Set theoretic line segmentation and graph based strategy for bilingual Kannada-English OCR

Umesh R S, Peeta Basa Pati and A G Ramakrishnan

Department of Electrical Engineering, Indian Institute of Science, Bangalore, India - 560 012

1 Introduction

India is an inherently multilingual nation and most of its people rarely communicate in a single language. There is always a mix of English or the neighbouring state language with the local language. This is true with many printed documents too, such as forms, school text books, certificates and official orders of the state or central Governments. Thus, an OCR to be widely useful in any Indian state, must necessarily be at least bilingual, capable of handling both the state language and English. Interestingly, many a times, the script changes at the level of the word within the same sentence or a text line in such printed documents. This paper, therefore, reports an approach for a bilingual OCR that can recognize both Kannada and English words from the same printed document.

In general, the design of a bilingual OCR might proceed by first segmenting the words from the document image, after the necessary initial steps of noise cleaning, page layout analysis and text segmentation. After identifying the script of a word, the characters of the word may be recognized using a single OCR engine, that employs two sets of training samples, one for English and the other for the Indian script. In our case, the idea is to recognize the script of each word, and accordingly send it to the Kannada or English recognition engine, as appropriate. This is because, the technology of English OCR is well developed and one could easily use a commercially available OCR for English. Figure 1 shows a schematic of our system.

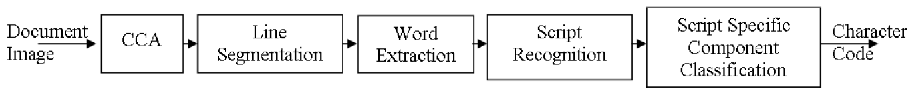


Fig. 1: A schematic of the proposed bi-lingual Kannada-English OCR.

Here, we report an efficient script recognition scheme that is more than 99% accurate in discriminating Kannada from Roman. We also propose a novel segmentation and recognition scheme for Kannada, which could possibly be applied to many other Indian languages as well.

2 Kannada Script

Kannada is the official language of the South Indian state of Karnataka. It has its own script derived from Bramhi script. Modern Kannada alphabet has a base set of 52 characters, comprising 16 vowels (called as swaragalu) and 36 consonants (vyanjanagalu). There are 2 more consonants used in old Kannada, namely lla and rra, taking the total number of consonants to 38. Further, there are consonant modifiers (vattaksharas or conjuncts) and vowel modifiers. The number of these modifiers is the same as that of base characters, namely 52. Compound characters called aksharas are formed by graphically combining the symbols corresponding to consonants, consonant modifiers and/or vowel modifiers using well defined rules of combination. The script has its own numerals too. In addition to the base set of characters and numerals, the script includes special symbols used in poetry, shlokas (prayer chants), Kannada grammar, etc. Thus, the number of possible consonant-vowel combination aksharas is 38 X 16 = 608. Similarly, the number of possible consonant-consonant-vowel aksharas is 38 X 38 X 16 = 23104. While designing a character recognition system, if we consider each akshara as a separate class, the number of classes becomes prohibitively high. However, in Kannada, consonant-modifiers and some of the vowel modifiers are mostly printed separately from the base character. So, if we treat each connected component as a different class, the number of classes in recognition can be reduced by a great extent. Figure 2 shows the vowels, consonants, all the consonant-vowel combinations & some of the consonant-consonant-vowel aksharas of the letter ka, all the conjuncts and the Kannada numerals.

3 Segmentation

The task of segmenting a document image into text and non-text, lines, words and eventually into individual characters is of fundamental importance for optical character recognition. The current study does not address the problem of separation of text from non-text. The input to this scheme is the bounding box information of various independent connected components in the skew-corrected, de-noised, component-labeled image. We propose a set-theoretic approach based on the bounding-box information for segmenting the text lines from the document image. This method is distinct from the ones proposed by [1, 2] which are based on the horizontal projection profiles.

3.1 line segmentation based on conneced components

This can be seen as the task of identifying unique non-intersecting sets of image components that form various text lines. Every component in the image needs to belong to only one text line. Fig. 3 shows two text lines of a Kannada document using only the bounding boxes of the connected components. Rows 1, 2, 3 and 4 denote particular rows of pixels. Labels a, b, c, etc. denote bounding boxes of specific connected components.

To detect and delineate text lines, we consider all the connected components intersected by each horizontal scan line. We compute the inter-component distances and look for close neighbours. If the distances of close neighbours are below a predefined

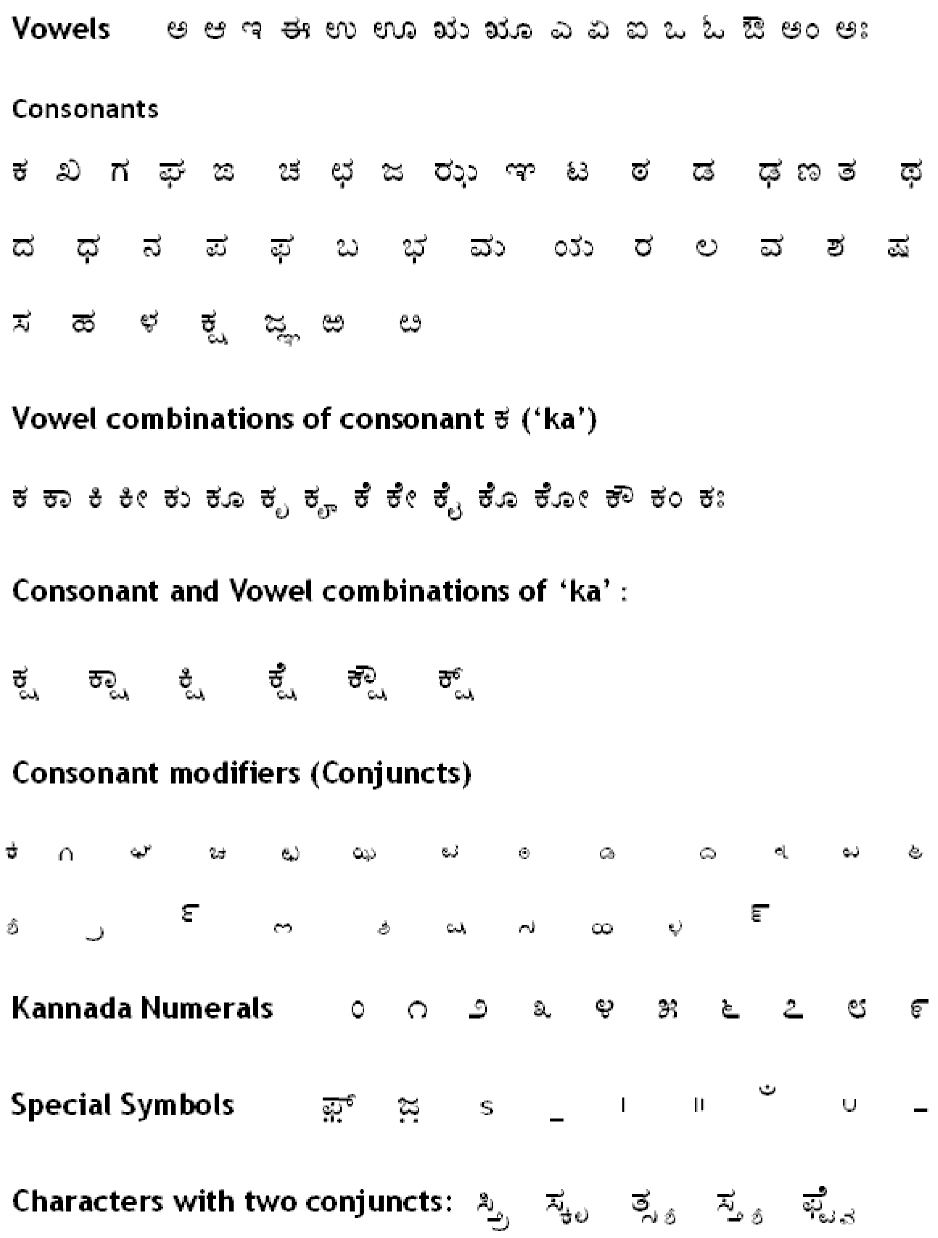

Vowels ಅ ಆ ಇ ಈ ಉ ಊ ಋ ೠ ಎ ಏ ಐ ಒ ಓ ಔ ಅಂ ಅಃ
Consonants
ಕ ಖ ಗ ಘ ಙ ಚ ಛ ಜ ಝ ಞ ಟ ಠ ಡ ಢ ಣ ತ ಥ
ದ ಧ ನ ಪ ಫ ಬ ಭ ಮ ಯ ರ ಲ ವ ಶ ಷ
ಸ ಹ ಳ ಕ್ಷ ಜ್ಞ ಱ ೞ
Vowel combinations of consonant ಕ ('ka')
ಕ ಕಾ ಕಿ ಕೀ ಕು ಕೂ ಕೃ ಕೄ ಕೆ ಕೇ ಕೈ ಕೊ ಕೋ ಕೌ ಕಂ ಕಃ
Consonant and Vowel combinations of 'ka' :
Consonant modifiers (Conjuncts)
Kannada Numerals ೦ ೧ ೨ ೩ ೪ ೫ ೬ ೭ ೮ ೯
Special Symbols
Characters with two conjuncts:

Fig. 2: Kannada character set.

