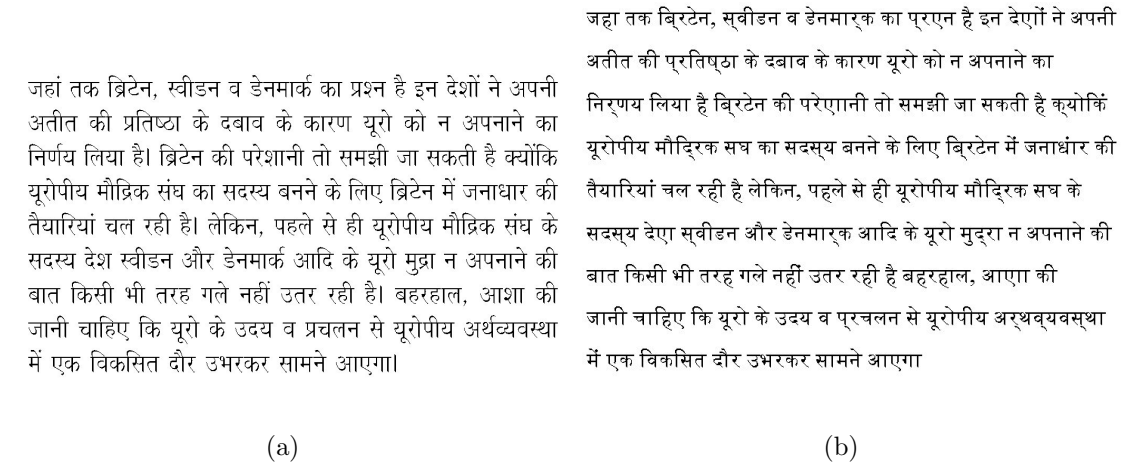
(2) *Incorrect character segmentation.* Segmentation is a challenging task, es-

Table 3.8: Result evaluation of the Hindi-English dictionary, where "A1" is the character accuracy with respect to "Chars", and "A2" is the character accuracy with respect to "Recognized". "A" is the word accuracy.

Pages	Chars	Recognized	Correct	A1	A2	Words	Correct	A
p0098	451	443	407	90.24%	91.87%	110	79	71.82%
p0160	317	311	272	85.80%	87.46%	73	54	73.97%
p0179	480	477	409	85.21%	85.74%	113	67	59.29%
p0401	294	290	264	89.80%	91.03%	71	53	74.65%
p0799	437	451	379	86.73%	84.04%	80	50	62.50%
p0987	405	402	359	88.64%	89.30%	67	39	58.20%
p1023	343	338	303	88.34%	89.64%	64	44	68.75%
Total	2727	2712	2393	**87.75%**	**88.24%**	578	386	**66.78%**

pecially for scanned Hindi images. If images contain noise, the average width and height of characters may not be obtained accurately. During segmentation, many decisions, such as identifying conjunct/shadow characters, determining lower modifiers, and determining stroke width, are based on these statistics. Incorrect statistics can significantly degrade the performance of character segmentation, therefore degrading the final performance of recognition.

(3) *Missing punctuation such as commas, periods, and parentheses.* Often the space between words and the following punctuation is small. Punctuation can therefore be merged with Hindi words during word segmentation. As a direct result, these symbols can influence negatively the distribution of the features used to perform segmentation. Although we tried to solve this by detecting punctuations first, some cases were incorrect.



(a) (b)

Figure 3.17: OCR result of the PDF converted ideal image. (a) Original image. (b) OCR result.

(4) *Character mis-classification due to noise.* This typically happens with *Open Header* and *Middle Bar* characters. Noise may cause an *Open Header* character to become a closed header character, or cause the detected vertical bar in a *Middle Bar* character to shift. The classes rarely overlap, so if one character is mis-classified, it most likely will not be recognized correctly.

(5) *Character similarity.* There exist Hindi characters with similar appearances. Sometimes the scanning noise makes them appear almost indistinguishable. During recognition, they may have the same confidence value, which can then make the final selection of OCR output of this character ambiguous. Higher level context or language models may help solve this problem.

(6) *Special symbols which are noise-like.* Some special symbols (including notations) are visually noise-like, although the position of these symbols relative to the text provides strong context. Sometimes, these symbols are removed as noise, other times noise is left and results in incorrect modifiers. This causes incorrect

classification and degrades the OCR's performance.

To improve performance, we consider all the factors discussed above and strive to remove their effects. Since noise causes a majority of the incorrect recognition, applying new denoising techniques, such as the approach in [136], or making the parameters more flexible with varying image quality can improve performance. For incorrect recognition caused by mis-classification, more rules or new classes may be added to increase the accuracy.

As a post-processing step, a number of known ways can increase OCR's accuracy for free text, and most use either general or domain specific lexicons. The recognized terms are looked up in the lexicon, and if they are present, no further analysis is required. If they are not present, we must assume an OCR error occurred and attempt to select the correct term. Often, this is accomplished with a distance measure between terms, typically based on character recognition confusion probabilities.

We use OCR to process bilingual dictionaries, and these types of documents introduce several inherent problems for OCR correction. First, we have a source, that, by nature, has few instances of some words, yet has a fairly complete coverage of the language. In the case of Hindi, the lexicons are not sufficiently complete to warrant statistical correction approaches. Although we have significant amounts of electronic text to generate a lexicon, most naturally occurring text provides only a limited coverage of the language. We have performed experiments with a limited lexicon, but not surprisingly, the overall recognition rates decrease. When a term is spelled correctly, but does not appear in the lexicon, we actually introduce more

errors by mapping these words incorrectly.

The second problem appears similar to the question the chicken and egg problem. Since we have no ground truth information (and it was not feasible to obtain it), it is difficult to estimate the OCR confusion probabilities needed for intelligent distance measures. We are currently exploring various character level correction schemes, but they rely on a statistical distribution of character bi- or tri-grams that may not be accurate for dictionaries.

In general, if we are trying to produce OCR systems for low density languages, large amounts of electronic text may not be available, so we are exploring other ways to semi-interactively identify common confusions.

3.4 Summary and Future Work

We have presented an adaptive Hindi OCR system that uses a generalized Hausdorff image comparison implemented as part of a rapidly retargetable language tool. The system includes three stages: (1) script identification; (2) character segmentation; and (3) training sample creation and character recognition. Based on the generalized Hausdorff image comparison, the system retargets easily to different Hindi fonts or even a different script, provided the segmentation can use the same or a similar approach. The OCR is also designed to recognize unknown special characters and to provide a simple interactive interface for the user to add them. The OCR (designed and implemented in one month) was applied to a complete Hindi-English bilingual dictionary and a set of ideal images extracted from Hindi documents in PDF format. Experiments result in an average recognition accuracy

of 87.82%, while the accuracy for ideal images was 95%, both at the character level, without any spell checking.

A major thrust of future work will be to perform OCR correction or to resolve ambiguity among candidates. One advantage is we assign confidence as a side effect of recognition. By setting two thresholds on the Hausdorff distance, the forward and reverse Hausdorff fractions under the constraint of these two thresholds can compute the confidence of a character and word recognition. This confidence has a real value between 0.0 and 1.0, making it more intuitive and usable than current commercial OCR software such as *ScanSoft Developer's Kit 2000* from ScanSoft and *FineReader Engine* from ABBYY. For these packages, the confidence of each character is a boolean value, which gives the same weight to all characters when computing a word's confidence.

Given the confidence of characters and words, we can further consider the word correction based on a dictionary search. The word correction engine would determine whether a word should be replaced with another correct word from the dictionary, significantly improving the recognition performance at both character and word level. Since a possibility exists that the recognized result may be over-corrected, the word correction also can provide a probability for the replacement, which tunes the correction. Details of this correction are discussed in [73, 74].

Another advantage involves the adaption to different image qualities. For example, if a scanned document image has a poor quality, the Hausdorff thresholds can be set to lower values, making the classifier and recognizer more tolerant. If the image has high quality, the thresholds can be set to higher values, speeding the

recognition process.

The next recognition step is to apply new, possibly multiclassifier techniques, and combine them with the current Hausdorff classifier to provide improved performance. The approach assumes we can train the system using a small number of samples, so the new classification techniques must also have this property.

Our OCR system was designed with the assumption that no vast amount of training samples are available, so it easily extends to the recognition of other languages or scripts under the following two conditions: (1) symbols of the same class; (2) the words can be segmented into characters. Once segmented, glyphs can use the same classifiers for recognition. Segmentation, however, differs greatly between Hindi and other non-Indic languages. For Chinese, segmentation is straightforward because character spacing is fixed. For many Latin fonts, kerning must be considered, while languages such as Arabic must consider touching characters during segmentation. Under the two assumption conditions, the segmentation can be changed based on the characteristics of the new script, while the recognition will be adapted through exemplars. The user can create new rules to classify characters, obtain samples from document images, set output codes for each character, set thresholds for the Hausdorff distance, and perform recognition. If extracting features that classify characters is difficult, the operator can put the whole set of characters into a single class, compute the Hausdorff distance between the segmented characters and each character in the class, then perform recognition. Although the system was designed to deal specifically with Hindi text, it was modularized so we can apply different components for other languages. Overall, our goal is to build a toolkit of

components that can be reused to build OCR capabilities rapidly for new languages.

Dealing with special characters can also extends to the recognition of some symbols that are difficult to segment correctly (such as complex ligatures). The user can include them in the samples and perform recognition. The only thing the user needs to do before performing recognition is to make sure the encoding of these special characters is correct. Finally, a key to making our system generally adaptive is to consider how to allow system parameters to dynamically adjust for changes in image quality.

Chapter 4

Automatic Training Sample Creation for OCR

4.1 Introduction

Optical character recognition (OCR) is one of the successful applications of the research in the computer vision and pattern recognition field. The technology for some scripts, such as Latin, Cyrillic, Chinese, Japanese, and Korean is fairly mature. Commercial OCR systems on the market today can provide recognition results with reasonable accuracy for high-quality printed documents. However, current technology takes an omnicentric view and provides general solutions, with trade-offs between performances and applications. To maintain the generality, no existing OCR systems can guarantee high accuracy across a full range of documents, which makes it difficult to optimize an existing system for a specific need. The system needs to be retrained with respect to a specific need such as a special character set, special fonts, and symbols. Providing training samples for an OCR is, "a high-skill, tedious, and thus often prohibitively expensive manual effort" [113]. Some researchers have focused on relieving this critical restriction to the automatic analysis of printed documents. For example, Liang et. al [82] proposed a methodology for special symbol recognition. Working on the output of an available OCR system, special symbols were detected and recognized based on the recognition confidence level. The research of Sarkar et. el either focused on the automatic creation of

training samples by aligning noisy and synthetic text-line images [112] or on the automatic classifier selection [113], and both required the support of ground-truth text. Ho and Nagy [44] presented an OCR with no shape training, where isolated character images were clustered based on the metric distance, then labeled according to a corpus. Their approach required the support of an available corpus and assumed extracted connected components can separate most characters. However, these two conditions are not always satisfied either because of the corpus availability or the poor quality of documents with many touching characters.

Preparing ground truth for an OCR system can also be a training sample creation procedure. The existing approaches to automatic ground-truthing divide into two categories: (1) truthing of scanned images; and (ii) degrading images with a degradation model. The approaches described in [62, 60] and [45] fall in the first category, where the synthetic images were generated from the electronic text, then printed and scanned. To perform automatic ground-truthing, ground-truthed bounding boxes of characters are matched with the scanned document images. The approach described in [138] belongs to the second category, where the generated synthetic images were degraded. Different models can be applied to degrade the synthetic images. Baird presented a brief survey of degradation models in [6].

In the approaches mentioned above, the noisy document images were generated from the synthetic images directly or indirectly. The layout obtained from the synthetic images was the same as the noisy images, and the character bounding boxes on the noisy images were known. The alignment of the synthetic image and the noisy image is performed by computing a transformation. However, if

the given images are scanned from newspapers or magazines, simulating the noise and obtaining the character bounding boxes are not trivial, especially when many touching characters exist.

We propose a novel methodology to automatic training sample creation. This approach does not require the synthetic image have the same layout as the noisy image, or even the same font. Obtaining the unknown character bounding boxes on the noisy image, i.e. the character segmentation, is controlled by the characters on a synthetic image. In this approach, graph theory is applied to optimize the segmentation. The same approach works under three different situations seen in the following sections.

4.2 Automatic Training Sample Creation Based on Electronic Text

Assuming electronic text is available, in Unicode, and has line breaks, the process of automatic training sample creation is shown in Figure 4.1. A synthetic image is generated based on the document's Unicode text, using the closest font to the original content. Given the electronic text, different approaches can generate the synthetic image [138, 128]. The Microsoft foundation class (MFC) library also provides a powerful Unicode package, which makes it convenient for the user to deal with the input and output of Unicode. We use the MFC library to generate the synthetic images using a specific font, size, and weight. Although the font is user-specified given the content of the real document, it can be identified automatically with methods proposed in Chapter 2, or [91], [137]. Using the MFC, the layout and bounding boxes of all components (zone, text-lines, words and characters) on the

synthetic image are obtained easily, so a hierarchical structure forms with these four levels.

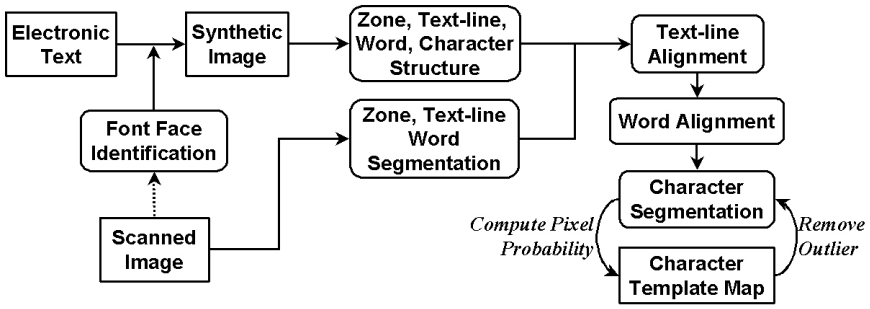


Figure 4.1: The flowchart to generate ground-truth automatically.

4.2.1 Preprocessing of the Scanned Documents

The scanned document image is deskewed, and words are extracted, both using approaches described in Chapter 2. Using average word height and width obtained from each page, the separated components (such as the dot of 'i' or 'j', and accents) merge to form characters. Words with determined boundaries then organize first into text-lines and further into zones. For each organized text-line, the 'meanline' and 'baseline' used for content alignment are detected using the horizontal projection of the text-line image (shown in Figure 4.2). The 'meanline' and 'baseline' of each word are refined based on the horizontal projection of the word image, and the *x-height* H_x (the distance between the 'meanline' and 'baseline', shown in Figure 4.2) can perform normalization during the matching. At the end of this stage, a hierarchical structure with three levels (zone, text-line and word) appears.

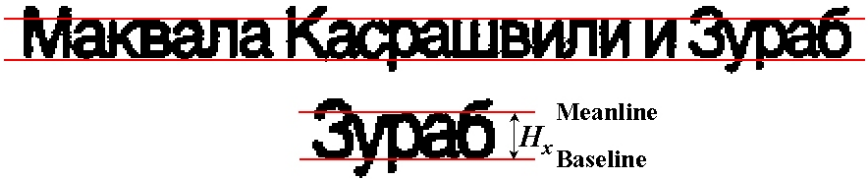


Figure 4.2: The meanline, baseline, and x-height detection.

4.2.2 Alignment of Text-lines and Words

The hierarchical structures obtained above are matched by performing text-line alignment and word alignment. Each text-line image is first normalized to the same *x-height*. The matching procedure consists of the following two steps:

(1) *Coarse matching*. The coarse matching applies to the vertical projection. As shown in the Figure 4.3, spacing in the text-line image is removed before computing the vertical projection profile. The profile with fewer values then elongate by simple linear interpolation to have the same length L as the other. Given two profiles $p_1(x)$ and $p_2(x)$, the mismatch degree D_m of the two profiles is evaluated using the following metrics:

$$D_m(p_1, p_2) = \frac{1}{L} \sum_{x=1}^{L} |p_1(x) - p_2(x)| \tag{4.1}$$

Two text-lines with D_m less than a predefined value (2 in our experiment) are considered a coarsely matched pair.

