Studies in Recognition of Telugu Document Images

Thesis submitted in partial fulfillment
of the requirements for the degree of

Masters By Research
in
Computer Science and Engineering

by

Venkat Rasagna Reddy. K
200402021
rasagna@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2013

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled “Studies in Recognition of Telugu Document Images” by Venkat Rasagna Reddy K., has been carried out under my supervision and is not submitted elsewhere for a degree.

June 30, 2013

Adviser: Prof. C. V. Jawahar

I dedicate this work to my loving grandfather

Komatireddy Yella Reddy, who made me dream big & my dear friend

Jinesh K. J., who showed me the value of life

Acknowledgments

I would like to thank Dr. C. V. Jawahar for his role as a guide and mentor to me in the past six years. I gratefully acknowledge the motivation, technical and philosophical guidance that he has given me throughout my undergraduate and graduate education. I thank Prof. R. Manmatha for his insightful inputs into my research work.

I thank Prof. P. J. Narayanan, Dr. Anoop M. Namboodiri & Dr. Jayanthi Sivaswamy for providing an excellent research environment at CVIT, IIIT Hyderabad. I would also like to thank the staff of CVIT, specially Administrative assistant Mr. R. S. Satyanarayana, and Project Coordinator Mr. A. Phaneendra. I am grateful to Pramod Sankar and the Annotation team Rajan Bharani and co. for providing the material and the groundtruth for my research work. I am also grateful for CVIT for funding my graduate education.

Its been my pleasure to actively collaborate and work with Sesh, Anand Kumar, Neeba, Karthika, and Jinesh. Without Chetan, Maneesh, Pratyush, Naveen, Pavan, Chandrika, Chhaya, Priyanka, Raman Jain, Suman Karthik, the CVIT lab would not have been creative, intelligent, noisy and fun. My best friends Avinash, Dileep, Harinath, Mahaveer, Manohar, Pradeep, Rakesh, Sreekanth and Rohit, made my life so enjoyable. I am so fortunate to meet these guys.

Above all I am thankful to my parents Padmaja, Ravinder Reddy, my sister Lahari, Ravinder Reddy. S and family and all those from CVIT and other research labs who had at some point in time helped me with their unconditional support and unbounded patience.

Abstract

The rapid evolution of information technology (IT) has prompted a massive growth in digitizing books. Accessing these huge digital collections require solutions, which will enable the archived materials to be searchable. These solutions can only be acquired through research in document image understanding. In the last three decades, many significant developments have been made in the recognition of Latin-based scripts. The recognition systems for Indian languages are very far behind the European language recognizers like English. The diversity of archived printed document poses an additional challenge to document analysis and understanding. In this work, we explore the recognition of printed text in Telugu, a south Indian language.

We begin our work by building the Telugu script model for recognition and adopting an existing optical character recognition system for the same. A comprehensive study on all the modules of the optical recognizer is done, with the focus mainly on the recognition module. We then evaluate the recognition module by testing it on the synthetic and real datasets. We achieved an accuracy of 98% on synthetic dataset, but the accuracy drops to 91% on 200 pages from the scanned books (real dataset). To analyze the drop in accuracy and the modules propagating errors, we create datasets with different qualities namely laser print dataset, good real dataset and challenging real dataset. Analysis of these experiments revealed the major problems in the character recognition module. We observed that the recognizer is not robust enough to tackle the multifont problem. The classifier's component accuracy varied significantly on pages from different books. Also, there was a huge difference in the component and word accuracies. Even with a component accuracy of 91%, the word accuracy was just 62%. This motivated us to solve the multifont problem and improve the word accuracies. Solving these problems would boost the OCR accuracy of any language.

A major requirement in the design of robust OCRs is the invariance of feature extraction scheme with the popular fonts used in the print. Many statistical and structural features have been tried for character classification in the past. In this work, we get motivated by the recent successes in object category recognition literature and use a spatial extension of the histogram of oriented gradients (HOG) for character classification. We conducted the experiments on 1.46 million Telugu character samples in 359 classes and 15 fonts. On this data set, we obtain an accuracy of 96-98% with an SVM classifier.

Typical optical character recognizer (OCR) only uses local information about a particular character or word to recognize it. In this thesis, we also propose a document level OCR which exploits the fact that multiple occurrences of the same word image should be recognized as the same word. Whenever

the OCR output differs for the same word, it must be due to recognition errors. We propose a method to identify such recognition errors and automatically correct them. First, multiple instances of the same word image are clustered using a fast clustering algorithm based on locality sensitive hashing. Three different techniques are proposed to correct the OCR errors by looking at differences in the OCR output for the words in the cluster. They are character majority voting, an alignment technique based on dynamic time warping and one based on Progressive Alignment of multiple sequences. In this work, we demonstrate the approach over hundreds of document images from English and Telugu books by correcting the output of the best performing OCRs for English and Telugu. The recognition accuracy at word level is improved from 93% to 97% for English and from 58% to 66% for Telugu. Our approach is applicable to documents in any language or script.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Background . . . 1
1.2 Motivation . . . 3
1.3 Review of literature of Indian Language OCRs . . . 4
1.4 Review of literature of Telugu language OCR . . . 5
1.5 Objectives of the study . . . 6
1.6 Significance of the work . . . 7
1.7 Major contributions . . . 7
1.8 Scope of the work . . . 8
1.9 Organization of thesis . . . 9
1.10 Summary . . . 10

2 Design and analysis of OCR system for Telugu language . . . 11
2.1 Telugu Language . . . 11
2.1.1 Script . . . 11
2.1.2 Character/Word formations . . . 15
2.1.3 Unicode rules for akshara formation . . . 17
2.1.4 Challenges . . . 20
2.1.4.1 European and Indian Language OCRs . . . 21
2.1.4.2 Challenges specific to Telugu language . . . 22
2.2 Architecture of the OCR system . . . 24
2.2.1 Preprocessing . . . 25
2.2.2 Segmentation . . . 26
2.2.3 Recognition . . . 26
2.2.3.1 Feature Extraction . . . 27
2.2.3.2 Classification . . . 28
2.2.3.3 Support Vector Machines . . . 29
2.2.3.4 Multi class Classification . . . 31
2.2.4 Post processing . . . 33
2.3 Adapted Telugu OCR system . . . 33
2.3.1 Converter . . . 35
2.3.2 Anti-converter . . . 37
2.4 Results . . . 38
2.4.0.1 Synthetic dataset . . . 38
2.4.0.2 Real dataset . . . 39

2.5 Testing . . . 40
2.5.1 Class A : Laser print dataset . . . 40
2.5.2 Class B : Good real dataset . . . 41
2.5.3 Class C : Challenging real dataset . . . 41
2.5.4 Error Analysis . . . 42
2.6 Discussions . . . 47

3 Multifont character classification in Telugu . . . 49
3.1 Introduction . . . 49
3.2 Classification Architecture . . . 51
3.2.1 Feature Extraction . . . 51
3.2.2 Character classification . . . 52
3.2.3 Classifier Architecture . . . 52
3.3 Dataset . . . 53
3.4 Experimental Observations . . . 54
3.5 Features and Classifiers . . . 56
3.5.1 Spatial Histogram of Oriented Gradients (SPHOG) . . . 57
3.5.2 Principal Component Analysis . . . 59
3.6 Results and Discussions . . . 60
3.7 Summary . . . 62

4 Error correction using post-processing techniques . . . 64
4.1 Introduction . . . 64
4.1.1 Dataset and Challenges . . . 65
4.2 Previous Work . . . 67
4.3 Word-Image Clustering . . . 69
4.3.1 Locality Sensitive Hashing . . . 69
4.3.2 Clusters from Hashing . . . 70
4.3.3 Features for Word-Image Clustering . . . 71
4.3.4 Evaluating Clusters . . . 71
4.4 Word Error Correction . . . 72
4.4.1 Fusion of OCR results . . . 73
4.5 Alignment based Error Correction . . . 75
4.5.1 Pairwise Dynamic Time Warping (DTW) . . . 76
4.5.2 Progressive String Alignment . . . 76
4.6 Experiments and Results . . . 78
4.6.1 Controlled Experiments . . . 78
4.6.2 Results on Book Dataset . . . 79
4.6.3 Error Analysis . . . 80
4.6.4 Effect of Clustering Errors . . . 81
4.7 Summary . . . 82

5 Conclusions . . . 85

Appendix A: Appendix . . . 88
A.1 Characters List . . . 88

Bibliography 93

List of Figures

Figure Page

1.1 Digitizing the books by scanning them into images 1

2.1 A new word is formed when a symbol combines with the word [3] 12
2.2 Composition of a word from multiple morphemes 12
2.3 Vowels in Telugu language 13
2.4 Modifiers of corresponding vowels [Vowel diacritics] 13
2.5 Consonants in Telugu language 14
2.6 Conjunct Consonants of corresponding consonants 14
2.7 Effect of vowel diacritics on the consonant 'ka' 15
2.8 Consonant conjuncts with the consonant 'ka' 15
2.9 Figure with n=0, n=1, n=2 16
2.10 Akshara with 1, 2, 3, 4 components 16
2.11 Unicode Table of Telugu language 17
2.12 Unicode Table of Telugu language 18
2.13 Unicode rules for symbols 19
2.14 Unicode rules for aksharas 20
2.15 Examples of Indian Language Scripts [62] 22
2.16 Cuts introduced by the fonts 23
2.17 Merges introduced by the fonts 23
2.18 Spatial distribution of components in Telugu 23
2.19 Architecture of OCR system for Telugu 24
2.20 Skew correction of a Telugu image 25
2.21 Distance-Angle plot nearest neighbor components (a) and (b): Note the clusters of nearest neighboring components at angles $\pm$ 90 and 0 degrees with the horizontal in English. Component Size Graphs in (c) and (d): Note that the component sizes vary widely in Telugu compared to English. 26
2.22 Connected components 27
2.23 Overview of Recognizer 27
2.24 Decision boundary separating the training data 29
2.25 SVM Classifier 30
2.26 DDAG architecture 32
2.27 Overview of the converter 35
2.28 An example showing the working of the converter module, the process is generally stopped after the output of the converter. 35
2.29 Both the component arrangements should generate the same Unicode 36

2.30 Occurrence of components in a different order for the same akshara 36
2.31 Occurrence of components in a different order for the same akshara 37
2.32 Lower components in Telugu language 37
2.33 Upper components in Telugu language 37
2.34 Similar characters in Telugu 42
2.35 Original word and Obtained word 43
2.36 Input and output of segmentation 43
2.37 Missing components because of segmentation 44
2.38 Cuts in aksharas 44
2.39 Merge of aksharas 44
2.40 Blobs in word image 45
2.41 Noise because of faulty segmentation from Figure 2.35 45
2.42 Multiple fonts in a page 46

3.1 Challenges in character classification of Telugu. First row shows similar character pairs from Telugu. Second row shows how the same character gets rendered in different fonts 50
3.2 General architecture of the character classification module 51
3.3 A glimpse of a few images belonging to the same class in the dataset 55
3.4 Variation in accuracy with increase in number of fonts 55
3.5 Accuracy and confused pairwise classifiers 56
3.6 Toy example of constructing a three-level pyramid. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, we subdivide the image at three different levels of resolution. Next, for each level of resolution and each channel, we count the features that fall in each spatial bin. 58
3.7 PCA example: 1D projection of 2D points in the original space. 59
3.8 Accuracy and number of Eigen vectors 61
3.9 Eigen vectors and their magnitude. 62

4.1 An example Telugu document image in our collection. This particular book is the Telugu version of the famous novel *Don Quixote* 65
4.2 Examples of visual complexity in the Telugu script. In (a), all the components with the same color belong to the same character. (b) The main consonant that forms the character is shown in orange, its vowel modifier is shown in green, and the consonant-conjunction is shown in red. Note that the related components could be spatially distributed around the main consonant. Example pairs of visually similar characters are shown in (c). In the second pair, the completion of circle on the left-bottom changes the character. In the last two pairs, the dot in the middle distinguishes between them. . . . 66
4.3 A sample segment from a document image in our collection. Note the large amount of degradations in the characters. Some of the words such as the second word on lines 1 and 2 are not even human readable. 68
4.4 Example clusters obtained using LSH over word images. Noisy variants are highlighted in red. 71
4.5 Extraction of profile features and their fixed length versions. For each word image, Upper, Lower, Projection and Transition Profiles are computed. These features are run through a DFT, and the top 84 coefficients chosen for each. 72
4.6 Cluster performance across various radii for querying LSH, of English and Telugu books. 73

4.7 The Fusion operator for word error correction. Erroneous OCR results from multiple instances of the same word are fed to it. The Fusion operator then predicts the correct recognized text for these words together. 74
4.8 Character Majority Voting based word correction. 1st column shows the word images in the cluster, 2nd column gives the OCR outputs. The remaining columns show the alignment across the strings. Note variants and errors in red. 74
4.9 Example word error correction. 1st column word images in cluster, 2nd OCR output, 3rd CMV and 4th DTW output. Note variants and errors in red and corrections in green. 77
4.10 Progressive String Alignment based error correction. The first 4 rows are aligned outputs of the symbols. Last row is the output obtained if at least a majority of the characters agree. The final string is correctly identified in this case. 78
4.11 Effect of Number of words in cluster over the word recognition accuracy. Over synthetic datasets, the word error correction improves with more points in the cluster. The Dynamic Programming based error correction always outperforms the CMV. 79
4.12 Effect of Word Length on word recognition accuracy. The OCR performs poorly with longer strings, since there are more characters that the OCR could go wrong at. The word error correction seems to correct this trend, as seen from the fairly steady plot. . . 80
4.13 Effect of cluster size on word error correction accuracy of Telugu and English Books. It can be seen that the improvement in word accuracies over the base OCR is significantly larger for bigger clusters. 82
4.14 An example where the errors were corrected by both CMV and DTW methods, inspite of errors in more than half the words in the cluster. 83
4.15 An example Telugu word that could not be corrected, because of large number of OCR errors for one particular character. 83
4.16 A typical example where CMV fails, but DTW correctly aligns and rectifies the OCR errors. 84

A.1 Classes used in Telugu OCR 88
A.2 Classes used in Telugu OCR 89
A.3 Classes used in Telugu OCR 90
A.4 Classes used in Telugu OCR 91

List of Tables

Table Page

2.1 Results on synthetic dataset with different splits 39
2.2 Results of Telugu OCR on real book data 40
2.3 Results of Telugu OCR on Laser print dataset 41
2.4 Results of Telugu OCR on Good real dataset 41
2.5 Results of Telugu OCR on Challenging real dataset 42

3.1 Fonts used in the dataset 54
3.2 Comparative results on a smaller set of Telugu Multifont Data Set 61
3.3 Classification accuracy: No of classes = 359, No of samples = 1453950 62

4.1 Effect of word length on the word error correction of the different algorithms. See Section 4.6 for details. 81
4.2 Comparison of word error correction between LSH based clustering and pure clusters. Note that the difference between the word recognition accuracies are insignificant for English, and around 4%-5% for Telugu. 84

Chapter 1

Introduction

1.1 Background

This is an era of digitization. Many attempts are being made to digitize knowledge. Since books are the source of knowledge, one can argue that digitizing the books is digitizing knowledge. Digital information enables effective access, reliable storage and long term knowledge. This led to the emergence of Digital Libraries, which are huge collection of document images. Advancements in information technology like cheap storage and imaging devices played a key role in their development. Efforts are being made to digitize literary, artistic and scientific works. Apart from better access and easier preservation, they can also be made available to the global community freely.

Figure 1.1 Digitizing the books by scanning them into images

Michael Hart came up with an idea of making the famous texts available freely to everyone through Project Gutenberg in 1971 [8]. It is the oldest digital library named in the honor of Johannes Gutenberg, the 15^{th} century German printer who started the type printing press revolution. Large scale digitization projects are underway at Google, the Million Book Project [9], and Internet Archive [6]. With continued improvements in book handling and presentation technologies, digital libraries are rapidly growing in

popularity propelled by Google [1], Yahoo! [10], and MSN [7]. Just as libraries have ventured into audio and video collections, so have digital libraries such as the Internet Archive.

The advantages of a digital library as a means of easily and rapidly accessing books, archives and images of various types are now widely recognized by commercial interests and public bodies alike. Some of the advantages of the digital libraries as summarized by Million [57] are as follows:

- **No physical boundary** : The user of a digital library need not to go to the library physically; people from all over the world can gain access to the same information, as long as an Internet connection is available.
- **Round the clock availability** : A major advantage of digital libraries is that people can gain access 24/7 to the information.
- **Multiple access** : The same resources can be used simultaneously by a number of institutions and patrons.
- **Information retrieval** : The user is able to use any search term (word, phrase, title, name, subject) to search the entire collection.
- **Preservation and conservation** : Digitization is not a long-term preservation solution for physical collections, but does succeed in providing access copies for materials that would otherwise fall to degradation from repeated use.
- **Space** : Traditional libraries are limited by storage space, digital libraries have the potential to store much more information, simply because digital information requires very little physical space to contain them and media storage technologies are more affordable than ever before.
- **Added value** : Certain characteristics of objects, primarily the quality of images, may be improved. Digitization can enhance legibility and remove visible flaws such as stains and discoloration.

Extensive research is being carried out to make the digital content accessible to users through indexing and retrieval of relevant documents from the collection of images [20, 76], text [23, 5], video and audio [21, 35]. Despite the advantages of having digital libraries, the major issue is how to make these large document collections searchable. The documents need to be searchable, to access the relevant information in them. The current trends in the application of Document Image Retrieval techniques in the field of Digital Libraries are summarized in [55]. The associated key technological issues are how to make these large document collections searchable to access relevant information in them. Here we are only concerned with the textual content in the data. The direct way to make the documents searchable is by using the Optical Character Recognizers. The document image is fed into an Optical Character Recognizer, which gives Unicode(*text*) as output. This can be stored and used for retrieval. The task of searching the documents is thus reduced to searching the text. Optical Character Recognizers

can effectively solve this problem provided it has a high recognition rate, i.e., recognizing the text in the documents correctly. Hence, success of text image retrieval systems mainly depends on the performance of optical character recognizer.

1.2 Motivation

With the emergence of digital library, there has been a huge increase in the digitization of the documents. The enormous increase in the document image collections requires immediate solutions to enable the users access these collections. Extensive research is being carried out to make this huge data pool accessible and make the collection put to optimal use. This requires a vast knowledge of document understanding, specifically in the field of image recognition.

Optical Character Recognizers are at the heart of these retrieval systems. There has been a significant improvement in the recognition of the documents in Latin-based scripts and some of the oriental languages [59, 67, 79]. Many attempts are being made to develop robust document analysis systems [13, 19]. Latin scripts have high accuracy OCR systems with limited variations in fonts [45, 81]. Developing OCR systems is difficult compared to the traditional OCR systems. This is because of the presence of large number of symbols, spatial distribution of the script, lack of language models and the complexity of the language. The lack of proper segmentation techniques also adds to the problem. Because of these reasons and more, the Indian language OCRs are lagging behind [67]. These problems are additional challenges posed by the Indian language OCRs on European language OCRs.

Document related issues like quality, printing issues also pose challenge to the recognition. In addition to these the system must also adapt to the degradations like cuts, merges, etc. These are the common problems for OCR irrespective of the language. Attempts are being made to build OCR systems with high generalization capabilities [59]. However, when it comes to the suitability of converting an old book to text and providing a text-like access, the present OCRs are found insufficient [34]. The performance of these recognizers decline as diverse collection of documents (with unseen fonts and poor in quality) are submitted for recognition. Furthermore, the requirement of different OCR systems to deal with different languages is the drawback of recognition based approaches. Diversity of documents (that vary in quality, scripts, fonts, sizes and styles) that are archived in digital libraries also complicate the problem of document analysis and understanding to a great extent.

The motivation is to design robust recognizers for documents varying in quality, fonts, sizes and styles. In this work we work on novel approaches for accessing the content of printed document image collections that vary in quality, scripts, and printing.

Recently lot of effort is being put into the development of OCRs for the scripts of Indian languages [43, 14, 16]. Electronic representation and access of documents and other material in Indian languages have been lagging behind for many years. Developments in information technology in India has created a great demand for information access in Indian languages. The need for the content in the local languages is ballooning as Information Technology reaches beyond the English speaking sections of

the populace. This has a created a need for developing local language OCRs in developing nations like India. An OCR is a critical tool in creating electronic content in Indian languages as a rich tradition of literature and a vibrant publishing industry have existed in all of them for a long time.

Design of robust OCRs is still a challenging task for Indian scripts. The central module of an OCR is a recognizer which can generate a class label for an image component. Classification of isolated characters and thereby recognizing a complete document, is still the fundamental problem in most of the Indian languages. The problem becomes further challenging in presence of diversity in input data (for example, variations in appearance with fonts and styles.)

Recognition errors arising from the confusion of characters is a major problem for Indian language recognizers. Since many of the characters are similar, they randomly get misclassifieded. There are parts of characters where a minor degradation in one of them results in an appearance similar to the other. A number of Indian languages have a large number of components or symbols in each word. Thus even when component level recognizers perform well on very good documents, the word level, and hence the document level accuracies are not acceptable in practical situations. The use of language models for post processing is not very promising for many Indian languages like Telugu since the large vocabulary (number of possible words) makes dictionaries infeasible and it is necessary to model joint probabilities at the sub-UNICODE level.

1.3 Review of literature of Indian Language OCRs

The work on character recognition has started in the 1960's, where an OCR was developed to recognize characters like numbers and English alphabet. A comprehensive report on advancements in character and document recognition in the last 40 years is presented by Fujisawa [36].

Many attempts are being made to develop optical character recognizers for Indian Languages [37]. A brief discussion of some of the works on Indian scripts is reported in [16]. Recognition system of Indian scripts mainly use structural and topological features with tree based and Neural Network classifiers. Pal and Chaudhuri [67] present a review of the OCR work done on Indian language scripts. The following is an excerpt from their study on the state of recognition systems for Indian languages.

- *Devanagri* : Character recognition is Devanagri is based on K-nearest neighbor (KNN) and Neural Networks classifiers [15, 24]. Structural and run-based template was used as features. They reported an accuracy of 93%, and the dataset was not mentioned.

- *Bangla* : Tree classifier using stroke-based features were used in the OCR system for Bangla by Chaudhuri and Pal [25]. The accuracy was mentioned around 94-97%, and the dataset was not mentioned.

- *Gurmuki* : Lehal and Singh reported a complete OCR system for Gurmukhi Script [50]. They use two feature sets: primary features like number of junctions, number of loops, and their positions

and secondary features like number of endpoints and their locations, nature of profiles of different directions. They report an accuracy of 96.6% on a test dataset of 100 pages.

- *Tamil* : A Tamil [12] OCR using Time Delay neural Networks and Gabor Filters as feature, reports an accuracy of 90-97% on their test dataset of 2700 characters in 2003.

- *Malayalam* : An OCR system for Malayalam language is also available [67] in the year of 2003. A two level segmentation scheme, feature extraction method and classification scheme, using binary decision tree, is implemented. This system is tested on around 500 printed and real pages, and report an accuracy of 94 - 97%. Not enough technical details and analysis available for this system.

1.4 Review of literature of Telugu language OCR

The first known recognition system for Telugu language was reported by Rajasekhar *et. al* [71] in 1977. The problem was then not adequately looked into, but for some isolated investigations. Rajasekhar *et. al* [71] developed a two stage syntax aided character recognition system with 50 primitive features. Sequential template matching was used to identify these features. The individual characters were then represented by joining and super imposing these primitives. The characters were then recognized by a process called curve coding. Even though the experimental results were promising, its validity on the practical data set was not investigated.

Neural Networks were also used for the purpose of recognition. Sukhaswami *et. al* [80] used optimized memory model of Hop-field neural network for recognition. They overcame the limitations in storage capacity by combining multiple neural networks which work in parallel.

Pujari *et. al* [69] designed a recognizer which exploited the inherent characteristics of the Telugu Script. They proposed a method which uses wavelet multi-resolution analysis for the purpose of extracting features and associative memory model for recognition tasks. This method avoids feature extraction process and uses the inherent characteristics of the character by a clever selection of Wavelet Basis function which extracts the invariant features of the characters. It has a Hopfield-based Dynamic Neural Network for the purpose of learning and recognition. However, though it is normally not suitable for image processing application.

Negi *et. al* [65] presented a practical approach to Telugu OCR which limited the number of templates to be recognized to just 370, avoiding issues of classifier design for thousands of shapes or very complex glyph segmentation. They used a compositional approach using connected components and fringe distance template matching for testing the OCR. They reported an accuracy of around 92% .

Jawahar *et. al* [43] described a character recognition process from printed documents containing Hindi and Telugu text. The bilingual recognizer was based on Principal Component Analysis followed by support vector classification. This attains an overall accuracy of approximately 96.7% on synthetic dataset.

Even though the accuracies are reported around or above 90%, they fail on the real data sets. These accuracies were reported on specific fonts used in the training set, and generally on small datasets.

1.5 Objectives of the study

In this thesis work, we address the following problems.

- Build a recognition system for the textual content in document images. To this end, an attempt is made to look into the following things.
  - Analyze Telugu script and their language rules for a building a script model for better document understanding.
  - Developing an Optical Character Recognition system for Telugu, where much research work needs to be focused.
  - Explore machine learning and pattern recognition algorithms for designing feature extraction, classification and post processing mechanisms for recognition of document images.
  - Performance analysis of the OCRs on synthetic, as well as real-life document images that are poor in quality and vary in printing.
- Addressing the multifont problem in Telugu, which is a major bottle neck for most of the recognizers.
  - Explore the invariance of feature extraction scheme with the popular fonts used in the print in character classification.
  - A study on the effect of statistical and structural features used on multiple fonts for character classification in Telugu.
  - Analyze the effect of spatial extension of the histogram of oriented gradients (HOG) used successfully in Object classification for character classification.
  - Test it on large number of fonts popular in Telugu.
- Improving the accuracy of the OCR by designing post-processors for classifying the confusing classes.
  - Analyze the effect of the symbol level classification errors on the word accuracy.
  - Incorporate the information from the entire document to reduce the word error rates.
  - Using the visual similarity and multiple occurrence of the words in improving OCR accuracy.
  - Use of pattern matching techniques to deduce the correct output for the confusing classes.
  - Developing a document level OCR which exploits the fact that multiple occurrences of the same word image should be recognized as the same word.

1.6 Significance of the work

This research work achieves significance due to the following main facts.