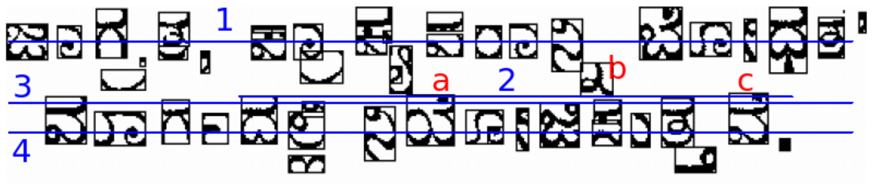


Fig. 3: Two text lines of a Kannada document. The bounding boxes of some of the connected components of successive lines intersect the same scan line, preventing line segmentation using horizontal projection.

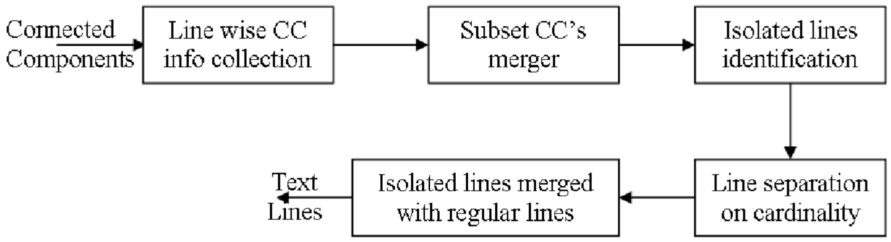

Connected Components
Line wise CC info collection
Subset CC's merger
Isolated lines identification
Line separation on cardinality
Isolated lines merged with regular lines
Text Lines

Fig. 4: A schematic of the proposed line segmentation algorithm.

threshold, then the set of intersected components is considered to belong to the same text line. Thus, the components intersected by scan lines 1 and 4 in Fig. 3 form distinct text lines, while those intersected by scan line 2, namely *a*, *b* and *c* do not, since they lie far spread apart. Similarly, scan line 3 also fails this test. At the end of this procedure, we are left with disjoint sets of components, and their spans (heights) decide the extent of these lines. A few isolated, left out components either fall within the span of one of the lines already found, or are assigned to the line containing their nearest component. Fig. 4 presents the proposed line segmentation algorithm, schematically. Let the labeled image be denoted as $L_{m\times n}$. Let the components in L be denoted by $c_i, i = 1 \ldots p$. Let each pixel row of L be denoted by $R_j, j = 1 \ldots m$. Then, set-theoretic segmentation can be described by the following steps-

- Form $S_j = \{c_i | c_i \cap R_j \neq \phi\}, j = 1 \ldots m, i = 1 \ldots p$ (form sets of components intersected by each pixel row)
- If $S_j \subset S_k$, set $S_j = \phi,\ j, k = 1 \ldots m, j \neq k$ (remove proper subsets of components; their supersets contain the same information)
- $\forall S_j \neq \phi, j = 1 \ldots m$ if $F(S_j, \gamma) =$ FALSE, $S_j = \phi$. F is a binary function based on the inter-component distances of elements of S_j and a threshold γ.
- $\forall S_j \neq \phi$ if $S_j \cap S_k \neq \phi,\ j, k = 1 \ldots m,\ j \neq k,\ and\ (S_j - S_k) \cap S_i = \phi,\ \forall i \neq k,\ S_j = \phi$ (merge sets that have some common components and the rest, unique components)
- $\forall c_k \in \left\{\{c_i, i = 1 \ldots p\} - \{\bigcup_{j=1}^{m} S_j\}\right\}$ (components not associated with any of the sets), find $min\|c_k - c_l\|,\ c_l \in \left\{\bigcup_{j=1}^{m} S_j\right\}$. Let the minimum occur at c_q. If

$c_q \in S_t$, then $c_k \in S_t$ (associate the component with the set containing its nearest connected component).
– $S_k \neq \phi,\ k = 1 \ldots m$ will now be disjoint sets of components. Each set comprises components that form a text line.

3.2 Word and character segmentation

Following line segmentation, one normally obtains information on the columns, which do not pass through any bounding box on a given line. By studying the distribution of the spacing information so obtained from all the lines of the document, one can threshold the distances to obtain word and character separation information.

However, we propose an alternative weighted distance spacing measurement which may give better results in case of documents containing multiple font sizes. Whenever the font size increases, so do the word and character spacing. So, if the spacing between any two adjacent components is weighted as a function of the sizes of the components across the gap (average of heights), one may obtain a more meaningful information about the spacing. This is effectively performed by using the bounding box information we employed for line segmentation. Thus, it turns out to be an additional benefit of using connected components to start with.

4 Script Recognition

Our recognition mechanism is based on separate engines for Kannada and English. Thus, once the document is segmented into text lines, words and components, we perform script recognition. Lot of research has been reported on script recognition at a paragraph/block or line level. While the former assumes that a full document page is of the same script, the latter imagines documents to contain text from multiple scripts but changing only at the level of the line. Though the latter is a realistic assumption in some cases, many Indian language documents have the script changing from one word to the next. In Fig. 5, we show a bi-script document, where the presence of interspersed English words in a Kannada document is clearly seen.

Script identification acts as a preliminary level of filtering to reduce the search complexity. Here, efficacy of various combinations of two different features and three different classifiers is evaluated for the bi-script recognition task. An initial design of a filter-bank that generates good accuracy for various script identification tasks, followed by a feature selection algorithm to choose the minimal best subset of these features to deliver the best output is reported.

4.1 Gabor and DCT based identification

Pati [3] studied the structural properties of eleven Indian scripts before designing an identifier for these scripts. Based on that study, we decided to employ features that are both frequency and direction sensitive. This, in our opinion, would be best able to discriminate between the scripts. Thus, we employ a multi-channel filter bank, using Gabor

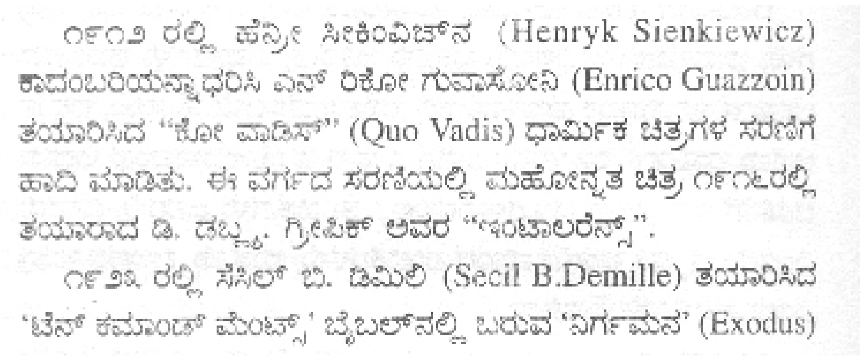

೧೯೧೨ ರಲ್ಲಿ ಹೆನ್ರೀ ಸೀಕಿಂವಿಚ್‌ನ (Henryk Sienkiewicz)
ಕಾದಂಬರಿಯನ್ನಾಧರಿಸಿ ಎನ್ ರಿಕೋ ಗುವಾಸೋನಿ (Enrico Guazzoin)
ತಯಾರಿಸಿದ "ಕೋ ವಾಡಿಸ್" (Quo Vadis) ಧಾರ್ಮಿಕ ಚಿತ್ರಗಳ ಸರಣಿಗೆ
ಹಾದಿ ಮಾಡಿತು. ಈ ವರ್ಗದ ಸರಣಿಯಲ್ಲಿ ಮಹೋನ್ನತ ಚಿತ್ರ ೧೯೧೬ರಲ್ಲಿ
ತಯಾರಾದ ಡಿ. ಡಬ್ಲ್ಯು. ಗ್ರಿಫಿಕ್ ಅವರ "ಇಂಟಾಲರೆನ್ಸ್".
೧೯೨೩ ರಲ್ಲಿ ಸೆಸಿಲ್ ಬಿ. ಡಿಮಿಲಿ (Secil B.Demille) ತಯಾರಿಸಿದ
'ಟೆನ್ ಕಮಾಂಡ್ ಮೆಂಟ್ಸ್' ಬೈಬಲ್‌ನಲ್ಲಿ ಬರುವ 'ನಿರ್ಗಮನ' (Exodus)

Fig. 5: Sample bi-script document showing interspersed Kannada and English words, necessitating script recognition at the word level.

functions [4], [5]. We have used a radial frequency bandwidth of one octave. This is because, the cortical cells in the visual cortex V1 of primates are observed to have that bandwidth and the coding of natural images is best attained at this bandwidth [6]. An angular bandwidth of 30 degrees is chosen for this experiment. After a careful evaluation of the various radial frequencies & angles for their usefulness for this application [4], we decided to use three radial frequencies and 6 angles. The radial frequencies chosen are 0.125, 0.25 and 0.5 radians, while 0, 30, 60, 90, 120 and 150 degrees are the angles chosen.

Discrete Cosine Transform (DCT) concentrates the information content in relatively fewer coefficients. For natural signals and images, the data compaction of DCT is close to that of the optimal KL transform. But unlike KLT, DCT is not dependent on the data. The symmetry of its transform matrix is exploited to obtain efficient hardware & software implementations [7]. Most image and video coders employ DCT. It has also been employed for other applications such as pitch modification for speech synthesis [8]. Pati [9] has used DCT coefficients for machine recognition of printed Odiya characters.