(2) *Fine matching*. The coarsely matched pair is verified by comparing two text-line images. After removing spaces and normalizing these two images to the same *x-height*, the smaller image is resized to equal the other one. Denoted by $f_1(x, y)$ and $f_2(x, y)$, the similarity $S(f_1, f_2)$ of two text-line images is measured by a Hamming

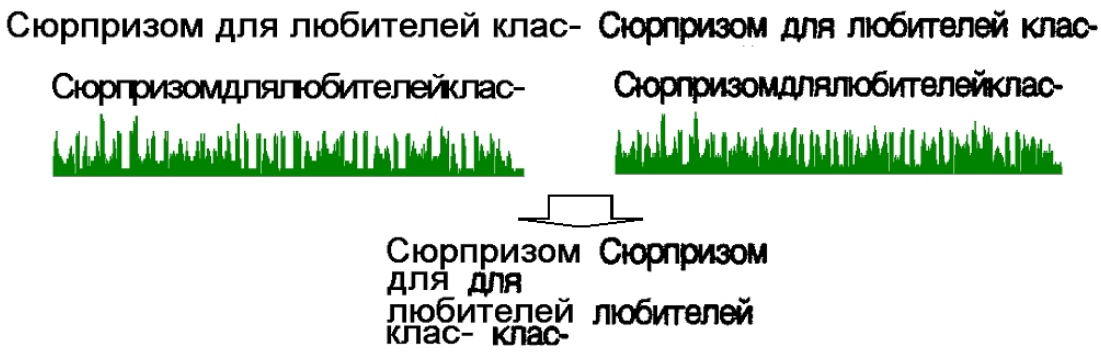


Figure 4.3: Aligned text-line examples.

distance and computed as follows:

$$S(f_1, f_2) = 1.0 - \frac{1}{WH} \sum_{x=1}^{W} \sum_{y=1}^{H} |f_1(x, y) - f_2(x, y)| \tag{4.2}$$

where W and H are the image width and height, respectively.

Word alignment is performed on the words in each matched pair found in the text-line alignment. Since the numbers of words in the two matched text-lines may differ, merging and splitting words may occur on the noisy document text-line image. Before checking the match, the spacing within each word is removed, and the smaller image is resized to equal the other one again. The similarity of two word images is computed with Eq. 4.2. Figure 4.3 shows an example of obtaining word alignment from text-line alignment.

4.2.3 Automatic Character Segmentation

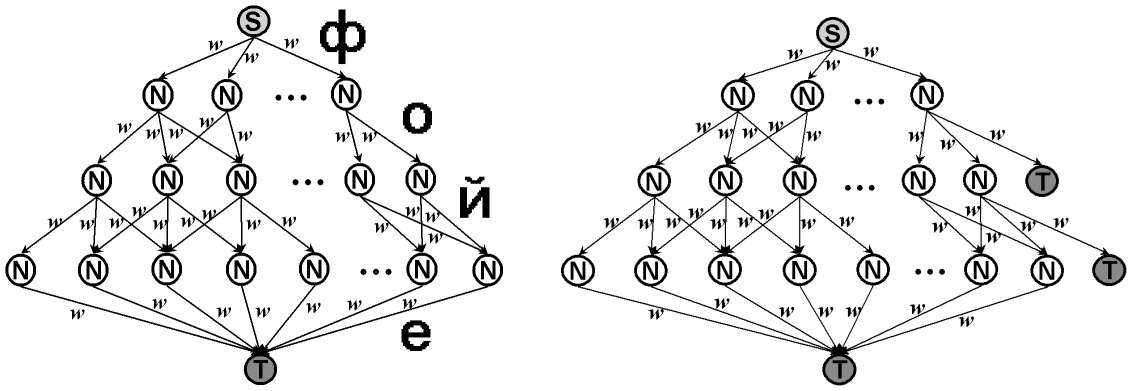
The alignment procedure cannot guarantee a match for each word on the noisy image. For words with a match, the number of characters and the content are known. The character segmentation is controlled by the matched ideal word image. For those words without match, the number of characters and the word content are

unknown, and the segmentation cannot be easily controlled. Thus, the segmentation procedures must be different. Since most of the words will be segmented by the controlled segmentation procedure, we can obtain reliable template maps for all the characters, and use them to control the unmatched character segmentation.

To find the optimal character segmentation of a word, a weighted directed acyclic graph (WDAG) representing all the possible character segmentations is created as follows:

- A vertex represents a potential segmentation position.
- The source vertex represents the left word boundary.
- The target vertex represents the right word boundary.
- An edge between two vertices represents one character candidate.
- The weight of an edge is the complement of character similarity.

Traversed vertices of the shortest path from the source to the target are the optimal segmentation positions. Therefore, finding the optimal character segmentation converts to a problem of finding a WDAG's shortest path. The controlled character segmentation (for the matched word) and the uncontrolled character segmentation (for the word with no match) using the WDAG are described in the following two subsections.



(a) (b)

Figure 4.4: The created weighted directed acyclic graph (WDAG) for optimized character segmentation. (a) Controlled character segmentation. (b) Uncontrolled character segmentation.

Controlled Character Segmentation

As shown in Figure 4.4(a), in the WDAG created for the word with a match, one source vertex (labeled S) and only one target vertex (labeled T) exist. The level of edges is fixed and equal to the number of characters in that word. The synthetic word image controls the possible segmentation position of the word. Suppose the character to be extracted has an aspect ratio r on the synthetic image, then for each left position p_l, the possible right position p_r of this character is in the range $[p_l + h \cdot (r - \epsilon), p_l + h \cdot (r + \epsilon)]$, where h is the height of the word and ϵ is a predefined shift of aspect ratio. The value of ϵ has little importance and only affects the number of possible positions. However, a larger ϵ means a larger graph and will slow the segmentation optimization. For each candidate between one pair of $[p_l, p_r]$, the image is first resized to equal the controlling synthetic character image's size. Then the similarity Sim of the resized image and the controlling image is computed with

Eq. 4.2, and the weight of the edge between these two vertices is assigned $1.0 - Sim$. The same procedure is repeated until the last character in a given word is complete. The created WDAG has $N + 1$ levels, where N is the number of characters in this word, and no edge exists between vertices in two nonadjacent levels. As an example, the Figure 4.4(a) shows a five-level WDAG created for a four-character word, where the character image shown along each right edge level is the controlling synthetic character image.

After the WDAG is created, the Dijkstra algorithm [30] finds the shortest path from source vertex S to the target vertex T, and the traversed vertices of this shortest path are the optimal segmentation positions.

After performing the controlled character segmentation on all the words with a match, the same character instances (characters with the same Unicode value) are normalized and combined to generate a template map TM using the following formula:

$$TM(i,j) = \sum_{k=1}^{N_{inst}} g_k(i,j), \quad 1 \leq i,j \leq N \tag{4.3}$$

where N_{inst} is the number of the character instances, and $N \times N$ is the normalization size of the images. As shown in the Figure 4.1, the generation of the template map is iterative. If the similarity of one instance and the existing template is too low, the instance is considered the outlier and creates a new template. The similarity is computed using the formula describe in the following subsection.

Uncontrolled Character Segmentation

For words without match found on the synthetic image, we have no knowledge about characters contained in the word, even the number of characters that should be segmented from the word. Thus, the level of the WDAG is undetermined and the edge weights must be calculated using a different approach. Since most of the words are segmented using the "controlled character segmentation" in Section 4.2.3 and the same characters combine to create template maps, the edge weights of the WDAG can then be computed based on these templates, as described later.

By denoting the minimum and maximum average aspect ratio of all the templates as r_{min} and r_{max}, respectively, and given a left position p_l of a possible character, the possible right position p_r of this character satisfies the following condition:

$$h \cdot r_{min} \leq (p_r - p_l + 1) \leq h \cdot r_{max}$$

where h is the word height. After deciding the right position p_r, $p_r + 1$ is considered the next possible left position. The above procedure is iterated until the right boundary of the word falls in the range $[h \cdot r_{min}, h \cdot r_{max}]$. Each p_r that is the right word boundary is labeled a target vertex T. Obviously, the WDAG created using the above procedure also has one source vertex S, but has more than one target vertex T in different levels. To compute the edge weight between one pair of $[p_l, p_r]$, we first compute the similarity of the character image and all templates. The similarity between the normalized image $f(x, y)$ and a template $g(x, y)$ can be computed as:

$$Sim(f, g) = 1.0 - \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} w(x, y) |f(x, y) - g_b(x, y)| \tag{4.4}$$

where $g_b(x, y)$ is the binary image converted from $g(x, y)$ by a simple thresholding, and

$$w(x,y) = \begin{cases} 1.0 & \text{if } g_b(x,y) \text{ is background} \\ g(x,y)/N_{inst} & \text{if } g_b(x,y) \text{ is foreground} \end{cases} \tag{4.5}$$

The weight w of the edge between p_l and p_r is then assigned $1.0 - Sim_{max}$, where

$$Sim_{max} = \max_{\forall g}(Sim(f, g)) \tag{4.6}$$

The Dijkstra algorithm [30] finds the shortest path from source S to all targets T, and the optimal segmentation SEG_{opt} is decided as follows:

$$SEG_{opt} = \underset{SEG \in \Lambda}{argmin}(\frac{L_s}{N_c}) \tag{4.7}$$

where Λ represents all possible segmentations, L_s is the shortest path, and N_c is the number of characters in this segmentation.

4.2.4 Segmentation Experimental Results

The proposed approach was first applied to a collection of scanned Cyrillic newspaper and magazine pages. Because of the scanning quality, most of the characters touch on images. The experimental results show this approach is effective and efficient, however, quantitative evaluation of the performance is difficult. We chose six pages and counted the segmentation errors manually; the results are shown in the Table 4.1. The results were compared with a commercial software Capture Development System 12 (CDS12) from ScanSoft. Since it is troublesome to count the segmentation errors with a high error rate, we cannot provide a quantitative evaluation of the CDS12 segmentation results, but only show the comparison result in

Figure 4.5. For the presented approach, most segmentation errors occur in the title fields. A significant difference between the title fonts and regular content or having the title overlay a graphic background made both the controlled and uncontrolled segmentation challenging.

Table 4.1: Segmentation error rate for the Cyrillic documents.

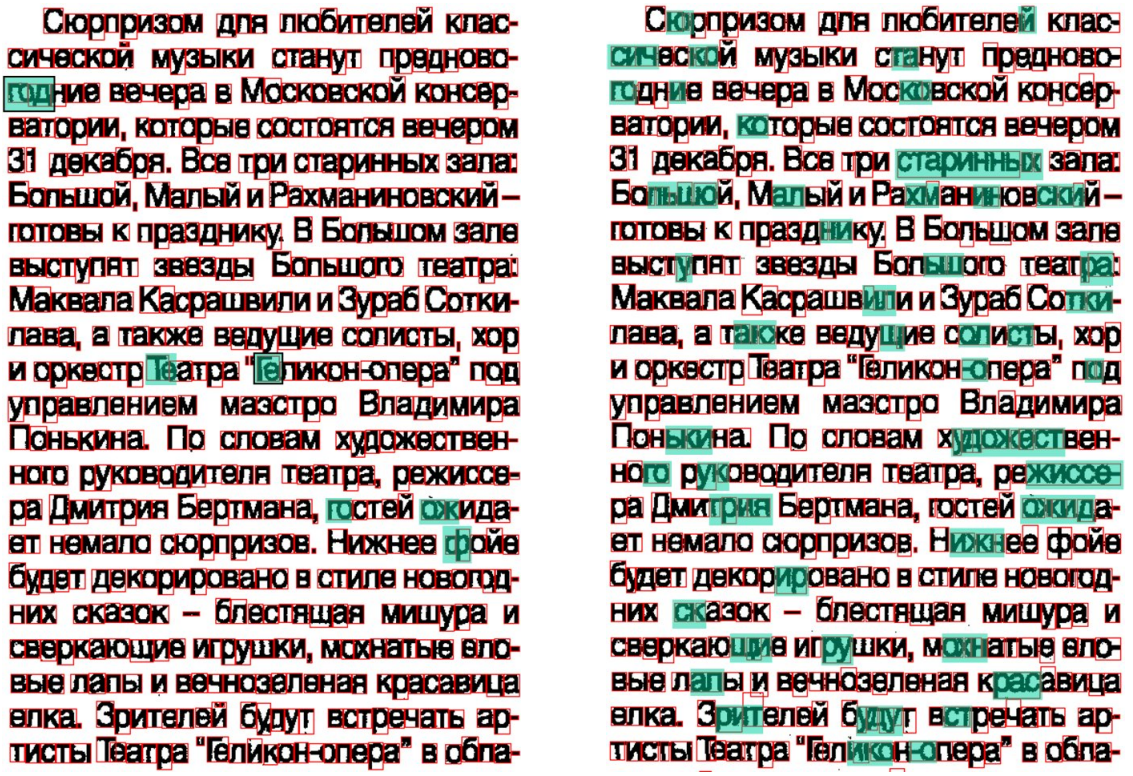
	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**Average**
Error Rate (%)	0.86	1.26	1.13	1.10	0.80	1.43	1.10

The same approach was also applied to 40 pages of documents in the University of Washington English Document Database I. The pages were selected to have many touching characters. Figure 4.6 shows the sorted error distribution of the 40 pages compared to the commercial software Capture Development System 12 (CDS12) from ScanSoft. It is obvious that the average performance of the presented approach is higher than CDS12.

By browsing pages with more errors, we found the following factors caused the higher error rate:

- **Italic words.** We assume characters can be segmented using a vertical line. However, italic words contain significant character kerning, which do no satisfy the assumed conditions.

- **Page warping.** Photocopying caused page warping, where straight text-lines overlap. Text-lines with significant overlapping affect first the text-line matching and further the word matching.



(a) (b)

Figure 4.5: The segmentation result comparison of the proposed approach and CDS12 (errors are marked using gray boxes). (a) The result of the proposed approach. (b) The result of CDS12.

- **Embedded formula.** Many professional journals are embedded heavily with mathematic formulas. Since embedded formulas often have a special format and alignment, the symbol segmentation of the mathematic formula also requires special handling.

- **Different font.** We assume page content has a single font, and the synthetic image was generated with that font. The assumption is not strictly satisfied for real documents because different fonts often represent a variety of content

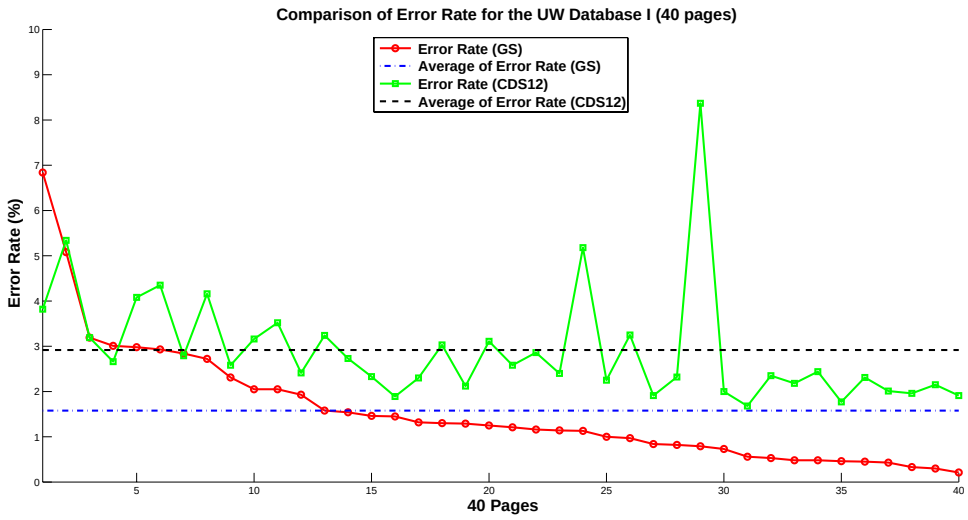


Figure 4.6: Error rate for the UW database I (GS: The proposed graph based segmentation. CDS12: Capture Development System 12.).

fields. In the experiment, given the font of the major text is correct, content with different fonts can still be matched correctly. Most errors involving font occur when a special font has extensive character kerning.