- Large collection of document images are emerging because of the emergence of the digital libraries all around the world. We need efficient tools to access these document collections for achieving optimal benefits from these collections. Optical Character Recognizer is one of the tool which helps in attaining this goal.
- The extensive research that is being done on the Optical Character Recognizers should also reach the Indian languages. We need to bridge the gap between the technology that is being used in the document retrieval system for Indian languages with the help of better and higher accuracy recognition systems.
- Document images in digital libraries are from diverse languages. With the advancements in information technology, there is a huge demand for the documents in local languages. This raises the need for ease accessibility of the available relevant documents in Indian languages.
- Multifont problem is a bottleneck for any language recognizer. It will be difficult to have a specific recognizer for a specific font. With the many diverse fonts in the digital library, it is essential to remove this stumbling block.
- The exponential growth of computation resources paves the way for recognition strategies based on machine learning schemes. The new generation OCRs need not recognize an isolated word. It can learn from its recognition experience obtained from the rest of the printed documents.
- A number of Indian language documents have a large number of components or symbols in each word. Thus even when component recognizers perform reasonably well on very good documents, the word level, and hence the document level accuracies are not acceptable in practical situations. There is a pressing needing for improvement in the accuracies of these OCRs.
- Recognition errors arising from the confusion of characters is a major problem for Indian language recognizers. Since many of the characters are similar, they randomly get misclassified. There are parts of character where a minor degradation in one of them results in an appearance similar to the other. With good post processors these issues can be addressed.

1.7 Major contributions

The main aim of the thesis is to address the problem of character classification in Telugu language. We perform a comprehensive analysis from a recognition point of view. Apart from analysis the OCR system for Telugu, we also try to solve the multi font problem in Telugu, there by trying to remove the bottleneck of Multifont. This can be used for solving the font problem in other languages. We also

try to design a post processor system which uses the information in the document to correct the OCR output. This can be used for languages where the traditional post processors fail. The challenges in this work include (i) Developing and analysis a character recognition system for Telugu, (ii) Solving the multi font problem with 15 popular fonts in Telugu on a challenging dataset of nearly 1.5 million, (iii) Designing a new post processor which uses the document information on Telugu and extending the results to English.

- Improved and analyzed the OCR recognition system for large number of classes on real book datasets.
- Buoyed by the success of the feature extraction techniques in Object Recognition, we tested the same to solve the multi font problem in Telugu along with the traditional feature extraction techniques.
- Experiments are conducted on 1453950 Telugu character samples in 359 classes and 15 fonts. On this data set, we obtain an accuracy of 96-98% with an SVM classifier.
- The word error rate of any optical character recognition system is usually substantially below its component or character error rate. This is especially true of Indic languages in which a word consists of many components.
- A document level OCR has been proposed which incorporates information from the entire document to reduce error rates.
- We demonstrate this technique for Telugu, where the use of post-processing techniques are not promising. We show a relative improvement of 28% for long words and 12% for all words which appear twice in the corpus.
- We also extend this technique to English language, where we improve the accuracy from 93% to 97%.

1.8 Scope of the work

We have modeled the Telugu language into 459 distinct classes. This is followed by developing an anti-converter and converter based, which is purely language based. Once the system has been refined and an end to end word OCR is developed, it was then tested on real book dataset. We have achieved an accuracy of 90.78% on this book dataset. To further analyze the errors, we have created a dataset of laser print dataset, good real dataset and a challenging data set. We achieved an error rate of 97.14%, 89.86% , 81.33% on these datasets.

We show that high classification accuracies can be obtained for character classification problem with the help of SPHOG-SVM combination. Left out confusions are associated only to a small percentage

of the classifier and a post-processing classifier with an uncorrelated feature set can successfully boost the overall classification performance. Our experiments are conducted on 1453950 Telugu character samples in 359 classes and 15 fonts. On this data set, we obtain an accuracy of 96-98% with an SVM classifier.

A document level word recognition technique is presented, which makes use of the context of similar words to improve the word level recognition accuracy. An error correction technique is presented to improve word accuracy of the raw OCR. An efficient clustering algorithm was used to speed up the process. The experimental results show that the word level accuracy can be improved significantly from about 58% to 66% for Telugu, and from 93.8% to 97% for English. The proposed technique may also be applied to other (Indian) languages and scripts. Future extensions may include the use of techniques to handle unique words by creating clusters over parts of words.

1.9 Organization of thesis

The thesis is organized into five chapters. The main focus of the thesis is to solve the character recognition problem in Indian languages, specially Telugu. We start with the OCR system for Telugu, and analyze the system. We then try to solve the multi font problem for Telugu. And finally we look at the new document level OCR, which incorporates information from the entire document to reduce word error rates. This is an overview of the work in this thesis.

In the second chapter we look into the Telugu language model. This includes the language constructs and the Unicode rules. We then look into the Optical Character Recognizer for Telugu. We analyze each module of the OCR system like segmentation, classification and post processing. We present the results of the working OCR on some of the document images. Then we perform qualitative analysis by working with different datasets. This gives us an insight into the errors of the OCR.

The third chapter is about the attempt to solve the multi font problem. Here we look into the design of robust OCRs with the invariance of feature extraction scheme with the popular fonts used in the print. Many statistical and structural features have been tried for character classification in the past. Motivated by the recent successes in object category recognition literature and use a spatial extension of the histogram of oriented gradients (HOG) for character classification. Our experiments are conducted on 1453950 Telugu character samples in 359 classes and 15 fonts. On this data set, we obtain an accuracy of 96-98% with an SVM classifier.

In the fourth chapter, we propose a document level OCR which exploits the fact that multiple occurrences of the same word image should be recognized as the same word. Whenever the OCR output differs for the same word, it must be due to recognition errors. We propose a method to identify such recognition errors and automatically correct them. First, multiple instances of the same word image are clustered using a fast clustering algorithm based on locality sensitive hashing. Three different techniques are proposed to correct the OCR errors by looking at differences in the OCR output for the words in the cluster. They are Character Majority Voting, an alignment technique based on Dynamic Time

Warping, and one based on Progressive Alignment of multiple sequences. We demonstrate the approach over hundreds of document images from English and Telugu (an Indic script) books by correcting the output of the best performing OCRs for English and Telugu. The recognition accuracy at word level is improved from 93% to 97% for English and from 58% to 66% for Telugu. Our approach is applicable to documents in any language or script.

1.10 Summary

In this section, we have discussed the importance and the evolution of digital libraries. We need to have tools to use these digital libraries effectively. OCR is one such tool which helps in converting these image collections to text databases. We have reviewed the literature of the existing Indian/European language OCRs. Differences between European and Indian language OCRs have also been discussed. The objective of this thesis is to build a recognition system for Telugu documents, analyze the defects of the OCRs and propose solutions to address these defects. This will help in improving the accuracies of the OCRs for any language (Telugu in particular), which will facilitate the development of efficient tools for Document collections.

We have also looked at the organization of the thesis briefly. In the next chapter we look at the design of a Telugu OCR. This depends on the effective language modeling and the architecture of the OCR. We give some insight into the language rules and model. We then look into the basic architecture of OCR and processes like preprocessing, segmentation, recognition and post processing. We test the performance of this adapted OCR system on real data set. We then perform qualitative analysis by working with different quality datasets. This gives us an insight into the errors of the OCR.

Chapter 2

Design and analysis of OCR system for Telugu language

2.1 Telugu Language

Telugu is a language primarily spoken in the state of Andhra Pradesh, India. It is the language with the third largest number of native speakers in India[3].Written in a script originated from the Brahmi script, Telugu is a south-central Dravidian language influenced by Sanskrit and Prakrit, as well as Urdu.

The earliest known inscriptions containing Telugu words appear on coins that date back to 400 BC. The first inscription entirely in Telugu was made in 575 AD and was probably made by Renati Cholas [3], who started writing royal proclamations in Telugu instead of Sanskrit. Telugu developed as a poetical and literary language during the 11th century. Until the 20th century Telugu was written in an archaic style very different from the everyday spoken language. During the second half of the 20th century, a new written standard emerged based on the modern spoken language. In 2008 Telugu was designated as a classical language by the Indian government[4].

Telugu is often considered an agglutinative language, wherein certain syllables are added to the end of a noun in order to denote its case. Grammatically, in Telugu, Karta (nominative case or the doer), Karma(object of the verb) and Kriya (action or the verb) follow a sequence. Telugu also has a Vibhakthi (preposition) tradition. Owing to the fact that virtually every word in Telugu ends with a vowel sound, European travelers in the 19th century often referred to Telugu as the “Italian of the East”[3].

2.1.1 Script

Onamaalu, or the Telugu alphabet consists of 56 symbols - 16 vowels, 36 consonants, and 4 other symbols. It is highly conducive for Phonetics. It has more letters than any Indian language. Some of them are introduced to express fine shades of difference in sounds. Today only 12 vowels, 31 consonants are being used. Since we are building a generic model, we have included all 56 characters in our model. The above mentioned numbers can vary.

Telugu is an inflecting language, Telugu nouns are inflected for number (singular, plural), gender (masculine, feminine, and neuter) and case (nominative, accusative, genitive, dative, vocative, instrumental, and locative). Here certain syllables are added to the end of a noun in order to denote its case: For example, the declension of *Ramudu* (masculine singular) is shown in the figure 2.1.

Nominative:	Ramudu	రాముడు	(డు du)
Accusative:	Ramuni	రాముని	(ని ni)
Instrumental:	Ramuniki	రామునికి	(కి ki)
Dative:	Ramuniki	రామునికి	(కి ki)
Ablative:	Ramudininchi	రాముడినించి	(నించి ninchi)
Genetive:	Ramunidi	రాముడిది	(ది di)

Figure 2.1 A new word is formed when a symbol combines with the word [3]

Telugu is often considered an agglutinative language; words are often composed of multiple morphemes. For example, *nuvvostanane* (if you say you will come) is formed from the individual words *nuvvu*, *vastaanu*, and *ante* as shown in the figure 2.2.

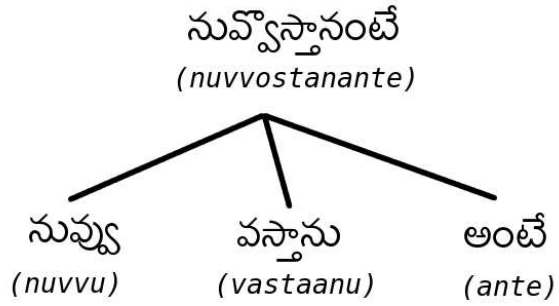


Figure 2.2 Composition of a word from multiple morphemes

New words can be formed when a symbol combines with a word or when multiple words combine. Telugu vocabulary is thus very huge. That is why it is very difficult to use a dictionary as a post processor to correct the output of the classifier.

The symbols in the language can be broadly divided into the following classes.

- Vowels
- Consonants
- Vowel diacritics
- Conjunct consonants

Telugu script is written from left to right and consists of sequences of simple and/or complex characters. The script is syllabic in nature, the basic units of writing are syllables. Since the number of possible syllables is very large, syllables are composed of more basic units such as vowels (achchu or swaram) and consonants (hallu or vyanjanam). The list of vowels are shown in the figure 2.3.

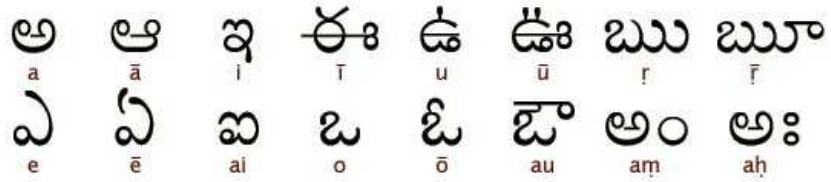


Figure 2.3 Vowels in Telugu language

Vowels can exist independently. When consonants combine with other vowel signs, the vowel part is indicated orthographically using signs known as matras or vowel diacritics. The shapes of maatras differ greatly from the shapes of the corresponding vowels. These are shown in the figure 2.4. Vowel diacritics, which can appear above, below, before or after the consonant they belong to, are used to change the inherent vowel.

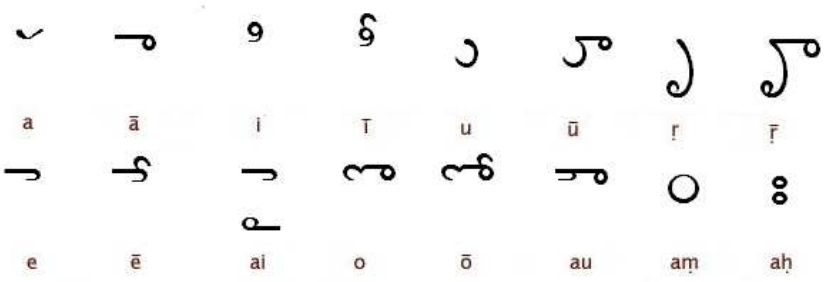


Figure 2.4 Modifiers of corresponding vowels [Vowel diacritics]

Figure 2.5 Consonants in Telugu language

The consonants in the consonant clusters take different shapes in reference to the context it is utilized in. Consonants are presumed to be pure consonants without any vowel sound in them. However, it is traditional to write and read consonants with an implied 'a' vowel sound. The consonants are shown in figure 2.5 The corresponding half consonants or conjunct consonants are shown in figure 2.6.

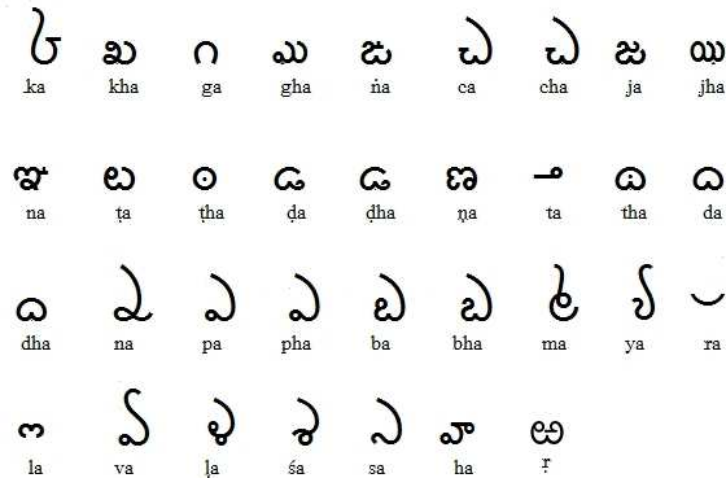


Figure 2.6 Conjunct Consonants of corresponding consonants

2.1.2 Character/Word formations

In this section we will be discussing about the character formations in Telugu. This will be helpful in how Unicode is written in Telugu which will be discussed in the coming sections. Let us look at some of the combinations.

- Vowels : They can exist independently. The vowels are shown in the figure 2.3
- Consonants + Vowels : The consonants shown in the Figure 2.5 is implied with a vowel 'a'. The effect of the vowel diacritics on the consonants can be seen in the 2.7

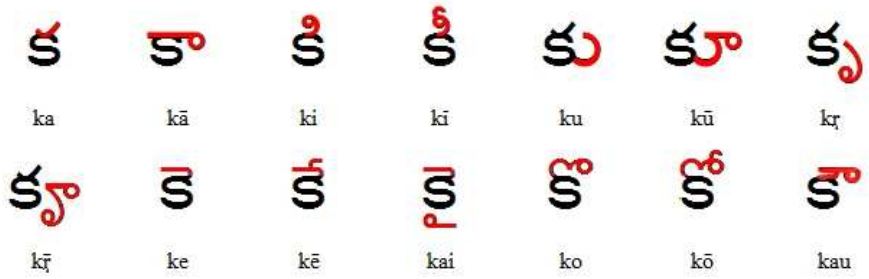


Figure 2.7 Effect of vowel diacritics on the consonant 'ka'

- Consonants + Conjunct Consonants : Consonant conjuncts shown in the fig 2.6 cannot exist independently. They need a consonant to form an akshara. The placement of the consonant conjuncts with the consonant 'ka' can be seen in 2.8

Figure 2.8 Consonant conjuncts with the consonant 'ka'

- Consonants + Vowel diacritic + (n x Conjunct Consonants) : Usually the value of n can be from 0 to 2. In the figure 2.8, the vowel diacritic used is 'a'. Aksharas with value of n ranging from n=0 to 2 is shown in 2.9.

Figure 2.9 Figure with n=0, n=1, n=2

Possible aksharas = Vowels + ((Consonants X Vowel diacritics) X (Conjunct X Conjunct .. n times)

  - For n=0, aksharas = 16 + (36 x 16) = 592
  - For n=1, aksharas = 16 + (36 x 16) + (36 x 16 x 36) = 21,328
  - For n=2, aksharas = 16 + (36 x 16) + (36 x 16 x 36 x 36) = 747,088

But in reality the numbers of aksharas are less than 10 thousand. The above numbers are only in theory.

The number of components in an akshara can have any value between 1 and 4. In the Figure 2.10 we can see the components varying from 1 to 4 in the akshara.

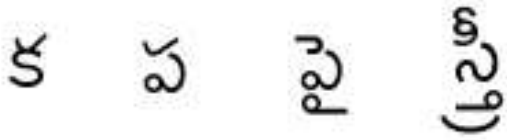


Figure 2.10 Akshara with 1, 2, 3, 4 components

2.1.3 Unicode rules for akshara formation

Unicode is a standard for computers to make them able to show text in different languages. Unicode standards are created by the Unicode Consortium. The goal is to replace current character encoding standards with a single worldwide standard for all languages. Currently, there are different ways to encode Unicode, UTF8 is the most common one used. In this section we will describe the general rules for Unicode in Telugu.

Hex code for the Unicode characters in Telugu is shown in figure 2.11 and 2.12.

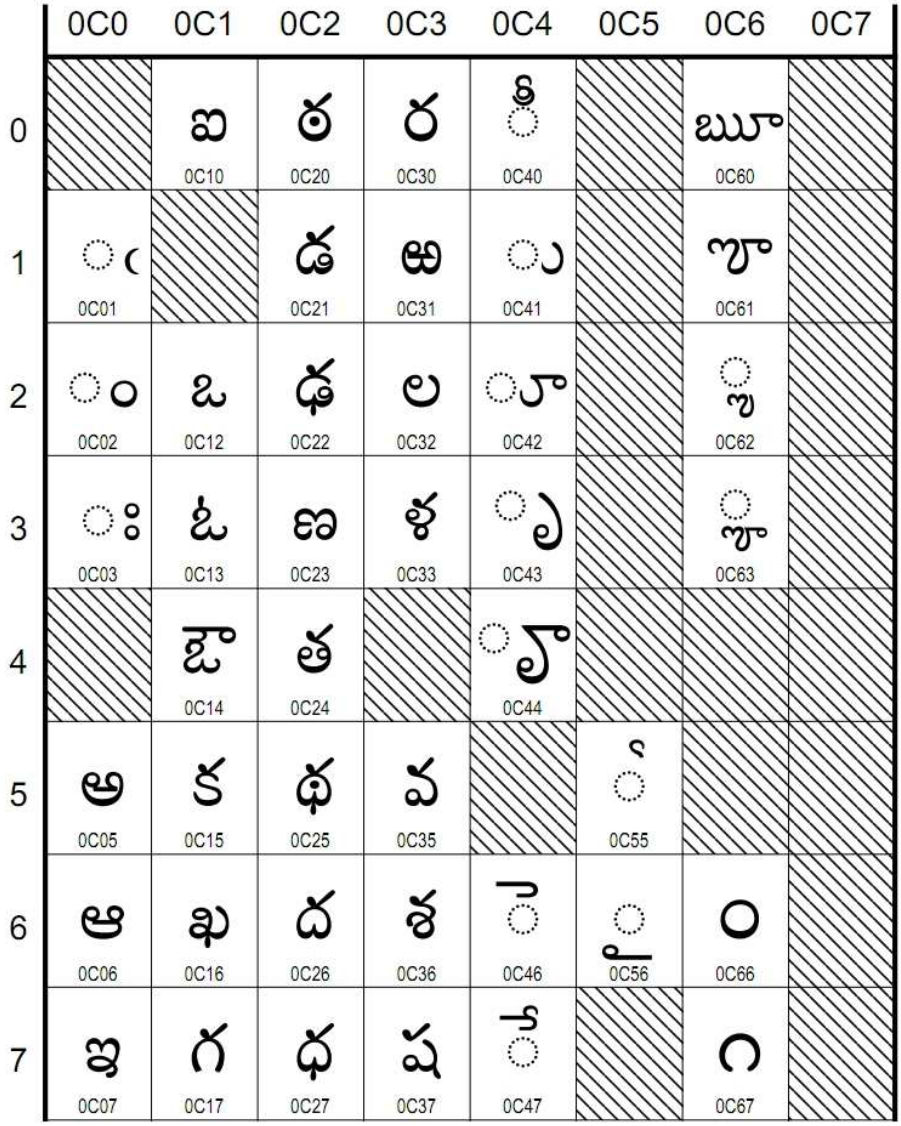


Figure 2.11 Unicode Table of Telugu language

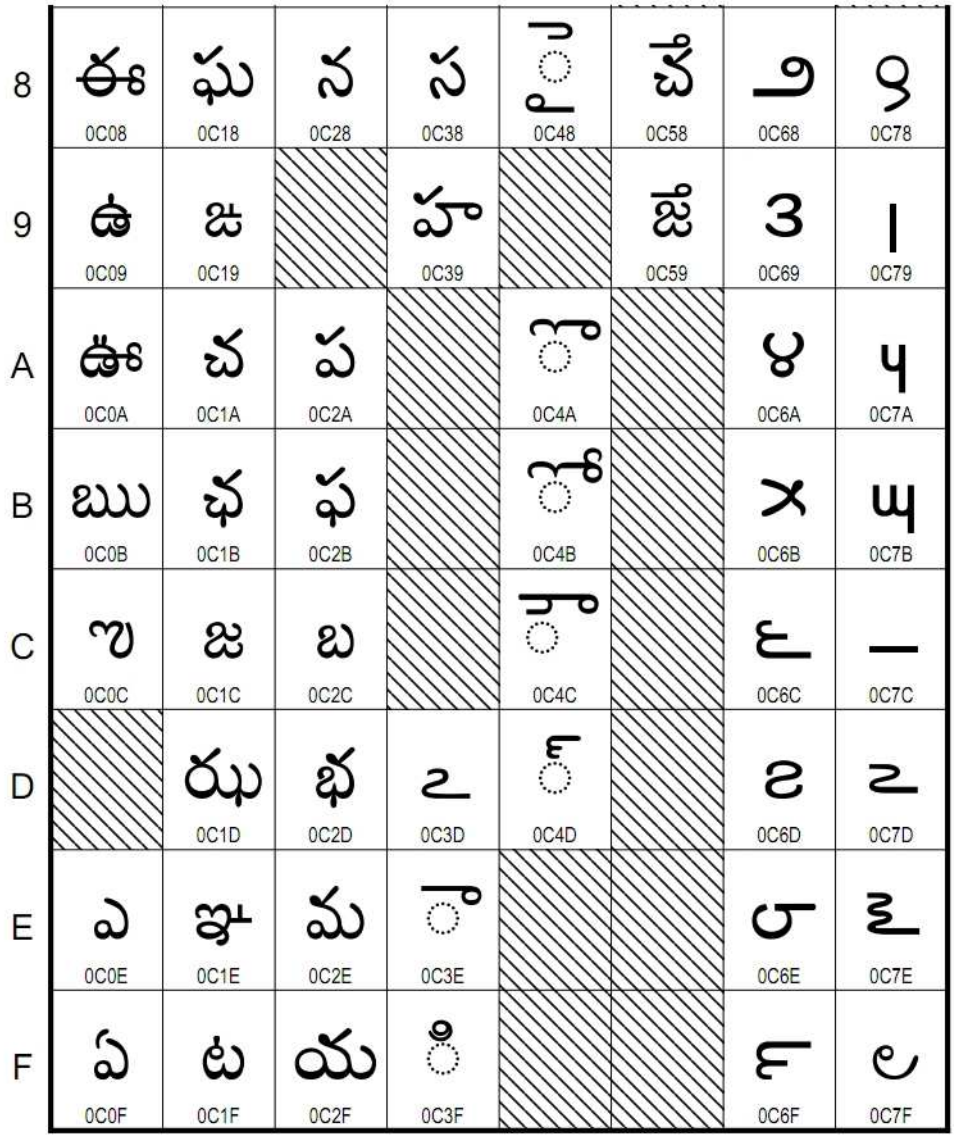


Figure 2.12 Unicode Table of Telugu language

The above figures contain the Hex Unicode for the vowels, vowel diacritics, and consonants. There is no Unicode for conjuncts. Unicode for conjunct is nothing but Unicode of halanth + Unicode of the consonant from which the conjunct was formed. So a consonant becomes a conjunct if the Unicode of halanth is written before it.

Basic rules for Unicode are described in 2.13. Vowel and consonant can exist independently, whereas matras and conjuncts require a consonant. These are rules for displaying a symbol.

Symbol	Rules	Character	Unicode (Hex)	Characteristics
Vowel	u(vowel)		0C05	*independent*
Vowel diacritic	u(vowel diacritic)		0C3E	*dependent*
Consonant	u(consonant)		0C15	*independent*
Conjunct	u(halant) +u(consonant)		0C4D;0C15	*dependent*

Figure 2.13 Unicode rules for symbols

After seeing how the symbols are assigned Unicode, now let us look into the rules for akshara Unicode. As mentioned earlier, matras and conjuncts can't exist independently, they need a consonant. So when a consonant is with a matra, the Unicode of consonant is written first and then the Unicode of matra as showed in 2.14 (a). Similarly when a conjunct comes along with a consonant, the Unicode of consonant is followed by the Unicode of halanth and then finally the Unicode of the conjunct. This can be seen in 2.14 (b). If a conjunct consonant exists with a vowel diacritic, then the matra is written at the last, 2.14 (c). When a consonant has more than one conjuncts, then the order in which the conjuncts occur doesn't matter. This can be seen in 2.14 (c) and (d). But the vowel diacritic must always come in the end, if one exists. These rules will be used in building the converter and the anti-converter. Converter converts the class labels into Unicode, and the anti-converter converts the Unicode into the class labels. More about converter and anti-converter will be explained in the coming sections.

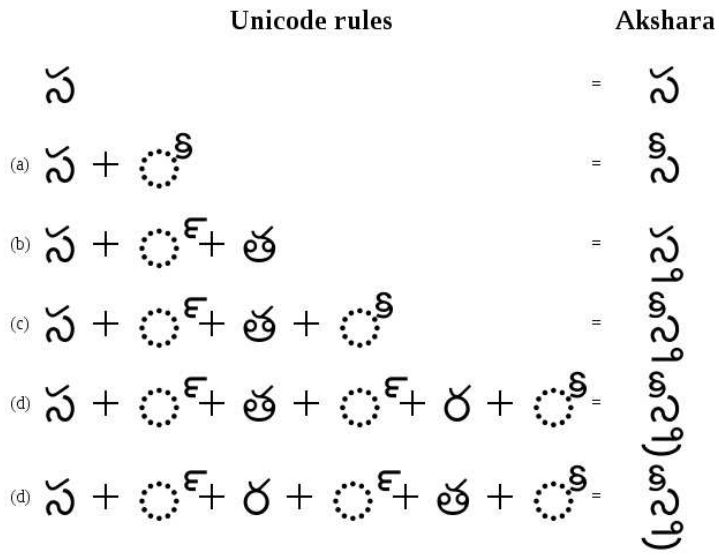


Figure 2.14 Unicode rules for aksharas

2.1.4 Challenges

OCR accuracy is affected by a number of factors. They may be external or internal. External errors are because of the errors in the input which are not handled by the OCR. Internal errors are the errors induced by the OCR system at various stages. George Nagy *et. al* [60] have conducted a survey on the common problems in the OCR system. These problems reduce the OCR accuracy dramatically. The general problems in the OCR are as follows:

- *Imaging Defects* : These defects are introduced between the printing process and the page image being submitted to the OCR.
  - The image can be noisy because of poor scanning.
  - Ink blobs merging with the distinct characters.
  - Cuts induced because of folding the paper or foreign material.
  - Printed text degradation because of aging, poor quality paper or ink.
  - Error in the image because of faulty printing.
- *Similar symbols* : The recognition of the characters is primarily done through their shape. Similar characters like the letter O and numeral 0 (zero) is difficult even for the humans. We can't recognize them distinctly, but can be identified with the context. Confusing classes increases the recognition error.
- *Typography* : Typography is the art, or skill, of designing communication by means of the printed word. Good typography is transparent: it reveals rather than hides the message. The OCR input

is often with layouts, typefaces, and type sizes that are far from ideal for OCR. The multi font problem in OCRs is still unsolved.

