Applying the OCRopus OCR System to Scholarly Sanskrit Literature

Thomas M. Breuel

DFKI and University of Kaiserslautern
Kaiserslautern, Germany
`tmb@informatik.uni-kl.de`

Abstract. OCRopus is an open source OCR system currently being developed, intended to be omni-lingual and omni-script. In addition to modern digital library applications, applications of the system include capturing and recognizing classical literature, as well as the large body of research literature about classics. OCRopus advances the state of the art in a number of ways, including the ability easily plug in new text recognition and layout analysis modules, the use of adaptive and user extensible character recognition, and statistical and trainable layout analysis. Of particular interest for computational linguistics applications is the consistent use of probability estimates throughout the system and the use of weighted finite state transducers to represent both alternative recognition hypotheses and statistical language models. In my talk, I will first give an overview of these technologies and their relevance to digital library applications in the humanities, and then focus on the use of statistical language models and their use for the integration of OCR output with subsequent computational linguistic and information extraction modules.

1 Introduction

New digitization techniques, cheap storage, cheap imaging equipment, and fast networking are making large amounts of scholarly literature available in image format, including literature in, and about, Sanskrit. Unfortunately, much of this literature remains inaccessible as existing OCR systems cannot cope well with this type of content, for a variety of reasons.

Large scale scanning efforts, like Google Books and various Million Books projects, tend to rely on scanning methods that result in fairly low image quality relative to the kinds of scanning methods that commercial OCR systems tend to be developed for.

Recognition of Devanagari has been studied, and some academic and commercial systems have been developed, but their performance is not yet at the level of systems for English. Some of the reasons are the greater complexity of the Devanagari script, the large numbers of ligatures, and the large and unusual vocabulary used in academic and historical texts. In addition, scholarly and classical uses of Devanagari may use special typographic features not found in commercial uses of Devanagari.

In addition, scholarly texts are frequently multi-lingual and multi-script, mixing Devanagari and Latin script, and occasionally using Greek letters and IPA symbols.

Overall, for classics applications, we can at least distinguish the following document types:

– *original documents.* Some documents of scholarly relevance are indeed the original written records; these may be handwritten, severely degraded, and use unusual fonts and styles. For Sanskrit, with its long oral tradition and long history, such documents are quite rare.
– *original texts.* Many classical texts have been recorded, edited, and reprinted within the last several centuries, using modern printing techniques and modern typography. Such texts are of great scholarly interest, as are detailed comparisons between the different editions. Although the body of such books primarily consists of the original text, they often also contain scholarly commentary, marginal notes, footnotes, and annotations, requiring multi-script and multi-language OCR.
– *commentaries, analyses, textbooks.* These kinds of documents contain a mix of scholarly writing in modern languages, passages in classical languages, annotations, footnotes, references, and other content.
– *dictionaries, encyclopedias, catalogs.* These kinds of documents contain large amounts of structured information. They are often characterized by non-standard grammars, complex layouts, and complex mixes of scripts and languages.

The OCRopus system (Breuel, 2008) is a multi-lingual and multi-script open source document analysis and OCR system that is actively being developed. In this paper, I will review some of the directions we are taking in adapting OCRopus to the needs of digitizing and analyzing scholarly literature, with an emphasis on Sanskrit.

2 The OCRopus System

Unlike many commercial OCR systems, the OCRopus system uses a strictly feed-forward architecture (Figure 1). That is, each processing stage receives some input and computes an output representation that is then used by the next stage, with no backtracking. From a software engineering point of view, this greatly reduces coupling between components and makes it much easier to "plug in" other layout analysis and text recognition modules. From a pattern recognition point of view, it is not known how to model backtracking correctly statistically, making it difficult to evaluate, test, or optimize OCR systems based on backtracking.

The major stages of the OCRopus system are:

– *Preprocessing* consists of page dewarping and deskewing, noise removal, and page frame detection.

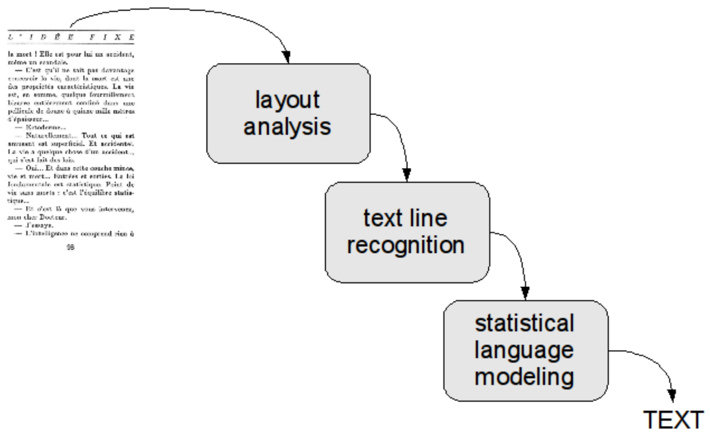


Fig. 1. The OCRopus system architecture is strictly feed-forward, with the three major stages being layout analysis, text line recognition, and statistical language modeling.