- **Reading order.** We did not assume explicitly the content reading order. However, in the text-line matching step, matched lines with higher values were considered landmarks that broke the contents into several parts, with further matching operations applied within these parts. If the detected reading order is not the same as the electronic text, this results in many unmatched text-lines and words. The unmatched words are segmented using uncontrolled segmentation, reducing the accuracy.

4.2.5 OCR Experimental Results

Trained by five pages' automatic segmentation results, an OCR was tested on 20 pages of a Cebuano-English and transliterated Iraqi Arabic-English dictionaries. In the OCR, the Nearest-Neighbor classifier applied to two sets of features, template pixels and Zernike moments, was used for recognition. Using the template pixels as features, the classification was based on a weighted Hamming distance computed with formula 4.4.

The second set of features were Zernike moments. Moment descriptors have been studied for image recognition and computer vision since 1960s [125]. Teague [124] first introduced the use of Zernike moments to overcome the shortcomings of information redundancy present in the popular geometric moments. Zernike moments are a class of orthogonal moments which are rotation invariant and can be easily constructed to an arbitrary order. And it was shown in [68, 67] that Zernike moments are effective for the optical character recognition (OCR).

The Zernike polynomials are a set of complex, orthogonal polynomials defined over the interior of a unit circle $x^2 + y^2 = 1$. The form of these polynomials is:

$$V_{nm}(x, y) = R_{nm}(x, y)exp(jm \cdot tan^{-1}\frac{y}{x})$$

where n is a non-negative integer, m is an integer such that $n - |m|$ is even and $|m| \leq n$, and $R_{nm}(x, y)$ is the radial polynomial defined as:

$$R_{nm}(x, y) = \sum_{s=0}^{n-|m|/2} (-1)^s \frac{(n-s)!}{s!(\frac{n+|m|}{2} - s)!(\frac{n-|m|}{2} - s)!}(x^2 + y^2)^{n/2-s}$$

Zernike moments are the projection of the image function onto these orthogonal basis functions. For a digital image $f(x, y)$, the Zernike moments of order n with

repetition m are defined as:

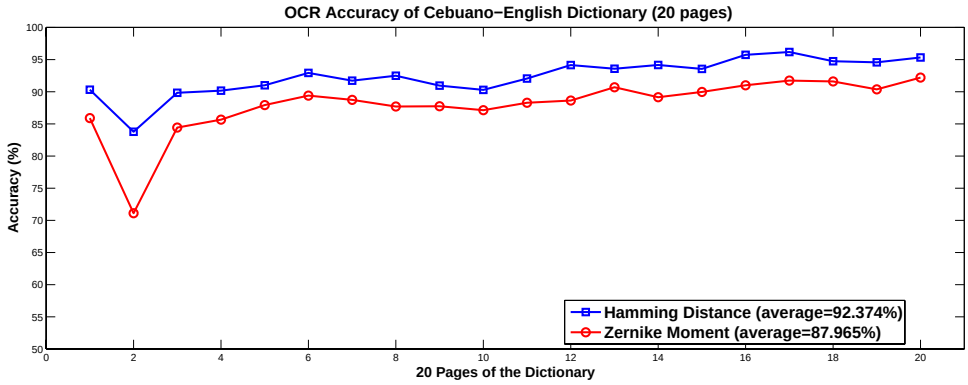
$$A_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x,y)[V_{nm}(x,y)]^*, \quad x^2 + y^2 \leq 1$$

It has been shown in [68, 67] that the magnitudes of Zernike moments are rotation invariant, and $A_{n,-m} = A_{nm}$, thus we only use $|A_{nm}|$ for features. To make the extracted Zernike moments be scaling and translation invariant, each input character is normalized to a predefined dimension and the center of the unit circle is moved to the centroid of the input image.

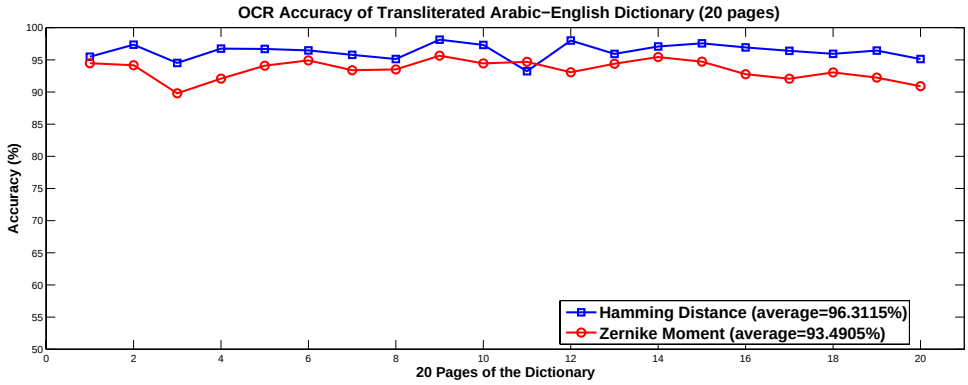
Figures 4.7(a) and 4.7(b) show the results for the above-mentioned dictionaries. Accurach higher than 92% for both dictionaries demonstrates that the presented approach can be easily retargeted to create a new OCR system for a totally new collection of documents.

4.3 Automatic Training Sample Creation with Limited User Feedback

In the previous section, we assume availability of the scanned documents' electronic text. However, the electronic text is not always available. In this section, we propose a new framework to design an adaptive OCR system. Given scanned document images, the adaptability lies in the automatic training sample extraction and clustering with limited user interaction. Clustered samples are labeled by the user and used to optimize automatic character segmentation. This approach does not require the support of the ground truth text and a corpus, which is essential for the processing of noisy document images where many characters touch each other. The system is shown in Figure 4.8 and we focus only on the extraction of training



(a)



(b)

Figure 4.7: OCR evaluation on two bilingual dictionaries. (a) Cebuano-English dictionary. (b) Transliterated Arabic-English dictionary

samples.

The scanned document is processed as described in section 4.2.1. As shown in Figure 4.8, the training part consists of three steps: (i) Template initialization; (ii) Iterative template refinement; and (iii) Template combination and labeling.

4.3.1 Template Initialization

The extraction of templates from an image without any prior knowledge begins with an initial set. For each segmented word, we extract the connected components

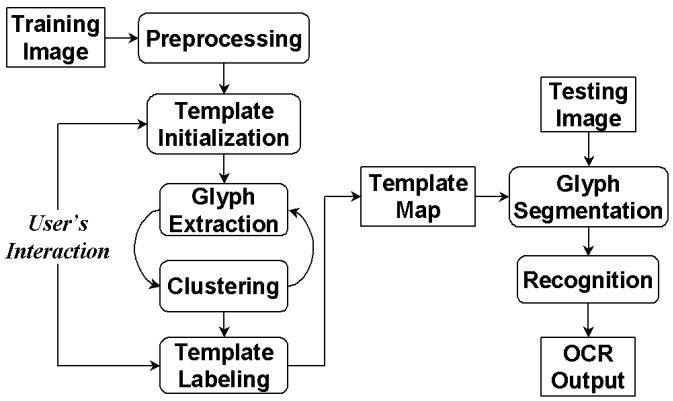


Figure 4.8: Flow chart of the proposed adaptive OCR system

(a glyph) of this word, in which accents or separated dots are merged. With the assumption that a regular character is neither too wide nor too narrow, a connected component satisfying the following conditions is considered a template candidate:

(1) The aspect ratio falls in the range $[r_{low}, r_{high}]$;

(2) The area is larger than A_{min};

where r_{low} and r_{high} are the predefined low and high aspect ratio thresholds respectively, and A_{min} is the area threshold (We found that $r_{low} = 0.2,\ r_{high} = 1.0,\ A_{min} = 5$ is a good selection in our experiment.). Extracted candidates are indexed and normalized into images with size $N \times N$ ($N = 32$). A symmetric similarity matrix $\mathcal{M}$ with dimension $K \times K$ is then computed, where K is the number of candidates. The value of the element in cell (i, j) of $\mathcal{M}$ is defined as the similarity between the ith and the jth candidate. The similarity between two normalized images $f_1(x, y)$ and $f_2(x, y)$ is measured by a Hamming distance and computed as:

$$S(f_1, f_1) = 1.0 - \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} |f_1(x, y) - f_2(x, y)|$$

Both $f_1(x, y)$ and $f_2(x, y)$ are binary images, therefore S has a value range $[0.0, 1.0]$. The matrix $\mathcal{M}$ can be described with a complete weighted graph, where each vertex represents a candidate and the weight of edge between vertex 'i' and 'j' has value $\mathcal{M}_{ij}$. Based on a predefined threshold S_{th}, the complete graph can be converted to a disconnected graph consisting of several subgraphs by removing edges with weight smaller than S_{th}. The extracted candidates are then clustered by extracting the subgraphs which are the cliques of the disconnected graph. Working on the converted disconnected graph, the clustering is described as follows:

---

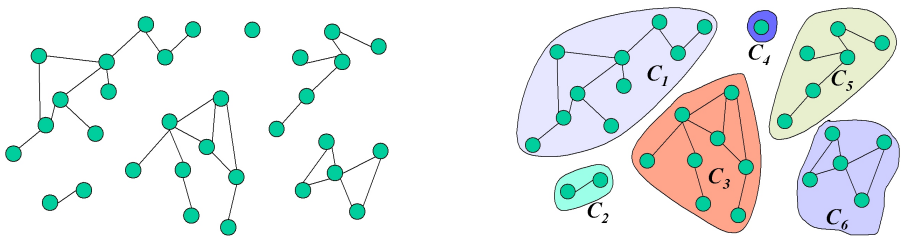
1. **for** each vertex g_i
2. **for** each cluster C_k
3. **for** each vertex g_j in cluster C_k
4. **if** vertices g_i and g_j are connected, **then**
5. put g_i in C_k and exit for loops 2 and 3
6. **if** g_i does not belong to any C_k, **then**
7. create a new cluster C_{k+1} and put g_i inside

Figure 4.9(a) shows an example of the disconnected graph, and Figure 4.9(b) shows the clustering result in which 32 vertices are clustered into six clusters. To insure the initial complete graph converts into a disconnected graph, the threshold S_{th} should be a value less than but approaching 1.0 (0.9 in our experiments).

After the clustering, all glyphs in the same cluster C_k are combined to generate a template map TM_k using the following formula:

$$TM_k(i, j) = \sum_{g \in C_k} g(i, j), \quad 1 \leq i, j \leq N$$

The total number of glyphs in each cluster is denoted N_{inst} which is useful for the computation of the weighted similarity in the following sections. Statistics of each



(a) (b)

Figure 4.9: Illustration of the clustering.

template, such as the aspect ratio, the adjusted aspect ratio (the ratio of the width and the x-height), and the connectivity, are noted, which help the normalization, segmentation, and recognition of characters.

4.3.2 Iterative Template Refinement

When the initial templates are clustered, we traverse all the word images and extract the matched parts. As illustrated in Figure 4.10(a), the matching procedure consists of the following two steps:

(1) *Coarse matching based on the vertical projection profile.* Before computing the vertical projection profile, the word image and the template are normalized to have the same *x-height*. The template projection is considered a window and slid along the word projection. At each position x of the word projection, we evaluate the mismatch degree as:

$$D_m(x) = \frac{1}{W} \sum_{i=1}^{W} |p_t(i) - p_w(x+i)|$$

where W is the normalized template width, and $p_t(i)$ and $p_w(i)$ are the projection values of the template and the word at the position i respectively. A

coarse match at position x is found if $D_m(x)$ is smaller than a predefined threshold D_{th} (2 in our experiments). As an example, we found two coarse matching positions for the template **a** in the Figure 4.10(a).

(2) *Fine matching based on similarity.* The obtained components in the first step are further evaluated by computing their similarity with the template, and those with similarities higher than a predefined threshold are the final matches. The character candidate image is binary, while the pixel values of the template map $g(x, y)$ are in a range $[0, N_{inst}]$, therefore $g(x, y)$ is first converted into a binary image $g_b(x, y)$ by a simple thresholding. The similarity of a character image $f(x, y)$ and a template $g(x, y)$ is redefined as a weighted similarity which has the following form:

$$S_w(f, g) = 1.0 - \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} w(x, y)|f(x, y) - g_b(x, y)|$$

where the weight $w(x, y)$ is defined as follows:

$$w(x, y) = \begin{cases} 1.0 & \text{if } g_b(x, y) \text{ is background} \\ g(x, y)/N_{inst} & \text{if } g_b(x, y) \text{ is foreground} \end{cases}$$

(a) (b)

Figure 4.10: The procedure of iterative template refinement.

Most of the glyphs that can be matched to one of the templates are extracted, and the templates are updated based on these newly extracted matches. Obviously

only characters in the template set can be extracted. However, for each word, the newly extracted components may leave another isolated component which can be added to the template set. We then check the remaining part of each word, if one part satisfies the two conditions described in Section 4.3.1, it is considered a new template candidate. The same clustering procedure as described in Section 4.3.1 is performed, and new template maps are generated. As shown in Figure 4.10(b), we can extract two new template candidates **N** and **K** from the current words after the match. The new templates can be used to find more matches, with additional templates generated by the same procedure. This process is iterated until no new template forms.

Typically, after the above processing, most words are segmented. The remaining components of each word are either wide characters not in the template maps or noisy characters. In the last step, we pair any two remaining components by considering each remaining component as a sequence, and extract the common component using a longest common subsequence (LCS) algorithm [43, 48]. All template maps can be generated using the LCS algorithm, in theory. However, complexity of the algorithm ($O((r+n)logn)$ and large number of words of each page makes applying LCS algorithm to all words impractical.

4.3.3 Template Combination and Labeling

Theoretically, the generated templates for the same character do not need to be combined if they are labeled correctly. However, a large number of templates will significantly slow the segmentation and recognition procedure. In addition,

non-character components can be extracted by the LCS algorithm and added to the templates. In this step, we first perform template combination and pruning, then label each template with a correct Unicode value. The procedure to perform template combination is: with a similarity threshold S_{tth} defined, we compute the similarity $S_t(g_1, g_2)$ of two template $g_1(x, y)$ and $g_2(x, y)$. If $S_t(g_1, g_2) \geq S_{tth}$, then the two templates combine to generate a new template, and all statistics update accordingly. To make the similarity value of two templates fall in the range [0.0,1.0], the similarity $S_t(g_1, g_2)$ is measured by a weighted Hamming distance which is computed using the following formula:

$$S_t(g_1, g_2) = 1.0 - \frac{1}{N^2} \sum_{x=1}^{N} \sum_{y=1}^{N} \left| \frac{g_1(x, y)}{N_{inst1}} - \frac{g_2(x, y)}{N_{inst2}} \right|$$

where N_{inst1} and N_{inst2} are the instance number of glyphs that generate these two templates.

The post-processed templates are saved into images, the user is required to prune the templates by browsing and assigning Unicode values to them. As training samples, these templates can then be used to perform character segmentation and recognition. The detailed process to perform segmentation based on these templates is described in "uncontrolled character segmentation" of Section 4.2.

4.3.4 Experimental Results

The proposed approach was first applied to a collection of Cyrillic documents scanned from magazines and newspapers. In the scanned documents, most characters touch each other (shown in Figure 4.11). Comparing the proposed approach

with a commercial software CDS12, the results (both segmentation and recognition) are shown in Table 4.2. The ground truth used for evaluation is obtained by the alignment of synthetic images and the scanned images described in Section 4.2 and [92].

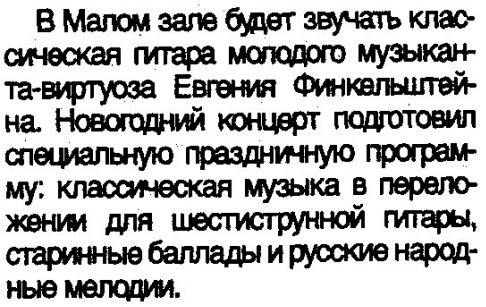

В Малом зале будет звучать классическая гитара молодого музыканта-виртуоза Евгения Финкельштейна. Новогодний концерт подготовил специальную праздничную программу: классическая музыка в переложении для шестиструнной гитары, старинные баллады и русские народные мелодии.