2.1.4.1 European and Indian Language OCRs

Recognition of documents in Indian languages is more challenging than recognition of documents in European languages. The complexity of the script, non-standard representations, and the amount of pattern recognition makes the building of Indian language OCRs, a challenging task. Some of the Indian language scripts are shown in the Fig 2.15.

- Indian languages have a large number of characters compared to European languages. This makes the recognition task challenging for the conventional classifiers.
- The visual similarity of the characters in Indian languages is very high. This increases the number of confusing classes, which in turn decreases the recognition accuracy.
- Research on OCRs for Indian languages has become active only for the last 10-20 years, while the research in OCRs for languages like English has been going on for the last 70-80 years. Because of this there are only a few standard databases and very less statistical information regarding the document image characters for Indian languages.
- Unicode/display/font related issues in building, testing and deploying working systems, slowed down the research in the development of character recognition system.
- Large number of characters with complex grapheme made the recognition difficult.
- Because of the large number of characters, there is an increase in computational complexity and memory requirements.

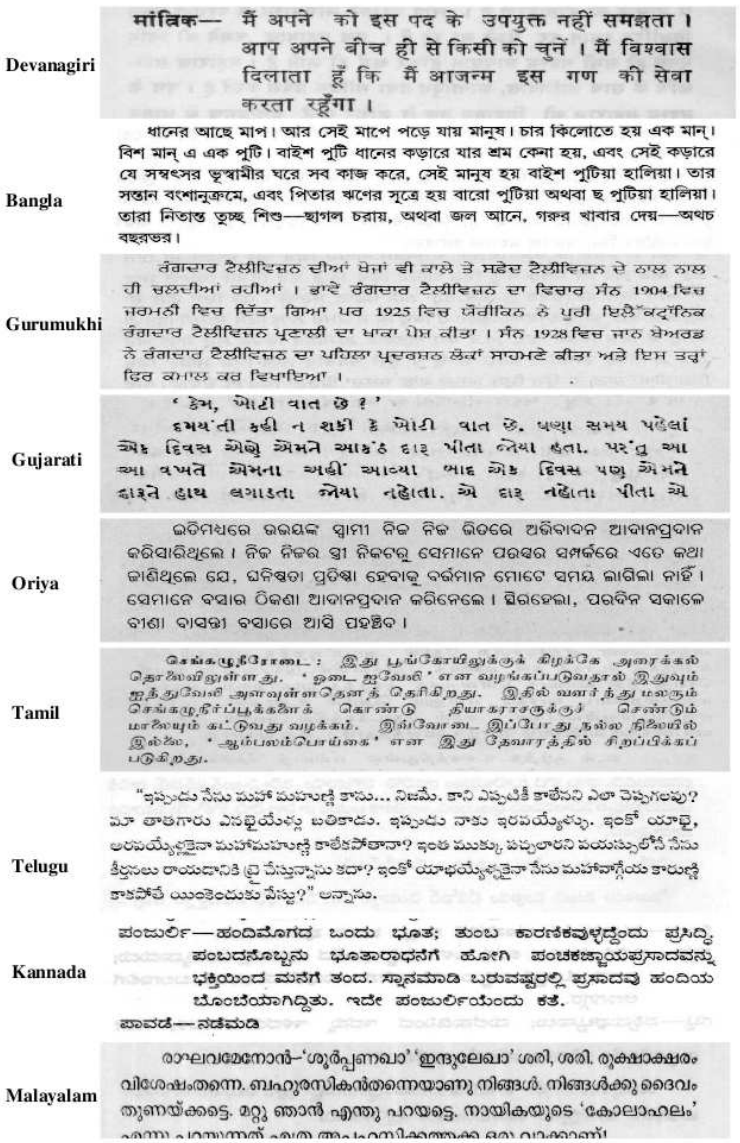


Figure 2.15 Examples of Indian Language Scripts [62]

2.1.4.2 Challenges specific to Telugu language

The main challenge in building the OCR for Telugu language is to deal with a large number of classes. Also we have to handle the large number of complex glyphs which are very similar in nature. However, recent advances in classifier design, combined with the increase in processing power of computers have all but solved the primary recognition problem.

- Rendering problems : This problem exists in most of the fonts. Because of the non-uniformity in the aspect ratio of the character glyphs, problems like non-uniform gaps appears in the documents. This results in touching characters or sometimes cut between the characters. These errors are then transferred from the electronic document to the print version. Cuts and merge examples because of the font are seen in the Fig 2.16, 2.17.

Figure 2.16 Cuts introduced by the fonts

Figure 2.17 Merges introduced by the fonts

- Spatial Complexity : The script is written from left to right in a sequential manner, many of these modifiers often gets distributed in a 1.5D manner (not purely left to right; they are also written top-to-bottom at places). Fig. 2.18 shows how Telugu script components occur spatially.

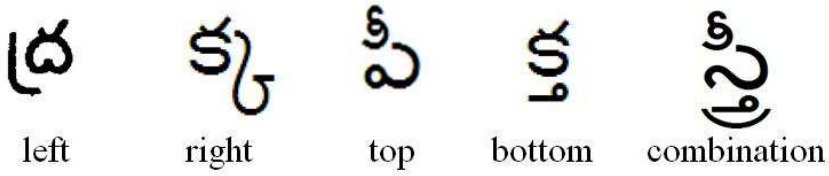


Figure 2.18 Spatial distribution of components in Telugu

- Lack of post processors for Telugu. Because of its vast vocabulary and language structure, it is very difficult to design a post processor for Telugu. As explained section 1.1.1, the number of words in Telugu are very huge.

2.2 Architecture of the OCR system

In this section, we present an overview of the OCR design used for OCRing the Telugu documents. OCRing is the process of extracting text from the document images. The input to the system is a scanned document, and the output of the system is Unicode or text in the document. The process of OCRing begins with pre-processing the scanned image. The pre-processing includes modules like noise removal, converting a color image into a gray scale image, thresholding the gray scale image into a binary image, and finally skew-correction of the image. Lay-out analysis is done after the pre-processing. This includes, various levels of segmentation like block/paragraph level segmentation, line level segmentation, word level segmentation and component/character level segmentation. Each symbol is passed as an input to the recognizer, where the features are extracted.

The recognizer has a base classifier with a very high performance. This recognizes the isolated symbols. Errors in the classifier will propagate, if not avalanched into the next phase. The performance hence is boosted by analysis of errors or confusions and providing extra knowledge to the system. But the symbol classifier cannot work in the presence of splits, merges or excessive noise. These are handled at the word level, which uses the symbol recognizer internally. The overall design of the OCR system for Telugu is shown in Figure 2.19. Lets look at the individual modules in detail.

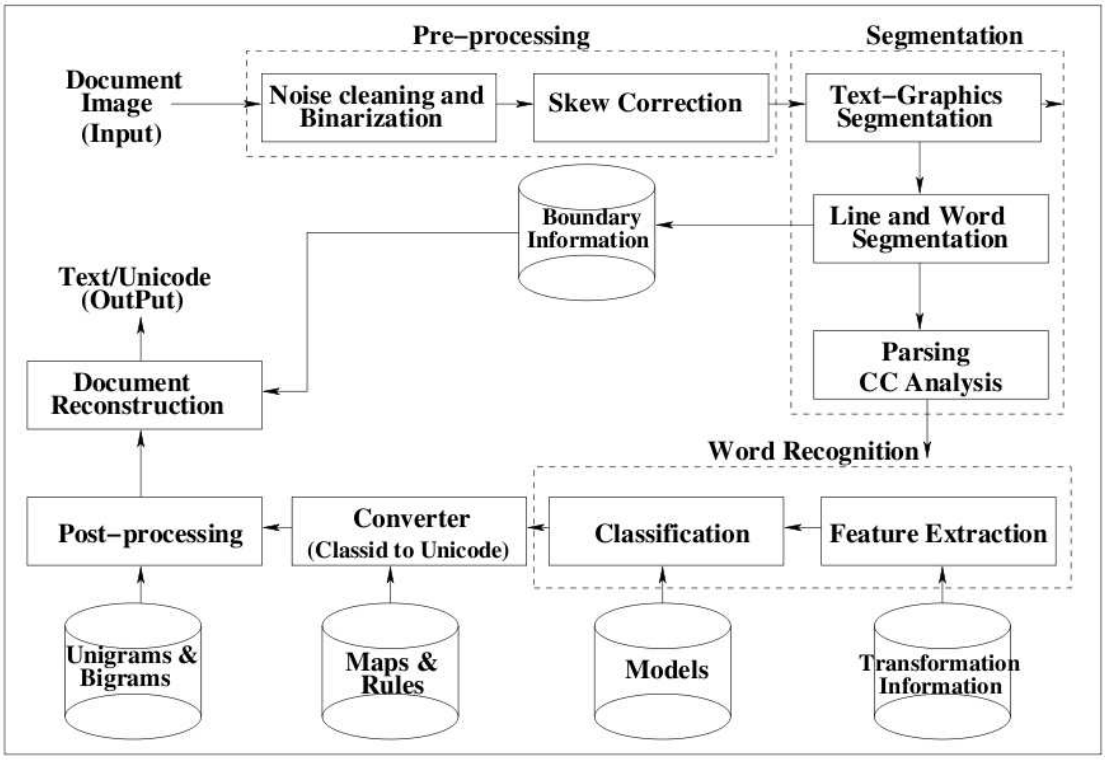


Figure 2.19 Architecture of OCR system for Telugu

2.2.1 Preprocessing

- This is the first step of OCRing. The image is first converted from a color image into a gray scale image ,i.e, black and white where the pixel values range from 0 to 255. Then using an adaptive threshold, the image is converted into a binary form, i.e, pixels with value 0 or 1.
- The image is then subjected to denoising. Noise is an undesirable by-product of image capture or scanning. It is a random variation of brightness or color information in the images. Methods like median filtering works well with most of the documents.
- Skew detection and correction are applied to the image. The process of detecting the slant of a misaligned image is skew detection. The method of straightening the image is skew correction. Telugu script is complex because of the glyph distribution, so methods like component distribution estimates which are used in English documents do not work well. Horizontal projection profile based techniques yields better results even though they require multiple lines of text.

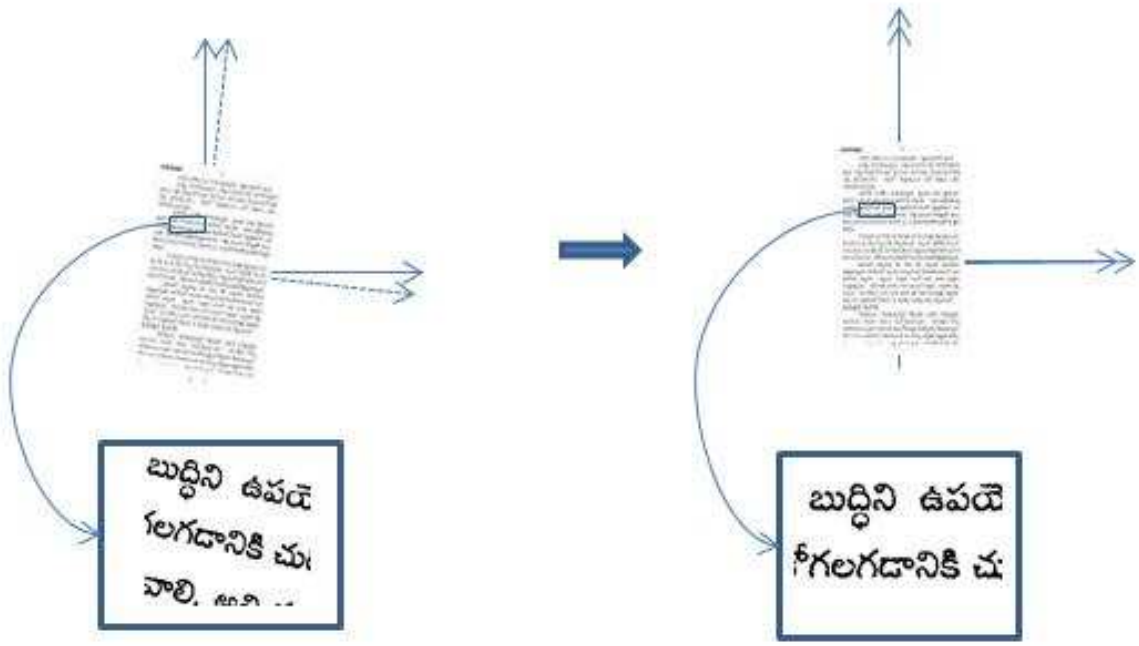


Figure 2.20 Skew correction of a Telugu image

2.2.2 Segmentation

Segmentation extracts homogeneous regions of text, graphics etc. from document images. Subdividing the text into words is of paramount importance in recognition systems as the input is set of components. The shape of connected components also provides critical information for the segmentation of text blocks. In roman scripts, components do not vary much in shape or size. This nature of the script helps in segmentation. However this is not possible in Telugu because the connected component sizes vary a lot. Sesh *et. al* [47] conducted the experiments on Distance-Angle plot nearest neighbor components in Telugu and English. They have also looked at the size of the components in Telugu and English. The results are as shown in Figure 2.21.

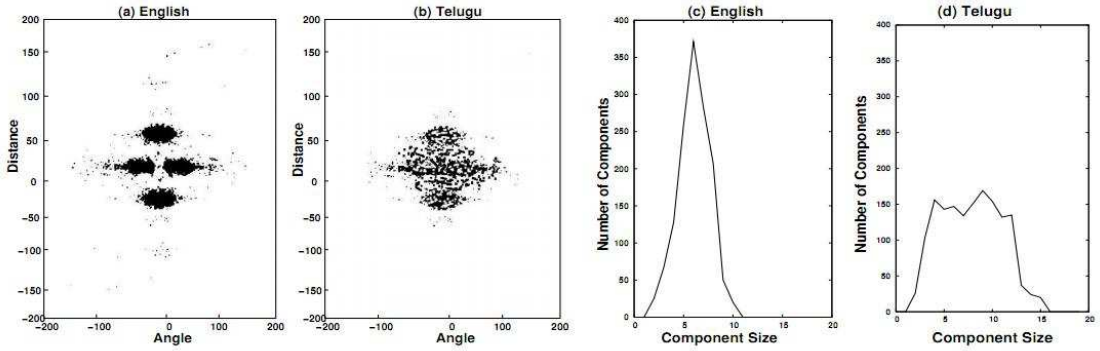


Figure 2.21 Distance-Angle plot nearest neighbor components (a) and (b): Note the clusters of nearest neighboring components at angles $\pm$ 90 and 0 degrees with the horizontal in English. Component Size Graphs in (c) and (d): Note that the component sizes vary widely in Telugu compared to English.

There are three stages involved in the text segmentation.

- *Line Segmentation*: This is to extract the lines from the pages.
- *Word Segmentation*: Extracting words from the obtained lines.
- *Symbol Segmentation*: Then finally symbol segmentation. This usually involves extracting the connected components in the word. Connected components are shown in Figure 2.22

2.2.3 Recognition

Recognition module is the heart of OCR. It consists of feature extraction, classification and word recognition modules. The input to the recognition module is a word image. The connected component(CC) analysis which was discussed in segmentation is used in the recognition module. This is a word recognizer, than a component recognizer. We cannot solve the problems like cuts and merges,

Figure 2.22 Connected components

and other degradations by components alone. By using the information from the components, these problems can be tackled at word level. So, even though CC is a segmentation technique, it is used in the recognition module. The overview of the recognition module is shown in Figure 2.23. Now let's look at the modules in detail.

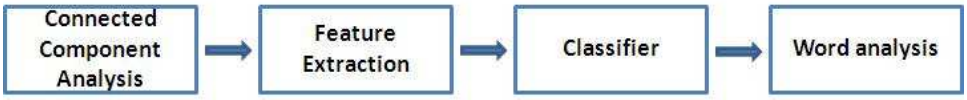


Figure 2.23 Overview of Recognizer

2.2.3.1 Feature Extraction

Devijver and Kittler [29] define 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". Depending on the specific recognition problem and available data, different feature extraction methods fulfill this requirement to a varying degree. The extracted features must be invariant to the expected distortions and variations that characters may have in that application. Moreover the "curse of dimensionality" [30, 42] cautions us that with a limited training set, the number of features must be kept reasonably small, if a statistical classifier is to be used.

Choosing a Feature extraction method plays a vital role in achieving high recognition accuracies. Many popular feature extraction techniques in character recognition can be found in [32]. In a recent work, Neeba and Jawahar [40] had looked into the performance of different features in Malayalam, they are directly expendable to other scripts. They showed through experiments, that the image(IMG) feature is the best. They have also shown that the statistical features perform much better than the shape based orientation features. This is mainly because of the presence of large number of classes.

So once we obtain the individual components after Connected Component Analysis, we normalize / rescale the image into 20 x 20. We then transform the 20 X 20 feature into a 400 X 1 feature. This 400 vector is used as the IMG feature.

However, the pattern classification becomes very cumbersome because of high dimensionality of the features. Therefore it is necessary to reduce the dimensionality of the feature vector without the loss of information. Dimensionality reduction techniques like Principal Component Analysis (PCA) and Linear Discrimination Analysis (LDA) transform the features into a lower dimensional space without much loss in the information.

Principal Component Analysis (PCA) is a transformation where the data set receives a new coordinate system, in which new axes follow the direction of greatest variance in the data set. Linear Discrimination Analysis (LDA) is to find a linear combination of features which characterize or separate two or more classes of objects. Both techniques look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data. PCA on the other hand does not take into account any difference in class.

2.2.3.2 Classification

Classification is a mapping from input data into defined subsumptive output categories. It is the task of assigning a label to input from a set of classes. This generally involves two tasks; training and testing. The classifier learns the association between samples and their labels from labeled samples in the training set. The classifier is then tested on the samples in the test set. The error analysis is done to evaluate the classifier's accuracy by comparing the output of the classifier and the label of the sample. In general, a classifier with minimal testing error is preferred.

Large numbers of classifiers exist in the literature. Nearest neighbor classifier is one of the most popular classifiers. K-nearest neighbor (KNN) is a supervised learning algorithm, which is an extension of the nearest neighbor classifier. Here the result of a new instance is classified based on majority of the category of K- nearest neighbors. Another classifier, which samples by a series of successive decisions, is a decision tree. We also have neural network classifiers: Multi-Layer Perceptron (MLP) and Convolution Neural Network (CNN). Naives Bayes (NB) classifier, is known to be mathematically optimal under restricted settings. Then there are Support Vector Machines (SVM). Support Vector Machines have become very popular for high accuracies and its ability to generalize.

The choice of classifier depends on its performance on the character symbols. Neeba *et. al* [40] have conducted an empirical study on the performance of various classifier schemes on Malayalam and Telugu languages. They have concluded that SVM with Decision Directed Acyclic Graphs (DDAG) gave the best results. We use SVM for classification task [68, 85]. SVM takes a set of input data and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-probabilistic binary linear classifier. They have the ability to identify the decision boundary with maximal margin. This results in better generalization, which is a highly desirable property for a classifier to perform well on a novel dataset. This scheme provides a compact representation of the dataset, since the design of SVMs allows the number of support vectors to be small compared to the total number of training examples. This results in the reduction of computation and storage requirements to

implement SVM classifier on large datasets. With smaller VC dimension, support vector machines are less complex, and have higher accuracies.

2.2.3.3 Support Vector Machines

Support Vector Machines are considered by many to be the best supervised learning algorithms. They are often binary classifiers, i.e., they solve a two class classification problem. SVMs are optimal margin classifiers. Margin is a “gap” separating data. Let us look at the intuition behind the margin.

Consider the Fig 2.24; in which X’s represent positive training samples (y=1), O’s represent negative training samples (y=-1). A decision boundary (also called the separating hyper plane) is also shown, and three points have also been labeled A, B and C. Notice that the point A is very far from the decision boundary. If we are asked to make a prediction for the value of y at A, it seems we should be quite confident that y = 1 there. Conversely, the point C is very close to the decision boundary, and while it’s on the side of the decision boundary on which we would predict y = 1, it seems likely that a small change to the decision boundary could easily cause the prediction to be y = -1. Hence, we are much more confident about our prediction at A than at C. The point B lies in-between these two cases, and more broadly, we see that if a point is far from the separating hyper plane, then we may be significantly more confident in our predictions. Given a training set, we have to get a decision boundary that allows us to make all correct and confident (meaning far from the decision boundary) predictions on the training examples.

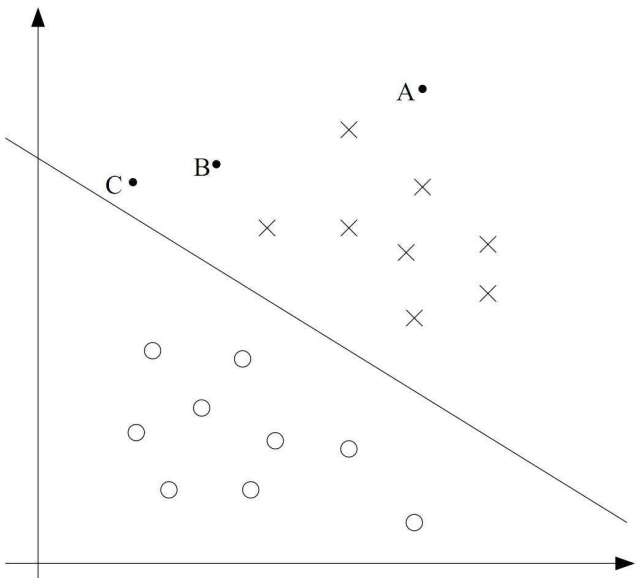


Figure 2.24 Decision boundary separating the training data

Consider labeled training dataset $[x_i, y_i]$ where i = 1,2, .. *l*, $y_i \in$ -1,1, and $x_i \in \Re^d$. Also d is the dimensionality of the dataset. Suppose we have a hyper plane that separates the positive from the negative examples. The points *x* which lie on the hyper plane satisfy *wx* + b = 0, where *w* is normal to the hyper plane and b is the offset. For the linearly separable case, the support vector algorithm simply looks for the separating hyper plane with largest margin; the point where the training data satisfy the following constraints.

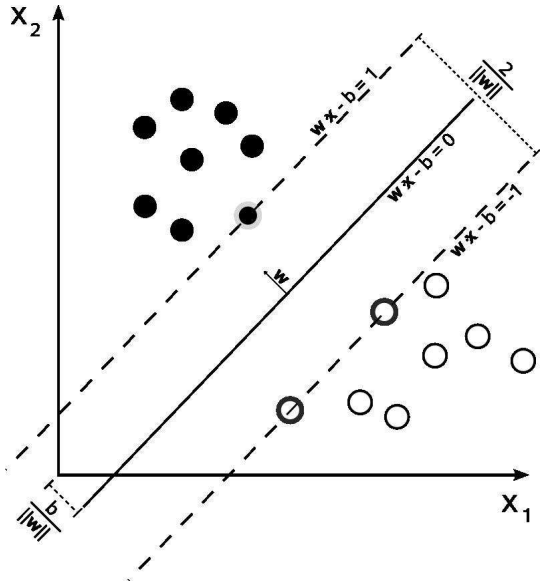


Figure 2.25 SVM Classifier

$$w.x_i + b \geq 1 \ for \ y_i = +1 \tag{2.1}$$

$$w.x_i + b \leq 1 \ for \ y_i = -1 \tag{2.2}$$

Identification of the optimal hyper plane for separation involves maximization of an appropriate objective function, i.e, solving the following quadratic optimization problem during training an SVM. Maximize:

$$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j K(x_i, x) \tag{2.3}$$

subject to the constraints $\alpha_i \geq 0, i = 1, 2, .., l$ and

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{2.4}$$

where α_i are the Lagrangian multipliers corresponding to each of the training data points x_i. The result of the training phase is the identification of a set of labeled support vectors x_i and a set of coefficients

α_i Support vectors are the samples near the decision boundary and most difficult to classify.
Class labels are

$$f(x) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x)) \tag{2.5}$$

where the function *K* is the kernel function. It is defined as $K(x, y) = \phi(x).\phi(y)$, where $\phi : \Re^d \rightarrow H$ maps the data points in *d* dimensions to a higher dimensional (possibly infinite dimensional) space H. For a linear SVM, *K(x, y) = x.y*. We do not need to know the values of the images of the data points in *H* to solve the problem in H. By finding specific cases of Kernel functions, we can arrive at Neural Networks or Radial Basis Functions. More detail discussion and a tutorial on SVM is available in [22].

Binary classifiers like SVM are basically designed for two class classification problems. However, because of the existence of a number of characters in any script, optical character recognition problem is inherently multi class in nature. The field of binary classification is mature, and provides a variety of approaches to solve the problem of multi class classification. Most of the existing multi class algorithms address the problem by first dividing it into smaller sets of a pair of classes $\binom{N}{2}$ and then combine the results of binary classifiers using a suitable voting methods such as majority or weighted majority approaches.

2.2.3.4 Multi class Classification

Multi class SVMs are usually implemented as combinations of two-class solution. The original technique for multi class classification using SVMs is known as the one-versus- rest technique [31]. More recently an improved technique, one-versus-one [38] has been developed. In the case of one-versus-rest, the i^{th} class is trained with all of the examples in the i^{th} class against all other examples, resulting in an N - 1 classifiers (where N is the number of classes). The class label of the test result is decided by combining the N classifier outputs using, say winner-take-all method. Where as, one-versus-one consists of a combination of binary classifiers, only two classes are used to train each classifier. In an N classification problem, one-versus-one technique results in N (N - 1) classifiers. The decision is made by combining these N (N - 1) classifier outputs using some voting strategy (e.g. majority vote, max wins, etc.).