For an image f(x,y), the DCT coefficient matrix F(u,v) is given by,

$$F(u,v) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f(x,y) \cos\left(\frac{\pi u(2x+1)}{2\,M}\right) \cos\left(\frac{\pi v(2y+1)}{2\,N}\right) \tag{1}$$

where M and N are the number of rows and columns of the image matrix; u and v are the frequency indices along the x and y directions, respectively. Figures 6 (a) & (b) demonstrate the feature extraction mechanisms using Gabor and DCT functions for recognizing Roman from the Kannada script.

We have used three different classifiers to decide about the script of the test words: (i) the nearest neighbor classifier (NNC), (ii) linear discriminant classifier (LDC), and (iii) the support vector machines (SVM's). Nearest neighbor has been a standard and time tested classifier. This classifier has proven to deliver good output, when we have a class representative training set. Here, Euclidean distance of the test pattern is evaluated in the feature space, with each of the training patterns. The class value of the nearest

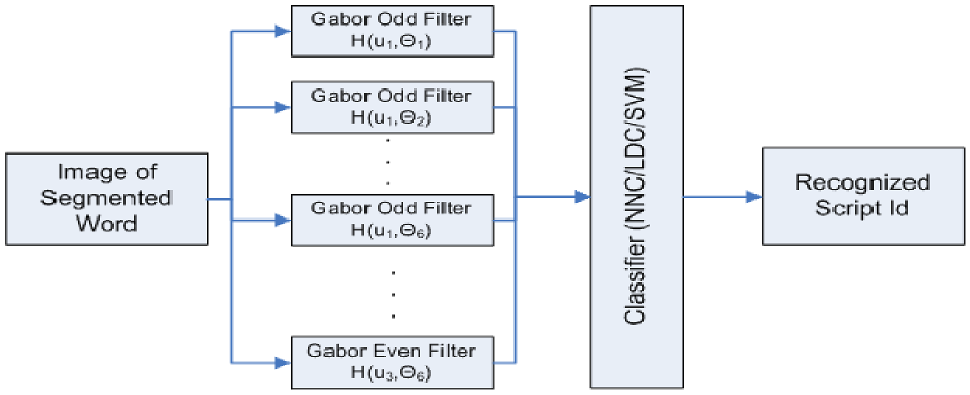


Image of Segmented Word
Gabor Odd Filter $H(u_1,\Theta_1)$
Gabor Odd Filter $H(u_1,\Theta_2)$
Gabor Odd Filter $H(u_1,\Theta_6)$
Gabor Even Filter $H(u_3,\Theta_6)$
Classifier (NNC/LDC/SVM)
Recognized Script Id

(a)

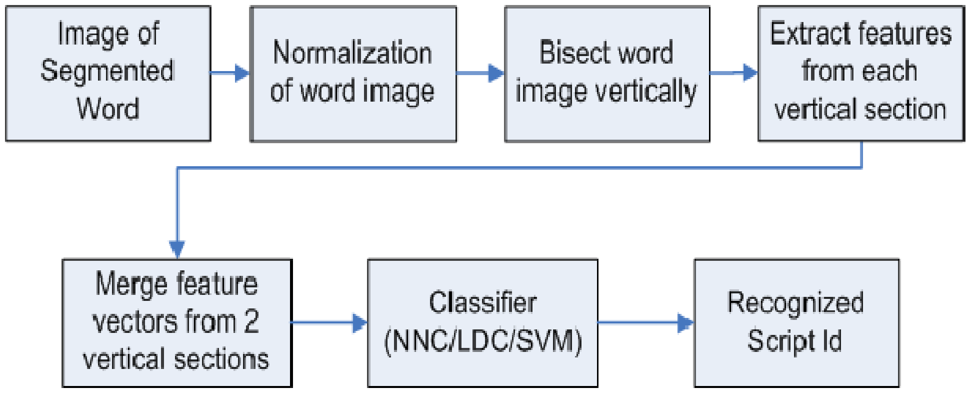
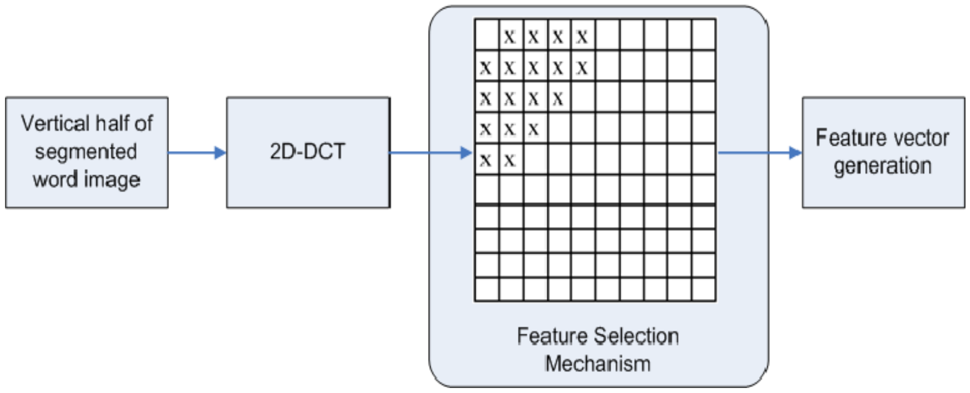


Image of Segmented Word
Normalization of word image
Bisect word image vertically
Extract features from each vertical section
Merge feature vectors from 2 vertical sections
Classifier (NNC/LDC/SVM)
Recognized Script Id

(b)


Vertical half of segmented word image
2D-DCT
Feature Selection Mechanism
Feature vector generation

(c)

Fig. 6: Extraction of feature vectors from word images for script recognition. (a) shows the extraction of Gabor features, while (b) demonstrates that of DCT features. (c) indicates those DCT coefficients that are included as part of the feature vector, to make the dimension same as that of Gabor.

neighbor is assigned to the test pattern. A linear discriminant function partitions the feature space using a hyper-plane. The two sides of this plane represent the two classes. The class value of the test pattern is decided based on which side of the plane it lies.

Among the discriminant approaches for classification, the most recent is the Support Vector Machine [10], where the optimal hyper-plane decides the separation between individual classes of patterns. The creation of a unique model to represent a class, derived by training the model with prototypes of each class, aids in maximization of the correct classification rate. We have used the SVMTorch – II, a toolbox designed and developed by Collobert and Bengio [11]. Of the various kernel functions this toolbox provides, we have used the Gaussian function with the total variance of the dataset, in a case by case basis, chosen as the standard deviation of the kernel function.

4.2 Results of script identification

Images are scanned from bi-script documents, such as books, newspapers and magazines. Variation in printing style and size are ensured. About 100 scanned pages are segmented by an automated process [9]. From this collection of word images, 20,000 distinct words each are selected for both Kannada and English. Of these, 7000 are used for training and the balance 13,000 word images are used for testing.

The performance is presented in percentage (%), which gives the average recognition accuracy for both the involved scripts. Table 1 presents the results separating Roman from Kannada script with different feature-classifier combinations. The recognition rate we have achieved equals or exceeds 99% with Gabor transform features for any classifier. The best result of 99.9% is obtained with the nearest neighbour classifier.

Table 1: The recognition accuracies of the various feature-classifier combinations for the bi-script case involving English and Kannada words.

GT-NNC	GT-LDC	GT-SVM	DCT-NNC	DCT-LDC	DCT-SVM
99.9	99.0	99.7	98.2	90.3	97.5

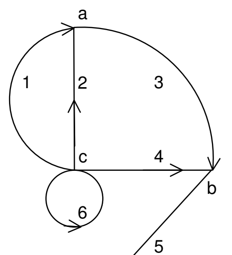
5 Component Classification

5.1 Introduction

The problem of recognizing printed Kannada characters has been studied in [1, 2]. As we find, it is one of being able to distinguish data from a large number of classes, typically around 450. This we find is the case with many Indian scripts [12]. The problem of classifying data from a large number of classes has been studied in [13]. The study shows a steep increase in computational costs with increase in the number of classes. Hence, most attempted solutions reduce the classes one needs to distinguish, by breaking characters into smaller subunits, usually using a script dependent segmentation technique [1, 2]. Each subunit is individually recognized, and a conclusion is drawn about the character based on the recognized subunits. However, the rules for segmenting the characters are usually script specific, and entail many assumptions. This makes it hard to adapt any such technique to other scripts. In what follows, we report a novel and effective strategy to solve this problem and discuss its merits and limitations. In Kannada and many other Indian scripts, a character can contain more than one 8-connected independent image components (which we shall refer to as 'components'). In this study, we report only about component classification, since it plays a central role in the whole recognition task. Recognition of characters from components is an important task that may need considerable attention, but is not the focus of this study.

The proposed strategy is based on the fundamental observation that, distinct component classes differ usually either in the number or kind of strokes or junctions they are composed of, or in the nature of stroke interconnections. In this study, we cast this inherent property into a graph-theoretic framework by associating thinned (single pixel wide) form of each component with a planar graph. The strokes are mapped to the edges of the graph, and the stroke junctions are mapped to vertices. A component can thus be associated with the incidence matrix of its graph representation. We report here our study on how this mapping can be used for the task of component classification.