Figure 4.11: Part of the scanned Cyrillic document image.

Table 4.2: Segmentation and OCR result comparison (ADP: The proposed approach; CDS12: Capture Development System 12.).

Operation	Approach	P1	P2	P3	P4	P5	P6
Segmentation	**ADP**	97.33	97.61	97.39	98.09	98.01	96.38
	CDS12	82.44	87.06	81.88	90.17	95.35	84.06
OCR	**ADP**	75.38	76.15	79.85	82.52	80.57	77.71
	CDS12	79.38	71.76	26.44	75.47	25.21	56.05

The same approach was also applied to 40 pages of documents in the University of Washington (UW) English Document Database I, where pages with many touching characters were selected. The sorted segmentation and OCR results of our system versus CDS12 are shown in the Figure 4.12.

The experimental results demonstrate that for a specific collection of docu-

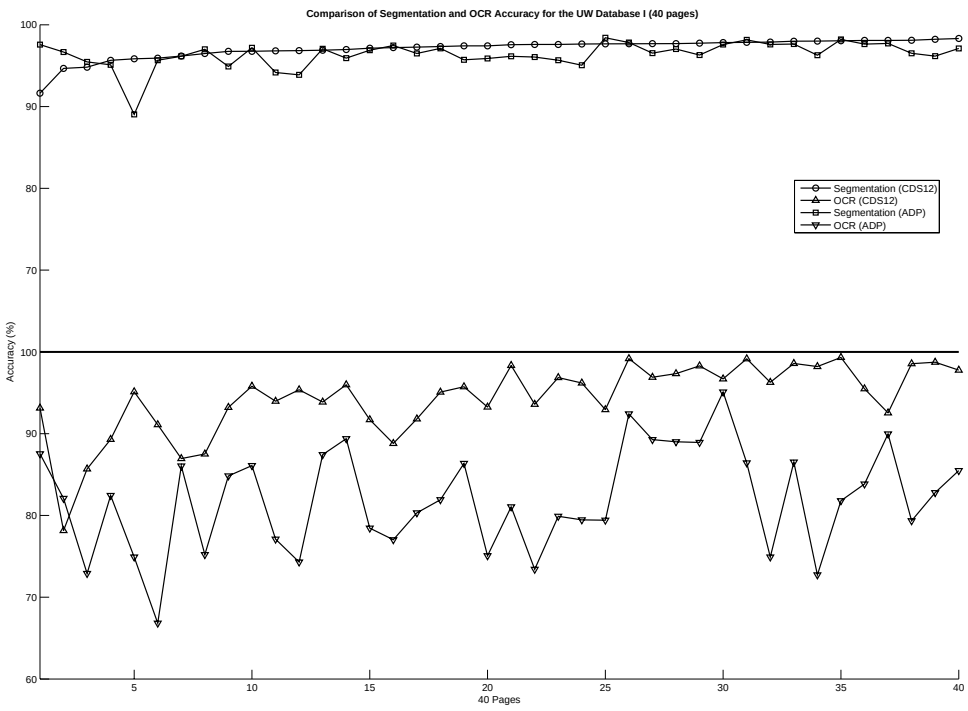


Figure 4.12: Segmentation and OCR result comparison with CDS12.

ments with many touching characters (for example, more than 90% of characters in the Cyrillic document are touching characters), the proposed method can extract training samples efficiently to retrain the system, improving the performance. For documents with fewer touching characters (the UW documents contain about 20% touching characters), the proposed approach can provide comparable segmentation result, but the OCR accuracy is lower than the commercial software. Considering good segmentation results, the simple template matching approach can be replaced with other classification algorithm to improve performance. By examining the results, we also found that the following factors affect the performance:

- **Isolated noise.** Some noise could satisfy the initial template conditions and be considered a template. This should be removed with the user's help.

- **Wide characters.** Wide characters, such as 'm' and 'w', are often over-segmented, so post-processing is necessary to improve performance.

- **Broken characters.** Broken characters may appear as noise. So if a character cannot be extracted as a connected component, the user may remove the broken part.

- **Disconnected characters.** If one character is disconnected, it must be handled specifically and carefully.

- **Character kerning.** We assume characters in a word can be segmented using the vertical lines, so the kerning of some specific characters or italic words makes it difficult to segment vertically.

- **Incorrect LCS.** The extracted LCS may contain more than one character, which requires the user's action to examine the results.

- **Template combination.** Templates might combine different but similar character templates (such as 'e', 'c' and 'o'), so combinations must be performed carefully.

4.4 Automatic Training Sample Creation for Languages with Shadowed Characters

Although the two approaches of automatic training sample creation described above obtained satisfying results, they are based on an implicit assumption that vertical segmentation can separate adjacent characters. However, many Asian scripts,

such as Arabic, Khmer and Devanagari, can contain many shadowed characters and particular attention is required to segment them. Besides the shadowed character case shown in Figures 3.11 and 3.12, Figure 4.13 shows some shadowed characters for Khmer and Arabic script. In this chapter, we use the Khmer script to demonstrate how to create training samples for languages with shadowed characters.

(a) (b)

(c) (d)

Figure 4.13: Shadowed characters and character segmentation for Khmer and Arabic script. (a) Shadowed Khmer characters. (b) Character segmentation of word in (a). (c) Shadowed Arabic characters; (d) Character segmentation of word in (c).

4.4.1 Introduction of Khmer

Khmer is written in the Cambodia and Vietnamese languages, and Tampuan and Krung. Khmer has about 33 consonants, 12 independent vowels, and 16 dependent vowel signs [3], which are shown in Figures 4.14, 4.15, and 4.16.

The following are some particularities of Khmer script writing:

(1) It is written from left to right, with characters being placed also above and below the main line of writing.

1	ក	kâ	6	ច	châ	11	ដ	dâ	16	ត	tâ	21	ប	bâ	26	យ	yô	30	ស	sâ
2	ខ	khâ	7	ឆ	chhâ	12	ឋ	thâ	17	ថ	thâ	22	ផ	phâ	27	រ	rô	31	ហ	hâ
3	គ	kô	8	ជ	chô	13	ឌ	dô	18	ទ	tô	23	ព	pô	28	ល	lô	32	ឡ	lâ
4	ឃ	khô	9	ឈ	chhô	14	ឍ	thô	19	ធ	thô	24	ភ	phô	29	វ	vô	33	អ	’â
5	ង	ngô	10	ញ	nhô	15	ណ	nâ	20	ន	nô	25	ម	mô						

Figure 4.14: Khmer consonants.

1	ឥ	ĕ	3	ឧ	ŏ,ŭ	5	ឫ	rœ̆	7	ឭ	lœ̆	9	ឯ	ê	11	ឱ	aô
2	ឦ	ei	4	ឩ	âu	6	ឬ	rœ	8	ឮ	lœ	10	ឰ	ai	12	ឳ	au

Figure 4.15: Khmer independent vowels.

1	◌ា	a—éa	5	◌ឺ	œ	9	◌ើ	aeu—eu	13	◌ែ	ê				
2	◌ិ	ĕ—ĭ	6	◌ុ	ŏ—ŭ	10	◌ឿ	œă	14	◌ៃ	ai—ey				
3	◌ី	ei—i	7	◌ូ	o—u	11	◌ៀ	iĕ	15	◌ោ	aô—oŭ				
4	◌ឹ	ŏ	8	◌ួ	uŏ	12	◌េ	é	16	◌ៅ	au—ŏu				

Figure 4.16: Khmer dependent vowels.

1	◌្ក	k	6	◌្ច	ch	11	◌្ដ	d	16	◌្ត	t	21	◌្ប	b	26	◌្យ	y	30	◌្ស	s
2	◌្ខ	kh	7	◌្ឆ	chh	12	◌្ឌ	th	17	◌្ថ	th	22	◌្ផ	ph	27	◌្រ	r	31	◌្ហ	h
3	◌្គ	k	8	◌្ជ	ch	13	◌្ឍ	d	18	◌្ធ	t	23	◌្ព	p	28	◌្ល	l	33	◌្អ	’
4	◌្ឃ	kh	9	◌្ឈ	chh	14	◌្ណ	th	19	◌្ទ	th	24	◌្ភ	ph	29	◌្វ	v			
5	◌្ង	ng	10	◌្ញ	nh	15	◌្ណ	n	20	◌្ន	n	25	◌្ម	m						

Figure 4.17: Khmer subscript consonant.

◌ំ ◌ះ ◌ៈ ◌ៗ ◌ៈ

Figure 4.18: Khmer diacritics.

(2) Words are not separated by spaces. A space in Khmer is a punctuation sign similar to a comma.

(3) A word is composed of clusters, sometimes also called syllemes. They are not a proper syllable, as syllables are a unit of consonants and vowels pronounced in one breath. Consonants pronounced after a vowel are part of the syllable, but not part of the cluster or sylleme.

(4) Consonants have two forms: (i) Normal form written in the main line of text (shown in Figure 4.14); (ii) Subscript form, placed under another consonant and read after the normal consonant, without any vowel sound between them (shown in Figure 4.17).

(5) Other signs, called independent vowels, behave like consonants (even if many of them have vowels sounds), and for typographical purposes they can be considered almost as consonants.

(6) Each vowel has a specific location in reference to the main consonant of the sylleme.

(7) Some vowels have two graphs, which have to be placed before and above the consonant, below and after, below and above, or before and after the consonant (shown in Figure 4.16).

(8) Some diacritical signs (shown in Figure 4.18) exist for part of syllemes. In special cases, one of these diacritical signs may change shape and location, depending on the vowels that follow it.

4.4.2 Controlled Khmer Character Segmentation

Compared with section 4.2, the main operations to obtain the Khmer character bounding boxes from a scanned image also progress from text line alignment to word alignment, and then from word alignment to character alignment. It also differs as follows:

(1) By considering space as word separator.

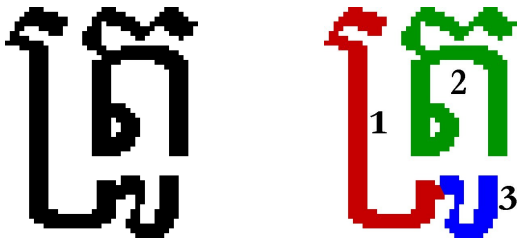
(2) By optimizing segmentation at the sylleme level.

(3) By post-processing each extracted sylleme, and extracting superscript and subscript if necessary.

The post-processing of extracted sylleme is based on the following definition in theory [118]:

> *A Khmer sylleme is always composed of: (1) A normal consonant or independent vowel (one and only one); (2) At most two subscript consonants or independent vowels; (3) At most two diacritic signs; (4) At most one vowel.*

Some shadow characters, and the superscript and subscript consonants may make vertical segmentation impossible, so we design a "pixel picking" mask for each sylleme to separate shadowed characters and superscript and subscript consonants as well. Figure 4.19 shows the "pixel picking" mask of a Khmer sylleme with shadow character and subscript consonant. The procedure to separate characters based on the mask is as follows:



(a) (b)

Figure 4.19: How to segment shadow characters of Khmer. (a) A sylleme containing shadow characters. (b) The "pixel picking" mask of each character (each filled cell means to pick the pixel of this position and labeled with the index of the character).

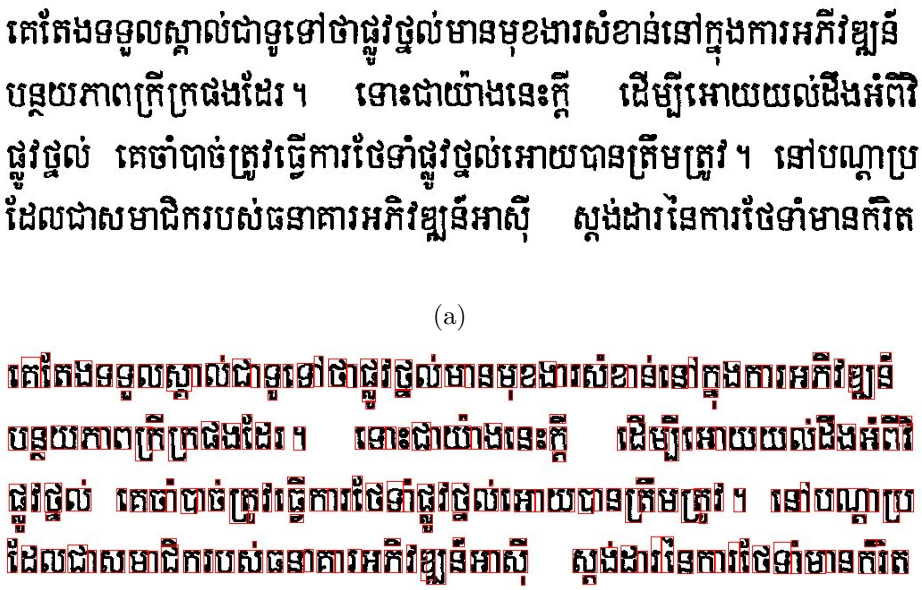
(1) Normalize the mask to the same size as the matched sylleme on the scanned document.

(2) Pick the pixels covered by the mask.

(3) For each remaining pixel not covered by any mask, find its nearest neighbor among the covered pixels and assign the same label.

4.4.3 Khmer OCR Results

The proposed approach was applied to a set of real Khmer documents. Figure 4.20 shows the Khmer character segmentation results of a scanned document. Using two pages of scanned documents as training pages (each page contains about 1,500 characters), part of the generated template maps are shown in Figure 4.21. The OCR results using the Hamming distance and Zernike moments are found in Table 4.3. The performance with average accuracy 84% does not look very satisfying. However, the evaluation is based on results without any post-processing.



(a)

(b)

Figure 4.20: Example of controlled character segmentation of Khmer document. (a) The original scanned document; (b) The segmented characters.

Table 4.3: Khmer document OCR results.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Average
Hamming	82.68	83.78	82.58	82.15	79.99	81.40	83.00	81.55	82.47	80.24	81.98
Zernike	84.96	84.54	84.40	83.44	83.11	84.03	84.43	83.47	85.18	82.06	83.96

Similar to the Devanagari script described in Chapter 3, the phenomenon of difference between reading order and coding order also exists for Khmer script. We strongly believe after correcting the reading order based on the language gramar, the OCR results can be improved significantly.

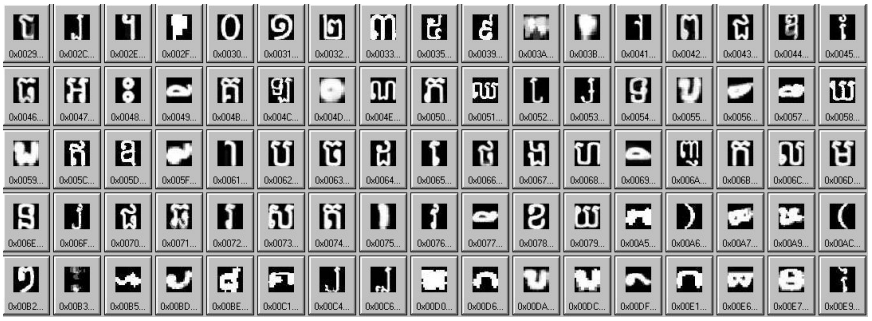


Figure 4.21: Part of the generated template maps.

4.5 Generalization of the Approach

Although the presented approach in the section 4.4 is described specifically for the Khmer script, the idea can be generalized to process all scripts with shadow characters. When vertical segmentation cannot be applied to the character level, the script's controlled segmentation must have specific handling. However, almost all script content can be organized into units to which vertical segmentation can be applied. For the Khmer script, this unit is the sylleme. Therefore, the automatic training sample creation occurs as follows:

(1) Organize the electronic content into units to which vertical segmentation is applicable.

(2) Obtain the "pixel picking" mask when generating a synthetic image.

(3) Segment the scanned document into units defined by (1), using the graph theory.

(4) Separate possible shadow characters, using the "pixel picking" mask.