Graphs and trees are effective means for integrating multiple classifiers [68, 70]. A tree is a simple graph such that there is a unique simple non-directed path between each pair of vertices of a graph. Decision trees are classical examples of a tree- based classifier combination scheme. In a decision tree classifier, decision is taken at each node based on some of the attributes and samples traverse down the tree along the selected path. Another possible tree-classifier is by designing the tree such that each node partitions the classes into mutually exclusive sets. A decision directed acyclic graph (DDAG) can be designed to solve the same problem in a better manner without partitioning into mutually exclusive sets of classes. This can reduce the risk in making a mistake by giving too much importance to the root node, or the nodes that are encountered initially. It has been shown that the integration of pairwise classifiers

using DDAG results in better performance as compared to other popular techniques such as decision tree, Bayesian, etc. [68].

DDAG is defined as follows [68]. Given a space X and a set of boolean functions $F = f : X \rightarrow 0, 1$, the class DDAG F of Decision DAGs, on N classes over F are functions which can be implemented using a rooted binary DAG with N leaves labeled by the classes where each of the K = N (N - 1)/2 internal nodes is labeled with an element of F. The nodes are arranged in a triangle with the single root node at the top, two nodes in the second layer and so on until the final layer of N leaves. The i^{th} node in j^{th} layer, j ¡ N , is connected to the i^{th} and i + 1-st node in the j + 1-st layer.

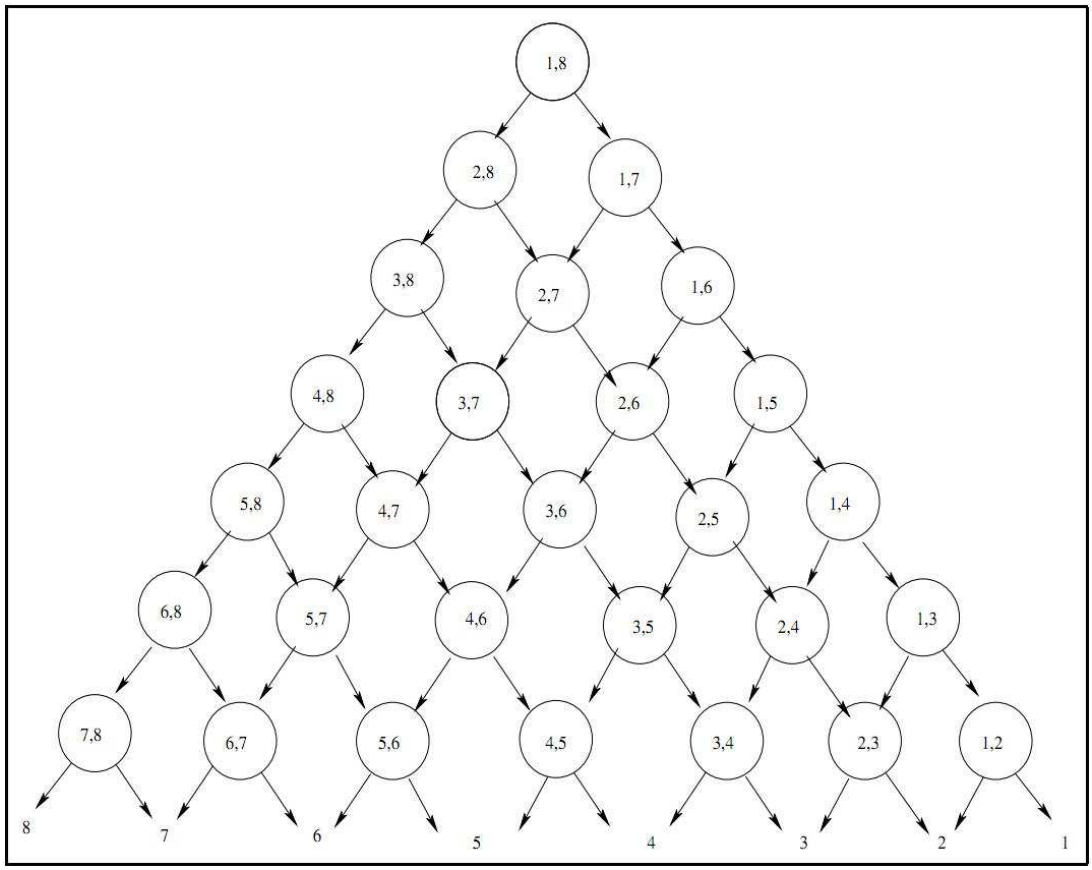


Figure 2.26 DDAG architecture

We construct a directed acyclic graph (DAG) using the one-vs-one method, where each node in the graph corresponds to a two-class classifier for a pair of classes [145]. The multi class classifier built using DDAG for an 8-class classification problem is depicted in Figure 2.26. It can be observed that the number of binary classifiers built for a N class classification problem is N (N - 1)/2. The input vector is presented at the root node of the DAG and moves through the DAG until it reaches the leaf node where the output (class label) is obtained. At each node a decision is made concerning to which class the input

belongs. For example, at the root node of Figure 2.26, a decision is to be made whether the input belong to class 1 or class 8. Samples from other classes get classified into left or right sub tree. Practically each node in the evaluation sequence removes one possible class. If it does not belong to class 1, it moves to the left child. In the same way, if it is does not belong to class 8, it moves to the right child. This process is repeated at each of the remaining nodes till it reaches the leaf node where the decision is reached to classify the input as one of the eight classes. The classifier creates two-class models that are used for classification and recognition purpose.

2.2.4 Post processing

The input of this module is the class labels given by the classifier module. The output of this module is the Unicode of the word, which is according to the language rules. Class labels are numbers ranging from 1 to n, in this case n=459. The class labels are then reordered according to the language, they are then mapped into unicode. This process of mapping the class labels into unicode is done by a module called converter. The reverse mapping of unicode to class labels is done by a module called anti-converter. We will discuss the converter and the anti-converter in detail, in the later part of this chapter. The unicode obtained from the converter module is sent to the word error correction module.

This is for correcting the OCR unicode output. This eliminates the Unicode errors which might be generated in the earlier module. Let's take the example of complete akshara *pa* again. In the Unicode mapping list Unicode of *pa* is Unicode of complete akshara *pa*, while the Unicode of *talakattu* is the Unicode of *talakattu*. So, while forming the Unicode of the word, the converter will give the output as Unicode of *pa* + *talakattu*. But the Unicode of *talakattu* is not necessary. But if there is a vowel form of *i* instead of *talakattu* then we cannot ignore the Unicode of *i*. So such language issues are handled in this module.

Not only Unicode corrections, but other error correction techniques like unigram and bigram can also be used in this module. For languages which have a limited vocabulary like English, dictionary techniques can be used. Since, Telugu has a large vocabulary as explained in section 1.1.1, so we have proposed a post processing technique by fusing the results of word clusters. We will get into the details of this technique later in the thesis. Karthika *et.al* [58] proposed a post processing scheme which uses statistical language models at sub character level to boost word level recognition results. They used a multi stage graph representation to formulate the recognition task as an optimization problem..

2.3 Adapted Telugu OCR system

We have adapted the Malayalam OCR [63] for building the recognition system for Telugu language. We have specifically worked on modeling the recognizer for the Telugu language. This involved many language specific issues like the script model and classes in Telugu, etc.. Some of the contributions in this module are as follows

- Analyzing Telugu script and their language rules for building a script model. We have to identify the distinct components in Telugu. After thorough examination of the script, we have identified 442 distinct classes. But to make the converter module, simple we have extended the number of classes to 459. The extra classes which are added are the symbols which are already occurring in the 442 classes, but occur as half consonants. The list of classes in Telugu can be found in the Appendix.

- Remodeling the Malayalam Optical Character Recognition system for Telugu language. This involved preparing the synthetic training dataset for building the model file. This dataset was used to preparing the model file for the Telugu language OCR. We have also extracted the component images from the ground truth, to prepare the classifier model to test on real data images.

  The classifier was trained with a minimum of 200 samples per each class. The total number of classes were 442. Since we are using an SVM classifier with DDAG architecture, the number of pair wise classes will be ${}^{n}C_{2}$, where n=442. Therefore the number of pairwise classifiers are ${}^{442}C_{2}$ = 97,461. The total number of samples used for training are 200 x 442 = 88,400 samples. The training set also includes the degraded samples, which are added to generalize the data.

- Exploring techniques for post-processing like reordering, converter and anticonverter. Telugu script is a unique script, as the components can appear on left, right, top or bottom of another component. We have discussed these difficulties in detail in section 2.1 . Apart from the reordering techniques, we also have devised techniques to convert the class label into Unicode and vice versa.

- Designing a testing mechanism for large scale testing of the recognizer. We have also created a mechanism to test the OCR effectively on both the synthetic and real dataset. With this we can analyze with following performance measures.

  - Symbol level accuracies : Percentage of the number of correctly classified components by total number of components. This is calculated using the edit distance concept [2].
  - Word level accuracies : Percentage of the number of correctly classified words by total number of words.
  - Unicode level accuracies : Percentage of the number of correctly classified Unicode words by total number of Unicode words.

- Performance analysis of the OCRs on synthetic, as well as real-life document images that are poor in quality and vary in printing. We have tested the OCR on large synthetic and real datasets. These results are discussed in detail in the next section.

  Now, lets look at the post processing module in detail.

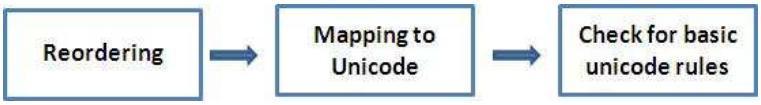


Figure 2.27 Overview of the converter

2.3.1 Converter

Converter is the module where the class labels are mapped into the Unicode. This is a language dependent module. The mapping is one-to-one, but it also has to include the Unicode rules. An example showing the functioning of the converter module is shown in Fig 2.28

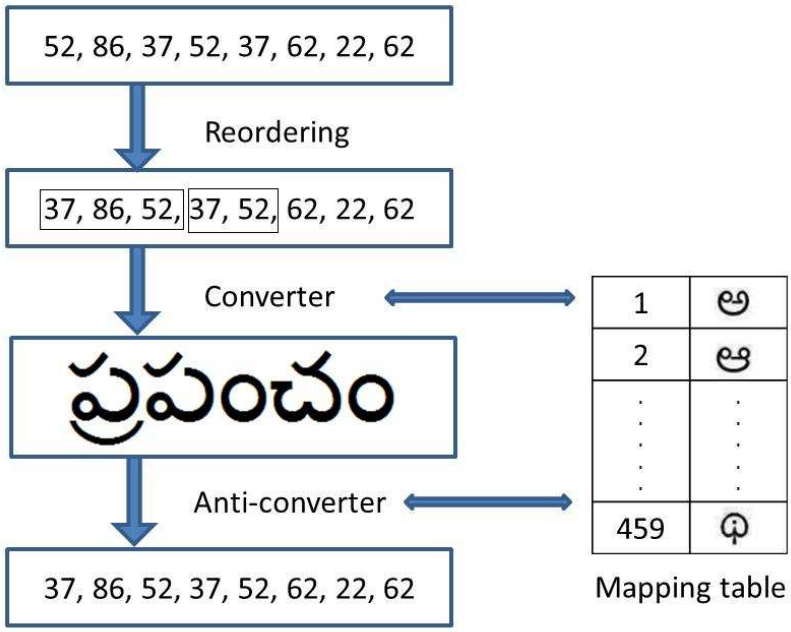


Figure 2.28 An example showing the working of the converter module, the process is generally stopped after the output of the converter.

So, before we map the labels to its corresponding Unicode, we have to reorder the components. The Unicode for different ordered components can be seen in Fig 2.29.

Now in Fig 2.30, the vowel form of 'a' ,i.e, *'talakattu'* need not always follow the consonant form of *'pa'* after connected component analysis. This creates confusion whether *'talakattu'* belongs to the component before that or after it. This confusion is difficult to resolve without using the properties of the components.

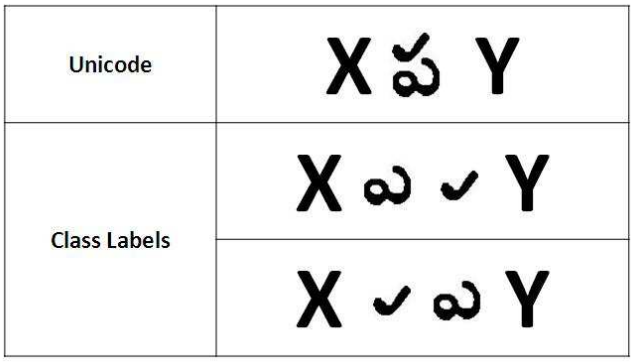


Figure 2.29 Both the component arrangements should generate the same Unicode

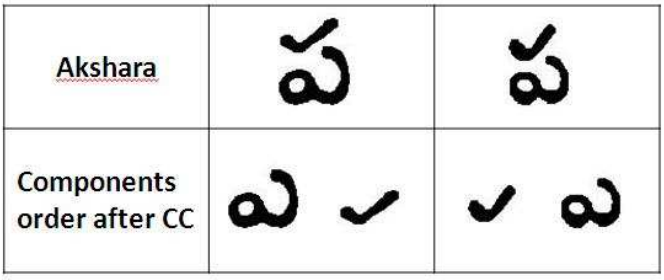


Figure 2.30 Occurrence of components in a different order for the same akshara

Now, consider the example shown in the Fig 2.31 . X is followed by ”pa” which is made up of 2 components and then Y. In the figure two cases showing the confusion with the different occurrence of the components in shown. It is difficult to resolve this problem, without using the knowledge about the components. We have a decision to decide whether *“talakattu”* can belong to X or pa and pa or Y in the shown example. The confusion increases if X or Y are made up of more than one components.

So, this module uses the knowledge about the components. The common categories among these are the lower components & upper components. In the previous example ”talakattu” is referred as lower component. A few more examples showing the lower components are shown in Fig 2.32 and upper components are shown in Fig 2.33.

We use the image properties, i.e., with the location or the placement of the upper/lower component to decide, to which component it belongs. But we cannot completely classify based on the image properties because sometimes the lower/upper components can be classified as belonging to an half consonant. So, we also use the prior knowledge about the components and the image properties to resolve the reordering problem.

Akshara	X ప Y	X ప Y
Components order after CC	X ఒ ◡ Y	X ◡ ఒ Y

Figure 2.31 Occurrence of components in a different order for the same akshara

Figure 2.32 Lower components in Telugu language

Figure 2.33 Upper components in Telugu language

Since this is a language dependent module, we have a list which tells us the Unicode corresponding to the components. The components after reordering are mapped with the Unicode using the list described earlier.

2.3.2 Anti-converter

Anti converter performs the reverse function of the converter. This is useful for comparing the OCR output at symbol level rather than word level. This helps us in avoiding the errors that can occur during Unicode conversion or reordering. The mapping from Unicode to class labels is not a one-one mapping rather one-many mapping. For example in the Figure 2.30, we have seen that *talakattu* can occur to the left of *pa* or to the right of it. Hence, for the Unicode of *pa* there are two combinations. Let us refer such an akshara as a complex character. When we have the Unicode of a word, there can be many complex characters and the output therefore will be

the number will be the number of permutations that can occur between the complex characters. If we have three complex characters in a word with 2 components, then the number of possible combinations we can have are *2 x 2 x 2 = 8* combinations. So, the anti-converter will output all the possible sequences for a Unicode word, even though all the sequences may not be valid. Error analysis uses all the generated sequences to compare with the output of the OCR. This also used for symbol level decomposition of data with ground truth.

2.4 Results

We have looked into the modules and the working of OCR, now we will be analyzing it through the results. Before we test it on the real book images, let us start by testing it on the synthetic dataset. The reason for using the synthetic dataset is because of the following reasons:

- A book might not contain all the classes/symbols of the language. For example, in English language it is tough to find words with ”z”, while words with ”a” are many.
- To have uniformity across the classes, i.e., same number of samples in all the classes. In the above example, we have seen that the frequency of ”a” and ”z” vary a lot.
- Synthetic data set is very easy to generate as compared with the book data set.
- To avoid the degradation and segmentation problems one might encounter, while processing the book data. The function of the classifier is to assign a label for the component. Problems like cuts, merges, noise, blobs will reduce the accuracy of the classifier, and hence we do not know if the classifier is good or bad. By using the synthetic data, we focus solely on the classification problem. This helps us in separating the errors caused by thresholding, segmentation and any other ocr module from the errors caused by the classifier, and hence gives us a clear picture of its symbol accuracy.

2.4.0.1 Synthetic dataset

Synthetic data has been generated from the QT tool which uses the Unicode as the input. Degradations like noise and rotation have been used to make it as close as possible to the real data set. 5% of the synthetic data has been used to train the classifier, while 95% was used as the test data set. The same has been repeated with splits of 10-90 and 15-85 for training-testing. The purpose of training and testing on different splits is to observe the effect of splits on the accuracy of the classifier. This also helps us in understanding, how quickly the classifier is able to learn from the data.

The accuracy of the OCR for 5-95 split is 97.12%, 10-90 is 97.85%, and for 15-85 it is 98.23% as shown in the table 2.1. It can be observed that even though the training data is doubled(10%) and tripled(15%), the effect on the increase in accuracy is marginal. From this experiment, we can infer

Training(%)	Testing (%)	Symbol Accuracy(%)
5	95	97.12
10	90	97.85
15	85	98.23

Table 2.1 Results on synthetic dataset with different splits

that a small amount of data is sufficient to train the classifier. Apart from the amount of training data required for the classifier, we can also know the confusing classes. Getting the list of the confusing classes, where classifier is not performing can be very helpful. This helps us in designing new post processing techniques to correct the OCR output.

2.4.0.2 Real dataset

After performing the experiments on synthetic data, we will test the OCR on real data. We have observed that the results on the book data set fell drastically, when the model file built on the synthetic data set was used. This is because of the vast difference in symbol models between the synthetic dataset and the real data set. Therefore trained data from the real dataset is to be used. For this experiment we have used 200 pages from 10 different books, which are completely annotated, i.e., ground truth at word level was available. The pages were chosen in such a way that most of the classes are covered. For training, we need the data at symbol level. So, we have used the anti-converter and connected components to annotate the data at symbol level. Anti-converter gave us the symbol level information from the ground truth.

Mapping the symbol level information with the components is a trivial task because of the presence of cuts and merges. So to get the training data, we only used the word images without cuts, merges and noise. With the above assumption the mapping of the components became one-one (prior information about the font is known). The case where the word has a cut and a merge were ignored. After extracting the symbols for every class, they were visually verified. In this way the training dataset was created.

Even though the pages were chosen from 10 books, it still had the following problems

- Number of samples in the classes varied.
- Some classes did not have any symbols.

To tackle the above problems, synthetic data was added to those classes with less or no symbols at all. Only 5% of the extracted symbols were used as training dataset. This is around 3% of the book size, as we have excluded symbols with cuts, merges and noise. Training samples were taken only from 2 books, i.e., 3% of 2 books data, where as testing samples were from 10 books including the 2 books used for training. The results for the experiments can be found in the table 2.2.

Books [Training]	2
Books [Testing]	10
Pages	200
Symbol Accuracy	90.78%

Table 2.2 Results of Telugu OCR on real book data

We have obtained an accuracy of 90.78%. The classifier with an accuracy as high as 98% when used for evaluating on the real data set, the accuracy has dropped drastically. Some of the reasons are as follows.

- Change in the fonts of the books.
- Segmentation errors
- Cuts and merges
- Noise
- Confusing classes
- Reordering problems
- Foreign characters (English, etc..)

We could not exactly know the effect of the degradation on the character OCR accuracy. There was a need to examine the degradation effect carefully.

2.5 Testing

To study the effect of degradations on the OCR, we shall test them on different quality images. We have chosen document images and categorized them as class A, B and C. Class A is for clean and good dataset. This is without cuts, merges or other degradations. Class B has a slight degradation compared with Class A, but better than Class C. The qualification has been done manually based on the visual appearance of the images and also considering the OCR's performance on the previous dataset.

2.5.1 Class A : Laser print dataset

We have corrected and verified to eliminate the segmentation errors. We also have ensured that there were no cuts or merges in the dataset. The main reasons for taking these precautions is to look at the

OCR accuracy without any degradations or errors from the other modules of OCR like preprocessing and segmentation. This will tell us exactly, how the classifier is performing on the clean dataset. This helps us in determining any mistakes at the recognizer level. We can focus only on the recognizer module, with these setting. The results of the recognizer can be seen in the Table 2.3. As expected the performance of the OCR on these pages was very good. The errors were mainly because of the misclassification of the samples, and foreign samples due to multiple fonts. These are both down to the inability of the classifier.

Dataset	Pages	Symbol Accuracy
Class A	25	97.14%

Table 2.3 Results of Telugu OCR on Laser print dataset

2.5.2 Class B : Good real dataset

This dataset consists of images from the books with reasonable quality. This image collection had images with cuts and merges. Unlike the previous laser print dataset this dataset was more real, i.e, it consisted of all types of degradations which occur normally. The laser print dataset was generated and scanned to avoid all the bookish degradations, while this dataset was taken from the books with reasonable degradations and different fonts. This dataset is more original, and gives us the cases where the OCR is failing in the real book experiments. The results of these experiments can be seen in the Table 2.4.

Dataset	Pages	Symbol Accuracy
Class B	25	89.86%

Table 2.4 Results of Telugu OCR on Good real dataset

2.5.3 Class C : Challenging real dataset

Challenging real dataset is the degraded version of class B dataset. Many more degradations like noise and rotations were added to further degrade this dataset. This is done to see the robustness of the classifier in the presence of cuts and merges. Since the connected components were being used the classifier was not able to do much in the presence of cuts and merges. This is the main drawback of the connected components. The experiments on this dataset highlights the need for a strategy to tackle these degradations outside the classifier. The results of these experiments can be seen in the Table 2.5.

Dataset	Pages	Symbol Accuracy
Class C	25	81.33%

Table 2.5 Results of Telugu OCR on Challenging real dataset

2.5.4 Error Analysis

We have looked at the datasets with different attributes. We have also seen the accuracies fall from 97.14% to 81.33%. Now let's look at the errors, we have encountered. We will analyze the errors, which are reducing the OCR accuracy in general and then we will look into the specific errors that are causing problems for the datasets.

- *Misclassification :* Indian languages and in specific Telugu language has a lot of similar characters. They are similar in shape and size. For example in the Fig 2.34, a loop separates "a" from "aa". Similarly a loop separates the two characters in case b, c. In case d, even though the two characters are similar, their aspect ratios differ. Even though the classifier accuracy is 100% for most of the pairwise classifiers, the problematic cases like these contribute in the decrease of the overall accuracy.

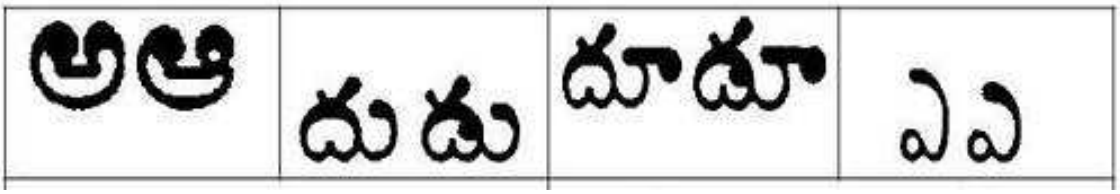


Figure 2.34 Similar characters in Telugu

- *Reordering problems :* Even though most of the reordering problems are addressed through Unicode rules and some post-processing, there still exists some cases which go wrong. There is a need to address this problem thoroughly. This can be done only at the word level and not at the component level. Figure 2.31 gives a detailed explanation of this problem.

- *Unicode problems :* Unicode problems still exist in most of the Indian languages, as they are in the process of development. Figure 2.35 shows a case where the original word can't be rendered in the current settings. This one of the many cases where the Unicode settings go wrong. In the figure 2.35, the Unicode of the original word, according to the rules, renders the obtained word image. Though the classifier recognizes the components, correctly because of these problems the recognizer accuracy reduces.

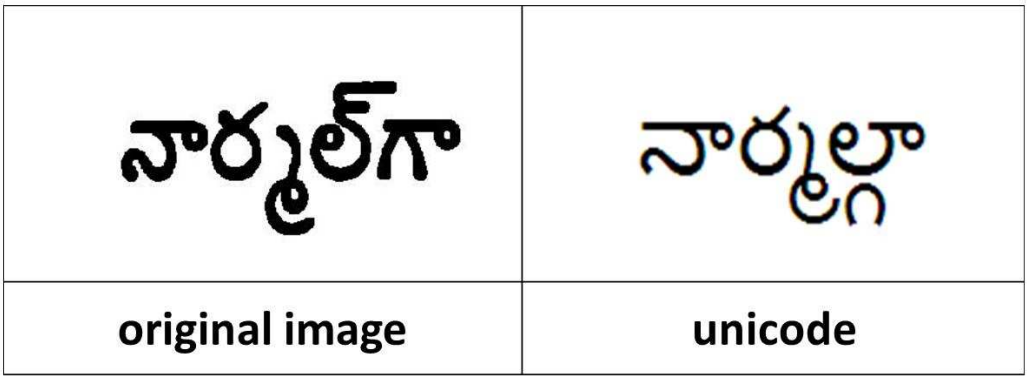


Figure 2.35 Original word and Obtained word

- *Segmentation errors :*

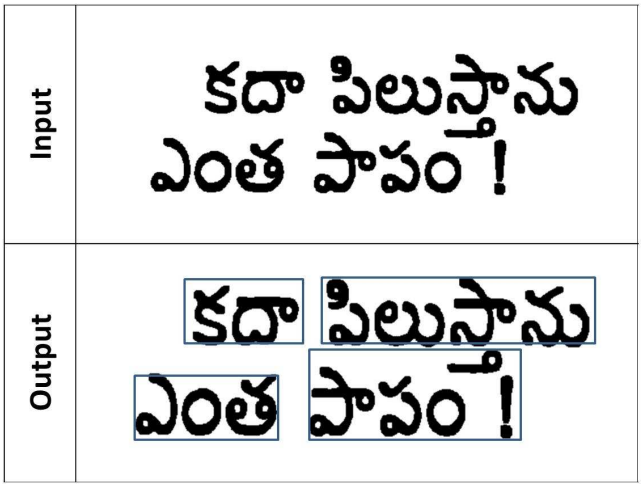


Figure 2.36 Input and output of segmentation

Segmentation problem is a very complex problem in Indian languages, especially in languages like Telugu where the components are rendered in all directions. The problem is already discussed in the section 2.2.2. Here we look into the errors caused by segmentation. Fig 2.36 shows the input and output of a part of a page.

Fig 2.37 shows the segmentation error where a component is missing. Sometimes, this also causes the components to produce cuts.

- *Cuts :* The accuracy of the classifier is mainly affected by the degradations. Degradations like cuts, merges, blobs, and noise are the some of the main factors which reduce the accuracy. When a character which should has a single component is split into two, the character is said to have a

పిలుసాను	పిలుస్తాను
Error	Expected output

Figure 2.37 Missing components because of segmentation

cut. The cuts majorly occur due to the slender shape of the aksharas. The contour is affected by the cut. Figure 2.38 shows some of the major cuts in the akshara.