We introduce our graph based processing of components through the illustration in Fig. 7, which depicts a planar graph with $\{a, b, c, d\}$ as vertices and $\{1, 2, 3, 4, 5, 6\}$ as edges. The signed incidence matrix for the graph is given by the center part of the table in Fig. 7. We adopt a systematic way to assign directions to edges. If the edge has a horizontal alignment, we assign the left end with a positive sign. Similarly, a vertically aligned edge has a negative sign associated with the lower end. When an edge has a mix of both the alignments, we give priority to the alignment that has a larger distance. In case of equal alignments in both the directions, we give preference to the height over width. Assigning directions to the loops is based on the relative position of the vertex with respect to the loop (left/right or above/below etc.). The column named hp denotes the sum of the entries of the corresponding row in incidence matrix. The final column gives the sum of the absolute values of the row entries. Similarly, the row vp denotes the sum of the corresponding column entries of the incidence matrix, whereas the last row gives the degrees of the vertices (sum of absolute values of column entries). Later, we use these entities to represent a component.



	a	b	c	d	hp	
1	1	0	-1	0	0	2
2	1	0	-1	0	0	2
3	-1	1	0	0	0	2
4	0	1	-1	0	0	2
5	0	-1	0	1	0	2
6	0	0	1	0	1	1
vp	1	1	-2	1		
	3	3	4	1		

Fig. 7: A sample graph and its signed incidence matrix

5.2 Graph Representations for components

Some OCRs use thinned versions (single pixel width representations) of components in classification, since it renders the task robust to font thickness and style variations. In the proposed scheme, we use thinning to obtain information about junctions and strokes in the character.

A 2D convolution of the binary image of a thinned component with a 3×3 mask of ones, yields different values at junctions and edges. Pixels anywhere on any edge get a value of 3; pixels that define terminal vertices get a value of 2 and pixels that belong to vertices of order 3 and higher get a value above 3. Thus, one can segregate edges from vertices and map the component to a planar graph. We adopt a systematic way to assign directions to edges and associate a signed incidence matrix with the component.

We can expect some of the components belonging to different classes to have same incidence matrices. Also, due to font variations, we can expect a single component class to have multiple graph representations. Besides, visually similar looking components could end up having very different matrix representations. Before we proceed further, we see whether the number of edges and vertices in the graph representation of components alone can be used to subdivide the component classes.

This segregation leads to four important categories of character components, which we call 'Object kind i', $i = 1 \ldots 4$. These categories denote those components which can be mapped to a point ($i = 1$), closed loops ($i = 2$), simple edges joining two vertices ($i = 3$) and objects with more than one edge/vertex ($i = 4$), respectively. It is interesting to know the prevalence of each of these categories in a general document of a script. Such an analysis is shown for Kannada script in Table 2, wherein, the number of components of various kinds found in 113 images obtained from 3 different books scanned at 300 dpi is shown in relative and absolute terms. Observe that about 30% of the samples were simple edges connecting two terminal vertices. It is interesting to note that if we segregate the learning process into three distinct branches (excluding objects which map to a single point), based on the kind of object we are dealing with, we would have already achieved one level of simplification of the problem we started with (one of classifying from about 450 classes). Thus, we begin to see the advantages of simple graph based features that we have defined. However, this still leaves us with the major

Table 2: Statistics of various kinds of objects collected from 113 scanned Kannada document pages.

Object Kind	1	2	3	4
Number of samples (%)	1781 (1.5)	4958 (4.3)	33806 (29.3)	74661 (64.8)

task of segregating complex objects of kind 4, which form the major chunk of the data we get (nearly 65%). The rest of the chapter is dedicated to solve this problem.

Before obtaining incidence matrices, we need to order the edges and vertices in a manner that remains invariant under normal font variations. However, since vectors do not have a natural ordering, any spatial ordering scheme can produce a different incidence matrix for very small changes in the position of the vertices or edges. To circumvent this problem, we order the edges based on their relative lengths. However, since vertices do not have such an attribute, we resort to the notion of "vertex significance", wherein we consider the sum of the lengths of the edges incident at a vertex as an attribute to rank the vertex. i.e, a vertex at which longer and/or more edges are incident has a higher ranking than the vertex with fewer and/or shorter edges. An incident matrix with edges and vertices ordered in this manner yields a robust representation for a component under normal font variations. Ties can either be broken arbitrarily (as we did), or systematically, based on factors like position etc. We next explore ways in which we can use the incidence matrix information for the task of classification.

Every classification scheme inherently uses a distance measure to compare patterns and this plays a decisive role in determining classification efficiency. We build such a notion from the signed incidence matrices.

5.3 Distance Measures

The area of graph and sub-graph matching addresses the problem we are faced with, and has been a subject of intense study [14]. However, we report a simple, yet useful strategy for this purpose. We resort to a few discrete, qualitative notion of distance, instead of quantitative ones. It is needless to say that this is not the best we can do with the information we have.

Let I and I_u be the signed and the unsigned incidence matrices, respectively, of a component. The rows of I are ordered in the descending order of relative edge lengths. The columns of I are stored in the rank of "vertex significance". From the graph, we collect the information in four separate row vectors, which we call $R_i, i = 1 \dots 4$ and define as follows:

- $R_1 = [no.of\,edges \quad no.of\,vertices]$
- $R_2 = [Euler\,no. \quad no.of\,non\,zero\,elements\,in\,I]$
- $R_3 = [vert.proj.of\,I_u \quad horz.proj.of\,I_u]$
- $R_4 = [vert.proj.of\,I \quad horz.proj.of\,I]$

where "vert proj" and "horz proj" stand for vertical and horizontal projections respectively, as defined in the context of Fig. 7. Clearly, R_1 and R_2 contain information about the gross features of the graph. R_3 contains partial information from the unsigned incidence matrix I_u and R_4 contains partial information from the signed incidence matrix in a condensed form.

Given two representations R_i^m and R_i^n $i = 1 \dots 4$, for samples numbered m and n, we define the qualitative distance between them as follows-

- if $R_i^m = R_i^n\ for\ i = 1 \dots j$, then they are said to ***match at level j***
- if $R_1^m \neq R_1^n$, then they are said to be ***unrelated***

Based on this notion of distance, we define the kinds of relationships the various component classes may develop, from the observations of graph representations of the training samples. Let the training samples belong to classes $C_k, k = 1, 2 \dots w$. Let $R_i^{a,m}, m = 1 \dots, p$ and $R_i^{b,n}, n = 1 \dots, q, i = 1 \dots 4$ be the representations of training samples (p and q in numbers) associated with classes C_a and C_b, respectively.

- for some samples r and s, if $R_i^{a,r} = R_i^{b,s}, \forall i$, and $a \neq b$, then, C_a and C_b are said to be in ***conflict at level 4***
- for some samples r and s, if $R_i^{a,r} = R_i^{b,s}, i = 1, 2, 3 \ and \ i \neq 4$, and $a \neq b$, then C_a and C_b are said to be in ***conflict at level 3*** (similarly, we can define ***conflicts at levels 2 and 1***)
- Classes in conflict are said to be ***related***
- if for no samples r and s, $R_1^{a,r} = R_1^{b,s}$, then classes C_a and C_b are said to be ***unrelated***
- for some samples r and s, if $R_i^{a,r} = R_i^{a,s}, \ i = 1 \ldots j < 4 \ and \ R_{j+1}^{a,r} \neq R_{j+1}^{a,s}$, then r and s are ***equivalent representations of class C_a up to level j***

Equipped with these qualitative distance measures between patterns and classes, we proceed with the classification strategies we use.

5.4 Classification strategy

The classification strategy we have adopted combines both deterministic and probabilistic decision making. First, using training samples, we try to learn the various ***conflicts*** among classes and ***equivalent representations*** for every class. After this learning, given a test pattern, we use the graph features R_i to see if we can assign a unique class label. In case of ***conflicts***, this will not be possible. However, in this process of finding a unique class label, the graph based features effectively narrow down the bigger classification task into a smaller one of classifying a sample to one among the conflicting classes. From here, we use probabilistic classification to assign a label to the test pattern. The important point to observe here is, the switch to conflict resolution can happen at any level (from 1 to 4), since conflicts can be defined at various levels. Usually this will be dictated by certain trade-offs between accuracy and computational cost, as we show later. The classifier we use to resolve conflicts is the Support Vector Machine (SVM) [15]. Though it is not mandatory to use SVMs, their utility in character recognition has been well documented [2, 13]. We have chosen RBF kernel, and the SVM formulation "support vector classification" for multi-class classification, for reasons elaborated in [13, 15].

5.5 Training

During learning, we build a table of representations that can guide us through the deterministic phase of classification. The table is grown by incrementally adding every new representation that we come across in the training set. A representation here denotes values of $R_i, i = 1 \ldots 4$ along with the class label. Every class will retain only its unique representations in the table. All ***conflicts*** and ***equivalent representations*** will be stored in the table. We also keep a count of repetitions of all representations. These can be used to obtain estimates of prior probabilities of equivalent representations of a particular class and also the prior probabilities of a particular representation belonging to conflicting classes, which can be used later for prediction.