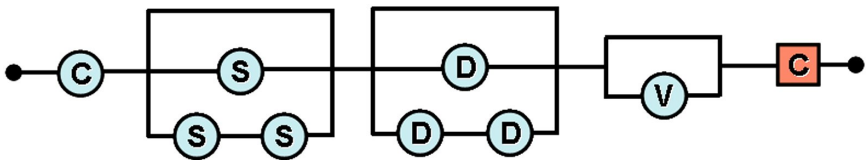


Figure 4.22: The pattern recognition machine for Khmer script.

Pattern matching can organize the content into units. The unit is described using a *regular expression*, and the organization of the content is carried out by pattern matching through a pattern recognition machine. For Khmer script, each unit is a sylleme. Based on the definition, a sylleme can be described using the following *regular expression*:

$$\mathbf{C(1+S+SS)(1+D+DD)(1+V)}$$

where:

- C is a consonant or independent vowel
- S is a subscript consonant or independent vowel
- D is a diacritic sign
- V is a vowel

The pattern recognition machine to recognize this unit (a pattern) from a string is constructed as shown in Figure 4.22. Performing efficient pattern matching after the pattern recognition construction falls outside this thesis's range, but can be referred to [72, 63].

Due to the large variety of different scripts, besides the generalization of the above approach, the following factors may require specific handling:

(1) New characters generated because of ligature. For instance, Devanagari script described in the previous chapter, has a large amount of new characters generated from ligature. Some of them can be segmented into two or more parts, but most of them are impossible to segment, so these characters must be recognized as a single character.

(2) Difference between reading order and coding order. The reading order and coding order of both Devanagari and Khmer script are different, which means the output codes are different from what displayed on the image. Once this happens, the recognition results must be post-processed based on some grammars of the language.

(3) Characters containing two or more connected components. Connected component extraction is a good way to segment characters. Usually, each extracted connected component represents a single character. However, almost all scripts, such as Devanagari, Khmer, Arabic, Chinese, and Latin, have several characters that contain two or more connected components. During segmentation and recognition, these characters must be handled specifically.

(4) Different forms of the same character. In some script, such as Arabic, each character has different forms depending on the location of which in the word. These characters must be handled specifically based on their locations, which often make the segmentation and recognition more challenging.

Chapter 5

Adaptive Logical and Semantic Segmentation

5.1 Introduction

The process of document layout analysis can be divided into two tasks: physical segmentation and logical analysis. Physical segmentation usually divides a page into zones with specific physical characteristics. Logical analysis labels each extracted zone with a specific function or logic label. The structural complexity of different documents makes it difficult to design a generic document analysis tool that can be applied universally. Furthermore, since logical analysis is often based on the physical segmentation result, the performance of the physical segmentation module is crucial for understanding the document image, and it dominates the results.

Dictionaries are members of a class of documents that are designed for easy search [32]. Their structure is typically regular and repeating, and "keys" are distinguished as access points for each entry. The format varies from simple word-to-phrase translation pairs through full descriptions that contain parts of speech, related forms, and examples of usage. Our goal is to capture the salient structure of these entries and label each element appropriately. Because of the regular structure, we are typically able to provide a relatively small number of training samples for each dictionary, and then have the system learn the features necessary for correct segmentation and labeling.

We present an approach that combines physical and logical segmentation, and can segment pages with repeating structures by learning the physical and semantic features that characterize the functionality of unique entries. A bootstrap technique, when applied to the generation of training data, improves the accuracy of training and segmentation. Before describing our page segmentation approach in details, we provide a brief literature survey in the following section.

5.2 Related Work

In most document analysis systems, pages are segmented first into different levels of entities based on physical features. With journal articles, for example, the page can be represented with a hierarchical structure of zones, text-lines, words, and characters. The segmentation is performed on physical features such as spacing, relative position, and text-line attributes. After obtaining the physical segmentation result, logical analysis is applied to the highest level – zones. For journal articles, zones can be classified as title, author, abstract, body, and references. Work relevant to the page segmentation is typically found in the logical layout analysis literature ([79] gives an overview of traditional methods). In our system, it is essential that we are to learn a dictionary's structure, since that structure is typically consistent throughout a given dictionary but varies widely between dictionaries.

Liang et al. [83] presented a probability-based text-line identification and segmentation approach. Their approach consisted of two phases: an offline statistical training and online text line segmentation. In the online text line segmentation phase, an iterative, relaxation-like method found an optimal partition of source

entities by improving a conditional probability. Kopec and Chou [75] applied a stochastic approach to build Markov source models for text line segmentation under the assumption that a symbol template was given and the zone (or text columns) had been previously extracted. Given the physical layout structures of document images modeled by a stochastic regular grammar, Kanungo and Mao [96, 61] used a generative stochastic document model for a Chinese-English dictionary page. A weighted finite state automaton modeling the projection profile at each level of the document physical layout tree segmented the dictionary page at all levels. Lee and Ryu [81] proposed a parameter-free method to segment document images with various font sizes, text line spacing, and document layout structures. Also some segmentation methods performed the segmentation based on rules that were either manually set up by the user [96, 79] or learned automatically by training [94, 111, 75].

5.3 Bootstrapping Logical and Semantical Segmentation

The structured page segmentation problem we present here addresses the general problem of identifying repeating structures by learning the physical and semantic features that characterize them. Unlike traditional page segmentation problems where zones are characterized by spatial proximity, we often find that publishers of documents with multiple entries (dictionaries, phone books, and other lists) use different font properties (bold, italics, size etc) and layout features (indentation, bullets, etc.) to indicate a new entry. Although such characteristics vary for different documents, they are often consistent within a single document, and hence the task suggests learning techniques. Furthermore, these entries may occur in a single

physical zone which traditional document analysis approach did not deal with. For dictionaries, one classified entry may extend across several columns or zones. Some zones may be ignored (i.e. classified as noise) because they have no interest for logical labeling (for example, a page number, header, or footer). In a dictionary, our functional segmentation essentially inserts a new level (an entry) between the zone and text line of the typical hierarchical representation. The segmentation is based not only on the structural features of the page, but also on the structural features of the entry. Therefore, this problem can be viewed as a combination of physical and logical segmentation because (1) pages are first segmented into physical zones; (2) one physical zone can be functionally segmented further into multiple entries; and (3) extracted entries are classified into different logical types.

A significant contribution to the effectiveness of entry segmentation results from application of a bootstrap technique for the generation of new training samples. Bootstrapping helps make the segmentation adaptive and improves the segmentation performance. We start with OCR results that include text size, font, face, text line, and text zone information. The goal of entry segmentation is to segment each page into multiple (sometimes partial) entries or alternatively to organize multiple lines of text as a single entry. Since the extraction of text lines in dictionaries is relatively straightforward, the problem of entry segmentation can be posed as the problem of finding the first (or last) line of each entry. The segmentation procedure is iterative and illustrated in Figure 5.1, and each iteration consists of the following three steps:

(1) Feature extraction: The segmentation system is trained automatically using a

small set of labeled samples (5-10 entries), and entry features are extracted.

(2) Segmentation: Pages are segmented based on the extracted features.

(3) Correction and bootstrapping: The segmented results are fed back to the user, who can make corrections to the small subset with errors. Based on the corrected segmentation results, bootstrapping samples are generated and used to retrain the system.

To warrant the training, we concentrate only on documents with a significant number of pages. Correction and training require an operator who knows the document structure.

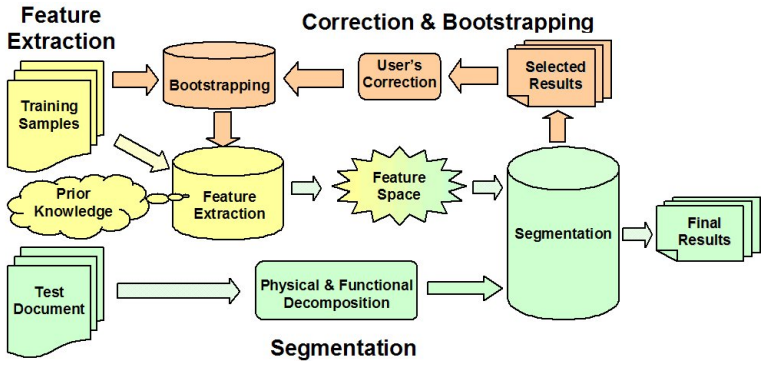


Figure 5.1: Diagram of the page segmentation approach

5.3.1 Feature Extraction

Based on a study of different types of structured documents, a feature pool containing all possibly useful features is created. Parameters listed as follows have been shown to be useful for the segmentation. The training module will select and use a subset of these features. Examples are shown in Figure 5.2.

- *Special symbols*: Special symbols such as punctuation, numbers, and other non-alphabet symbols are often used to start a new entry, end an entry, or mark the continuation of a text-line.

- *Word font, face, and size*: Word font, face, and size (especially the features of the first word in each entry) are often important entry features. In a dictionary page, for example, the first word of each entry (typically the headword) can appear bold, all capital letters, in a different font, or larger than the rest of the entry.

- *Word patterns*: Words often form distinguishable patterns which can be used to describe the entry structure consistency.

- *Symbol patterns*: Combined with other symbols or regular characters, special symbols can form consistent patterns to represent the beginning or ending of an entry.

- *Line structures*: The indent, spacing, length, or height of text-lines in an entry can be contained in the line structures to represent the entry features.

- *Other features*: Other features can also be used, such as spacing between adjacent entries, the position of text, script type, word spacing, and character case.

During the training (feature extraction) phase, each extracted feature is assigned a probability. Based on estimated probabilities, each feature is assigned a

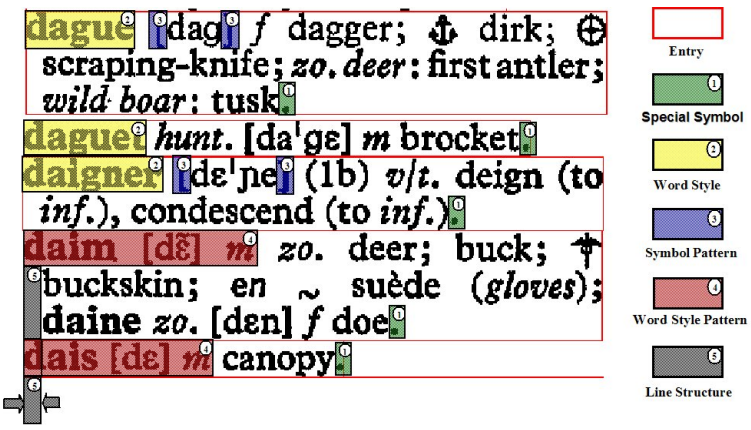


Figure 5.2: Some useful features for segmenation.

weight to compute the entry score from all extracted features. The detailed procedure is as follows:

(1) Count the occurrence of different features in training samples;

(2) Compute the feature occurrence rate as the feature probability. Suppose a total of N training entries and K extracted features, then for feature i ($1 \leq i \leq K$), the probability can be computed as: $p_i = \frac{K_i}{N}$, where K_i is the number of occurrences of feature i.

(3) Assign feature weights based on the following computed probability:

$w_i = \frac{p_i}{A} \times 100$, where, $A = \sum_{i=1}^{K} p_i$ and $1 \leq i \leq K$

(4) Considering the extracted features as a formed feature space, each entry is projected to this space and a voting score is computed as follows:

$FV = \sum_{i=1}^{K} w_i S_i$ where $S_i = 1$ if the feature i occurs, otherwise $S_i = 0$

(5) Obtain the minimum, maximum, and average voting scores of entries; these

values will be used as thresholds in the segmentation stage.

5.3.2 Segmentation

The segmentation is an iterative procedure that maximizes the feature voting score of an entry in the feature space. Based on the extracted features, a document can be segmented into entries by searching for the beginning and ending text lines of an entry. This search operates like a threshold-based iterative procedure, and the threshold can be estimated from the training set. Considering a relatively small number of text lines on one page, this search can be accomplished with brute-force search. The approach is iterative, the training set can be generated by bootstrapping, and initial segmentation result is refined step by step. The segmentation procedure is described as:

(1) Search for candidates from the first text line in one zone by feature matching. This operation is equivalent to determining if the first line in one zone is the beginning of a new entry or a continuation of an entry in the previous zone or previous page.

(2) Search for the entry end. This operation can be replaced with the searching of the next entry's beginning, since the beginning of one entry is the end of the previous entry.

(3) Remove the extracted entries, and iterate until all new entries are identified.

Once we obtain the initial segmentation results, the results are traversed and, if necessary, two simple operations (splitting, merging) are applied. It should be noted

that some features (such as spacing between entries and text line indent) are so important and reliable that they must be given special attention. In our approach, during the procedure of feature score computation, we consider entry spacing and text line indent as rules if they appear as features. If one of the rule is broken, a punishment (a negative score) is applied to the feature score. In this way, entry spacing and text line indent are treated as the most important features and thus have dominant influence on the final segmentation result. Details of this approach was published in [31].

5.3.3 Correction and Bootstrapping

Due to many structured documents' complexity, it is difficult to determine the optimal value of some parameters. We attempt to learn as much as possible about the features of the given training set. In our approach, a new training set is generated from the original set and selected new segmentation results. This technique is the bootstrap technique described in section 2.1.7.

Bootstrap samples can be generated from the original training samples, from the new segmentation results, or from a combination of both. Considering the situation where the original training set is a small set, we always generate bootstrap samples from the combined set of original training samples and selected segmentation results.

Before combining the segmentation results with original training samples to generate bootstrap samples, the operator corrects the original segmentation results by performing one or more of the following operations:

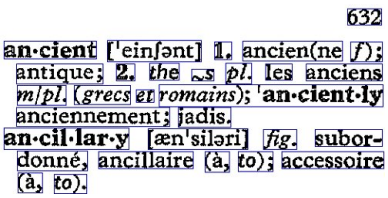
- Splitting: divide one segmented entry into two or more individual entries
- Merging: combine two or more adjacent entries into one single entry
- Resizing: change the size of a segmented entry
- Moving: change the bounding box position of a segmented entry
- Removing: erase a segmented entry
- Relabeling: change the type label of an entry

Let $X_{N_i} = \left\{x_1^i, x_2^i, ..., x_{N_i}^i\right\}$ be a set extracted from the set of original training samples and new selected segmentation results for entry type i, where x_j^i $(1 \leq j \leq N_i)$ are the feature vectors with each vector element the probability of the specific feature entity. We generate a bootstrap sample set $X_{N_i}^B = \left\{x_1^{b_i}, x_2^{b_i}, ..., x_{N_i}^{b_i}\right\}$ with size Ni from the original set X_{N_i}. The procedure to generate bootstrap samples described in section 2.1.7 is applied here.

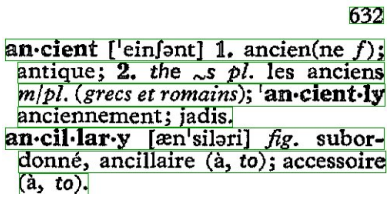
We assume the document has consistent functional structure, but some features may occur in only one entry type. For example, the line indent feature will not appear in a single line entry, and the special ending symbol may not appear in an entry that continues on the next page. So, we generate the bootstrap samples for each predefined entry type.

5.4 Experimental Results

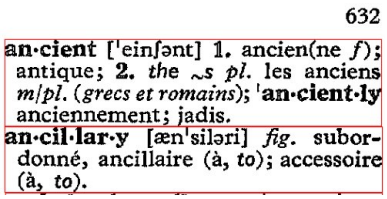
We have applied this approach to the segmentation of three categories of structured documents: (i) dictionaries; (ii) voice transcriptions; and (iii) phone books.



(a) Word



(b) Text line



(c) Entry (page number is noise)

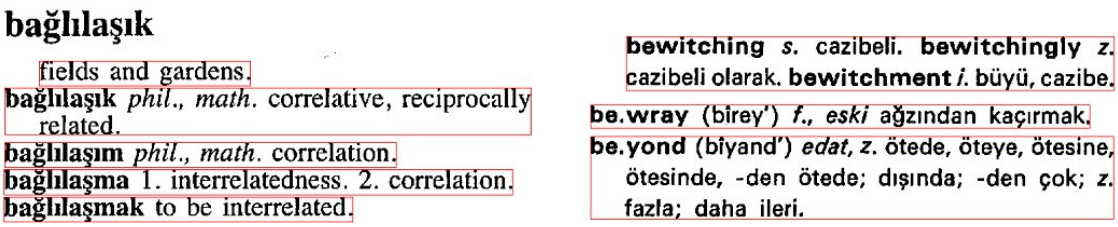
Figure 5.3: English-French dictionary segmentation results.