- *Layout analysis* performs a separation of text and images on a page, and divides the text regions into text columns, paragraphs and lines.
- *Text line recognition* recognizes linear sequences of characters and outputs the corresponding text and style information, in a format that allows for both segmentation and recognition alternatives.
- *Statistical language modeling* selects the best interpretation of the text taking into account prior knowledge about the vocabulary, orthography, and grammar of the target language.

OCRopus uses only a small number of data types internally, making it easy to reuse parts of OCRopus in other systems, and making it easy to incorporate other systems into OCRopus:

- Images are used for representing input data, as well as page layouts, segmentations, and other spatial data structures.
- Weighted finite state transducers are used for representing the output of text line recognition (that is, text with associated segmentation and recognition alternatives), as well as statistical language models.
- The hOCR format is used for representing the final output of the OCR system; it encodes OCR information in completely standards-compliant HTML files. By building upon existing HTML standards, hOCR automatically has a rich set of representations for typographic phenomena in the world's major languages.

OCRopus is primarily a set of C++ libraries implementing a variety of different document analysis and OCR-related functions. OCRopus contains a set of simple standard internal C++ abstract interfaces for major document analysis and OCR functionality, like isolated character recognition, text line recognition, segmentation, and layout analysis. In addition, most of the OCRopus functionality is available through scripting languages, and major functions are available as command line programs (Figure 2).

```
while pages:nextPage() do
   pages:getBinary(page_image)
   segmenter:segment(page_segmentation,page_image)
   regions = RegionExtractor()
   regions:setPageLines(page_segmentation)
   page = PageNode()
   for i = 1,regions:length()-1 do
      regions:extract(line_image,page_image,i,1)
      lattice = make_FstBuilder()
      bpnet_recognizer:recognizeLine(lattice,line_image)
      bestpath2(line,lattice,langmod)
      page:append(line:utf8())
   end
   document:append(page)
end
```

Fig. 2. The top-level driver loop (in Lua) for multi-page, multi-column OCR with statistical language modeling. Users can easily modify this loop, for example, to use a different recognizer, to change the way language modeling is handled, to construct a different output format, or to pre-process pages or lines to remove noise.

OCRopus can be used in a variety of ways for document analysis and OCR:

– Users can access it through hosted web services, submitting images through a web browser or command line web client, and obtaining document analysis and OCR results as web pages or images.
– Users can run OCRopus from the command line or use it from shell scripts.
– Users can customize how OCRopus performs OCR by scripting the OCR engine in Lua (a widely used, easy-to-use scripting language); in fact, the top-level driver loops are all written in Lua and can be modified to suit particular needs (e.g., different image preprocessing, different output formats).
– Users can add C++ functions and components to OCRopus. Components that conform to the existing OCR-related interfaces can be used as drop-in replacements for functions like layout analysis and text line recognition. Pre-existing OCR systems can be integrated into OCRopus in this way.

3 Preprocessing

The OCRopus OCR system has a fully scriptable library of efficient image processing and statistical operations, including efficient binary morphology and geometric transformations. Obvious uses of these tools include noise removal and other document cleanup.

However, many standard document analysis algorithms can also be formulated as image processing tasks. For example, layout analysis by smearing, layout analysis using the docstrum, and similar high-level operations can be expressed in terms of binary morphology, distance transforms, marker morphology, and statistics.

Even simple shape recognition problems, such as finding staffs, bars, circles, and lines can be expressed using these primitives.

4 Layout Analysis

OCRopus features a number of different layout analysis methods:

– The primary layout analysis method used in OCRopus is based on computational geometry algorithms for finding whitespace and robust least square matching.
– We are developing a trainable statistical layout analysis method that will be included in future versions of OCRopus.
– XY cuts and Voronoi methods are widely used methods from the literature. In benchmarks, we find that these methods do not genernally perform as well as the previous two methods, but they may be useful in some specialized applications.
– Custom "smearing" and morphologically based layout analysis methods can be explicitly scripted in Lua and used in place of one of the built-in methods.

Two key questions that arise in the context of applications to scholarly Sanskrit literature are how well these algorithms work on Devanagari, and how well they work on mixed text.