Also, we collect features (moment and 2d spline) of components belonging to different classes in different files. Thus, we generate as many feature files as the no. of classes. For every conflict at every level, we use features from conflicting classes to build a SVM classifier. e.g., Suppose we find components from 3 different classes are in conflict at level one (share the same representation, say $R_1(m)$). They can also be in conflict again at level 1 with some other $R_1(n)$. Each of this is a distinct conflict. For each such conflict at each level, we build a separate SVM classifier, leading to a huge chest of classifiers. We maintain a table to help us pick the right classifier when a conflict arises during testing. We call this as classifier pointer table. We are now equipped to tackle any conflict that arises during testing.

5.6 Prediction

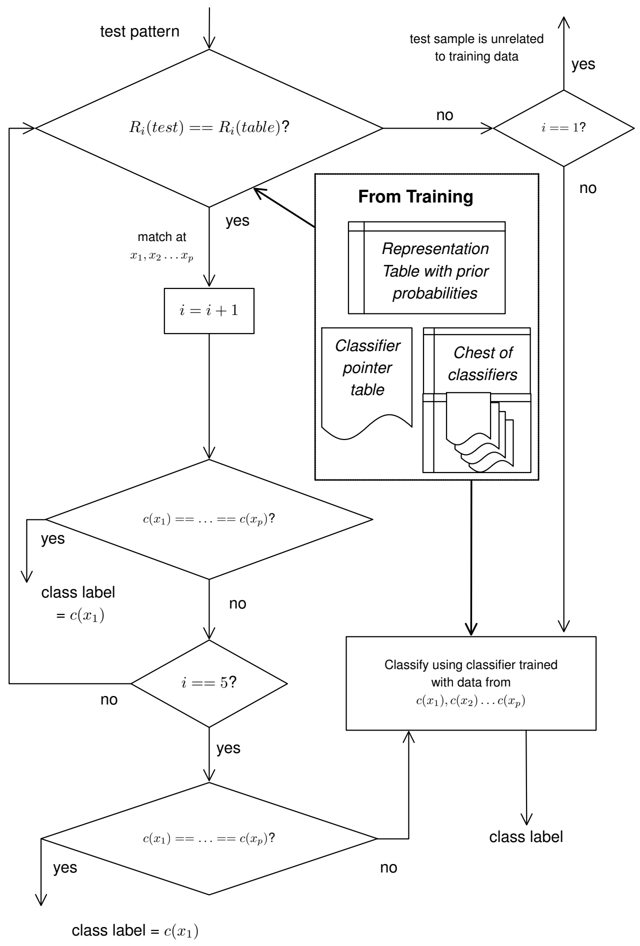
Fig. 8 shows a schematic of the classification strategy for the components. Given a test pattern, we obtain its graph representation and look up in the representation table to find whether such a representation exists in it (a simple string comparison with existing representations). If a unique match is found (no conflicts), we decide that the test pattern belongs to the corresponding class. This classification is completely deterministic and dictated by the graph structure.

In case of conflicts, we use the classifier specifically trained with samples from the conflicting classes at the highest level of representation, to classify the pattern. e.g., given a test pattern, if it so happens that we do not find a match for R_4 of the test pattern in the representation table, but find that conflicts exist at level 3, then it is obvious that conflict exists at levels 2 and 1 also (due to the very nature by which the representation table is generated). Though we can choose a classifier trained to resolve conflicts at any of the levels 1, 2 or 3, we choose the classifier trained to resolve conflicts at level 3, i.e., we narrow down the conflicts as much as possible using the graph based representation. This, as we show later, reduces the computational complexity by a great extent, as compared to trying to resolve conflicts at lower levels. When we do not find any match based even at level 1, (i.e., the sample is *unrelated* to any training data), we can declare that the test pattern does not belong to any of the given classes. Thus, this scheme provides an opportunity for decision making at many levels.

5.7 Experiments, Results and Discussion

Data Sets

Synthetic Data Set- This data set contains computer generated character components, digits and punctuations from 21 different Kannada fonts, including a few decorative ones. The font size is varied in the range $50 - 100$ in steps of 10 and the characters are printed at 100 dpi. 446 component classes are found in the whole script, including the ones in multiple-component characters. We have $33,552$ samples in all, providing, on an average, about 75 samples for each class. The factors that influenced the choice of this data set are font shape and size variations and the availability of samples from all possible components in fairly uniform numbers. The size variation in the data set mimics the usual sizes of components in documents scanned at 300 dpi and higher.



test pattern
$R_i(test) == R_i(table)$?
no
$i == 1$?
yes
test sample is unrelated to training data
no
yes
match at $x_1, x_2 \ldots x_p$
From Training
Representation Table with prior probabilities
Classifier pointer table
Chest of classifiers
$i = i + 1$
$c(x_1) == \ldots == c(x_p)$?
yes
class label = $c(x_1)$
no
$i == 5$?
no
yes
Classify using classifier trained with data from $c(x_1), c(x_2) \ldots c(x_p)$
class label
$c(x_1) == \ldots == c(x_p)$?
yes
no
class label = $c(x_1)$

Fig. 8: The strategy for classifying the connected components. x_i denotes the i^{th} entry in the representation table. $C(x_i)$ denotes its class label.

Real Data Set- This data set contains $50,316$ objects of kind 4, obtained from three different books with pages randomly chosen from scans performed at 300 and 600 dpi. The components in this data set belong to 281 classes, leading to around 179 training samples per class.

Table 3 shows the composition of the synthetic and real data sets. While the real data set contains only objects of kind 4, the synthetic data set contains components of every kind. However, Table 3 shows information regarding only the objects of kind 4 in both the data sets, obtained after extracting bounding boxes of the components. As can be observed, the real data set has fewer classes but more average samples per class than the synthetic data set.

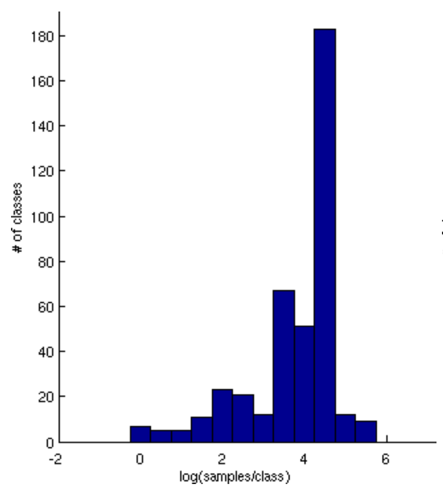
Fig. 9 shows the histogram of the number of classes as a function of the logarithm of the number of samples per class. We observe that a large number of classes in the synthetic data set (about 180) have more or less similar number of examples (about 90). We can also observe that there are few classes which have less samples per class. But, it is evident that the synthetic data set is better than the real data set in representing various classes. From Fig. 9b, we see that there are huge variations in the number of samples per class. This is substantiated by the figures in Table 3, where the standard deviation is higher than the mean. We observe that more than half the classes have far fewer samples per class than the synthetic data set, while a few classes had too many, leading us to conclude that this real data set is biased. All the results that we present in this section need to be interpreted taking into account this observed bias. For this reason, we believe that the results from the synthetic data set better represent the general properties of the proposed scheme.

Table 3: Statistics regarding objects of kind 4 in synthetic and real data sets. Values of component height and width are given in number of pixels.

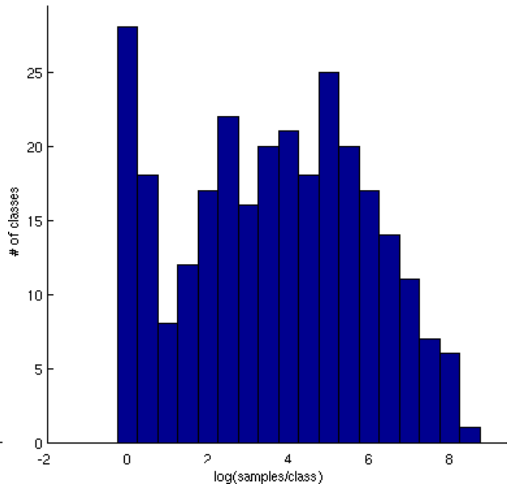
Synthetic Data set (406 classes, 26851 samples)			
	Comp. ht.	Comp. wd.	#test samples/class
mean	55.4	76.4	66
std	71.1	34.6	45

Real Data set (281 classes, 50316 samples)			
	Comp. ht.	Comp. wd.	#test samples/class
mean	48.9	47.6	179
std	14.0	18.0	579

Features for SVM classifiers We choose features that do not require image scaling. Thus, we use normalized central moment features [16] and 2d least squared B-spline features [17]. Though moment features have been extensively studied and used, the use of 2d B-splines for this purpose has received relatively little attention. Since these are solutions to sets of linear equations, they are not computationally very expensive. Our limited comparative studies seem to show that 2d least squared B-spline coefficients can be effectively used as features for component recognition. Fig. 10 shows the 8-fold



(a) Synthetic Data set



(b) Real Data set

Fig. 9: Statistics on the number of samples per class in the data sets. x-axis:ln(#samples/class), y-axis: No. of classes.

cross validation results on the synthetic data set obtained from libsvm [15] using an RBF kernel with 25 B-spline features, on objects of kind 4.