The experimental results are shown as follows.

5.4.1 Dictionary Segmentation Results

The segmentation approach was applied to five dictionaries with different structural features: French-English dictionary (613 pages), English-French dictionary (657 pages), Turkish-English dictionary (909 pages), English-Turkish dictionary (1152 pages) and Cebuano-English dictionary (1163 pages). The French-English and the English-French pages are taken from the same bilingual dictionary, so they have the same features. Figures 5.3 and 5.4 show the segmentation results of some of these dictionaries, and Table 5.1 gives the evaluation results.

Figure 5.5 shows the performance improvement for the segmentation of the first four dictionaries after bootstrapping. The evaluation results from the statistical information of 50 pages of each dictionary. The initial segmentation used four training entries. Iterations following the initial segmentation were based on adding



(a) (b)

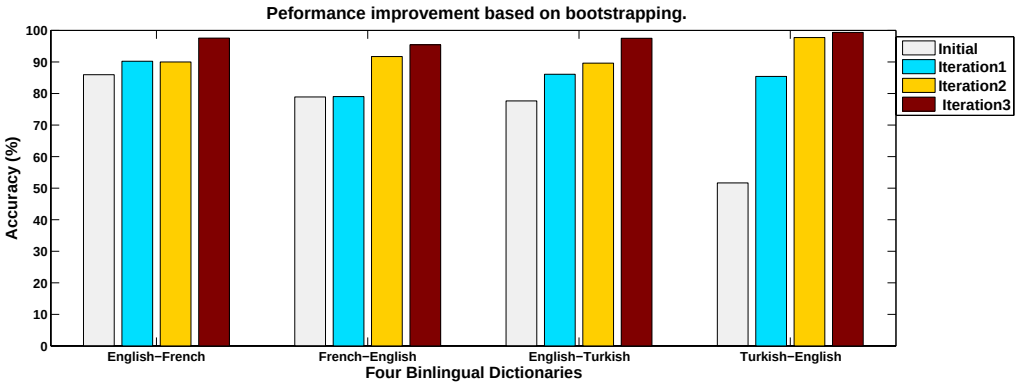
Figure 5.4: Segmentation of Turkish dictionaries. (a) Turkish-English dictionary segmentation results (with many single-line entries). (b) English-Turkish dictionary segmentation results (different entry features).

Table 5.1: Segmentation results of five dictionaries.

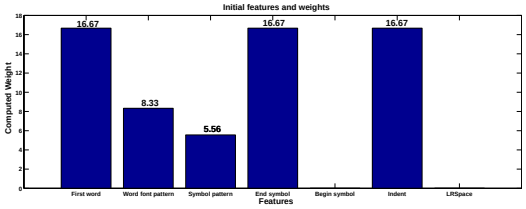
Document	Page No	Total Entries	Correct Entries	Incorrect Entries	False Alarm	Mislabeled Entries
EnglishFrench	635	20174	**96.11%**	3.89%	0.21%	0.80%
FrenchEnglish	75	2423	**97.90%**	2.10%	0.25%	0.49%
EnglishTurkish	96	3517	**99.26%**	0.74%	0.23%	0.31%
TurkishEnglish	70	2654	**98.98%**	1.02%	0.08%	0.38%
CebuanoEnglish	50	2152	**99.21%**	0.79%	0.00%	4.46%

different numbers of training entries used to generate bootstrap samples. The chart in Figure 5.5(a) demonstrates the segmentation can be refined step by step by applying the bootstrap technique. Figures 5.5(b) and 5.5(c) show the extracted features and assigned weights in the initial step and after bootstrapping respectively. It can be seen the weights changed after bootstrapping.

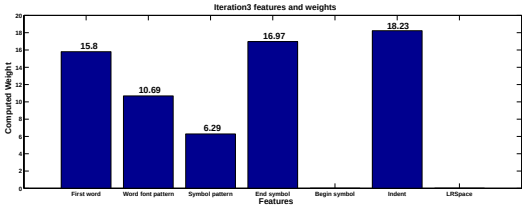
The evaluation of results is shown in Table 5.1. Obtaining ground truth on a large data set is very time-consuming, so the evaluation is based only on the available ground truth of these dictionaries.



(a) Accuracy Improvement



(b) Initial Features and Weights



(c) Features and Weights after Bootstrapping

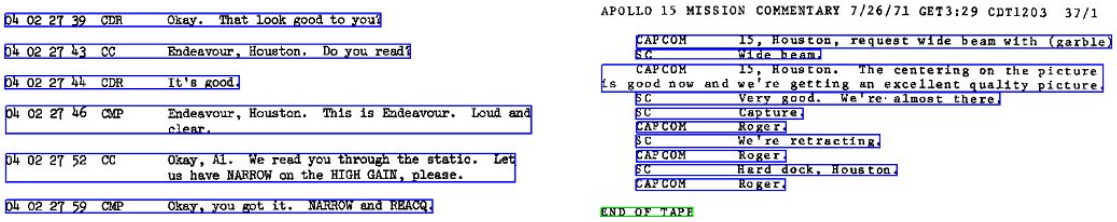
Figure 5.5: Progressive performance improvement based on bootstrapping (four dictionaries)

5.4.2 APOLLO 15 Voice Transcription Segmentation Results

For the dictionary parsing problem, we wish to segment the dictionaries into entries that can be tagged and used as lexical resources. Typed transcriptions of audio content provide a related challenge. We are currently integrating into an audio retrieval interface the audio, video and photographs from the Apollo 15 mission. The audio and scanned images of typed transcripts are from the Lunar Module (LM), the Command Module (CM), and mission control. We aim at synchronizing the images of the transcriptions to the audio. First, we segment the transcript images into spoken units and label the times, sources, and spoken text regions.

While this text is not complicated, unparameterized segmentations will not be as accurate as a modeled segmentation. Figure 5.6 shows the segmentation results of the transcriptions. These transcriptions contain five parts (around 3400 pages), and each part has different structural features. Table 5.2 shows the evaluation results of the segmentation based only on available ground truths.

Compared with the segmentation results of five dictionaries, these transcription documents have relatively simple structures and more obvious structural features. The segmentation results are more accurate than the dictionary segmentation results, with accuracy from 98.07% to 99.87%. Due to physical and logical noise (uninterested entry), the "false alarm" and "mislabeling" errors are significantly higher than the dictionary results. The highest "false alarm" error rate is 8.37% (vs 0.25% for dictionaries), and the highest "mislabeling" error rate is 2.75% (vs 0.49% for dictionaries).



(a) (b)

Figure 5.6: Segmentation results of voice transcription ((a) and (b) have different features).

Table 5.2: Segmentation results of transcripts.

Document	Page No	Total Entries	Correct Entries	Incorrect Entries	False Alarm	Mislabeled Entries
AS15_CM	92	2112	**99.20%**	0.80%	3.65%	0.23%
AS15_LM	94	1969	**98.07%**	1.93%	3.71%	0.48%
AS15_PAO	101	1123	**99.20%**	0.80%	8.37%	1.38%
AS15_PAC	96	1540	**99.22%**	0.78%	0.19%	2.75%
AS15_TEC	93	1513	**99.87%**	0.13%	0.00%	0.59%

5.4.3 Other Structured Document

In addition to dictionaries and transcriptions, we also tested the approach's robustness by applying it to a phone book, with results shown in Figure 5.7. The last text-line is ignored as noise (uninterested part).

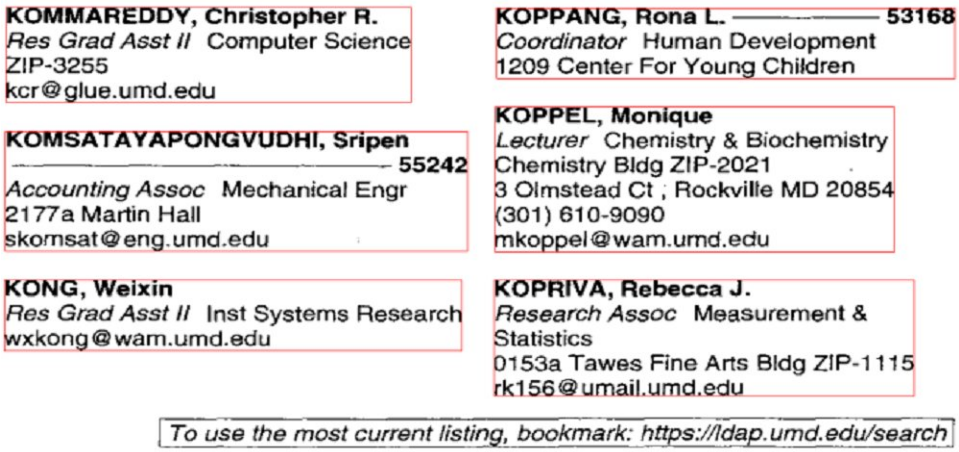


Figure 5.7: Segmentation of contact information list.

5.5 Specific Handling of Documents with Tables

The approach presented above can handle most documents with different structures. However, documents with table contents require specific handling because they often contain more information which should be extracted. Basically, besides the horizontal segmentation which can be conducted using the above approach, each segmented entry requires a further vertical segmentation, which generally segment the content in one entry to table cells by detecting table columns. Although table content may have a large varieties of layout, spacing and vertical line are usually the only possible column separators. So column detection is usually equivalent to the detection of spacing or long vertical lines. Figure 5.8 shows examples of two different table layouts, which use spacing and vertical line as separators, respectively.

5.5.1 Detection of Column Separators

The detection of the column separator of table is based on the following two assumptions: (1) Column number is known; (2) Columns don't have overlapping on all pages; (3) A table header exists either on the first page or on every page. Suppose there are N columns on every page, and the page size is $W \times H$, where W is width and H is height, the detection procedure is described as follows:

(1) Extract the sub-image in bounding box $[l\ t\ r\ b] = [\frac{W}{10}\ \frac{H}{6}\ \frac{9W}{10}\ \frac{5H}{6}]$, where l, t, r, and b are the *left*, *top*, *right*, and *bottom* coordinates, respectively.

(2) Extract connected components from the sub-image and consider the narrow and high components as vertical line candidates.

Page 4

Meeting Number		Recommended Spelling or New Name or Version accepted by Cabinet	Old Name/s Spelling	District	Degree Square
3		Bojanamane	Boja Namane	Central	2227
6	*	Bokaa	Bokaa	Kgatleng	2426
12		Bokspits	Boks Pits	Kgalagadi	2620
17		Bokwi	Bokwi (Island)	Ngamiland	1923
19		Bolelantokwe	Bolelantokwe	Central	2325
17		Bontse	Bontse (Pan)	Kweneng	2423
5	*	Bonwapitse	Bona Pitse	Central	2326
11		Borakalalo	Borakalalo	Kweneng	2425
16		Boritse	Beretse, Boritse (Pan)	Kweneng/ Ngwaketse	2323

(a)

NOME	ACIDENTE	LOCALIZAÇÃO			DOCUMENTO	OBSERVAÇÕES
		GEOGRÁFICA		ADMINISTRATIVA		
		LATITUDE	LONGITUDE			
Água Bambe	Ribeira	0° 16'.0 N	6° 31'.3 E	Freg. Neves	01	
Água Banco	Ribeira	0 12.5 N	6 40.0 E	Freg. Ribeira Afonso	06	
Água da Bandeira	Ribeira	0 19.2 N	6 34.7 E	Freg. Trindade	03	
Água Batatu	Ribeira	1 37.1 N	7 24.8 E	Freg. Conceição (Príncipe)	02	
Água Bôbó ou Água Picão	Ribeira	0 15.9 N	6 45.2 E	Freg. Santana	01,06	No Doc. 06 Água Bôbó, grafia não adoptada por se considerar incorrecta. No Doc. 01 designado apenas por Água Picão
Água Bôbó ou Água Sambué	Ribeira	0 19.3 N	6 44.8 E	Freg. Graça	01,04	No Documento 04 Água Bôbó, grafia não adoptada por se considerar incorrecta. No Doc. 01 designada por Água Sambué ou Bôbó. No Doc. 04 designada apenas por Água Bôbó
Água Bóia ou Água Budo ou Água Sola Ver Água Serra ou Guegue	Ribeira	0 19.1 N	6 44.8 E	Freg. Graça	01 04	
Água Bomba	Ribeira	0 14.8 N	6 38.8 E	Freg. Trindade	01,06	
Água Bonifácio	Ribeira	0 17.3 N	6 45.1 E	Freg. Santana	01,04	
Água Buanga	Ribeira	1 37.3 N	7 24.8 E	Freg. Conceição (Príncipe)	08	
Água Budo ou Água Bóia ou Água Sola Ver Água Serra ou Guegue	Ribeira	0 19.1 N	6 44.8 E	Freg. Graça	01 04	

(b)

Figure 5.8: Two tables using different column separators. (a) Spacing as separator. (b) Vertical line as separator.

(3) Select $N-1$ components from candidates by evaluating their locations and consider the selected $N-1$ components as column separators.

(4) If step 3 succeeds, then stop the procedure. Otherwise, the separators are spacing, continue the procedure.

(5) Organize the connected components extracted in the second step into words. Let the bounding box of a word be $[l\ t\ r\ b]$, a vertical projection based on each

word's box is calculated using the following formula:

$$P_{new}[i] = P_{old}[i] + (b - t + 1) \qquad i \in [l,\ r]$$

(6) Detect all spacing existing in P.

(7) Select $N - 1$ spacing locations by evaluating the spacing width and location simultaneously, and consider these spacing as column separators.

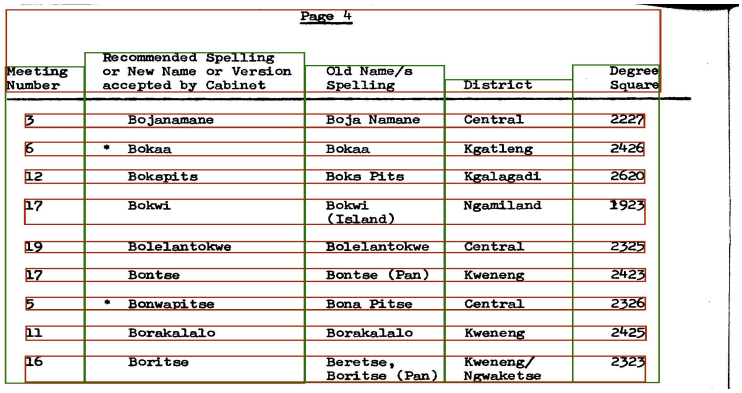
The reason that the above process was conducted on a sub-image is: because of the photocopying or scanning, the page margins often contain some noise which can affect the detection of spacing or vertical lines. By taking the sub-image, we can reduce the detection errors greatly.

5.5.2 Table Content Segmentation Results

The segmentation of entry into table cells is straightforward after obtaining the separator locations. The proposed approach was applied to two collections of table content documents. Results displayed in Figure 5.9 demonstrate the proposed approach is able to detect the table column correctly.