Figure 2.38 Cuts in aksharas

- *Merges :* Another major component of the degradation is merges. This occurs when the spacing between the characters is less. This sometimes also occurs because of low resolutions. When an akshara with a component touches the other akshara's component, it forms a new component. The aksharas are said to be merged in this case. Figure 2.39 shows the merged components.

Figure 2.39 Merge of aksharas

- *Blobs :* The above degradations including this, is mainly because of aging of the document, faulty printing or scanning and other factors. Blobs are mainly because of the faulty ink. Thresholding also contributes to the degradations. Figure 2.40 shows the blob, which destroys the original structure of the character.

Figure 2.40 Blobs in word image

- *Noise :* Noise in an image can be because of thresholding, segmentation, printing or various other issues. Figure 2.41 shows a part from the previous line in the word image. The recognizer thinks this noise as a part of the word image, and hence the error.

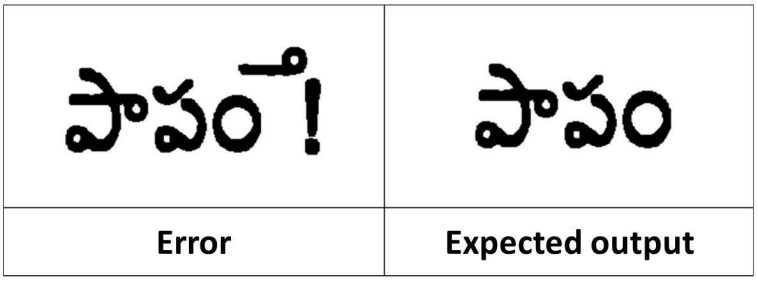


Figure 2.41 Noise because of faulty segmentation from Figure 2.35

- *Change in font :* The main bottleneck of OCR is multi font. Here in our experiments, we have observed that the recognizer trained on a particular font cannot be used on a different font. But having different recognizers for different fonts is again an impossible task, because then we have to employ a 100% font recognizer module. The recognizer must be robust enough to classify characters in different fonts. In Figure 2.42, we can see different fonts occurring in the same page. Sometimes the akshara might change its shape completely in another font. This is again a major problem, which reduces the accuracy of the recognizer. So, to develop an OCR which recognizes book of any font is a challenging task.

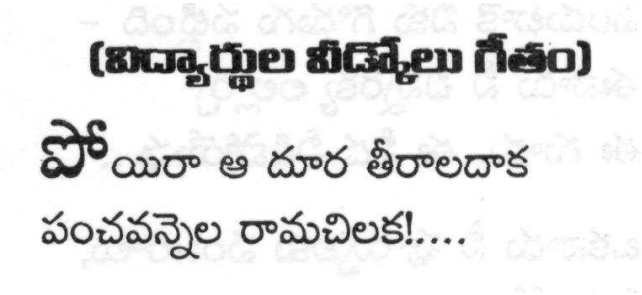


Figure 2.42 Multiple fonts in a page

Now, let's analyze the results of the OCR on these datasets.

- *Dataset A :* In this dataset the major errors are as follows :
  - Misclassification of the samples.
  - Multi font problem
- *Dataset B :* Along with the errors in Dataset A, the additional errors which occurred are as follows
  - Segmentation errors
  - Few degradation errors (cuts, merges, noise).
  - Unicode problems
  - Reordering errors
- *Dataset C :* Along with the errors in Dataset B, the additional errors which occurred are as follows
  - Many degradation errors (cuts, merges, noise).

2.6 Discussions

In the previous section, we have analyzed the major errors that deteriorate the performance of the OCR. The major inferences we have drawn from analyzing the three datasets A (Laser print dataset), B (Good real dataset), C (Challenging real dataset) are as follows :

- Multi font problem is a major bottleneck in the usage of the OCR. This problem is consistent in all the three datasets. Even on the Laser print dataset, the ocr performance on certain pages deteriorated because of the inability of the recognizer to classify the samples. There is a need for *robust classifiers which are font independent*.

- Degradations pose a serious threat to the accuracy of the recognizer. Unlike the multi font problem, this is an internal problem because of the classifier's inability, the degradation is an external problem. Multi font problem can be solved by the OCR by making the classifier robust and more general. Whereas the degradation depends on the input, which makes it impossible to predict. We can device strategies to tackle each degradation separately which in itself is complex and on top of that we need a complex mechanism to integrate them all. Hence there is a need for *a generalized and simple solution to the problem of degradation of any kind.*

  In the coming sections, we try to solve these two problems.

  A major requirement in the design of robust OCRs is the invariance of feature extraction scheme with the popular fonts used in the print. Many statistical and structural features have been tried for character classification in the past. In this work , we get motivated by the recent successes in object category recognition literature and use a spatial extension of the histogram of oriented gradients (HOG) for character classification. Our experiments are conducted on 1453950 Telugu character samples in 359 classes and 15 fonts. On this data set, we obtain an accuracy of 96-98% with an SVM classifier.

  Typical optical character recognizer (OCR) only uses local information about a particular character or word to recognize it. In Chapter 4, we propose a document level OCR which exploits the fact that multiple occurrences of the same word image should be recognized as the same word. Whenever the OCR output differs for the same word, it must be due to recognition errors. We propose a method to identify such recognition errors and automatically correct them. First, multiple instances of the same word image are clustered using a fast clustering algorithm based on locality sensitive hashing. Three different techniques are proposed to correct the OCR errors by looking at differences in the OCR output for the words in the cluster. They are Character Majority Voting, an alignment technique based on Dynamic Time Warping and one based on Progressive Alignment of multiple sequences. We also demonstrate the approach over hundreds of document images from English and Telugu (an Indic script) books by correcting the output of the best performing OCRs for English and Telugu. The recognition accuracy at word level is improved from 93% to

97% for English and from 58% to 66% for Telugu. Our approach is applicable to documents in any language or script.

We address the internal problem of OCR, i.e., making the classifier more robust to tackle multiple fonts. And the external problem of OCR, i.e., devising a post processing technique to tackle the problem of degradations and the misclassifications as well.

Chapter 3

Multifont character classification in Telugu

3.1 Introduction

Large repositories of digitized books and manuscripts are emerging worldwide [77]. Providing content-level access to these collections require the conversion of these images to textual form with the help of Optical Character Recognizers (OCRs). Design of robust OCRs is still a challenging task for Indian scripts. The central module of an OCR is a recognizer which can generate a class label for an image component. Classification of isolated characters and thereby recognizing a complete document, is still the fundamental problem in most of the Indian languages. The problem becomes further challenging in presence of diversity in input data (for example, variations in appearance with fonts and styles.) Multifont character classification is still a bottleneck in the development of OCRs for Indian languages.

Characters are first segmented out from page or word images. A set of appropriate features are then extracted for representing the character image. Features could be structural or statistical. Structural features are often considered to be sensitive to degradations in the print. A feature-vector representation of the image is then classified with the help of a classifier. Multilayer neural network, K nearest neighbor, support vector machines (SVM) etc. are popular for this classification task.
Classification of Indian scripts is challenging mainly due to the following reasons

- Large number of classes (compared to Latin scripts)
- Many pairs of very similar characters. (See Figure 3.1.)

In a recent work, Neeba and Jawahar [64] had looked into the success rates of character classification problem in an Indian context. Though their results are primarily on Malayalam, they are directly expendable to other scripts. They successfully solved the character classification problem (even in the presence of large number of classes) for limited number of fonts popularly seen in print.
They had argued that

- Multiclass classification solution can be made scalable by designing many pair-wise classifiers.

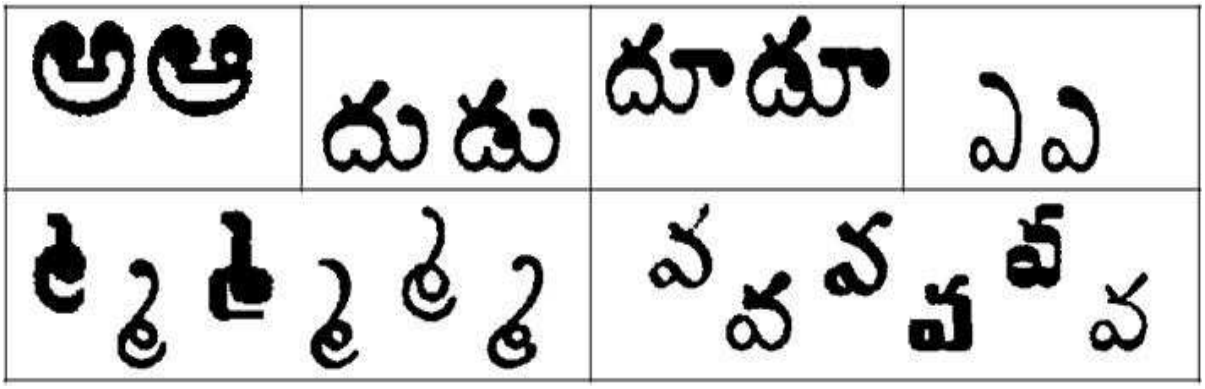


Figure 3.1 Challenges in character classification of Telugu. First row shows similar character pairs from Telugu. Second row shows how the same character gets rendered in different fonts

- Use of large number of features (of the order of few hundreds) makes the problems better separable and solvable with simple classifiers.

- When the dimensionality of the feature is made reasonably high, even the simple features like raw-pixels or PCA-projections provide satisfactory results.

A strong requirement of any robust character recognition system is the high classification accuracy, in the presence of multiple and diverse font sets. In this work, we explore the problem of character classification in a multifont setting. Though our studies are for Telugu script, we believe that these results are also expendable to other languages. Our objective is to demonstrate the utility of the histogram of gradients (HoG) [26] sort of features for character classification. We also show that the linear SVM with DDAG sort of classifier fusion strategy provides equivalent results to an Intersection kernel SVMs. We validate our experimental results on 1,453,950 Telugu character samples in 359 classes and 15 fonts.

Telugu like most other Indian scripts, have consonants, vowels and vowel-modifiers. In addition, there are also half consonants which gets used in consonant clusters. Though the script is getting written from left to right in a sequential manner, many of these modifiers often gets distributed in a 1.5D (not purely left to right; they are also written top-to-bottom at places) manner. Compared to most other Indic scripts, Telugu has large number of basic characters/symbols. Many of them are also similar in appearance. This makes the character classification problem in Telugu very challenging.

In the Figure 3.1, the top row represents the similar characters problem. In case 1, 2, 3 of the top row, the characters are just differentiated by a single loop. While in case 4 of the top row, aspect ratio separates the two components. The variations of characters in different fonts can be seen in the bottom row of Figure 3.1. The first case in the bottom row is the half consonant of the akshara *ma*. You can see that the loop of this component in some fonts is closed, while it is present in other fonts. This shows the diversity of multiple font datasets, and adds to the problem of multifont character classification.

Telugu character recognition has been attempted in the past with various features. Negi *et al.* [65] used fringe maps as the feature. The method was tested on 2524 characters. Jawahar *et al.* [43] did a more thorough testing (of close to one Million samples) of the character classifiers but with limited font variations as well as degradations. They used PCA, LDA etc. as the possible feature extraction scheme for Telugu character classification.

3.2 Classification Architecture

The general architecture of the character classification module is shown in Figure 3.2. The main modules of this architecture are the feature extraction module and the classifier module. Our main aim is to optimize this module for character classification. The main drawback with this module is that the classifier accuracy is bounded by the limitation of the feature extraction module. Now, lets analyze each of these modules in more detail.

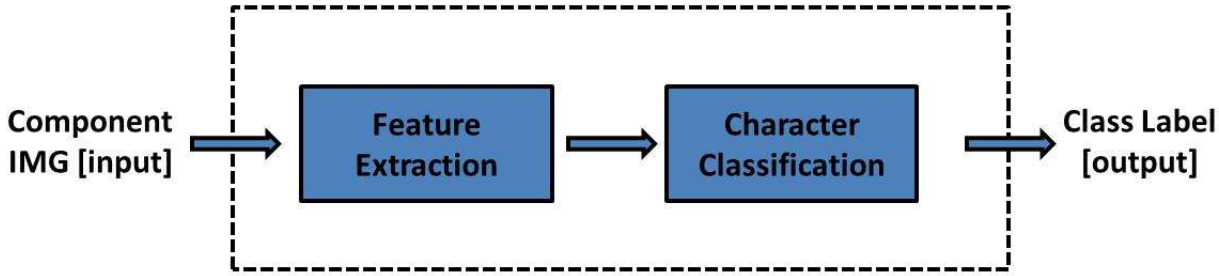


Figure 3.2 General architecture of the character classification module

3.2.1 Feature Extraction

The detailed description of this module has been given earlier in section 2.2.3.1. The main drawback of the earlier approach is trying to solve the problem by using the modules linearly. In this section we will be looking into the limitations of the feature extraction module. Some features are good for solving some cases, while the others for other cases. Lets consider two features vectors

- F1 - Count of the number of loops in the component.
- F2 - Aspect ratio of the component image.

In Figure 3.1, consider the top row as components. F1 can solve case 1, while F2 can't. Likewise F2 can solve case 4, while F1 can't. If I use F1 as a feature with a classifier and then my classifier would not be able to solve the problem of case 4, and similarly with F2 we cannot solve the case 1. Hence, the classifier is being limited by the generalized linear approach of the feature vector. So, we have to device a strategy to use the features F1, F2 according to our problem and make sure the limitations of the feature vector are not passed on to the classifier. We must note that selection of a feature extraction scheme is probably the most important factor in achieving high accuracy. So, we have to decide which feature extraction method is the best for the given classes.

Neeba *et. al* [64] observed that for better performance, a rich feature space is required for large class problems. If the feature space is rich, they could also become discriminative for most of the classifiers. Also, with a large feature vector, character classification can be solved with reasonable success.

$$Length\ of\ feature\ vector\ \propto\ Accuracy$$

But the only drawback is, as the length of the feature vector increases, the memory size of most of the classifiers increase enormously. Hence, it is not advisable to use lengthy feature vectors for all classes for memory and computational constraints. So, there is a need to find a feature extraction method which can deliver high accuracy and can also be applied for practical purposes.

3.2.2 Character classification

Classifiers have been discussed in detail earlier in section 2.2.3.2. A major challenge in the development of OCRs for Indian scripts come from the large character set, which results in a large class classification problem.

$$Number\ of\ classes\ \propto\ \frac{1}{Accuracy}$$

Neeba *et. al* [64] observed that SVM classifiers outperform other classifiers for character classification. A class of feature extraction scheme based on the use of raw image and its projection onto an uncorrelated set of vectors result in best performance. Also it was observed that the SVM classifiers with DDAG degrade gracefully, when the number of classes increases.

Support Vector Machine (SVM) is a technique widely used for leaning classification. SVM classifies by learning a separate hyperplane in some high dimensional feature space. The learned hyperplane is such that it strives for reducing the generalization error of the classifier by maximizing the margin between the support vectors. More about the SVMs can be found in section 2.2.3.3.

3.2.3 Classifier Architecture

The architecture of the classifier also plays a vital role in dealing with the multi class problem. A detailed discussion about the multiclass problem is in section 2.2.3.4. The two major architectures widely used are as follows:

- *1 Vs All :* In this structure, if we consider n classes, then a classifier will have one class as "+" samples, and the rest (n-1) classes as "-" samples. In this particular setting, we will have n classifiers for n classes. Though this setup requires less memory, the accuracies are generally low.

- *DDAG :* Decision Directed Acyclic Graphs (DDAG) have been explained in section 2.2.3.4 earlier. For n classes, it requires $\frac{n(n-1)}{2}$ or ${}^nC_2$ classifiers. Though it is computationally expensive, it generally gives very high accuracies. These are more efficient as the problem of classes is reduced to a set of binary pairwise classifiers.

3.3 Dataset

To solve the multifont problem in Telugu, we need to have a challenging font dataset with degradations. The dataset had the following attributes

- *Fonts :* 15 popular fonts in Telugu language have been used for the creation of this dataset. The list of font names are as shown in Table 3.1.

- *Font size :* Font size was varied to observe the effect of scaling on the classification. Font size of 12, 14 and 16 were used.

- *Resolution :* The generated pages were scanned in three resolutions namely 200 dpi, 300 dpi and 600 dpi. This is to vary the quality of images.

- *Degradations :* The components were extracted from the images through a semi supervised process. Degradations were then added to these components to make the dataset more challenging. Rotations of -2, -1, 0, 1 and 2 degrees were applied. Then salt and pepper noise of 0%, 2%, 4%, 6%, 8% and 10 % was added the set of images. 0% noise and 0% rotation implies using the original image.

- *Number of Classes :* The dataset was created with 359 classes from Telugu language.

So each class had (15 fonts X 3 font sizes X 3 resolutions X 5 rotations X 6 noise) = 4050 images.

Therefore the Dataset has (4050 X 359 classes) = 1,453,950 images

List of fonts
Anuradha
Chandra
Devi
Godavari
Gurazada
Harshapriya
Hemalatha
Nannayya
Pravalika
Shravani
Suma
Telangana
TeluguFont
Tikkana
Vasanti

Table 3.1 Fonts used in the dataset

3.4 Experimental Observations

We have initially conducted our experiments on a dataset of 100 classes. Linear SVM classifier with raw pixels as a feature vector has been used in these experiments. From the analysis of the misclassified samples, we have observed that the decrease in classification accuracy is mainly because of the following reasons.

- Increase in number of fonts

  We can observe the variation in the accuracy with increase in number of fonts in the Figure 3.4. The reasons for the decrease in accuracy is mainly due to the increase in number of similar classes. Another reason is the variation of the component in multiple fonts.

- Few confused pairwise classifiers The drop in the accuracy of the classifier is mainly because of the confusing classes. In Figure 3.5, we plot the cumulative accuracy over all pairwise confusions. As can be seen in Figure 3.5, the errors are associated with only certain pairs. If we can solve these errors in the pairs, then we can achieve high accuracies.

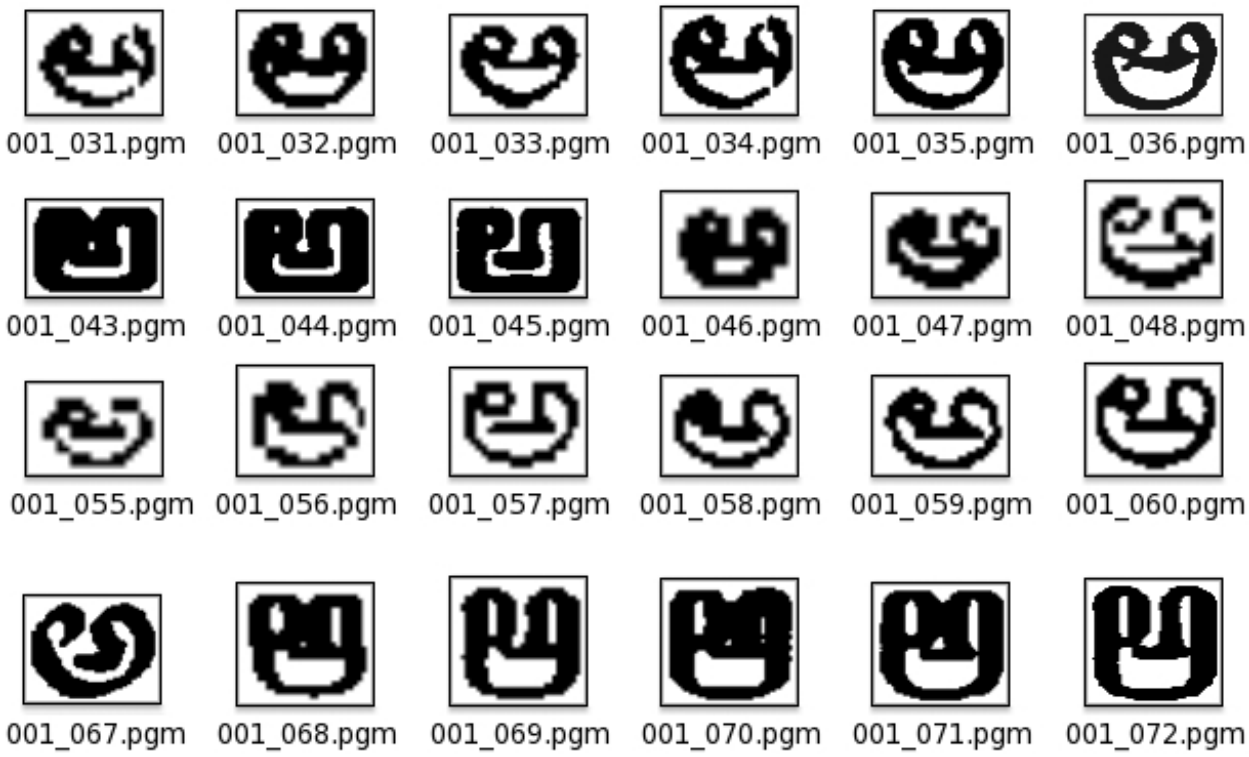


Figure 3.3 A glimpse of a few images belonging to the same class in the dataset

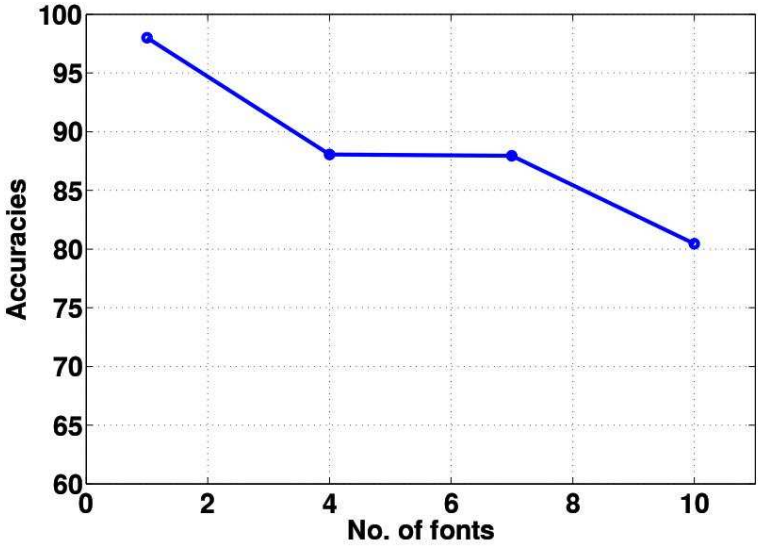


Figure 3.4 Variation in accuracy with increase in number of fonts

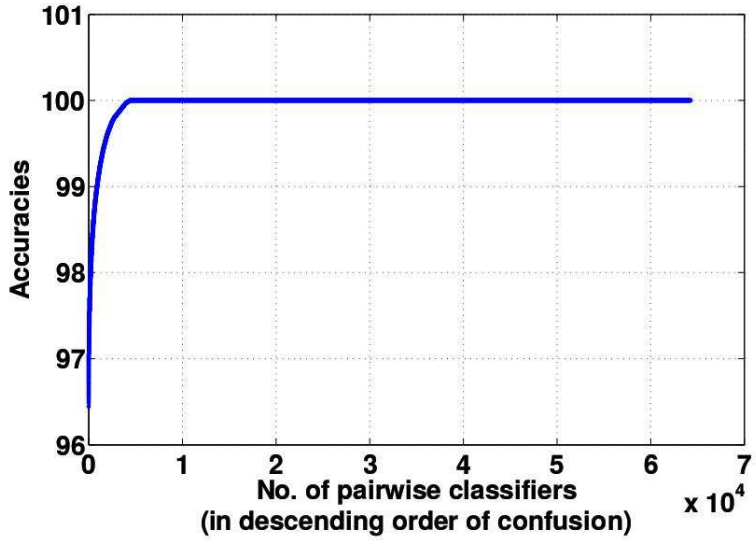


Figure 3.5 Accuracy and confused pairwise classifiers

3.5 Features and Classifiers

Recent years have witnessed significant attention in development of category level object recognition schemes with many interesting features. Histogram of oriented gradients (HOG), which was successfully used for detecting pedestrians [26], is one of the prominent and popular features for capturing the visual data, when there are strong edges. Naive histogram representation loses the spatial information in the image. To address this, spatial pyramid matching was proposed [48]. Similar to [54], we also employ a feature vector which captures spatial information and histograms of oriented gradients.

We are motivated by the recent classification experiments in multifont data sets [28] and handwritten MNIST and USPS digit data sets [54]. Many of these studies are limited to handwritten digits. There have been many studies in this area

- focusing on generalization of classification results to unknown fonts, and thereby solving the character 'category' recognition problem [28].
- accurately solving the handwritten digit recognition with many machine learning concepts [54].
- development of recognition algorithms with fewer training data or lesser resource usage.

Based on the conclusions obtained in our earlier work on character classification [64], we use SVM classifiers. SVM classifiers are the state of the art in machine learning, to produce highly accurate and generalizable classifier. The classification rule for a sample x is

$$sign(\sum_{i=1}^{nSV} \alpha_i \kappa(x, s_i) + b)$$

where s_is are the support vectors and $\kappa()$ is the kernel used for the classification. The Lagrangians α are used to weigh the individual kernel evaluations. The complexity of classification linearly increases with the number of support vectors. To make the classification fast, we can do the following [41]:

- Use linear kernels instead of nonlinear ones.
- Store the weight vector instead of the support vectors
- Use binary representation as well as appropriate efficient data structures and
- Simplification of repeating support vectors in a decision making path consisting of multiple pair-wise classifiers.

It was shown that the intersection kernel can be evaluated fast in many practical situations [53]. However, the comparisons are with that of complex kernels like RBF Kernel. Such classifiers are appropriate when the classes are not well separable. In the case of large class character recognition data set, most of the pair-wise classifiers could be linearly separable. The overall classification accuracy reduces due to

- cascading effects in the multiple classifier systems
- some of the pairs are difficult to separate with simple features.

In this work, we compare the IKSVM with linear SVM and prefer to go for linear SVMs due to the computational and storage advantages of the linear SVM over IKSVM. Based on the experimental results presented in the next section, we argue that

- object category recognition features are useful for the character recognition specially in presence of multiple fonts.
- linear SVMs perform very similar to IKSVM for most of the character classification tasks.
- Use of SPHOG sort of features can successfully solve the multifont character classification problem in Indic scripts.

Now let us look at the features used in detail.