Pre-processing Real OCR data can have holes in edge strokes as a result of noise and binarization. Thinning is extremely sensitive to the presence of holes. If spurious holes are not filled, they can lead to unwanted thinned characters, and thereby to wrong graphs. This, by far, seems to be the biggest challenge to graph based representation of noisy components. A good morphological pre-processing step can probably mitigate this problem to a great extent. We found morphological majority, followed by spurring to be sufficient to overcome most of these effects in synthetic data. However, we find that this may not be sufficient to tackle effects of noise in real data. It is needless to say that the efficacy of the proposed method would improve by adding more sophisticated morphological preprocessing and script specific processing steps that can render graph representations robust to noise and font variations.

Results and discussions We segregated the synthetic data set into the four kinds of objects discussed in section 5.2. The first column of Table 4 gives details regarding the components of various kinds in the synthetic database. The term without brackets is the object kind. The first term in brackets is the no. of classes contributing to this object kind and the second term in brackets is the no. of samples of the particular object kind found in this database and the last term in brackets gives the proportion of the particular kind of object in the database. Note that a single class can turn out to belong to different object kinds due to variations in fonts.

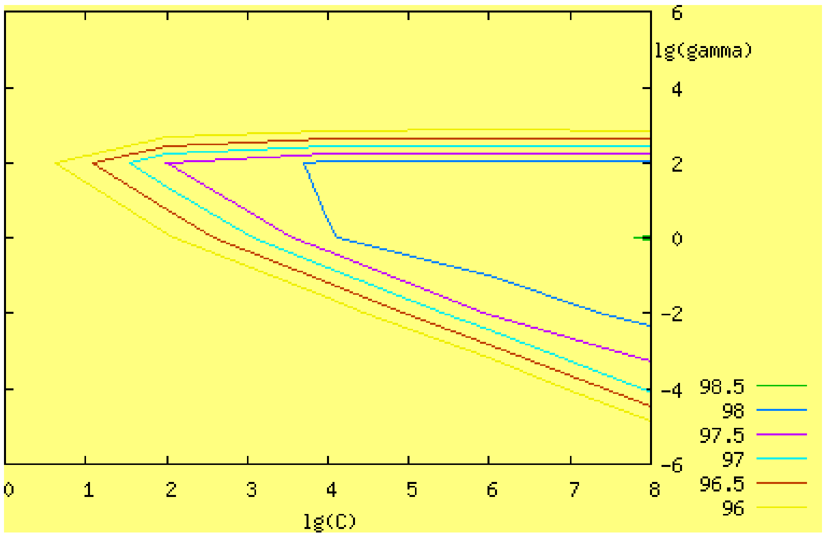


Fig. 10: Cross-validation results obtained using a multiclass SVM (RBF kernel) for objects of kind 4, with 25 2d-spline features. The log scales of the axes are in base 2.

These statistics differ from the statistics obtained from real data shown in Table 2, since these were synthetically generated. We can observe that 14 classes contribute to objects of kind 2 (closed loops) and 76 classes contribute to objects of kind 3 (simple edges). The fact that we can efficiently classify objects of kind 2 and 3 using dedicated classifiers is shown in column 4 of the same table.

However, objects of kind 4 have contributions from 414 classes. As we can observe, the relative cost of computation rises steeply for classifying objects of kind 4. The table shows that if we use 16 moment features, time to classify a pattern of kind 4 will be about 66 times more than the time needed to classify an object of kind 2, and with 25 spline features, it could be nearly 106 times longer. This is the difficulty we encounter when we try to classify objects of too many classes and particularly so, as the dimension of the feature vector increases. Hence, objects of kind 4 will be the focus of our study in the rest of this article. As can be seen from the last row of the same table, the spline features outperform the moment features at nearly the same computational cost.

We have built two separate classifiers on the lines of the proposed scheme to segregate objects of kind 4 in synthetic and real data sets. We have performed 8-fold cross-validation on each of the data sets. During each cross-validation run, $(7/8)^{th}$ of the data samples are used for training, using which, we build a representation table and derive features to build models to classify conflicting classes at various levels using SVMs. The rest of the data is used for testing. We have run two experiments, one with 16 moment features and the other with 25 spline features on the synthetic data set, while we used only 16 moment features in the real data set, since we observed similar trends with both features in the synthetic data set. We report results from both the classifiers and corresponding features. Since the results from the deterministic part of the strategy

Table 4: Cross-validation results for synthetic data set, obtained using SVM (rbf kernel) alone. mnts- Moment features, spl- 2d least squared spline features, Dim.- Dimensionality, (C,γ)- Values of C and γ for best accuracy, Cost- Relative cost of classification.

Obj. kind	features	Dim.	Best(%) Accu.	C,γ	SVs	Cost
2 (14)	mnts	16	98.35	16, 1	261	1.0
(977) (3.0%)	spl	25	98.26	16, 0.0625	170	1.0
3 (76)	mnts	16	98.67	64, 4	1701	6.5
(5178) (15.7%)	spl	25	98.82	64, 0.0625	1927	11.5
4 (414)	mnts	16	97.09	256, 16	17261	66.1
(26851) (81.4%)	mnts	25	97.38	256, 4	17360	103.9
	spl	25	98.51	256, 1	17669	105.8

are invariant to the chosen features (we used the same representation tables for both the features), they are reported only once for each classifier. In our discussions, we mostly quote results obtained with moment features from the synthetic data set and the corresponding results from the real data set are quoted in brackets.

We have chosen a strategy to obtain useful C and γ for training the SVMs for every conflict, since this is critical for obtaining good results. In order to achieve this, we have performed 3-fold cross-validations across a wide range of C ($2^0 - 2^6$ in powers of two) and γ ($2^{-8} - 2^2$ in powers of two) values. If the peak performance occurred at a unique pair of C, γ value, we have used the same for training the SVM. If many pairs of C, γ give the same performance, we have used the average values, since, as we see from Fig. 10, the regions of similar performance are nearly convex and hence, we expect the means to lie within these regions. However, the values so obtained need not be optimal. In reporting the results, for ease of reading, wherever appropriate, we have rounded the mean values of no. of classes, support vectors and conflicts to the nearest integer.

Table 5 shows the statistics obtained from the 8 representation tables generated by the cross-validation experiments. As expected, there are fewer unique strings at lower levels of representations. A fact that is clearly evident from Table. 5, is the relatively high variation observed in the real data set, in terms of graph representation. This is borne out by the fact that the total number of representations in the table is on an average, nearly half the size of the training data, while for the synthetic data set, it was only about a quarter. We believe that the main reason for this is that we have far few samples for many classes, as seen in Fig. 9b, and invariably, nearly all of them have to end up in the representation table, thereby increasing the table length. What is interesting is the fact that nearly 62% ($UR_4/\sum UR_i$) (86% for real data set) of all the variations occur only at level 4. i.e., if the representation table construction were to be truncated at level 3, on an average, the number of representations in our tables would be only about 12% (8%) of the overall training samples. This could be a pointer to the fact that we might not have had enough samples in both real and synthetic data sets for learning, especially at levels 3 and 4.

Table 6 shows information about the conflicts found in the tables. It shows that conflicts increase as the level of representation increases. Though this may seem counter intuitive, we need to observe that the no. of unique representations at every level also

Table 5: Statistics from the representation tables. # Tr. - No. of training samples, Reps. - Total representations in the table. $UR_i, i = 1 \ldots 4$ - "unique representations at level i" in the tables, (% total) - The means as a percentage of # Tr.

	# Tr.	Reps.	UR_1	UR_2	UR_3	UR_4
Synthetic Data set (26851 samples)						
mean	23550	6274	71	332	2356	4425
(% Tr.)	100.0	26.6	0.3	1.4	10.0	18.8
std	2.2	16.2	0.7	5.0	9.9	11.1
Real Data set (50316 samples)						
mean	43924	24677	107	237	3275	21705
(% Tr.)	100.0	56.2	0.2	0.5	7.5	49.4
std	16.4	19.6	2.3	1.5	5.8	33.9

increases, as shown in Table. 5, thus bringing down the no. of classes per conflict. Table 6 points to the important fact that we must expect the training times to be huge if we want to use the proposed scheme. As described in section 5.4, each conflict is eventually resolved using an SVM. We need to cross-validate and find optimal parameters for each of these SVMs. Thus, in total, we need to cross-validate and train on an average about 1477 SVMs (61+233+618+565) for synthetic data and 2533 SVMs (93+153+1148+1139) for real data. Although the number of training samples per SVM goes down as we go to higher levels of representation (which we shall see shortly), it still looks like a daunting task. However, given that the training is usually a one time job in the case of OCR engines, the highest importance is accorded to the classification time and accuracy. We shall shortly show that this exorbitant cost we incur in training pays off well during testing, and hence should not be a matter of concern in developing OCR engines on these lines.

The crucial information about what happens to computational costs is captured in the number of classes per conflict, since this determines the no. of support vectors and hence the cost of computation. Table. 6 shows this statistics too. As we can observe, the no. of classes per conflict (hence classifier) varies drastically. e.g., for level 4, the average no. of classes per conflict is 4 (3). But the maximum no. of classes is found to be 50 (14) for a particular SVM, thereby hinting that there should be many SVMs which distinguish only 3 or 2 classes. Hence, the average computational cost could be much lower, than what we perceive from this table. Similar is the case with other levels too.