5.6 Summary and Analysis

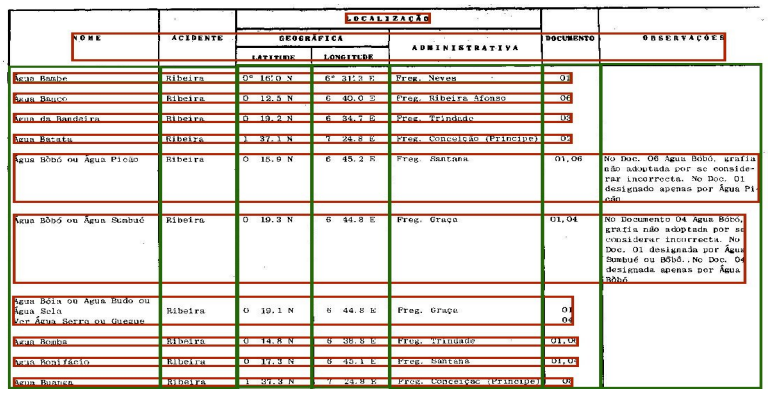
The presented approach to page segmentation uses a bootstrapping technique to learn a segmentation model. The segmentation system is first trained using a small set of samples (typically less than 10 entries) and the model is used to segment the whole set of documents. After the operator makes corrections to a selected set



Page 4

Meeting Number		Recommended Spelling or New Name or Version accepted by Cabinet	Old Name/s Spelling	District	Degree Square
3		Bojanamane	Boja Namane	Central	2227
6	*	Bokaa	Bokaa	Kgatleng	2426
12		Bokspits	Boks Pits	Kgalagadi	2620
17		Bokwi	Bokwi (Island)	Ngamiland	1923
19		Bolelantokwe	Bolelantokwe	Central	2325
17		Bontse	Bontse (Pan)	Kweneng	2423
5	*	Bonwapitse	Bona Pitse	Central	2326
11		Borakalalo	Borakalalo	Kweneng	2425
16		Boritse	Beretse, Boritse (Pan)	Kweneng/ Ngwaketse	2323

(a)



NOME	ACIDENTE	LOCALIZAÇÃO GEOGRÁFICA LATITUDE	GEOGRÁFICA LONGITUDE	ADMINISTRATIVA	DOCUMENTO	OBSERVAÇÕES
Água Bambe	Ribeira	0° 16'0 N	6° 31'3 E	Freg. Neves	07	
Água Banco	Ribeira	0 12.5 N	6 40.0 E	Freg. Ribeira Afonso	08	
Água da Bandeira	Ribeira	0 19.2 N	6 34.7 E	Freg. Trindade	03	
Água Batata	Ribeira	1 37.1 N	7 24.8 E	Freg. Conceição (Príncipe)	02	
Água Bôbó ou Água Picão	Ribeira	0 15.9 N	6 45.2 E	Freg. Santana	01,06	No Doc. 06 Água Bobó, grafia não adoptada por se considerar incorrecta. No Doc. 01 designado apenas por Água Picão
Água Bôbó ou Água Sambué	Ribeira	0 19.3 N	6 44.8 E	Freg. Graça	01,04	No Documento 04 Água Bôbó, grafia não adoptada por se considerar incorrecta. No Doc. 01 designada por Água Sambué ou Bôbó..No Doc. 04 designada apenas por Água Bôbó
Água Bóia ou Água Budo ou Água Sola ver Água Serra ou Guegue	Ribeira	0 19.1 N	6 44.8 E	Freg. Graça	01 04	
Água Bomba	Ribeira	0 14.8 N	6 38.8 E	Freg. Trindade	01,05	
Água Bonifácio	Ribeira	0 17.3 N	6 45.1 E	Freg. Santana	01,05	
Água Buanga	Ribeira	1 37.3 N	7 23.8 E	Freg. Conceição (Príncipe)	05	

(b)

Figure 5.9: Segmentation results of tables. (a) Spacing as separator. (b) Vertical line as separator.

of newly generated segmentation results, these corrected results are combined with the original training set to generate a set of bootstrap samples which are used to retrain the system. Starting with OCR results, this approach can be applied to the segmentation of any documents with repeating structure whose structure can be learned from training.

We applied this approach to many structured documents such as dictionaries and voice transcripts and obtained satisfying results. Experiment results show that

the bootstrap technique can improve the performance of segmentation even with a small set of training samples.

Many structured documents contain pictures, figures, tables, and/or other content, which makes the segmentation more difficult. We proposed an approach to segment pages with table contents by detecting column separators. Although accurate results were obtained for these document, for documents with mixed table content and regular text content, the process is still challenging. The future work can be extended to solve the segmentation of pages with these elements.

Chapter 6

Conclusions and Future Work

This thesis considered the problem of adaptive analysis and processing of structured multilingual documents. We presented a framework to extract information from multilingual documents, where adaptability was applied to every step, such as word-level script and style identification, font identification, OCR, and page segmentation. In this chapter we summarize the main results and propose new directions for future research.

6.1 Script Identification, Font Face, and Style Classification

The approaches to perform script identification, font, and style classification are conducted either at different levels (page or word) or using different features.

6.1.1 Main Results

Script Identification

We presented an approach to perform word-level script identification. Texture features were extracted using the isotropic Gabor filter bank. Four classification techniques, WED, k-NN, GMM, and SVMs were applied to perform the classification and results were compared. Bootstrapping technique was applied to the small training set to test the performance.

(1) Isotropic Gabor filter bank is able to capture the features of different scripts at the word level.

(2) Large number of training samples (19 pages) can produce better performance than a small number of training samples although the performance difference is often small.

(3) For large number of training samples, the k-NN classifier obtains the best performance, while the SVMs classifier is the most robust.

(4) For a small training set, the performance of k-NN, GMM, and SVMs is almost the same, and WED has the lowest performance.

Style Identification

We proposed an iterative approach to identify the word styles. Features used for identification were selected using a brute-force search. A Gaussian mixture model (GMM) was built for each character on every page. The word style was decided based on the voting of character styles. Already decided word style updated the styles of characters contained in that word. The GMMs were updated in each iteration. Experimental results compared with a commercial software showed the proposed approach was better performed.

Font Identification

A new texture operator, grating cell operator, was applied to extract texture features for page-level font identification. Operator parameters were selected based

on experimental results. Classification results based on features extracted using this operator and the isotropic Gabor filter were compared, and two classifiers, WED and BPNN, were applied and compared. Experimental results showed:

(1) The grating cell operator can more accurately capture texture features of different fonts. It detects only texture and does not respond to other image attributes such as edges.

(2) Traditional Gabor filter operator responds to both texture and other image attributes.

(3) Due to the characteristics of document images, strokes of characters of different fonts often create a series of gratings with different patterns, which make the grating cell operator more effective for extracting these pattern features.

(4) BPNN is more effective than WED classifier for font identification, implicating a more complicated classifier can improve the performance than the simple WED classifier.

6.1.2 Future Work

The future research directions for script identification, font and style classification are described as follows.

The script identification is a following process after document image preprocessing and word segmentation. Incorrect word segmentation, different word styles, and single-character words can affect the identification result. Therefore, removing or reducing the effect of these factors can absolutely improve the performance.

The grating cell operator has a much higher complexity than Gabor filters. Therefore, the approach can benefit from techniques reducing the computation complexity.

As a post-processor of OCR results, the word-level style identification depends on OCR performance. However, the same approach can perform as the feed-back to OCR system to improve the recognition performance. So combination with OCR should be the future research direction.

6.2 Adaptive Optical Character Recognition

In Chapter 3 and [86], we demonstrated how to design an OCR for a specific script rapidly, by the design of a Hindi OCR. Due to the large amounts of ligatures, the segmentation algorithm was designed specifically based on the characteristics of Devanagari script. And, the small training set made the generalized Hausdorff image comparison an effective recognition technique. Applying the designed OCR to a Hindi-English bilingual dictionary and ideal image converted from PDF file, the results are summarized below.

6.2.1 Main Results

(1) For scanned bilingual documents, the recognition accuracy at the character level reached 87.75% without spelling checking and correcting based on dicitionary search.

(2) For clean image, the accuracy came close to 95%, also without any post-

processing, such as error correction. The result confirmed the effectivness of the presented segmentation approach.

(3) With small training set, the generalized Hausdorff image comparison can be easily tuned to handle the large number of special Hindi characters caused by ligatures.

6.2.2 Future Work

A major thrust of future work will be to perform OCR correction or to resolve ambiguity among candidates. One advantage is we assign confidence as a side effect of recognition. The real values of the confidence make them more intuitive and usable than current commercial OCR softwares.

Given the confidence of characters and words, we can further consider the word correction based on a dictionary search. The word correction engine would determine whether a word should be replaced with another correct word from the dictionary, significantly improving the recognition performance at both character and word level.

Another advantage involves the adaption to different image qualities. When a scanned document image has a poor quality, the Hausdorff thresholds can be set to lower values, making the classifier and recognizer more tolerant. If the image has high quality, the thresholds can be set to higher values, speeding the recognition process. The setting of the thresholds can also be determined automatically based on the qualities of images.

The next recognition step is to apply new, possibly multi-classifier techniques, and combine them with the current Hausdorff classifier to provide improved performance.

6.3 Automatic Training Sample Creation for OCR

In Chapter 4, we presented an approach which automatically extracts training samples from scanned documents under different situations. When electronic text is available, the procedure is completely automatic by aligning scanned documents with synthetic images generated from electronic texts. However, limited user feedback is required when electronic text is unavailable while vertical segmentation is possible. Under all above situations, graph theory was applied to handle touching characters. Results working on different script, such as Cyrillic, Latin, and Khmer are summarized below.

6.3.1 Main Results

(1) Dijkstra algorithm is efficient and effective to optimize the segmentation of touching characters.

(2) The presented approach is able to provide segmentation results with high accuracy, which can be used as training samples of an OCR.

(3) When the training samples are ready, the application of graph theory can optimize the segmentation of touching characters. Compared with the current commercial software, this approach is extremely useful for handling documents

with a large number of touching characters.

(4) When the electronic text is unavailable but vertical segmentation is possible, the proposed approach can iteratively and automatically extract characters with limited user feedback.

(5) For some scripts, shadow characters caused by ligatures often make it impossible to conduct vertical segmentation to extract characters. Experimental result of Khmer script showed the proposed approach can still segment the character with high accuracy by applying "picking mask" technique.

6.3.2 Future Work

The degradation of documents caused by many factors, such as page warping from photocopy, character kerning from italic characters, and text line ambiguity from mathematics formula, reduced the performance. Handling of different degradation is desired in the future.

If the electronic text is unavailable, documents with significant broken characters make the extraction of training samples impractical. Techniques handling this case are required, image morphology might be a good solution.

The segmentation of shadow characters was generalized, however, specific handling still exist in the recognition phase. Generalization in the recognition phase is desired.

6.4 Adaptive Logical and Semantic Segmentation

A feature pool was created for the page segmentation. Extracted from several training samples, features were used to segment one page into entries logically and semantically. Bootstrapping technique was applied to refine the result based on user's feedback. Results are summarized as follows.

6.4.1 Main Results

(1) Useful features can be extracted from several training samples to conduct segmentation.

(2) Experimental results obtained from different document types including bilingual dictionaries, transcripts and phone-book list showed the proposed approach was effective for page segmentation.

(3) Bootstrapping technique can improve the performance progressively.

6.4.2 Future Work

This is the most stable phase of this thesis. The future work can be directed to the following directions: (1) Processing of documents with more complicated structures, such as pages with more complicated tables, forms or other types of regions. (2) The feature pool can be updated by new features learned from new documents. (3) Color documents often use different colors to represent different functions, therefore, color information should be extracted from a document before converting it into binary image to conduct OCR.

6.5 Conclusion

The main goal of the work presented in this thesis has been to contribute to structured multilingual document analysis through the development of an adaptive framework. The designed framework consists of the following major components where adaptability is applied to each part:

- A general word-level script identification approach based on global texture features.
- A general font identification approach using a new texture operator.
- An adaptive font style identification approach based on Gaussian mixture model.
- An adaptive OCR using generalized Hausdorff image comparison, demonstrated on Hindi.
- An automatic training sample creation framework under different situations.
- A bootstrapping physical and semantical page segmentation approach.

Every part of the designed framework can be extracted as an independent toolkit to perform multilingual document analysis without major modifications. It is proved by experimental results that the research herein can help extract information rapidly from multilingual documents which serve as important resource for language system such as cross language information retrieval and machine translation.

BIBLIOGRAPHY

[1] Finereaders. http://www.abbyy.com.

[2] Omnipage pro. http://www.scansoft.com.

[3] Technical papers. In *Second United Nations Conference on the Standardization of Geographical Names*, volume 2, pages 163–164, London, UK, 1972.

[4] A. Antonacopoulos. Page segmentation using the description of the background. *Computer Vision and Image Understanding*, 70(3):350–369, 1998.

[5] H. S. Baird. The skew angle of printed documents. In L. O'Gorman and R. Kasturi, editors, *Document Image Analysis*, pages 204–208. IEEE Computer Society Press, 1996.

[6] H. S. Baird. The state of the art in document image degradation modeling. In *4th IAPR Workshop on Document Analysis Systems*, pages 1–13, 2000.

[7] H. S. Baird, S. Jones, and S. Fortune. Image segmentation by shape-directed covers. In *Proc. Int'l Conf. Pattern Recognition*, pages 820–825, Atlantic City, NJ, 1990.

[8] H. S. Baird and G. Nagy. A self-correcting 100-font classifier. In *Proc. SPIE Symp. on Electronic Imaging: Science and Technology*, San Jose, CA, 1994.

[9] V. Bansal. *Integrating Knowledge Sources in Devanagari Text Recognition*. PhD thesis, Indian Institute of Technology, Kanpur, India, March 1999.

[10] V. Bansal and R. Sinha. Segmentation of touching and fused devanagari characters. *Pattern Recognition*, 35:875–893, 2002.

[11] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, England, Oxford, 1995.

[12] V. Blanz, B. Scholkopf, H. Bulthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3d models. In *International Conference on Artificial Neural Networks*, pages 251–256, Berlin, 1996.

[13] D. S. Bloomberg. Multiresolution morphological approach to document image analysis. In *1st International Conference on Document Analysis and Recognition (ICDAR)*, pages 963–971, Saint-Malo, Francs, 1991.

[14] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[15] L. Breiman. Heuristics of instability and stabilication in model selection. *The Annals of Statistics*, 24(6):2350–2383, 1996.

[16] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[17] A. Busch, W. W. Boles, and S. Sridharan. Texture for script identification. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(11):1720–1732, 2005.

[18] G. Cesarini, M. Gori, S. Marinai, and G. Soda. Structured document segmentation and representation by the modified x-y tree. In *Proc. 5th Int'l Conf. Document Analysis and Recognition*, pages 563–566, 1999.

[19] B. Chaudhuri and U. Pal. An ocr system to read two indian language scripts: Bangla and devanagari (hindi). In *Proc. 4th International Conference on Document Analysis and Recognition*, pages 1011–1016, Germany, 1997.

[20] B. Chaudhuri and U. Pal. Skew angle detection of digitized indian script documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):182–186, 1997.

[21] S. Chaudhury and R. Sheth. Trainable script identification strategies for indian languages. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 657–660, Bangalore, India, 1999.

[22] Y.-K. Chen and J.-F. Wang. Skew detection and reconstruction based on maximization of variance of transition-counts. *Pattern Recognition*, 33(2):195–208, 2000.

[23] A. Conway. Page grammars and page parsing: A syntatic approach to document layout recognition. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 761–764, Tsukuba Science City, Japan, 1993.

[24] R. Cooperman. Producing good font attribute determination using error-prone information. *The International Society for Optical Engineering Journal*, 3027:50–57, 1997.

[25] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[26] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Trans. Information Theory*, IT-13(1):21–27, 1967.

[27] J. G. Daugman. Complete discrete 2d gabor transforms by neural networks for image analysis and compression. *IEEE Trans. Acoustics, Speech and Signal Processing*, 36:1169–1179, 1988.

[28] P. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, London, 1982.

[29] D. Dhanya and A. Ramakrishnan. Script identification in printed bilingual documents. *Lecture Notes in Computer Science*, 2423:13–24, 2002.

[30] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[31] D. Doermann, H. Ma, B. Karagol-Ayan, and D. W. Oard. Lexicon acquisition from bilingual dictionaries. In *SPIE Photonic West Electronic Imaging Conference*, pages 37–48, 2002.

[32] D. Doermann, E. Rivlin, and A. Rosenfeld. The function of documents. *International Journal of Computer Vision*, 16(11):799–814, 1998.

[33] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (Second Edition).* John Wiley & Sons, Inc., New York, N.Y., 2001.

[34] B. Efron. Bootstrap methods: Another look at the jackknife. *Annual Statistics*, 7:1–26, 1979.

[35] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, 1993.

[36] F. Esposito, D. Malerba, and G. Semeraro. A knowledge-based approach to the layout analysis. In *3rd Int'l Conf. on Document Analysis and Recognition*, pages 446–471, 1995.