3.5.1 Spatial Histogram of Oriented Gradients (SPHOG)

We experiment with features constructed using histograms of oriented gradients which have become popular in the vision literature for representing objects [17, 26, 33, 48, 52] and scenes [66]. Each pixel in the image is assigned an orientation and magnitude based on the local gradient and histograms are constructed by aggregating the pixel responses within cells of various sizes. We construct histograms with cell sizes 14×14, 7×7 and 4×4 with overlap of half the cell size. The histograms at each level are multiplied by weights 1, 2 and 4 and concatenated together to form a single histogram which are

then used to train kernel SVMs. This is very similar to the spatial pyramid matching [48] when used with the intersection kernel (we differ in the overlapping grids). The various choices for the descriptor are as follows :

- **Oriented Derivative Filter**: The input gray scale image is convolved with filters which respond to horizontal and vertical gradients from which the magnitude and orientation is computed. Let *rh(p)* and *rv(p)* be the response in the horizontal and vertical direction at a pixel *p* respectively, then the magnitude *m(p)* and the angle *a(p)* of the pixel is given by :

$$m(p) = \sqrt{rh(p)^2 + rv(p)^2} \tag{3.1}$$

$$\alpha(p) = atan2(rh(p), rv(p)) \in [0, 360) \tag{3.2}$$

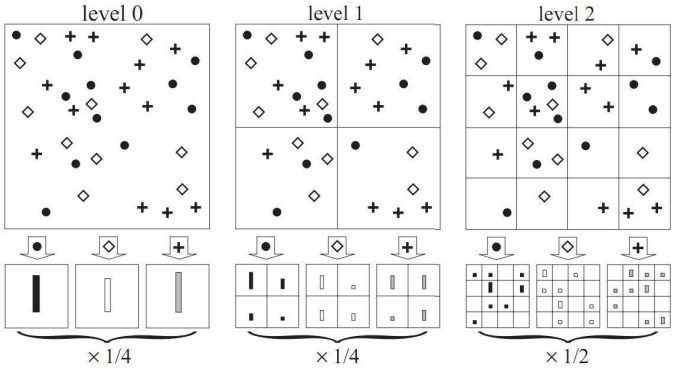


Figure 3.6 Toy example of constructing a three-level pyramid. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, we subdivide the image at three different levels of resolution. Next, for each level of resolution and each channel, we count the features that fall in each spatial bin.

- **Signed vs. Unsigned** The orientation could be signed (0-360) or unsigned (0-180). The signed gradient distinguishes between black to white and white to black transitions which might be useful for digits.
- **Number of Orientation Bin** The orientation at each pixel is binned into a discrete set of orientations by linear interpolation between bin centers to avoid aliasing.

The entire set of histograms is finally concatenated to form a single histogram. We refer this feature as SPHOG in the work.

3.5.2 Principal Component Analysis

Principal component analysis (PCA) can help identify new features (in a lower dimensional sub-space) that are most useful for representation [31]. This should be done without losing valuable information. Principal components can give superior performance for font-independent OCRs, easy adaptation across languages, and scope for extension to handwritten documents. Various extensions of PCA have also been proposed in computer vision, including Binary PCA [83]. While binary PCA models binary data as Bernoulli distribution, classical PCA models the image with Gaussian assumption. Both methods reduce the dimension of a dataset to reveal its essential characteristics and the subspace captures the main structure of the data. In the present work, we employ the conventional PCA for extracting relevant information from high dimensional datasets on which further transformation to classification sub-space is performed using LDA.

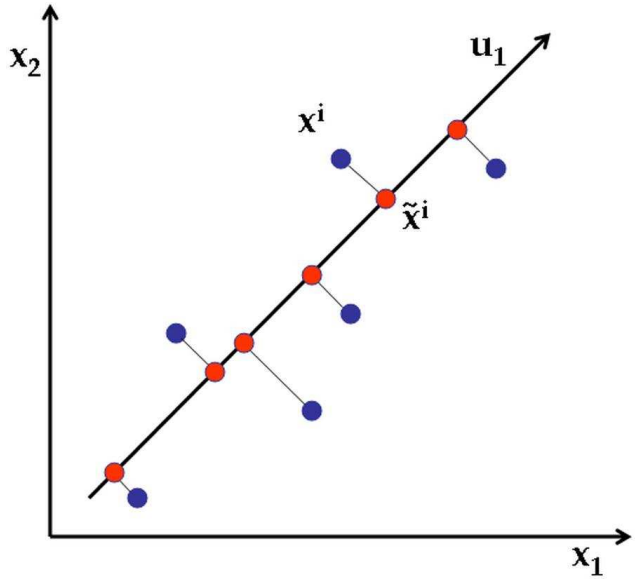


Figure 3.7 PCA example: 1D projection of 2D points in the original space.

Consider the i^{th} sample represented as an M-dimensional (column) vector x_i, where M depends on the image size. For a given training datasets, x_1, x_2, .., x_n we compute the covariance matrix (σ):

$$\sum = \frac{1}{N}\sum_{i=1}^{N}[x_i - \mu][x_i - \mu]^T \tag{3.3}$$

Then we need to identify minimal dimension M' such that

$$\frac{\sum_{i=1}^{K}\lambda_i}{Trace(\sum)} \geq \alpha \tag{3.4}$$

where λ_i is the i^{th} largest Eigen value of the covariance matrix $\sum$ and α is a limiting value. Eigenvectors corresponding to the largest M' eigenvalues are the direction of greatest variance. The M'th eigenvector is the direction of greatest variation perpendicular to the first through (M'-1)st Eigen vectors. The Eigen vectors arranged as rows in matrix **A** and this transformation matrix is used to compute the new feature vector by projecting as, y_i = $\mathbf{A}x_i$. With this we get the best one dimensional representation of the component images with reduced feature size. Principal component analysis (PCA) yields projection directions that maximize the total scatter across all classes. In choosing the projection which maximizes total scatter, PCA retains not only between-class scatter that is useful for classification, but also within-class scatter that is unwanted information for classification purposes. Much of the variation seen among document images is due to printing variations and degradations. If PCA is applied on such images, the transformation matrix will contain principal components that retain these variations in the projected feature space. Consequently, the points in the projected space will not be well separated and the classes may be smeared together.

Thus, while the PCA projections are optimal for representation in a low dimensional space.

3.6 Results and Discussions

We start by investigating the deterioration of performance with the number of fonts. For this purpose, we collected a character level with ground truth for Telugu data set in fifteen fonts. Number of classes which is common to all these fonts is 359. We first investigate the utility of raw pixels as a feature with a linear SVM classifier. For this experiment, we consider only the first 100 classes. Results of the variation of accuracy is plotted in Figure 3.4. It may be seen that with only one or limited fonts, the accuracies are acceptable, however, with the number of fonts increasing, the accuracy comes down significantly.

We now quantitatively show the results on a 100 class subset of the Telugu characters in 15 different and popular fonts. We show that the naive features, like raw pixels or PCA, are unable to address the significant font variation present in the data set.

Table 3.2 compares the performance of the four features in presence of two different SVM classifiers – Linear SVM(LSVM) and Intersection Kernel SVM (IKSVM). Linear SVMs are also implemented as One Vs All as well as DDAG [64].

It may be noted that the raw image features are not able to perform well when the number of fonts increases. This is expected because of the variation in the styles and shapes of the associated glyphs. It is surprising that the PCA, which was performing reasonably well for limited number of fonts [64] is also not able to scale well for the multifont situation.

A graph which shows the variation of the number of Eigen vectors (principal components) selected Vs the accuracy obtained is shown in Figure 3.8. The plot of magnitudes of Eigen values of the covariance matrix (used in PCA) is shown in Figure 3.9. These graphs explain that with an increase in the number of PCs the accuracy monotonically improves. However, the accuracy saturates at a level 91%,

Classifier	RawPixels	PCA	SPHOG	PCA-SPHOG
	d=784	d=500	d=2172	d=500
LSVM(OneVs All)	91.91	90.19	98.10	96.70
LSVM(DDAG)	94.19	93.84	97.25	97.51
IKSVM(OneVs All)	92.26	96.369	98.71	98.39

Table 3.2 Comparative results on a smaller set of Telugu Multifont Data Set

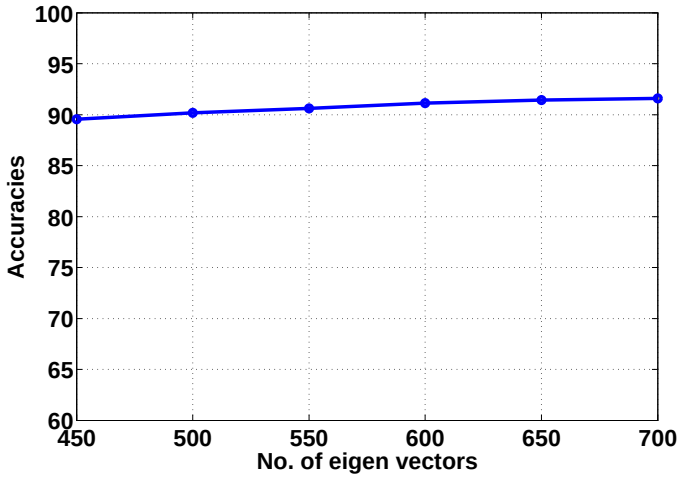


Figure 3.8 Accuracy and number of Eigen vectors

which is not an acceptable level of accuracy, we are looking for an OCR. On the contrary, the SPHOG features are performing consistently well for the large font data set, as can be seen in Table 3.2. PCA has been applied on the SPHOG feature as the dimensionality of the feature is large. Even with 23% of the SPHOG feature vector, accuracy close to the SPHOG result has been obtained.

In short, it is clear from the experiments conducted on a 100 class data set, that SVM classifier with SPHOG and PCA-SPHOG features, provide the most accurate classifiers. We have extended the results obtained for a full Telugu character set consisting of 359 classes. They summarize as follows:

Obtaining an accuracy of 96.4 on a truly challenging multifont data is significant. However, we would like to see the possibility of enhancing the accuracy further. For this, we analyze the confusions associated with all the pairwise classifications. As can be seen from Figure 3.5, the errors are associated with only certain pairs. In Figure 3.5, we plot the cumulative accuracy over all pairwise confusions. If we can address the errors in these pairs with the help of an additional classifier (we call them as

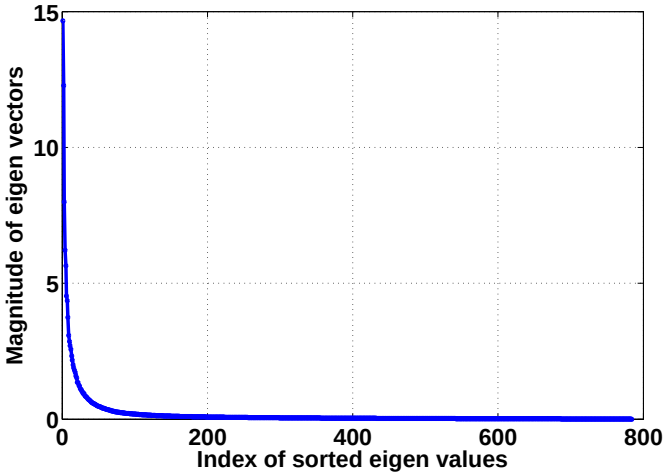


Figure 3.9 Eigen vectors and their magnitude.

Raw pixels with Linear SVM classifier results in an accuracy	81.05
SPHOG with Linear SVM classifier results in an accuracy	96.41
PCA-SPHOG with Linear SVM classifier results in an accuracy	92.95

Table 3.3 Classification accuracy: No of classes = 359, No of samples = 1453950

post-processing classifier), we can further enhance the accuracy (say to 98%). The detailed design and analysis of the post-processing classifier is beyond the scope of this work.

3.7 Summary

In this section we have addressed the problem of classification on multifont datasets. Initially we prepared a big multifont dataset. The composition of this dataset, which will be used as the multifont dataset is as follows. Each class had 15 fonts, 3 font sizes, 3 resolutions, 5 levels of rotations and 6 levels of noise, which amounts to 4050 images per class. For 359 classes, the total number of images is 1,453,950. We have seen the traditional architecture of the classifier with modules like feature extraction, character classification and classification architecture. We have observed that the classification accuracy of the classifier drops drastically with the increase in the number of fonts. Also the drop in accuracy is mostly associated with a few pairwise classifiers. This shows the inability of the feature in classifying these classes. Also we need to analyze which classification architecture is more suited for our problem. We have used 4 features and 3 classifiers to perform our experiments. We have used the SPHOG feature which is based on Object recognition to test on our dataset. This has increased the

component classification accuracy to 96.41%, compared to 81.05% of the raw image pixel feature which is currently being used. As the SPHOG feature is very large, we used PCA to reduce the feature size. Even with 23% of the SPHOG feature vector, we have obtained an accuracy of 92.95%, which is close to the original SPHOG feature results.

We show that high classification accuracies can be obtained for character classification problem with the help of SPHOG-SVM combination. Left out confusions is associated only to a small percentage of the classifier and a post processing classifier with an uncorrelated feature set can successfully boost the overall classification performance.

We have improved the component accuracy, but the word accuracy was still low. If we consider the component accuracy to be 96% and the average length of the word are 5. Then the probability of getting a correct word is (0.96) to the power of 5. This will give us a result of 81.5% approximately, which is very low considering the 96% component accuracy. So we need an algorithm which can exploit the intrinsic document constraints and the component information in the word, to boost the word accuracy. In chapter 4, we make an attempt to do the same. We try to boost the word accuracies using post-processing techniques. A document level word recognition technique is presented, which makes use of the context of similar words to improve the word level recognition accuracy. More details in the next section.

Chapter 4

Error correction using post-processing techniques

4.1 Introduction

A large number of online Digital Libraries (DL) these days, consist of scanned document images from books, newspapers, magazines, etc. The scanned documents are stored as images in such DLs, making it difficult to search their content. The text equivalent of these images is usually obtained using optical character recognition (OCR). For the Latin scripts, OCRs have had reasonable success, with commercial products available as well. However, for many non-European scripts, OCRs are not accurate enough yet. Indic scripts in particular are challenging and the main reason for poor OCR accuracy is the complexity of the script, visual similarity of different characters, and the distributed layout of writing [67, 37, 47].

An OCR usually does a recognition or classification at the local level based on the image of a specific character or word. In this work, we present a *document level OCR* which exploits the global context of a document to correct OCR errors. The proposed method exploits the multiple occurrences of a single word across a book, or a collection of books. Word images from a collection of documents are clustered using a fast and accurate clustering technique based on locality sensitive hashing (LSH) [27]. Then, words from each cluster are recognized using an OCR. Errors in recognition are identified by aligning characters/components using a dynamic time warping technique over the words in the cluster. The identified errors are corrected by a character majority voting procedure.

In this work, we further explore a newer and faster string alignment scheme. We present a thorough evaluation of our LSH based clustering technique. We demonstrate our approach on a larger dataset of multiple scanned books in an Indian script called Telugu. We further demonstrate the approach over hundreds of English language document images as well. The improvement in accuracy for words which occur at least thrice in the collection, is about 8% for Telugu and 4% for English.

4.1.1 Dataset and Challenges

Our dataset is obtained from scanned books in English and Telugu languages. Telugu is an Indian language which has more than 70 million speakers, a large number of newspapers and literature dating back several hundred years. For the experiments in this work, we use a dataset of two Telugu books, comprising 128 pages; while for English we obtain four books consisting of 722 pages. An example document image from the dataset is shown in Figure 4.1. The page-to-word segmentation is performed using the run length smearing algorithm (RLSA) [86]. The obtained image segments are extracted and stored as individual word-images.

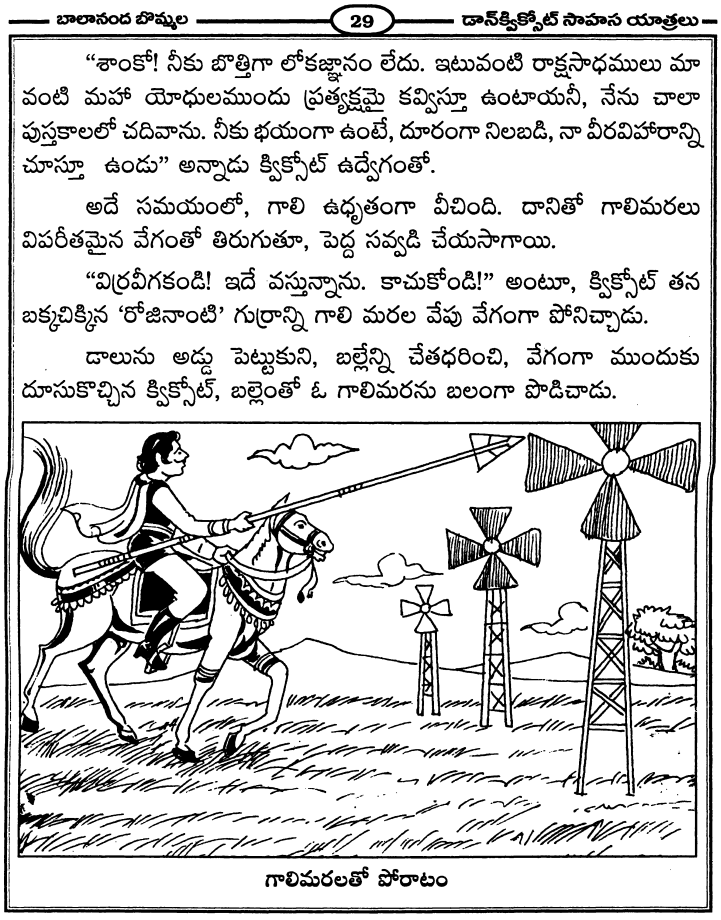


Figure 4.1 An example Telugu document image in our collection. This particular book is the Telugu version of the famous novel *Don Quixote*

There are many challenges toward accurately recognizing the text in our document images as listed below.

- **Script Complexity :** Indian language scripts, especially Telugu, consists of a large number of distinct characters; each character written as a conjoined consonant and vowel modifier. This format makes it hard to both segment a character, as well as recognize it with a classifier. A couple of example words are shown in Figure4.2. The vowel modifiers and consonant conjuncts could occur either before or after the main character, or above or below it.

- **Visual Similarity :** In Indian languages, there are a large number of visually similar character pairs. Subtle changes such as the presence/absence of a dot, or a small stroke could alter the sound of the character and the meaning of the word. This calls for more stronger/complex classifiers to discriminate between such characters.

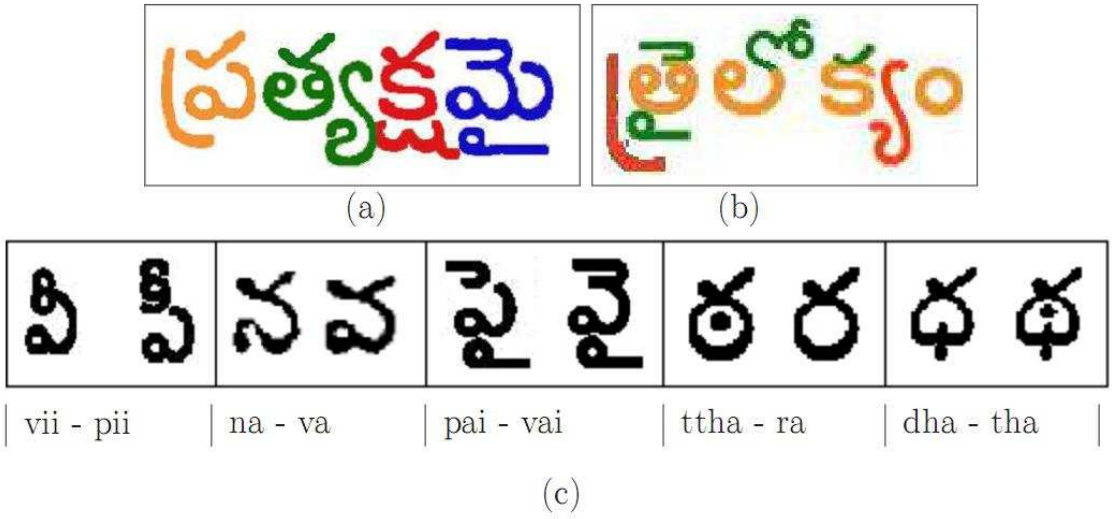


Figure 4.2 Examples of visual complexity in the Telugu script. In (a), all the components with the same color belong to the same character. (b) The main consonant that forms the character is shown in orange, its vowel modifier is shown in green, and the consonant-conjunction is shown in red. Note that the related components could be spatially distributed around the main consonant. Example pairs of visually similar characters are shown in (c). In the second pair, the completion of circle on the left-bottom changes the character. In the last two pairs, the dot in the middle distinguishes between them.

- **Degradations:** The poor accuracies are worsened by the presence of degradations in the document images. These images are generally degraded due to the age of the paper, noise from scanning, etc. An example is shown in Figure 4.3. Characters are frequently broken during noise cleaning and binarisation. Degradations are present in both the Telugu and English document

images. They result in multiple segments of each character which are all incorrectly recognized by the OCR.

- **Difficult OCR Post-Processing :** Moreover, it is hard to correct OCR errors using standard post-processing techniques. Since Telugu has a large vocabulary [18], it is both difficult to build a dictionary, as well as use it for error correction. For an estimate, to cover about 80% of the vocabulary, the size of the dictionary may be almost 50 times that for English. The applicability of statistical language models for error correction are limited as well. The number of bigram syllable types for Telugu is the highest among all languages [18]. The learning of bigram models for the language requires significant amount of training data.

- **Character to Word Accuracy:** In documents, each word consists of a number of components. A component is a set of pixels, which are continuous. For example, in English letter ”a” is a single component where as ”i” has 2 components. Thus, even when component level recognizers perform well on a document, the word and document level accuracies may not be acceptable in practical situations. Consider an English OCR with 95% character recognition accuracy and let the average number of components in a word be 5. The average word recognition error would be $1 - (0.95)^5$, or 22.6%.

In the next Section, we shall present the previous work related to this work. The word image clustering method is explained in Section 4.3. The schemes to correct OCR errors by fusing cluster information is presented in Section 4.4 and Section 4.5. Our experiments and results are detailed in Section 4.6, and we conclude in Section 4.7.

4.2 Previous Work

OCR systems have been well studied and engineered for the Latin script [55]. The performance of English language OCR systems is acceptable in the case of good quality scanned documents [82]. Google Books, for example, uses text from OCR to index books in its English language collection. However, work on the recognition of complex Indian scripts such as Telugu are quite limited [37, 43, 65]. Due to the unavailability of a large corpus most of the experiments were done only on a limited number of pages. The classification results were reported at character or at sub-character level. For example, Negi *et al.*[65] report a component level accuracy of 92% on 2524 components.

In this work, we use a base-OCR proposed by Jawahar *et al.* [43]. An SVM based ensemble of classifiers is learned for labeling. Each SVM performs a one-vs-rest classification for a given character. The classifiers for all the labels are arranged in a Decision Directed Acyclic Graph (DDAG). The appropriate architecture for this DDAG is learn from training data. For N distinct character labels, the DDAG consists of ${}^{N}C_2$ classifiers. The classification of a given character image takes only N steps to identify the label. In the Telugu script, the number of classes or N is typically 432. This classifier has a character

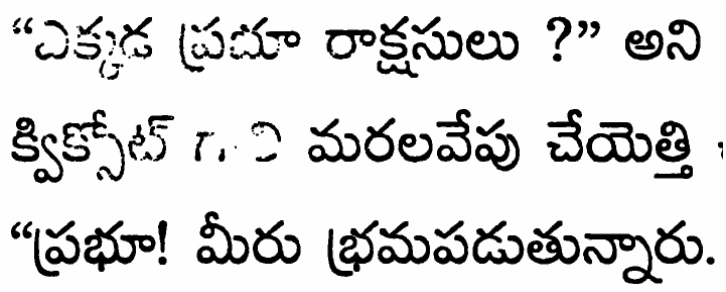


Figure 4.3 A sample segment from a document image in our collection. Note the large amount of degradations in the characters. Some of the words such as the second word on lines 1 and 2 are not even human readable.

recognition accuracy of 97.87% on 1 million synthetically generated and degraded character sets. The accuracies are much lower on real document images in part because of segmentation and degradation issues, as shall be seen in Section 4.6.

In the presence of OCR errors, post-processing is very useful to correct classifier mistakes. This could be achieved by building a character error model [44], which is then used to correct frequently confused characters. Another popular approach is to use a dictionary to rectify errors resulting in invalid words [51]. Statistical language models such as n-gram models are also quite successful in reducing the OCR errors [49, 61]. However, such post-processing is challenging for our dataset of Telugu books. But, the use of language models for post processing is quite challenging for many Indian languages like Telugu. This is because of the large number of possible words, which makes dictionaries challenging to build. Moreover, it is necessary to model joint probabilities for language models, at the sub-UNICODE level.

Most OCR literature is focused on recognizing individual characters and words with little work on exploiting the constraints like a book is mostly dominated by a single font, and a word can occur multiple times. Mathis and Breuel [56] introduce a style-conscious classification by learning mixture models for various styles of characters. Sarkar and Nagy [78] use the stylistic similarity within a document to improve recognition performance over handwritten numerals. Recent attempts at book level OCR include Xiu and Baird [87] who propose a mutual-entropy based model adaptation.

We investigate a different approach which exploits the similarity of word images in a book. This is based on the idea of word spotting introduced by Rath and Manmatha [73, 74] which looks at creating clusters of word images by image matching. They primarily focused on searching handwritten word images using Dynamic Time Warping (DTW). The one weakness of DTW is that it is slow. This was overcome by Kumar *et al.* [46] who showed that locality sensitive hashing could instead be used to index and search word images rapidly within a book. However, they did not perform explicit recognition.

Similar work was undertaken by Marinai [55], who explored clustering at character level by which he was able to recognize merged characters over one book. Tao and Hull [39] also proposed the use of word image clusters to improve OCR accuracy. Their clusters were built using image matching techniques like Euclidean distance matching and they used a simple majority voting technique with dictionary lookup to improve accuracy on 16 pages of English degraded using synthetic means. We note that their image matching techniques [73] and their majority voting technique are not very accurate or fast for the book level application we envisage.

This work instead uses locality sensitive hashing [27] for clustering word images. The clustered word images are all assumed to be the same word and their recognition outputs are aligned to find the most likely word. We investigate three techniques - the first one is an approximate technique called character majority voting, the second is based on alignment using dynamic time warping of all pairs of word images and the third a progressive alignment technique - to improve OCR accuracy.

4.3 Word-Image Clustering

The goal of clustering word-images, is to partition the word images in the collection into groups. Each group is expected to contain all instances of a given word from all the pages in the dataset. Hence the number of such groups would equal the number of unique words in the collection, which is unknown. This means the clustering has to proceed with agglomerative or divisive methods, with a carefully chosen stopping criterion. However, these techniques are computationally expensive, generally of the order $O(N^2 \cdot logN \cdot d)$, where d is the length of the feature vector and N is the number of word-images to be clustered. The clustering time becomes prohibitive, once the collection of word-images scales up.

One of the major time-consuming steps in clustering is the identification of Nearest Neighbors (NNs) that can be clustered together. By replacing the exact NN problem with an approximate-Nearest neighbor problem, one can achieve significant speedup in the clustering process [27, 46]. The approximate NN problem is solved using Locality Sensitive Hashing (LSH). LSH builds a hash over the dataset, which allows for quickly retrieving the approximate NNs for each test point. Once the NNs are obtained, clustering can proceed with significant speedup.