Table 7 shows the average no. of support vectors per classifier, which is a better estimate of how the computational need will go down, as we go to higher levels of representations. It shows that classifying data at level 4 could be nearly 19 (32) times quicker on an average (using moments), compared to classifying it at level 1. However, as we could have guessed by the huge deviations in the no. of classes per classifier, the no. of support vectors per classifier has a huge variance too. What is clear from the results of Table 7 is that the computational cost of classification using the proposed scheme at various levels is much lower than the cost of classification using a single multi-class SVM (M-SVM), whose results are shown in Fig. 10. However, this gain does not take into consideration the cost involved in searching the representation table

Table 6: Number of conflicts and classes per classifier at various levels.

	Level 1	Level 2	Level 3	Level 4
Conflicts at each level				
Synthetic Data set				
mean	61	233	618	565
std	1.0	2.9	5.2	4.1
Real Data set				
mean	93	153	1148	1139
std	1.0	2.5	8.0	12.3
Classes per classifier				
Synthetic Data set				
mean	46	16	5	4
std	48.7	22.5	8.6	4.7
max	182	123	119	50
Real Data set				
mean	43	27	7	3
std	37.1	34.2	9.9	1.7
max	118	117	92	14

Table 7: No. of support vectors per classifier (denoted as SVs/Cfr) and mean relative computational cost (mean. rel. cost) at various levels. M-SVM stands for the single multi-class SVM trained with samples from all classes.

SVs/Cfr	M-SVM	Level 1	Level 2	Level 3	Level 4
Synthetic Data set					
Normalized central moments (16 features)					
mean	15949	1598	445	99	84
std	33	2007	838	272	157
max	15983	8326	5218	4197	1883
mean. rel. cost	**189.9**	**19.0**	**5.3**	**1.2**	**1.0**
2d Least Squared splines (25 features)					
mean	16178	1584	467	137	101
std	28	2023	878	319	176
max	16229	8324	5426	4449	1926
mean. rel. cost	**160.2**	**15.7**	**4.6**	**1.4**	**1.0**
Real Data set					
Normalized central moments (16 features)					
mean	9554	1978	1204	222	61
std	62	1951	1764	522	75
max	9596	6161	6074	5458	995
mean. rel. cost	**157.2**	**32.6**	**19.8**	**3.7**	**1.0**
2d Least Squared splines (25 features)					
mean	8050	1762	971	199	61
std	59	1742	1518	470	68
max	8114	5607	5568	4803	617
mean. rel. cost	**132.0**	**28.9**	**15.9**	**3.3**	**1.0**

and obtaining graph based features, which we shall mention shortly. Assuming that these costs are relatively small, we see that classifying a component using the proposed scheme can give us acceleration factors of nearly 10 (5), 36 (8)), 162 (43) and 189 (157) respectively, as we go to higher levels of representation using moments features. This makes a huge difference to testing time, which is very crucial for OCR applications. This can make a big difference in practically being able to use a classifier for the task. Though results shown in Fig. 10 are very attractive, such a classifier can seldom be used in real-life applications, due to this single factor.

The lower speed up factors for real data set, observed in Table 7, can be attributed to the fact that far fewer classes are being classified in the real data set compared to the synthetic one. We believe that a real data set with a large number of classes would show results similar to the synthetic data set, since an increased number of classes would relatively increase the support vectors in the single multiclass SVM by a huge margin, as compared to the increase in support vectors in classifiers trained to resolve conflicts.

To know the true gain in speed, we must factor out the time spent on obtaining graph based features and in traversing the deterministic path of the scheme, wherein we employ string matching to narrow down to higher levels of representation. Some OCR algorithms thin or skeletonize the data to remove certain font based dependencies. Getting to graphs from this stage involves only binary morphological filtering, and simple calculations which can be implemented efficiently. Thus, we can safely exclude the role of this while estimating computational costs. In the deterministic part of the algorithm, we perform exact binary string search over small strings (maximum length being around 150 bytes). The search can be made quicker by building a heap structure from the representation table during training [18], leading to search times of $\theta(\ln(n))$, n being the number of entries in the representation table. This computation is much smaller than $\theta(s \times d)$ multiplications, where s and d represent the no. of support vectors and their dimension, respectively. This is especially true at lower levels of representations, where the strings are only a few bytes in length, but the number of support vectors is relatively large. The deterministic and probabilistic computational costs may become comparable only at higher levels (3 and 4), in which case, we see that the acceleration factors are above 100. Factoring out the computational needs for the deterministic part still leaves us with huge acceleration factors. Hence, we see that the SVM classification is the only major computational cost we incur at lower levels of representation. This behaviour becomes particularly useful when one would like to increase the number of features being used by the SVMs to resolve conflicts.

Moreover, the total entries in the representation table should logically reach a saturation limit as the number of training samples increases. Thus, as the size of the training set increases, the string search part of the algorithm may need relatively less and less time, as compared to SVM classification. In all, we can safely conclude that we can indeed realize large acceleration factors by resorting to the proposed scheme.

Though the proposed scheme looks computationally attractive, the performance of this scheme in terms of classification accuracy is also important. Table 8 shows the results obtained from the cross-validation study. The classification accuracy at a particular level refers to the accuracy obtained by restricting conflict resolution to that level. e.g., if we do not use the representations beyond R_1 and instead resolve conflicts by using

Table 8: Classification accuracies when classification is restricted to a particular level of graph based features.

Classification Accuracy (%)	Level 1	Level 2	Level 3	Level 4
Synthetic Data set				
Normalized central moments (16 features)				
mean	94.8	93.2	88.3	90.1
std	0.3	0.6	0.9	0.6
min	94.3	92.0	86.9	89.2
max	**95.4**	**94.0**	**90.1**	**90.8**
2d Least Squared splines (25 features)				
mean	95.8	94.3	88.6	90.7
std	0.3	0.3	0.6	0.5
min	95.5	93.8	87.5	89.8
max	**96.2**	**94.7**	**89.1**	**91.1**
Real Data set				
Normalized central moments (16 features)				
mean	96.3	95.9	87.4	82.8
std	0.2	0.2	0.6	0.6
min	96.1	95.6	87.1	82.3
max	**96.4**	**96.1**	**87.8**	**83.2**
2d Least Squared splines (25 features)				
mean	96.7	96.3	87.7	83.1
std	0.2	0.2	0.3	0.6
min	96.4	96.0	86.7	82.4
max	**97.0**	**96.6**	**88.2**	**84.1**

classifiers trained to resolve conflicts at level 1, we get results tabulated in the second column of Table 8. However, if we let the algorithm use the full extent of representation, we get results up to level 4 (last column). Due to lack of matching representations at higher levels, some decisions may have to be taken at lower levels. i.e, results at level 4 could have contained results obtained at lower levels. We observe that the classification accuracies obtained at levels 1 and 2, though slightly lower, are comparable to the ones shown in Fig. 10, thereby making a clear case in favour of this strategy by virtue of its classification speed. Table 8 also shows that decision making at lower levels (1 and 2), irrespective of whether we could make decisions at higher levels, can yield better accuracy. However, this comes at an increased computation cost, as shown in Table 7. We see a trade-off between accuracy and computational cost coming into picture at this juncture. But classifying at level 2 is still about 36 (8) times faster than what we would have done with a single SVM classifier, as we can infer from Table 7. Also, as expected, we observe that the classification accuracies obtained with splines is better than those obtained with moment features with both the synthetic and real data.

It is instructive to see what the contribution of each of the levels is, to the total classification. Table 9 shows such a break-up. We see that nearly 90% (63%) of the data can

Table 9: Contribution from each level for the total classification.

Contribution (%)	Level 1	Level 2	Level 3	Level 4
Synthetic Data set				
mean	0.12	3.98	4.97	90.07
std	0.05	0.50	0.45	0.61
Real Data set				
mean	0.17	3.43	33.36	63.01
std	0.03	0.09	0.42	0.43

be classified at level 4, which is a very good prospect from the point of view of computational cost. However, from Table 8, we find that we may have to sacrifice accuracy in this case. We observe that in the case of real data, about 30% of the components have no match with existing representations at level 4, clearly indicating that the learning of representations at level 4 is at its very early stages. As we find, it is better in the case of synthetic data, where only 10% needs to be classified at the lower levels.

We now try to analyze the various errors leading to relatively poor performance at higher levels of representation. It is also important to note the fact that in the proposed graph based scheme, there are two kinds of errors that can be committed at each level. First, the errors due to the deterministic part of the algorithm based on graph features. Second, the errors committed by individual SVMs. It is instructive to know which of these errors dominates at each level. This can provide insights to improve upon the achievable classification accuracy. Table 10 shows such statistics. We can split the errors that arise due to graph features into three types, referred to as Type 1, 2 and 3. Type 1 errors are those for which no match could be found even at level 1. Type 2 errors denote those instances where, though a unique string match is found, the label assigned

Table 10: Relative (%) errors when classification is restricted up to a particular level.