[37] K. Etemad, R. Chellappa, and D. Doermann. Page segmentation using wavelet packets and decision integration. In *Proc. International Conference on Pattern Recognition*, volume 2, pages 345–349, 1994.

[38] L. Fletcher and R. Kasturi. A robust algorithm for text string separation form mixed text/graphics images. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 10:910–918, 1988.

[39] B. Gatos, N. Papamarkos, and C. Chamzas. Skew detection and text line position determination in digitized documents. *Pattern Recognition*, 30(9):1505–1519, 1997.

[40] J. Ha and R. Haralick. Document page decomposition by the bounding-box projection technique. In *3rd Int'l Conf. Document Analysis and Recognition*, pages 1119–1122, Montreal, Canada, 1995.

[41] Y. Hamamoto, S. Uchimura, and S. Tomita. A bootstrap technique for nearest neighbor classifier design. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(1):73–79, 1997.

[42] R. M. Haralick. Document image understanding: Geometric and logical layout. In *Int'l Conf. on Computer Vision and Pattern Recognition*, pages 385–390, Seattle, WA, 1994.

[43] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the Association fro Computing Machinery*, 24(4):664–675, 1977.

[44] T. K. Ho and G. Nagy. Ocr with no shape training. In *15th International Conference on Pattern Recognition*, pages 27–30, Barcelona, Spain, 2000.

[45] J. D. Hobby. Matching document images with ground truth. *International Journal on Document Analysis and Recognition (IJDAR)*, 1(1):52–61, 1998.

[46] J. Hochberg, P. Kelly, T. Thomas, and L. Kerns. Automatic script identification from document images using cluster-based templates. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(2):176–181, 1997.

[47] J. J. Hull. Document image skew detection: Survey and annotated bibliography. In J. J. Hull and S. L. Taylor, editors, *Document Analysis Systems II*. Word Scientific, 1998.

[48] J. W. Hunt and T. G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[49] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[50] R. Ingold and D. Armangil. A top-down document analysis method for logical structure recognition. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 41–49, Saint-Malo, France, 1991.

[51] Y. Ishitani. Logical structure analysis of document images based on emergent computation. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 189–192, Bangalore, India, 1999.

[52] D. J. Ittner and H. S. Baird. Language-free layout analysis. In *IAPR 2nd Int'l Conf. on Document Analysis and Recognition*, pages 336–340, Tsukuba Science City, Japan, 1993.

[53] S. Jaeger, H. Ma, and D. Doermann. Identifying script on word-level with informational confidence. In *8th International Conference on Document Analysis and Recognition*, pages 416–420, Seoul, Korea, 2005.

[54] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice-Hall, Englewood Cliffs, 1989.

[55] A. K. Jain and B. Yu. Document representation and its application to page decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):294–308, 1998.

[56] A. K. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997.

[57] T. Joachims. *Advances in Kernel Methods-Support Vector Learning*, chapter Making Large-Scale SVM Learning Practical, pages 41–56. MIT-Press, 1999.

[58] J. Jones and A. Palmer. An evaluation of the two dimensional gabor filter model of simple receptive fields in cat striate cortex. *J. of Neurophysiology*, 58:1233–1258, 1987.

[59] S. Kahan, T. Pavlidis, and H. Baird. On the recognition of printed characters of any font or size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2):274–288, 1987.

[60] T. Kanungo and R. M. Haralick. An automatic closed-loop methodology for generating character groundtruth for scanned documents. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(2):179–183, 1999.

[61] T. Kanungo and S. Mao. Stochastic language models for style-directed layout analysis of document images. *IEEE Transactions on Image Processing*, 12(5):583–596, 2003.

[62] T. Kanungo and P. Resnik. The bible, truth, and multilingual ocr evaluation. In *SPIE Conference on Document Recognition and Retrieval*, pages 86–96, 1999.

[63] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. Technical Report TR-31-81, Harvard University, Cambridge, MA, 1981.

[64] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. New algorithms for skewing correction and slant removal on word-level. In *ICECS '99, 6th IEEE International Conference on Electronics, Circuits and Systems*, volume 2, pages 1159–1162, Pafos, Cyprus, 1999.

[65] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. Skew angle estimation in document processing using cohens class distributions. *Pattern Recognition Letters*,

20(11):1305–1311, 1999.

[66] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. Slant estimation algorithm for ocr system. *Pattern Recognition*, 34(12):2515–2522, 2001.

[67] A. Khotanzad and Y. H. Hong. Invariant image recognition by zernike moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):489–497, 1990.

[68] A. Khotanzad and Y. H. Hong. Rotation invariant image recognition using feature selected via a systematic method. *Pattern Recognition*, 23(10):1089–1101, 1990.

[69] S. Khoubyari and J. Hull. Font and function word identification in document recognition. *Computer Vision and Image Understanding*, 63(1):66–74, 1996.

[70] J. Kim, D. Le, and G. Thoma. Automated labeling in document images. In *Proc. SPIE Conf. Document Recognition and Retrieval VIII*, pages 111–122, San Jose, CA, 2001.

[71] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, 70:370–382, 1998.

[72] D. Knuth, J. Morris, and V. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[73] O. Kolak and P. Resnik. Ocr error correction using a noisy channel model. In *Human Language Technology Conference (HLT 2002)*, 2002.

[74] O. Kolak, P. Resnik, and W. Byrne. A generative probabilistic ocr model for nlp applications. In *HLT-NAACL 2003*, 2003. to appear.

[75] G. E. Kopec and P. A. Chou. Document image decoding using markov source models. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 16(6):602–617, 1994.

[76] J. Kreich, A. Luhn, and G. Maderlechner. An experimental environment for model based document analysis. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 50–58, Saint-Malo, France, 1991.

[77] M. Krishnamoorthy, G. Nagy, S. Seth, and M. Viswanathan. Syntactic segmentation and labeling of digitized pages from technical journals. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 15(7):737–747, 1993.

[78] P. Kruizinga and N. Petkov. Nolinear operator for oriented texture. *IEEE Transactions on Image Processing*, 8(10):1395–1407, 1999.

[79] F. LeBourgeois, S. Souafi-Bensafi, and J. Duong. Using statistical models in document images understanding. In *DLIA2001 Advance Program*, Seattle, WA, 2001.

[80] K. Lee, Y. Choy, and S. Cho. Geometric structure analysis of document images: A knowledge-based approach. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 22(11):1224–1240, 2000.

[81] S. Lee and D. Ryu. Parameter-free geometric document layout analysis. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 23(11):1240–1256, 2001.

[82] J. Liang, I. T. Phillips, V. Chalana, and R. Haralick. A methodology for special symbol recognitions. In *15th International Conference on Pattern Recognition*, pages 11–14, Barcelona, Spain, 2000.

[83] J. Liang, I. T. Phillips, and R. M. Haralick. An optimization methodology for document structure extraction on latin character documents. *IEEE Tran. Pattern Analysis and Machine Intelligence*, 23(7):719–734, 2001.

[84] C. Lin, Y. Niwa, and S. Narita. Logical structure analysis of book document images using contents information. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 1048–1054, Ulm, Germany, 1997.

[85] S. Lowther, V. Chandran, and S. Sridharan. An accurate method for skew determination in document images. *Digital Image Computing Techniques and Applications*, 1:25–29, 2002.

[86] H. Ma and D. Doermann. Adaptive hindi ocr using generalized hausdorff image comparison. *ACM Transactions on Asian Laguage and Information Processing*, 2(3):193–218, 2003.

[87] H. Ma and D. Doermann. Bootstrapping structured page segmentation. In *SPIE Conference Document Recognition and Retrieval*, pages 179–188, Santa Clara, CA, 2003.

[88] H. Ma and D. Doermann. Gabor filter based multi-class classifier for scanned document images. In *7th International Conference on Document Analysis and Recognition*, pages 968–972, Edinburgh, Scotland, 2003.

[89] H. Ma and D. Doermann. Adaptive word style classification using a gaussian mixture model. In *17th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 606–609, Cambridge, United Kingdom, 2004.

[90] H. Ma and D. Doermann. Word level script identification for scanned document images. In *SPIE Conference Document Recognition and Retrieval*, pages 178–191, San Jose, CA, 2004.

[91] H. Ma and D. Doermann. Font identification using the grating cell texture operator. In *SPIE Conference Document Recognition and Retrieval*, pages 148–156, San Jose, CA, USA, 2005.

[92] H. Ma and D. Doermann. A graph theoretic approach to automatic ocr training. In *8th International Conference on Document Analysis and Recognition*, Seoul, Korea, 2005.

[93] H. Ma, B. Karagol-Ayan, D. Doermann, J. Wang, and D. Oard. Parsing and tagging of bilingual dictionaries. *Traitement Automatique Des Languages*, 44(2):125–150, 2003.

[94] D. Malerba and F. Esposito. Learning rules for layout analysis correction. In *DLIA 2001 Advance Program*, Seattle, WA, 2001.

[95] S. L. Manna, A. M. Colla, and A. Sperduti. Optical font recognition for multi-font ocr and document processing. In *10th International Workshop on Database and Expert Systems Applications*, pages 549–553, Florence, Italy, 1999.

[96] S. Mao and T. Kanungo. Stochastic language models for automatic acquisition of lexicons from printed bilingual dictionaries. In *Document Layout Interpretation and Its Applications*, Seattle, WA, 2001.

[97] S. Mao, A. Rosenfeld, and T. Kanungo. Document structure analysis algorithms: A literature survey. In *SPIE Conference Document Recognition and Retrieval*, pages 197–207, Santa Clara, CA, 2003.

[98] R. McGregor. *The OXFORD Hindi-English Dictionary*. OXFORD DELHI, OXFORD UNIVERSITY PRESS, 1993. ISBN 0-19-864339-X.

[99] P. Mitchell and H. Yan. Document page segmentation based on pattern spread analysis. *Optical Engineering*, 39(3):724–734, 2000.

[100] G. Nagy. Twenty years of document image analysis in pami. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):38–62, 2000.

[101] G. Nagy, S. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *Computer*, 25(7):10–22, 1992.

[102] L. O'Gorman. The document spectrum for page layout analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(11):1162–1173, 1993.

[103] L. O'Gorman and R. Kasturi. *Document Image Analysis*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[104] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *1997 Conference on Computer Vision and Pattern Recognition*, pages 130–136, San Juan, Puerto Rico, 1997.

[105] U. Pal and B. Chaudhuri. Automatic identification of english, chinese, arabic, devnagari and bangla script line. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 790–794, Seattle, US, 2001.

[106] T. Pavlidis and J. Zhou. Page segmentation and classification. *CVGIP: Graphical Models and Image Processing*, 54(6):484–496, 1992.

[107] G. Peake and T. Tan. A general algorithm for document skew angle estimation. In *Proc. Int'l Conf. Image Processing*, volume 2, pages 230–233, 1997.

[108] N. Petkov and P. Kruizinga. Comptational models of visual neurons specialised in the detection of periodic and aperiodic oriented visual stimuli: Bar and grating cells. *Biological Cybernetics*, 76:83–96, 1997.

[109] W. Postl. Detection of linear oblique structures and skew scan in digitized documents. In *Proc. Int'l Conf. Pattern Recognition*, pages 687–689, 1986.

[110] P. Resnik. Mining the web for bilingual text. In *37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, College Park, MD, 1999.

[111] L. Robadey, O. Hitz, and R. Ingold. A pattern-based method for document structure recognition. In *DLIA 2001 Advance Program*, Seattle, WA, 2001.

[112] P. Sarkar and H. S. Baird. Decoder banks: Versatility, automation, and high accuracy without supervised training. In *17th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 646–649, Cambridge, United Kingdom, 2004.

[113] P. Sarkar, H. S. Baird, and X. Zhang. Training on severely degraded text-line images. In *7th International Conference on Document Analysis and Recognition (ICDAR'03)*, pages 38–43, Edinburgh, Scotland, 2003.

[114] M. Schmidt. Identifying speaker with support vector networks. In *Interface'96 Proceedings*, Sydney, Australia, 1996.

[115] A. Schreyer, P. Suda, and G. Maderlechner. Font style detection in documents using textons. In *The Third Document Analysis System Workshop, Assoc. for Pattern Recognition*, 1998.

[116] H. Shi and T. Pavlidis. Font recognition and contextual processing for more accurate text recognition. In *The Fourth International Conference on Document Analysis and Recognition (ICDAR'97)*, pages 39–44, 1997.

[117] P. Sibun and A. L. Spitz. Language determination: Natural language processing from scanned document images. In *Proc. 4th Conference on Applied Natural Language Processing*, pages 115–121, Stuttgart, 1994.

[118] J. Solâ. Displaying khmer. http://www.khmeros.info.

[119] A. L. Spitz. Determination of the script and language content of document images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(3):235–245, 1997.

[120] C. Suen, S. Bergler, N. Nobile, B. Waked, C. Nadal, and A. Bloch. Categorizing document images into script and language classes. In *Proc. Int'l Conf. Advances in Pattern Recognition*, pages 297–306, 1998.

[121] K. Summers. Near-wordless document structure classification. In *Proc. Int'l Conf. Document Analysis and Recognition*, pages 462–465, Montreal, Canada, 1995.

[122] T. Tan. Rotation invariant texture features and their use in automatic script identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(7):751–756, 1998.

[123] Y. Tateisi and N. Itoh. Using stochastic syntactic analysis for extracting a logical structure from a document image. In *Proc. Int'l Conf. Pattern Recognition*, pages 391–394, Jerusalem, Israel, 1994.

[124] M. Teague. Image analysis via the general theory of moments. *Journal of the Optical Society of America*, 70(8):920–930, 1979.

[125] C.-H. Teh and R. T. Chin. On image anlaysis by the methods of moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):496–513, 1988.

[126] Q. D. Trier, A. K. Jain, and T. Taxt. Feature extraction methods for character recognition-a survey. *Pattern Recognition*, 29(4):641–662, 1996.

[127] S. Tsujimoto and H. Asada. Major components of a complete text reading system. *Proc. IEEE*, 80(7):1133–1149, 1992.

[128] D. Turner. The freetype project. http://www.freetype.org.

[129] A. Vailaya, H. Zhang, and A. K. Jain. Automatic image orientation detection. In *Proc. Int'l Conf. Image Processing*, volume 2, pages 600–604, 1999.

[130] A. Vinciarelli and J. Luettin. Off-line cursive script recognition based on continuous density hmm. In *Proc. 7th International Workshop on Frontiers in Handwriting Recognition*, Amsterdam, 2000.

[131] R. von der Heydt, E. Peterhans, and M. Dursteler. Grating cells in monkey visual cortex: Coding texture? In B. Blum, editor, *Channels in the Visual Nervous System: Neurophysiology, Psychophysics and Models*, pages 53–73. Freund, London, U.K., 1991.

[132] R. von der Heydt, E. Peterhans, and M. Dursteler. Periodic-pattern-selective cells in monkey visual cortex. *J. Neuroscience*, 12:1416–1434, 1992.

[133] F. Wahl, K. Wong, and R. Casey. Block segmentation and text extraction in mixed text/image documents. *Graphical Models and Image Processing*, 20:375–390, 1982.

[134] B. Waked, S. Bergler, and C. Y. Suen. Skew detection, page segmentation, and script classification of printed document images. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, pages 4470–4475, San Diego, CA, 1998.

[135] A. Yamashita, T. Amano, Y. Hirayama, N. Itoh, S. Katho, T. Mano, and K. Toyokawa. A document recognition system and its application. *IBM J. Research and Development*, 40(3):341–352, 1996.

[136] Y. Zheng, H. Li, and D. Doermann. Text identification in noisy document images using markov random field. In *7th International Conference on Document Analysis and Recognition (ICDAR)*, pages 599–605, Edinburgh, Scotland, 2003.

[137] Y. Zhu, T. Tan, and Y. Wang. Font recognition based on global texture analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(10):1192–1200, 2001.

[138] G. Zi and D. Doermann. Document image ground truth generation from electronic text. In *17th International Conference on Pattern Recognition (ICPR)*, volume 2, pages 663–666, Cambridge, United Kingdom, 2004.

[139] A. Zramdini and R. Ingold. Optical font recognition using typographical features. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(8):877–882, 1998.