4.3.1 Locality Sensitive Hashing

The intuition behind Locality Sensitive Hashing is as follows. The dot product $a.x$ projects each feature vector onto a real line. This real line is chopped into equi-width segments of appropriate size w and hash values are assigned to vectors based on which segment they project onto [27]. The hash functions are chosen based on p-stable distributions that works for all $p \in (0, 2]$. The hashing technique uses several such hash functions so as to ensure that, for each function, the probability of collision (features falling into the same bin) is much higher for words which are similar than for those which are dissimilar. A hash function $g_i, i = 1, \ldots, l$ is used to compute a hash code into a second level of hashes

which stores the corresponding index for fast retrieval. This procedure is repeated L times, therefore each point will belong to L tables. For fixed a, b chosen uniformly from the range $[0, w]$, the hash function $h_{a,b}(x)$ is given by,

$$h_{a,b}(x) = \left\lfloor \frac{a.x + b}{w} \right\rfloor$$

The value of w is generally chosen to be 4.

4.3.2 Clusters from Hashing

After the hash is built over the word-images, each word is queried against the hash to obtain the approximate nearest neighbors (NNs) within a distance T. Given a query word feature, the L first level hash codes are determined and all the words within the corresponding second-level hash tables are retrieved. The NNs for a given query are used to form a cluster for that query. The query process is repeated for every unclustered word from the collection. Thus, all words from the document images are clustered into groups of similar words. The clustering procedure is outlined in Algorithm 1.

Input: Word Images W_j and Features $F_j, j = 1, \ldots, n$
Output: Word Image Clusters **O**
for *each* $i = 1, \ldots, l$ **do**
    for *each* $j = 1, \ldots, l$ **do**
        Compute hash bucket $I = g_i(F_j)$
        Store word image W_j on bucket I of hash table T_i
    end
end
$k = 1$
for *each* $i = 1, \ldots, n$ *and* W_i *unmarked* **do**
    Query hash table for word W_i to get cluster O_k
    Mark word W_i with k
    $k = k + 1$
end

Algorithm 1: Word Image Clustering

We observe that, in general, multiple occurrences of a given word are clustered together. This generally includes noisy images, same words with degradations at different characters, etc. Figure 4.4 shows example clusters for English and Telugu word images. Such clusters are the ideal candidates for correcting the OCR errors, since the errors in recognizing one character could be corrected by using evidence from other correctly recognized characters. Sometimes words with the same stem and small word-form variations may also be grouped in a cluster. Words with extra character/symbols are also grouped if the stem content is same.

Figure 4.4 Example clusters obtained using LSH over word images. Noisy variants are highlighted in red.

4.3.3 Features for Word-Image Clustering

Among various features available in the literature for representing word images, profile features are popularly used for word retrieval and recognition problems [73]. The profile features that are extracted, consists of:

- The Projection Profile: the number of ink pixels in each column.
- Upper and Lower Profile: which measures the number of background pixels between the word and the word-boundary
- Transition Profile: given as number of ink-background transitions per column.

Profiles for an example word are shown in Figure 4.5. Each profile is a vector whose size is the same as the width of the word. The dimensionality of the feature, therefore, varies with the word width. However, we require a fixed length description to use these features for LSH based clustering.

Fixed length descriptions for profile features may be obtained by computing a Discrete Fourier Transform (DFT) of the profiles [46, 75]. The noisy higher order coefficients of the DFT are discarded resulting in a robust representation for the word-images. We use 84 Fourier coefficients for each of the profile features. With this representation, word-images may be matched by comparing feature vectors using an Euclidean distance metric, which is now used for LSH.

4.3.4 Evaluating Clusters

During clustering, it is important to know the right radius T, with which to query the LSH for the NNs. With a low radius, different instances of the same word may be clustered separately. On the other hand a large radius may result in noisy clusters. The quality of clustering may be quantified using the cluster purity measure. Cluster purity measures whether a cluster is uncontaminated by items that are not supposed to be part of the cluster. However, this measure is biased towards small clusters. In the extreme case, if each word were its own cluster then the cluster purity would be 100%. This is

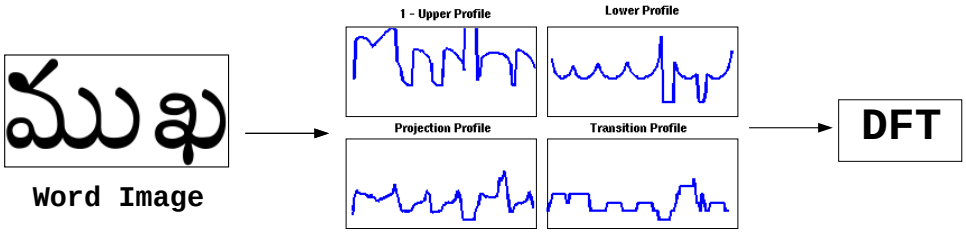


Figure 4.5 Extraction of profile features and their fixed length versions. For each word image, Upper, Lower, Projection and Transition Profiles are computed. These features are run through a DFT, and the top 84 coefficients chosen for each.

undesirable and hence we measure the clustering quality using the F-*score* measure, which is popular in the document retrieval community. The F-score is given as

$$F_\beta = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * recall + precision}$$

To compute the precision and recall, we evaluate a series of $N \cdot (N-1)/2$ pairwise relationships for each of the word images. If a pair of word-images belonging to the same word are clustered together, such a pair contributes to the true-positives(TP). If such a pair is assigned to different clusters, it would be counted as a false-negative(FN). When two word-images are clustered together, but belong to different words it is considered as a false-positive(FP). Based on these computed values, the precision and recall are obtained as

$$precision = \frac{TP}{TP + FP}, recall = \frac{TP}{TP + FN}$$

In our F-score computation, we would like to penalize false positives more than false negatives. This is achieved by using a $\beta > 1$, typically 5. This gives more weight to precision or cluster purity. The best performing T is obtained by evaluating the clustering over a subset of the collection. The plot of the F-score against various values of T is shown in Figure 4.6. For the Telugu script, a T of 40 seems to be the best performing for both English and Telugu. At this threshold, the cluster purity is about 95%.

4.4 Word Error Correction

In our approach, we propose to exploit the multiple occurrences of a word across the collection, to correct OCR errors for such instances. The idea is to propagate the OCR output at each character location across the cluster, which is then fused to identify the right word. The word error correction is based on the following assumptions:

1. Words generally repeat across a book. Each of the occurrences is visually similar to the other. The variations are limited to some of the characters.

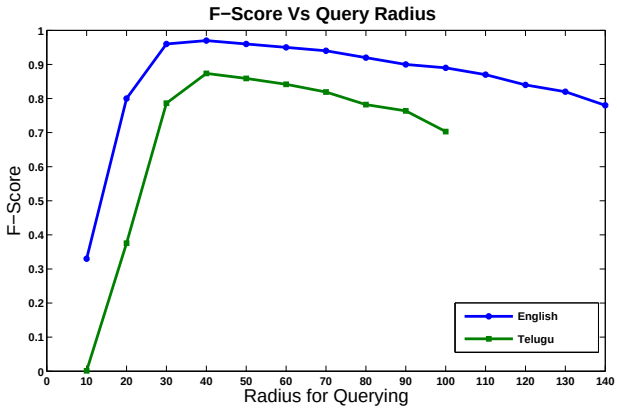


Figure 4.6 Cluster performance across various radii for querying LSH, of English and Telugu books.

2. OCR errors are typically non-deterministic. They occur at different locations of a word, for different instances of the same word.

3. OCR errors are local to certain characters. Specifically, the edit distance between the real word and the recognized word is generally small.

Let us assume that we are given a cluster consisting of word-images belonging to the same word, such as the one shown in Figure 4.7. The OCR output for these word-images is shown alongside. The fusion operator combines the given text strings of a cluster and generates the most likely outcome for the cluster label. An ideal fusion operator, would be able to generate the correct string for the cluster, even when all the words in the cluster are wrong. This is achieved by identifying the OCR errors, which are generally distributed across the word. Such errors could then be corrected by fusing information from the cluster.

A naive scheme would assign the most frequently occurring string to the entire cluster. However, it is possible that no one word enjoys a majority in the cluster, such as the case in our example above. Given these words alone, no single word could be used to label the entire cluster. To address this issue, we use a method called Character Majority Voting (CMV).

4.4.1 Fusion of OCR results

A basic fusion scheme is called the Character Majority Voting (CMV). CMV exploits the fact that the OCR errors are distributed across the word, and generally do not repeat the same errors. For example, in Figure 4.8, we can observe that for each character position, there are sufficient examples where the character was correctly recognized. The symbols of each word are lined up against one another. Then the candidates for each symbol position are chosen by selecting that character which is in the majority

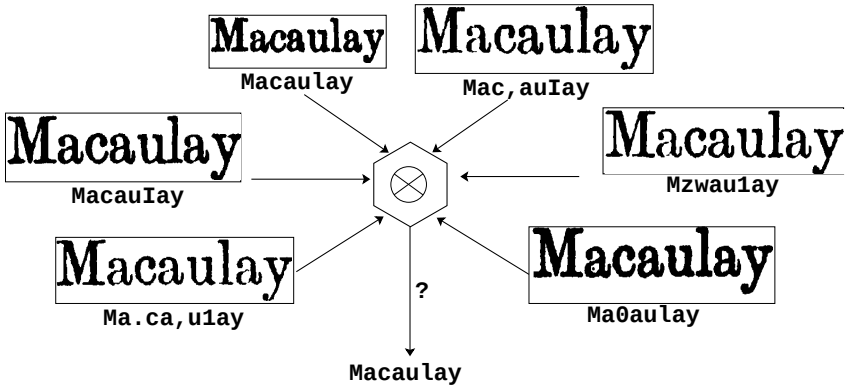


Figure 4.7 The Fusion operator for word error correction. Erroneous OCR results from multiple instances of the same word are fed to it. The Fusion operator then predicts the correct recognized text for these words together.

Word Image	OCR output	Symbols							
పరిస్థితి	పరిస్థితి	ప	◡	రి	స	్	థ	తి	
పరిస్థితి	నరిస్థితి	న	◡	రి	స	్	థ	తి	
పరిస్థితి	పరిస్థితి	ప	◡	రి	స	్	థ	·	తి
పరిస్థితి	పరిస్థితి	ప	◡	రి	స	్	ధ	తి	
పరిస్థితి	పరిస్థితి	ప	◡	రి	సి	థ	తి		
CMV	పరిస్థితి	ప	◡	రి	స	్	థ	తి	

Figure 4.8 Character Majority Voting based word correction. 1st column shows the word images in the cluster, 2nd column gives the OCR outputs. The remaining columns show the alignment across the strings. Note variants and errors in red.

at each position. If no candidate enjoys the majority, the existing candidate is not replaced. In CMV, the average length of the word is computed from the length of all words existing in the cluster.

Figure 4.8 demonstrates the working of CMV. It shows word images and errors in red in the first column (some of the small red circles are breaks in the character). The OCR symbols are then expanded in order (note that Indian languages use vowels to modify consonants and hence the modifier follows

the consonant in this symbol list). The CMV looks at the majority in each column and assigns that to be the output which is listed at the bottom.

The drawback of CMV is that it works well when the lengths of all the words in the cluster are the same. When the lengths of the strings are different, characters are stacked up in the wrong positions, which leads to errors in the correction step. This is avoided and improved upon by using a dynamic time warping based alignment, as detailed in Section 4.5.

4.5 Alignment based Error Correction

The length constraint of CMV may be violated with the presence of degradations in the characters, which makes the OCR to interpret and recognize a single character as multiple different characters. For example, in the third word of Figure 4.8 stray symbols are attached to the character, resulting in variable length strings for the word. In this case, the length of the majority of the OCR output was correct, which might not always be the case. The correction scheme needs to incorporate varying length OCR outputs in the cluster.

This alignment is performed using a Dynamic Programming (DP) technique called Dynamic Time Warping (DTW) [73]. The alignment costs $D[i, j]$, between the two words is populated using the recurrence relation

$$D(i,j) = \max \begin{cases} D(i-1, j-1) + d(w_{1,i}, w_{2,j}) \\ D(i-1, j) + 1 \\ D(i, j-1) + 1 \end{cases}$$

where $d(\cdot, \cdot)$ is the Kronecker delta function.

Let $\mathbf{C}$ be a cluster of n similar words obtained from the recognizer. Each word $w \in \mathbf{C}$ is aligned with other words $\mathbf{C} - w$ of the cluster. All matching characters of two words are aligned and the unmatched characters are identified as possible errors in word w of the recognizer. The unmatched characters are candidates to replace the erroneous characters in word w. Alignment of w with all other words of the cluster gives a group G of possible character replacements for errors. Hence, for every erroneous character $E_k, k = 1, \ldots, m$ of w there are $n-1$ possible corrections. The probability p_k of the possible correction is computed using maximum likelihood - in this case it is just the count of each candidate at that position divided by the total number of candidates at that position. If there is only one candidate at a position the probability is set to zero. A wrong character is replaced by a new character G_k for which p_k is maximum. The steps of this process are outlined in Algorithm 2. This procedure is repeated to correct every word of a cluster $\mathbf{C}$.

The DTW alignment occurs between pairs of strings, and one may find multiple ways of combining the various pairwise results from a cluster. We present two solutions, based on Progressive String Alignment (PSA) and fixed-length pairwise Dynamic Time Warping (DTW).

Input: Cluster $\mathbf{C}$ of words $W_i, i = 1, \ldots, n$

Output: Clusters $\mathbf{O}$ of correct words

for *each* $i = 1, \ldots, n$ **do**

  for *each* $j = 1, \ldots, n$ **do**

    if $j \neq i$ **then**

      Align word W_i and W_j

      Record errors $E_k, k = 1, \ldots, m$ in W_i

      Record possible corrections G_k for E_k

    end

  end

  Correct E_k if Probability p_k of correction G_k is maximum

  $O \leftarrow O \cup W_i$

end

Algorithm 2: Word Error Correction

4.5.1 Pairwise Dynamic Time Warping (DTW)

In the pairwise-DTW method, the right length L of the string is identified from the mode of the length distribution in the cluster. The set of words corresponding to this length is identified as S_L, the other words are the set $\mathbf{C} - S_L$. A DTW based alignment is performed between each $w \in S_L$ and $w' \in \mathbf{C} - S_L$. The characters for the two words are aligned, and a null-character is used to pad for unmatched characters. If the length of the aligned string is longer than the pre-defined length L, the characters corresponding to the null-string are removed. Thus all the aligned strings from each pairwise match between w, w', are now of the same length L. A character majority voting is now performed on these aligned strings to obtain the corrected word from the OCR outputs.

Figure 4.9 shows an example of a cluster (first column), the OCR output (second column) and the CMV and pairwise DTW corrections. The red ovals show problems with the images in the cluster and OCR errors. The green ones show the corrected outputs. In spite of 3 out of the 4 OCR outputs being wrong, the output can still be corrected.

4.5.2 Progressive String Alignment

In the previous method, it was required that the length of the word be known at each iteration. This was observed to be too restrictive, giving little scope to correct early errors. To handle this problem, we use a progressive string alignment method. This technique is linear in the number of words in the cluster [84] and hence much faster than pairwise DTW.

In progressive string alignment, we begin with a pair of strings w_1 and w_2 which are then aligned at characters that match. Once aligned, the character positions where the two strings mismatch are

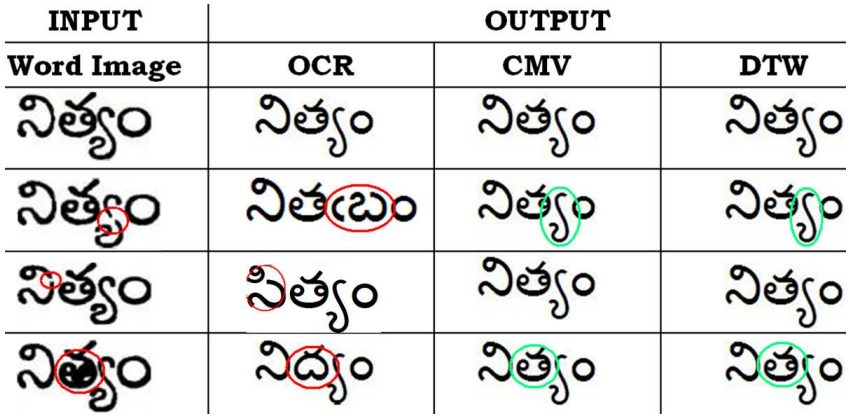
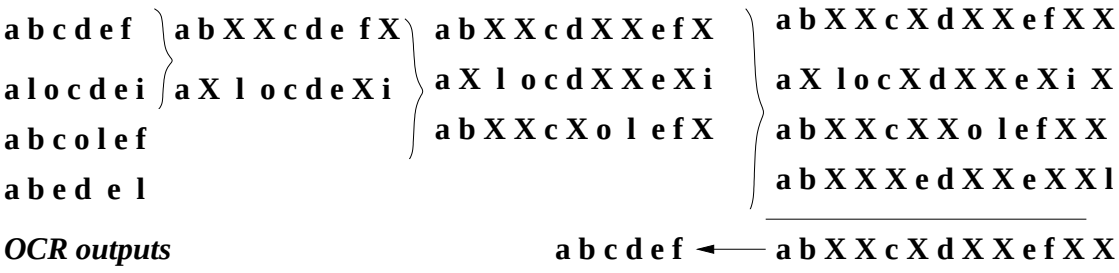


Figure 4.9 Example word error correction. 1st column word images in cluster, 2nd OCR output, 3rd CMV and 4th DTW output. Note variants and errors in red and corrections in green.

replaced by a null-character, say X. Let us call this aligned string as W_i, where i keeps track of the current iteration. In the next iteration, $i + 1$ a new word w_3 is aligned with W_1. The procedure is repeated until all the words in the cluster are aligned.

This procedure is illustrated below. Let us assume that the cluster of word-images corresponds to the text *abcdef*. Erroneous OCR outputs for these images is listed on the left column. At each step, the strings are progressively aligned as shown below:



To identify the correct text string from the aligned strings, we use the Character Majority Voting, discussed in Section 4.4.1. All character positions where the null-character X occurs more often than a regular character are ignored. By this procedure, the final string for the above example was correctly obtained as *abcdef*, in spite of multiple errors in the OCR outputs.

An example for alignment of Telugu words is given in Figure 4.10. The PSA algorithm aligns the symbols correctly creating gaps where symbols are not aligned.

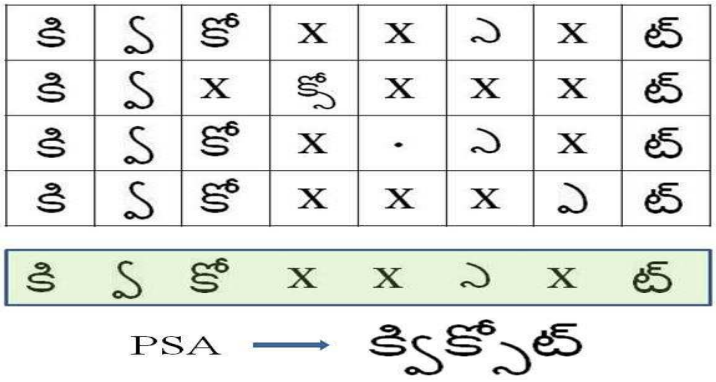


Figure 4.10 Progressive String Alignment based error correction. The first 4 rows are aligned outputs of the symbols. Last row is the output obtained if at least a majority of the characters agree. The final string is correctly identified in this case.

4.6 Experiments and Results

We begin with performing experiments over a controlled dataset. This allows us to work with labeled word images, allowing us to tune the algorithm parameters. We could also model ideal clustering results and identify the loss of accuracy due to LSH based clustering. The idea here is to assume perfect clustering and to study the effect of word length and the number of words in a cluster on the error detection and correction algorithms.

4.6.1 Controlled Experiments

This dataset has 5000 clusters. Each cluster has 20 images of the same word with different font sizes, and resolution. The words are generated by font-rendering using ImageMagick. These words are then calibrated (degraded) using the Kanungo degradation model [88] to approximate real data. The word generation process makes correct annotations available for evaluating the performance of the algorithm. The calibrated data sets were then divided into clusters with the number of words ranging from 2 to 20. This experiment examines the effect of the number of words in a cluster on accuracy. A Telugu OCR [43] was used to generate the OCR results. We expect the error correction to get better if there are more words in a cluster but to saturate beyond a certain number of words and this is what we see in Figure 4.11. Figure 4.12 shows that accuracy decreases with word length. We would expect the OCR word error rate to increase with word length if the symbol error rate is constant. In these experiments, the DTW technique performs consistently better than CMV.

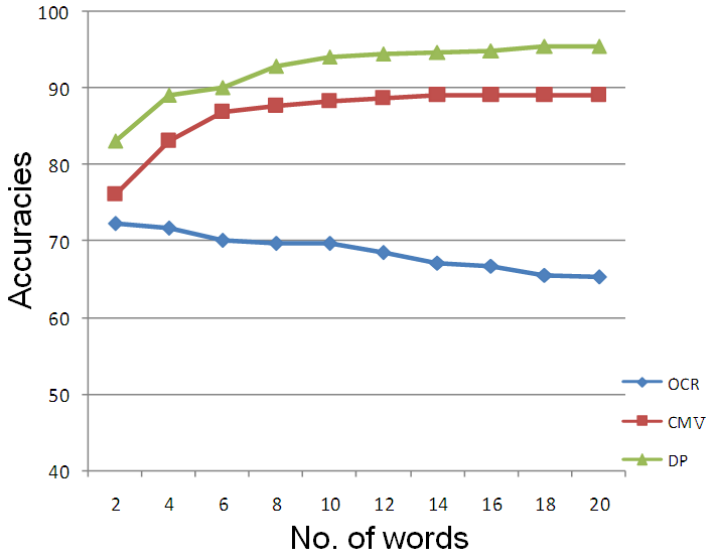


Figure 4.11 Effect of Number of words in cluster over the word recognition accuracy. Over synthetic datasets, the word error correction improves with more points in the cluster. The Dynamic Programming based error correction always outperforms the CMV.

4.6.2 Results on Book Dataset

We next test our method on our dataset of Telugu and English scanned books. To evaluate the performance, the words in the dataset were manually annotated. The base OCR used for English was ABBYY FineReader [11], and the Telugu OCR was the one presented in Jawahar*et al.* [43]. The symbol accuracy estimate thus obtained, was about 96.1% for English and 77% for Telugu. The word recognition accuracy of the English OCR was found to be almost 92.4%, while that of Telugu was about 58%. Since our ground truth was available at the word level, the symbol accuracy is estimated as the number of matching symbols between the given string and the ground truth. Thus, even when the component level recognition rate of Indian languages is acceptable, the word level recognition rate may be much lower. The results from the English and Telugu books are given in Table 4.1. Following word error correction, we observe that pairwise DTW gives the most error correction rate of all the methods except in one case where the progressive alignment is better. The average word recognition accuracy for all words appearing at least thrice in the Telugu books was found to be about 66%. Our next experiment evaluates the effect of word length on word error correction. The different rows are for words which are short (2-3 symbols), medium (4-5 symbols) or long ($>= 6$ symbols). The first observation is that, from the raw Telugu OCR accuracies, for short words the word accuracy rate is 16% lower than the symbol (or component accuracy). For long words the word accuracy rates are about 30% lower than the symbol accuracies. This is expected, since there is a higher probability of one of the symbols being erroneous

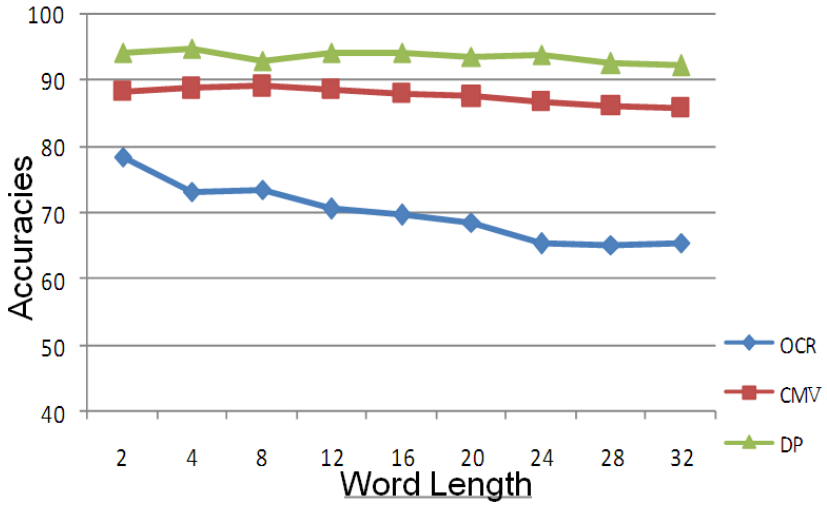


Figure 4.12 Effect of Word Length on word recognition accuracy. The OCR performs poorly with longer strings, since there are more characters that the OCR could go wrong at. The word error correction seems to correct this trend, as seen from the fairly steady plot.

in a longer word. Similar trends are found for the English OCR as well. Secondly, following word error correction, we observe that the correction of errors is almost uniform across various word lengths. This does not agree with what was reported in [72], which appears to be mostly because of a better Telugu OCR output over their data. Further, we see the effect of cluster size on word error correction in Figure 4.13. In larger clusters, the correction of OCR errors is more pronounced than in smaller clusters. For English, the red and blue lines diverge with increasing cluster size. This agrees with our observations from the synthetic dataset as well (Figure 4.11). Some qualitative examples of successful and a few failure cases for Telugu are shown in Figures 4.14, 4.15, 4.16.

4.6.3 Error Analysis

One of the major source of errors in our framework is when the words which are correctly recognized by the OCR, end up in clusters with many erroneous recognition results. The erroneous characters dominate the correct ones, thereby forcing an incorrect alignment result.

In the specific case of one of our Telugu books, the statistics of errors are as follows:

	Size	No. of Clusters	No. of Words	Symbol Accuracy				Word Accuracy			
				OCR	CMV	DTW	PSA	OCR	CMV	DTW	PSA
Telugu	Short	224	2442	82.8	88.2	88.2	83.4	71.2	80.2	80.2	77.9
	Medium	270	2149	74.0	80.2	81	74.4	51.3	58.3	59.1	58.6
	Long	137	786	66.6	73.0	73	68.3	32.2	40.7	41.5	38.7
English	Short	1805	25242	95.3	96.9	96.9	96.9	90.9	93.4	93.4	94.2
	Medium	1158	7501	97.7	99.3	99.3	99.1	96.0	98.9	98.9	98.4
	Long	823	4012	98.2	99.5	99.5	99.3	95.9	99.0	99.0	98.6

Table 4.1 Effect of word length on the word error correction of the different algorithms. See Section 4.6 for details.