Rel. Errors (%)	level 1	level 2	level 3	level 4
Synthetic Data set				
Graph	**Normalized central moments (16 features)**			
Type 1	0.5	0.4	0.3	0.5
Type 2	2.1	3.2	42.8	36.7
Type 3	61.5	72.8	44.5	43.5
Total	**64.1**	**76.4**	**87.6**	**80.7**
SVM	35.9	23.6	12.4	19.3
Graph	**2d Least Squared splines (25 features)**			
Type 1	0.7	0.5	0.3	0.6
Type 2	3.4	4.1	48.6	39.7
Type 3	74.9	82.4	44.0	48.5
Total	**79.0**	**87.1**	**92.9**	**88.8**
SVM	21.0	12.9	7.1	11.2
Real Data set				
Graph	**Normalized central moments (16 features)**			
Type 1	0.7	0.6	0.2	0.4
Type 2	1.5	5.3	22.4	54.7
Type 3	69.0	68.4	71.1	43.0
Total	**71.2**	**74.4**	**93.7**	**98.1**
SVM	28.8	25.6	6.3	1.9
Graph	**2d Least Squared splines (25 features)**			
Type 1	1.1	1.0	0.3	0.5
Type 2	3.0	6.0	23.9	55.1
Type 3	81.6	80.5	72.5	42.7
Total	**85.7**	**87.5**	**96.7**	**98.3**
SVM	14.3	12.5	3.3	1.7

based on graph features is wrong. Type 3 errors occur when the classifier pointer table associates a wrong classifier for the test pattern.

As we find from Table 10, most of the errors are indeed due to reliance on the graph based representation, especially of Type 2 and Type 3. We also observe that errors due to SVMs is much less in the case of splines, than in the case of moment features. We observe that at levels 1 and 2, most errors are of type 3, whereas errors of type 2 are very low. This indicates that we have been able to learn the existence of conflicts, or patterns that lead to conflicts, but have been unable to capture all the conflicting classes for various representations.

This is in contrast to the case of levels 3 and 4, where we observe that errors of type 2 are significant, indicating that the learning is oblivious to the fact that multiple classes share a common representation, let alone know which of them. This also explains the large contribution of errors of type 3 at these levels. This indicates that the algorithm has not seen enough training samples at these levels and is still at a very early stage of learning. Theoretically, errors of type 2 and 3 should be absent (or saturate to some value in practice) with larger training sets, since this is a deterministic strategy. Hence, to realize the true value addition provided by the proposed graph based scheme, we may need to use huge training sets.

Moreover, the representation of characters based on R_i is a very simple reduction of the information actually present in the signed incidence matrix. We can employ graph matching/comparison algorithms [14, 19], and/or other graph features, which may better the performance of the deterministic part of the scheme.

We can see that there is scope for improvement on the SVM front too (especially in the case of synthetic data). We attribute the errors committed by SVM to sub-optimal parameter selection and training. As mentioned earlier, we train each SVM based on an average of a set of best parameters obtained from three-fold cross-validation over a coarsely sampled, wide range of parameters [15]. This we should expect by no means to lead to best possible results with SVMs. Moreover, we had too few samples to train the SVMs. An increase in the number of training samples might yield better classification accuracy for each individual SVM. These observations make us believe that we can improve the performance here too. So, a better parameter criterion to choose the training parameters combined with search on a much finer parameter grid should definitely improve the results. A particular aspect that could perhaps be helping SVM classification is the fact that a set of conflicting classes need not necessarily have components that look visually alike, since the grouping is based on graph features. This can make the classification task easier when features that capture visual information are chosen. So, perhaps fewer and simpler features can suffice for the task on hand. We can even probably choose a different set of features to classify at different levels.

We now analyze other possible causes for the observed poor performance at higher levels. We believe the degradation in performance can be attributed to three main causes apart from the size of the training data set. The first is the inherent variation in real data. Perhaps we can try to alleviate this effect by using script-specific pre-processing schemes. e.g., discarding very small terminal edges, closing looping edges and coalescing very small non-terminal edges with vertices. The second contributor to the malperformance could be noise leading to holes in component images during binarization.

Presence of holes can dramatically alter graph representations, thereby mixing up representations, and increasing representations per class. This can also be minimized by better pre-processing techniques. However, the third and a more subtle factor to note, is the effect of scanning resolution. Since the deterministic part of the algorithm depends heavily on edge junctions, a coarse scan can lead to loss of junction information. This can make graph representation less effective. A component scanned at different resolutions can have very different graph representations. Thus, a relatively high scanning resolution (depending on the font size) may be needed in order to extract best results from the graph based schemes. This further emphasizes the need to use features which do not need image scaling for use with classifiers, since, as images get larger, resizing gets costlier. From Table 3, we observe that the components in the real data set have significantly different statistics compared to the synthetic data set, and are on an average, smaller especially in width. This could probably be due to particular classes being completely missed out in real data set. However, if indeed small sized components exist in the data set, they could lead to more conflicts at higher levels, as evidenced by Table 6, thereby making learning harder. Combinations of these factors need to be accounted for, during training and testing, for the graph based scheme to be effective.

From the above results and discussions, we see that there is a large scope for improvement in the whole methodology. The results mentioned here are only conservative estimates of what is potentially achievable with this scheme.

Extensions- In the experiments we discussed, we have not used the prior-information about which we mentioned in section 5.4. Using prior information can help us identify insignificant conflicts, which can be omitted in training, thereby reducing the training time. In conjunction with SVMs that can give probabilistic outputs [15], it can also help to improve the classification accuracy of the SVMs.

We could continue to use the strategy outlined here and use other graph features to further sub-divide the problem. A particularly useful extension could be to use edge labels obtained from an edge classifier (mapped to edge weights in graphs) to disentangle possibly merged components. The proposed method also seems to be potentially useful in online character recognition, where we can replace edge information by stroke information. In this case, we will not have the problem of ordering edges and vertices, since we obtain time information with each stroke. This may make this strategy very suitable for online character recognition.

6 Conclusion

A bilingual OCR system has been presented, with a robust segmentation strategy that does not depend on projections. We have demonstrated the use of graph based features in building fast and efficient classification engines for Kannada script. This can easily be extended to other scripts, since this is a script-independent strategy. Though the scheme involves a highly taxing learning phase, we have shown that it can yield a high speedup in testing, a key factor in designing OCR engines. We have discussed the scalability of the scheme and shown that it is well suited for learning from very large data sets. Similar ideas could be used to develop bilingual OCR's involving other Indian scripts.

References

1. B. Vijayakumar and A. G. Ramakrishnan, *Machine Recognition of Printed Kannada Text*, Document Analysis Systems V, Ed. D Lopresti, J Hu and R Kashi, pp. 37-48, Lecture Notes in Computer Science 2423, Springer Verlag, Berlin, 2002.
2. T. V. Ashwin and P. S. Shastry, *A font and size independent ocr system for printed kannada documents using support vector machines*, SADHANA Vol .27, Part 1, February 20002, pp. 35-58.
3. P B Pati, "Analysis of Multi-lingual Documents with Complex Layout & Content", PhD Thesis, Indian Institute of Science, Bangalore, INDIA, 2007.
4. P. B. Pati and A. G. Ramakrishnan, "Word level multi-script identification, Pattern Recognition Lett.(2008), doi:10.1016/j.patrec.2008.01.027
5. P. B. Pati, S. S. Raju, N. K. Pati and A. G. Ramakrishnan, "Gabor filters for document analysis in Indian Bilingual Documents," Proc. First Intl. Conf. on Intelligent Sensing and Information Processing (ICISIP-04), IEEE Publications, Chennai, India, 2004, pp.123 - 126.
6. D. J. Field, "Relation between the statistics of natural images and the response properties of cortical cells," Journal of Opt. Soc. of Am. A, vol. 4, no. 12, 1987, pp. 2379 - 2394.
7. K. R. Rao and P. Yip, Discrete Cosine Transform : Algorithms, Advantages, Applications. New York: Academic Press, 1990.
8. R. Muralishankar, A. G. Ramakrishnan, and P. Prathibha, "Modification of pitch using DCT in the source domain," Speech Communication, vol. 42, 2004, pp. 143 - 154.
9. P. B. Pati, "Machine recognition of printed Odiya text documents," Master's thesis, Indian Institute of Science, Bangalore, INDIA, 2001.
10. C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining and Knowledge Discovery, vol. 2, no. 2, 1998, pp. 955 - 974.
11. R. Collobert and S. Bengio, "On The convergence of SVMTorch, an Algorithm for Large Scale Regression Problems," Tech. Rep., Dalle Molle Institute for Perceptual Artificial Intelligence, Martigny, Switzerland, 2000.
12. Peeta Basa Pati and A. G. Ramakrishnan, *OCR in Indian Scripts: A Survey*, IETE Technical Review, Vol. 22, No. 3, May-June 2005, pp. 217-227.
13. Chih-Wei Hsu, Chih-len Lin. A comparison of methods for multiclass support vector machines. IEEE Trans Neural Networks. 13: 415-425, March (2002).
14. Bruno T. Messmer and Horst Bunke,*Efficient Subgraph Isomorphism Detection: A Decomposition Approach*, IEEE Trans. Knowl. Data Eng. 12(2): 307-323 (2000).
15. Chih-Chung Chang and Chih-Jen Lin, *LIBSVM: a library for support vector machines*, Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm
16. J. Flusser. Moment invariants in image analysis. In Proc. World Academy of Science, Engineering and Technology. Vol 11, Feb 2006.
17. C. De Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.
18. R. L. R. Thomas H. Cormen, Charles E. Leiserson. *Introduction to algorithms.* MIT Press/McGraw-Hill, 1990.
19. D. Lopresti and G. Wilfong. A fast technique for comparing graph representations with applications to performance evaluation. *International Journal on Document Analysis and Recognition*, 6(4):219–229, April 2003.