Words in Clusters (size $>=$ 3)	5198
Total Word Errors	2754
Corrected Errors	717
Introduced Errors	78
Effective Correction	639
% of errors corrected	23.2%

The 23% corrected words correspond to the 8% of all word errors corrected for this book.

4.6.4 Effect of Clustering Errors

Finally, we evaluate the effect of clustering errors in word error correction accuracies. Errors in clustering could result in mistakes during alignment and CMV. This is especially true for small clusters, where a different word could bias the votes for a wrong character. The comparison between pure clusters and LSH based clusters is presented in Table 4.2. For pure clusters, the labels of the dataset were used to cluster all occurrences of a word together. Similar to previous experiments, we consider only those clusters which have three or more instances.

As can be seen in Table 4.2, for English the word recognition accuracies from LSH based clusters is very close to that from pure clusters. For Telugu, we observe about 4%-5% difference in performance between LSH clusters and pure ones. This implies that one could obtain further improvement in word error correction with a better clustering performance.

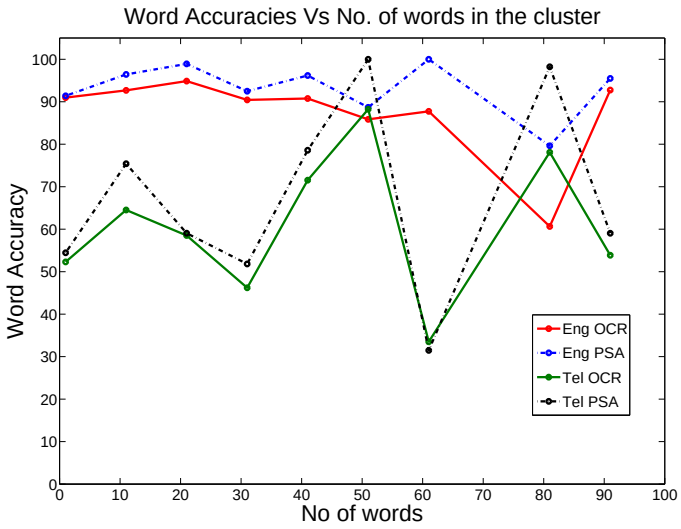


Figure 4.13 Effect of cluster size on word error correction accuracy of Telugu and English Books. It can be seen that the improvement in word accuracies over the base OCR is significantly larger for bigger clusters.

4.7 Summary

A document level word recognition technique is presented, which makes use of the context of similar words to improve the word level recognition accuracy. An error correction technique is presented to improve word accuracy of the raw OCR. An efficient clustering algorithm was used to speed up the process. The experimental results show that the word level accuracy can be improved significantly from about 58% to 66% for Telugu, and from 93.8% to 97% for English. The proposed technique may also be applied to other (Indian) languages and scripts. Future extensions may include the use of techniques to handle unique words by creating clusters over parts of words.

Image	Output							
చేరుకున్నారు.	చే	రు	కు	నా	ఽ	రు	.	
చేరుకున్నారు.	దే	రు	కు	నా	ఽ	రు	.	
చేరుకున్నారు.	చే	రు	కు	నా	ఽ	రు	.	
చేరుకున్నారు.	దే	.	రు	కు	నా	ఽ	రు	.
చేరుకున్నారు.	చే	.	రు	కు	నా	౩	రు	.
CMV	చే	రు	కు	నా	ఽ	రు	.	
DTW	చే	రు	కు	నా	ఽ	రు	.	
Groundtruth	చే	రు	కు	నా	ఽ	రు	.	

Figure 4.14 An example where the errors were corrected by both CMV and DTW methods, inspite of errors in more than half the words in the cluster.

Image	Output						
ప్రయాణం	౨	్	ఎ	యు	ణ	ం	
ప్రయాణం	౨	్	ఎ	యు	ణ	ం	
ప్రయాణం	౨	్	ఎ	యు	ణ	ం	
ప్రయాణం	౨	్	ఎ	యు	ూ	ణ	ం
ప్రయాణం	౨	్	ఎ	యా	‒	ణ	ం
CMV	౨	్	ఎ	యు	ణ	ం	
DTW	౨	్	ఎ	యు	ణ	ం	
Groundtruth	౨	్	ఎ	యా	ణ	ం	

Figure 4.15 An example Telugu word that could not be corrected, because of large number of OCR errors for one particular character.

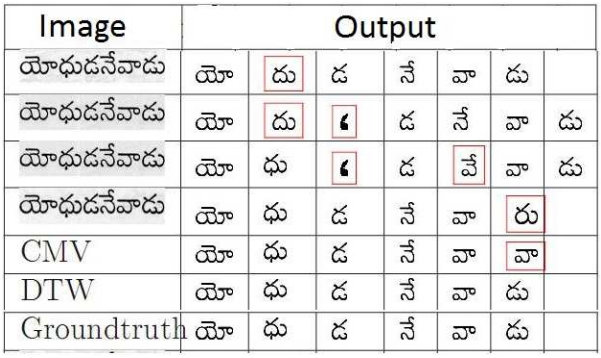


Figure 4.16 A typical example where CMV fails, but DTW correctly aligns and rectifies the OCR errors.

		LSH Clusters	**Pure Clusters**
English	OCR	92.4	92.6
	CMV	95.1	95.3
	DTW	95.2	95.3
	PSA	95.6	95.8
Telugu	OCR	57.6	58.2
	CMV	65.9	70.3
	DTW	66.3	70.7
	PSA	64.5	69.1

Table 4.2 Comparison of word error correction between LSH based clustering and pure clusters. Note that the difference between the word recognition accuracies are insignificant for English, and around 4%-5% for Telugu.

Chapter 5

Conclusions

The exponential rise in the number of documents being archived by many applications like Digital libraries requires tools for the effective access of these collections. The diverse nature and the large quantity of documents make this a challenging task. We attempt to solve this problem by developing robust optical character recognizers. The document analysis and understanding of Indian Languages has been lagging behind the recognition systems for languages like English. In this work we propose a classifier system for effectively and efficiently solving the character recognition problem for Telugu, where the character set is very large (in the order of hundreds). We have analyzed Telugu script and their language rules for a building a language model for better document understanding. We have explored machine learning and pattern recognition algorithms for designing feature extraction, classification and post processing mechanisms for recognition of document images. We have analysed the OCR's performance on synthetic documents. We have also extended this analysis to real life documents which are poor in quality. After extensive script analysis, we have designed an OCR for the recognition of Telugu printed document images. We have conducted experiments to evaluate its performance, in which we have got good results on reasonably diverse quality documents. However, the performance of the OCR varies with the diversity of document images under consideration.

Design of robust OCRs is still a challenging task for Indian scripts. The central module of an OCR is a recognizer which can generate a class label for an image component. Classification of isolated characters and thereby recognizing a complete document is still the fundamental problem in most of the Indian languages. The problem becomes further challenging in presence of diversity in input data (for example, variations in appearance with fonts and styles). We have shown that high classification accuracies can be obtained for character classification problem with the help of SPHOG-SVM combination. Experiments were conducted on 1453950 Telugu character samples in 359 classes and 15 fonts. On this data set, we have obtained an accuracy of 96-98% with an SVM classifier.

A document level word recognition technique is presented, which makes use of the context of similar words to improve the word level recognition accuracy. An error correction technique is presented to improve word accuracy of the raw OCR. An efficient clustering algorithm was used to speed up the process. We demonstrate our approach on a larger dataset of multiple scanned books in an Indian script

Telugu. We further demonstrate the approach over hundreds of English language document images as well. The experimental results show that the word level accuracy can be improved significantly from about 58% to 66% for Telugu and from 93.8% to 97% for English. The proposed technique may also be applied to other (Indian) languages and scripts.

Techniques which have been implemented and tested on Telugu language in this thesis are as follows:

- Optical Character Recognizer (OCR) for Telugu adopted from Malayalam OCR. [Chapter 2]
- Reordering and Unicode rule based models. [Chapter 2]
- Classifiers based on synthetic and real datasets. [Chapters 2, 3, 4]
  - Linear Support Vector Machine (SVM) classifier with One Vs. All architecture.
  - Linear SVM classifier with DDAG architecture.
  - Intersection Kernel SVM (IKSVM).
- Features used [Chapter 2,3,4]
  - Raw pixels
  - Principal Component Analysis (PCA)
  - SPatial Histogram Of Gradients (SPHOG)
  - PCA with SPHOG
- Locality Sensitive Hashing (LSH) for clustering. [Chapter 4]
- Word error correction algorithms [Chapter 4]
  - Majority voting
  - Pairwise Dynamic Time Wrapping [DTW]
  - Progressive String Alignment [PSA]

This work opens up many interesting problems in document understanding in the Indian Language context. In the thesis, we have discussed several problems which reduce the OCR's accuracy. Though we have provided solutions to these problems, a lot more has to be done.

- Left out confusions with the SPHOG-SVM combination are associated only to a small percentage of the classifier and a post-processing classifier with an uncorrelated feature set can successfully boost the overall classification performance.
- The SPHOG-SVM combination can also be used for any language, since it is language independent.

- The document-OCR incorporates information from the entire document to reduce word error rates. The improvement in accuracy for words which occur at least thrice in the collection is about 8% for Telugu and 4% for English. Future extensions may include the use of techniques to handle unique words by creating clusters over parts of words.
- The approach may be applied to improve the accuracy of any OCR run on documents in any language. Here we have shown for Telugu and English documents.

Appendix A

Appendix

A.1 Characters List

The class labels used for designing Telugu OCR are as show in the following figures:

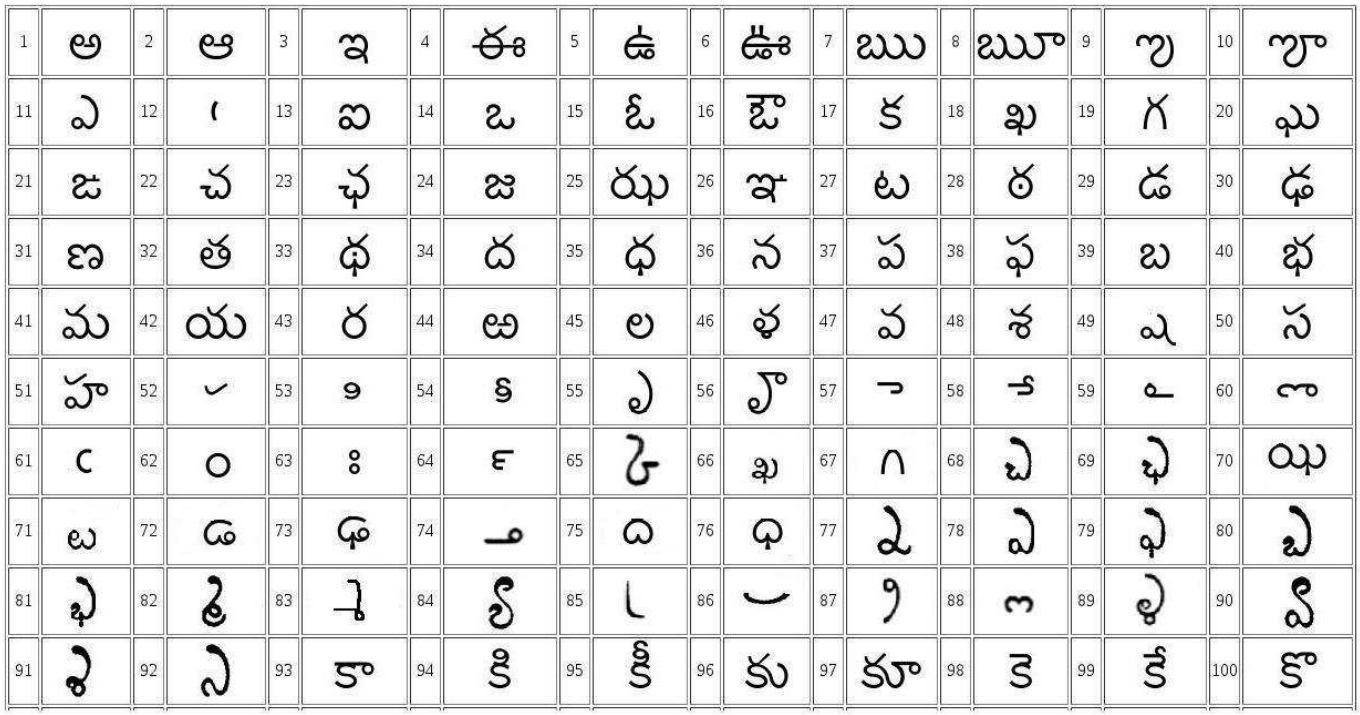


Figure A.1 Classes used in Telugu OCR

Figure A.2 Classes used in Telugu OCR

Figure A.3 Classes used in Telugu OCR

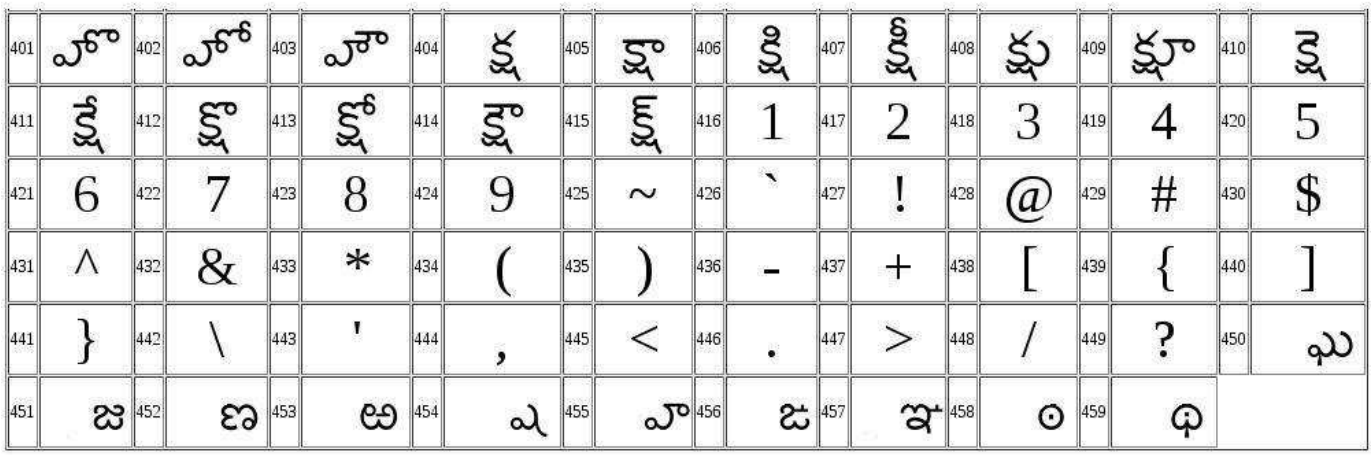


Figure A.4 Classes used in Telugu OCR

Related Publications

The work done during my masters has been disseminated to the following conferences:

- Venkat Rasagna, Anand Kumar, C.V. Jawahar and R. Manmatha, **Robust Recognition of Documents by Fusing Results of Word Clusters**, *Proceedings of 10th International Conference on Document Analysis and Recognition(ICDAR 09)*, 26-29 July, 2009, Barcelona, Spain.
- Venkat Rasagna, Jinesh K.J. and C.V. Jawahar, **On Multifont Character Classification in Telugu**, *Proceedings of International Conference on Information Systems for Indian Languages (ICISIL 2011)*, 09-11 March, 2011, Punjab, India.

Bibliography

[1] Google books. *http://books.google.com*.

[2] http://en.wikipedia.org/wiki/levenshteindistance.

[3] http://en.wikipedia.org/wiki/telugulanguage.

[4] http://www.omniglot.com/writing/telugu.htm.

[5] The indexing and retrieval of document images: A survey.

[6] Internet archive. *http://www.archive.org*.

[7] Open content alliance. *http://www.opencontentalliance.org*.

[8] Project gutenberg. *http://www.gutenberg.org/wiki/MainPage*.

[9] The universal digital library. *http://www.ulib.org*.

[10] Yahoo directory. *http://dir.yahoo.com*.

[11] ABBYY FineReader software from. http://finereader.abbyy.com/.

[12] K. H. Aparna and V. S. Chakravarthy. A complete ocr system development of tamil magazine documents. *Tamil Internet*, pages 22–24, 2003.

[13] K. S. Baird. Anatomy of a versatile page reader. *Proceedings of the IEEE*, 80(7):1059–1065, 1992.

[14] V. Bansal and R. M. K. Sinha. Integrating knowledge sources in devanagari text recognition system. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(4):500–505, 2000.

[15] V. Bansal and R. M. K. Sinha. A complete ocr for printed hindi text in devanagari script. In *Proceedings of Sixth International Conference on Document Analysis and Recognition*, pages 800–804. IEEE, 2001.

[16] V. Bansal and R. M. K. Sinha. A devanagari ocr and a brief overview of ocr research for indian scripts. *Proceedings of STRANS01, IIT Kanpur*, 2001.

[17] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 509–522, 2002.

[18] A. Bharati, K. Prakash Rao, R. Sangal, and S. M. Bendre. Basic statistical analysis of corpus and cross comparison among corpora. In *Proceedings of 2002 International Conference on Natural Language Processing, Mumbai, India*, 2002.

[19] M. Bokser. Omnidocument technologies. *Proceedings of the IEEE*, 80(7):1066–1078, 1992.

[20] C. Breiteneder and H. Eidenberger. Content-based image retrieval in digital libraries. In *International Conference on Digital Libraries: Research and Practice, Kyoto*, pages 288–295. IEEE, 2000.

[21] M. G. Brown, J. T. Foote, G. J. F. Jones, K. S. Jones, and S. J. Young. Open-vocabulary speech indexing for voice and video mail retrieval. In *Proceedings of the fourth ACM international conference on Multimedia*, pages 307–316. ACM, 1997.

[22] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[23] J. Chan, C. Ziftci, and D. Forsyth. Searching off-line arabic documents. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006*, volume 2, pages 1455–1462. IEEE, 2006.

[24] B. B. Chaudhuri and U. Pal. An ocr system to read two indian language scripts: Bangla and devnagari (hindi). In *Proceedings of International Conference on Document Analysis and Recognition*, page 1011. Published by the IEEE Computer Society, 1997.

[25] B. B. Chaudhuri and U. Pal. A complete printed bangla ocr system. *Pattern Recognition*, 31(5):531–549, 1998.

[26] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005.

[27] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual Symposium On Computational Geometry*, pages 253–262. ACM, 2004.

[28] T. E. De Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of International Conference on Computer Vision Theory and Applications*. Citeseer, 2009.

[29] P. A. Devijver and J. Kittler. Pattern recognition: A statistical approach. 1982.

[30] R. O. Duda and P. E. Hart. Pattern classification and scene analysis. *A Wiley-Interscience Publication, New York: Wiley, 1973*, 1, 1973.

[31] R. O. Duda, P. E. Hart, D. G. Stork, et al. *Pattern classification*, volume 2. wiley New York, 2001.

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

[33] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2008*, pages 1–8. IEEE.

[34] S. Feng and R. Manmatha. A hierarchical, hmm-based automatic evaluation of ocr accuracy for a digital library of books. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL'06*, pages 109–118. IEEE, 2006.

[35] J. Foote. An overview of audio information retrieval. *Multimedia Systems*, 7(1):2–10, 1999.

[36] H. Fujisawa. Forty years of research in character and document recognition–an industrial perspective. *Pattern Recognition*, 41(8):2435–2446, 2008.

[37] V. Govindaraju and S. Setlur. *Guide to OCR for Indic Scripts: Document Recognition and Retrieval*. Springer-Verlag New York Inc, 2009.

[38] T. Hastie and R. Tibshirani. Classification by pairwise coupling. *Annals of Statistics*, pages 451–471, 1998.

[39] T. Hong and J. J. Hull. Improving ocr performance with word image equivalence. In *Fourth Symposium on Document Analysis and Information Retrieval*, volume 2, pages 177–190, 1995.

[40] IEEE. *Empirical evaluation of character classification schemes*, 2009.

[41] P. Ilayaraja, N. V. Neeba, and C. V. Jawahar. Efficient implementation of svm for large class problems. In *19th International Conference on Pattern Recognition, ICPR 2008*, pages 1–4. IEEE.

[42] A. K. Jain and B. Chandrasekaran. Dimensionality and sample size considerations in pattern recognition practice. *Handbook of statistics*, 2:835–855, 1982.

[43] C. V. Jawahar, M. Pavan Kumar, and S. S. Ravi Kiran. A bilingual ocr for hindi-telugu documents and its applications. In *Proceedings of Seventh International Conference on Document Analysis and Recognition*, pages 408–412. IEEE, 2003.

[44] S. Kahan, T. Pavlidis, and H. S. Baird. On the recognition of printed character of any font and size. *IEEE PAMI*, 9(2):274–288, 1987.

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

[46] A. Kumar, C. V. Jawahar, and R. Manmatha. Efficient search in document image collections. In *Proceedings of the 8th Asian Conference on Computer vision*, pages 586–595. Springer-Verlag, 2007.

[47] K. S. Kumar, S. Kumar, and C. V. Jawahar. On segmentation of documents in complex scripts. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pages 1243–1247, 2007.

[48] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006*, volume 2, pages 2169–2178. Ieee, 2006.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *IEEE*, 86:2278–2324, 1998.

[50] G. Lehal and C. Singh. A complete ocr system for gurmukhi script. *Structural, Syntactic and Statistical Pattern Recognition*, pages 358–367, 2002.

[51] G. S. Lehal, C. Singh, and R. Lehal. A shape based post processor for gurumukhi ocr. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 1105–1109, 2001.

[52] D. G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision, ICCV 1999*, volume 2, pages 1150–1157. Ieee, 1999.

[53] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2008*, pages 1–8. IEEE, 2008.

[54] Maji Subhransu and Malik Jitendra. Fast and Accurate Digit Classification. *Technical Report, University of California, Berkeley*, 2009.

[55] S. Marinai. A survey of document image retrieval in digital libraries. In *9th Colloque International Francophone Sur l.Ecrit et le Document (CIFED)*, pages 193–198. Citeseer, 2006.

[56] C. Mathis and T. Breuel. Classification using a hierarchical bayesian approach. *Pattern Recognition*, 4:103, 2002.

[57] M. Million. *Recognition and Retrieval from Document Image Collections*. PhD thesis, International Institute of Information Technology, Hyderabad, August 2008.

[58] K. Mohan and C. V. Jawahar. A post-processing scheme for malayalam using statistical sub-character language models. pages 493–500, 2010.

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

[60] G. Nagy, T. A. Nartker, and S. V. Rice. Optical character recognition: An illustrated guide to the frontier. *Proceedings of Society of Photo-Optical Instrumentation Engineers (SPIE)*, 3967(1):58–69, 1999.

[61] P. Natarajan, E. MacRostie, , and M. Decerbo. The BBN Byblos Hindi OCR System. *Guide to OCR for Indic Scripts*, pages 173–180, 2009.

[62] N. V. Neeba. *Large Scale Character Classification*. PhD thesis, International Institute of Information Technology, Hyderabad, August 2010.

[63] N. V. Neeba and C. V. Jawahar. Recognition of books by verification and retraining. In *19th International Conference on Pattern Recognition, ICPR 2008*, pages 1–4. IEEE, 2008.

[64] N. V. Neeba and C. V. Jawahar. Empirical evaluation of character classification schemes. In *Seventh International Conference on Advances in Pattern Recognition*, pages 310–313. IEEE, 2009.

[65] A. Negi, C. Bhagvati, and B. Krishna. An ocr system for telugu. In *Proceedings of International Conference on Document Analysis and Recognition*, page 1110. IEEE, 2001.

[66] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[67] U. Pal and B. B. Chaudhuri. Indian script character recognition: a survey. *Pattern Recognition*, 37(9):1887–1899, 2004.

[68] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. *Advances in neural information processing systems*, 12(3):547–553, 2000.

[69] A. K. Pujari, C. D. Naidu, and B. C. Jinaga. An adaptive character recognizer for telugu scripts using multiresolution analysis and associative memory. *Indian Conference on Computer Vision, Graphics and Image Processing*, 2002.

[70] J. R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1):71–72, 1996.

[71] S. N. S. Rajasekaran and B. L. Deekshatulu. Recognition of printed telugu characters. *Computer graphics and image processing*, 6(4), 1977.

[72] V. Rasagna, A. Kumar, C. V. Jawahar, and R. Manmatha. Robust recognition of documents by fusing results of word clusters. In *10th International Conference on Document Analysis and Recognition, 2009. ICDAR'09.*, pages 566–570. IEEE, 2009.

[73] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2003*, volume 2, pages II–521. IEEE.

[74] T. M. Rath and R. Manmatha. Word spotting for historical documents. *International Journal on Document Analysis and Recognition*, 9(2):139–152, 2007.

[75] T. M. Rath, R. Manmatha, and V. Lavrenko. A search engine for historical manuscript images. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval*, pages 369–376. ACM, 2004.

[76] Y. Rui, T. S. Huang, and S. F. Chang. Image retrieval: Past, present, and future. In *International Symposium on Multimedia Information Processing*, volume 10, pages 1–23, 1997.

[77] K. Sankar, V. Ambati, L. Pratha, and C. V. Jawahar. Digitizing a million books: Challenges for document analysis. *Document Analysis Systems VII*, pages 425–436, 2006.

[78] P. Sarkar and G. Nagy. Style consistent classification of isogenous patterns. *IEEE PAMI*, 27(1):88–98, January 2005.

[79] C. Y. Suen, S. Mori, S. H. Kim, and C. H. Leung. Analysis and recognition of asian scripts-the state of the art. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 866, 2003.

[80] M. B. Sukhaswani, P. Seetharamulu, and A. K. Pujari. Recognition of telugu characters using neural networks. *International Journal of Neural Systems*, 6(3):317–357, 1995.

[81] K. Taghva, J. Borsack, and A. Condit. Effects of ocr errors on ranking and feedback using the vector space model. *Information processing & management*, 32(3):317–327, 1996.

[82] K. Taghva, J. Borsack, and A. Condit. Evaluation of model-based retrieval effectiveness with ocr text. *ACM Transactions on Information Systems*, 14(1):64–93, 1996.

[83] F. Tang, R. Crabb, and H. Tao. Representing images using nonorthogonal haar-like bases. *IEEE transactions on pattern analysis and machine intelligence*, pages 2120–2134, 2007.

[84] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[85] M. Vidyasagar. A theory of learning and generalization. *Book review*, 35(1983):1889, 1997.

[86] F. M. Wahl, K. Y. Wong, and R. G. Casey. Block segmentation and text extraction in mixed text/image documents. *Computer Graphics and Image Processing*, 20(4):375 – 390, 1982.

[87] P. Xiu and H. S. Baird. Whole-book recognition using mutual-entropy-driven model adaptation. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6, page 5, 2008.

[88] Q. Zheng and T. Kanungo. Morphological degradation models and their use in document image restoration. In *Proceedings of International Conference on Image Processing, 2001*, volume 1, pages 193–196. IEEE, 2001.