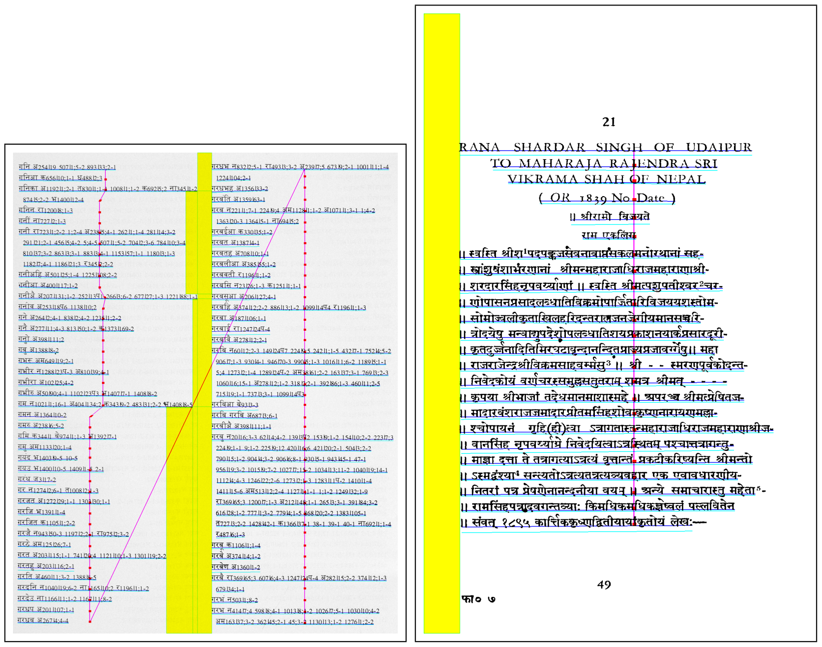
The primary layout method used in OCRopus performs layout analysis in several steps:

– First, it computes a whitespace cover of the background, by computing all locally maximal rectangles.
– From this whitespace cover, it selects column separators.
– Then it performs robust least square matching of text lines against the connected components on the page, subject to the constraint that text lines cannot cross column separators.
– Finally, it performs reading order determination by computing a "text-line-after" relationship between pairs of text lines and extending that partial order to a total order via topological sorting.

Generally speaking, OCRopus's layout analysis methods seem to work quite well on both types of input (see Figure 3). Text column styling is similar between the two languages and there is no need to modify the white space cover algorithm.

The fact that Devanagari largely lacks intra-word whitespace actually reduces the probability of column misdetection.

The constrained text line finder in OCRopus uses a "western style" text line model based on baselines and descenders; however, this model works well for Devanagari, since Devanagari characters effectively have size variations corresponding to descenders and ascenders. However, the statistics for those variations are somewhat different from those of Latin script, with a larger "x-height" relative to the size of the descenders and ascenders.



21

RANA SHARDAR SINGH OF UDAIPUR
TO MAHARAJA RAJENDRA SRI
VIKRAMA SHAH OF NEPAL

(OR 1839 No. Date)

॥ श्रीरामो विजयते

राम एकलिंग

॥ स्वस्ति श्रीश¹पदपङ्कजसेवनावाप्तसकलमनोरथानां सह-
॥ स्रांशुवंशाभरणानां श्रीमन्महाराजाधिराजमहाराणाश्री-
॥ शरदारसिंहनृपवर्य्याणां ॥ स्वस्ति श्रीमत्पशुपतीश्वर²चर-
॥ णोपासनप्रसादलब्धातिविक्रमोपार्जितारिविजययशस्तोम-
॥ सोमोज्वलीकृताखिलहरिदन्तरालजनजेगीयमानसच्चरि-
॥ त्रोदयेषु मन्वाद्युपदेशोपलब्धातिशयप्रकाशनयार्कप्रसारदूरी-
॥ कृतदुर्जनादितिमिरघटावृन्दानन्दितप्राज्यप्रजावर्गेषु॥ महा
॥ राजराजेन्द्रश्रीविक्रमसाहवर्म्मसु³ ॥ श्री - - स्मरणपूर्वकोदन्त-
॥ निवेदकोयं वर्णचरस्समुल्लसतुतराम् शमत्र श्रीमत् - - - -
॥ कृपया श्रीभाजां तदेधमानमाशास्महे ॥ अपरञ्च श्रीमत्प्रेषितज-
॥ मादारवंशराजजमादारप्रीतमसिंहशोवाकृष्णनारायणमल्ला-
॥ श्चोपायनं गृहि(ही)त्वा ऽत्रागतास्तन्महाराजाधिराजमहाराणाश्रीज-
॥ वानसिंह नृपवर्य्याग्रे निवेदयित्वाऽत्रस्थितम् पश्चात्तत्रागन्तु-
॥ माज्ञा दत्ता ते तत्रागत्याऽत्रत्यं वृत्तान्तं प्रकटीकरिष्यन्ति श्रीमन्तो
॥ ऽस्मद्वंश्या⁴ सन्त्यतोऽत्रत्यतत्रत्यव्यवहार एक एवावधारणीय-
॥ नितरां पत्र प्रेषणेनानन्दनीया वयम् ॥ अन्ये समाचारास्तु महेता⁵-
॥ रामसिंहपत्रादवगन्तव्याः किमधिकमधिकज्ञेष्वलं पल्लवितेन
॥ संवत् १८९५ कार्त्तिककृष्णद्वितीयायांकृतोयं लेखः——

49

फा० ७

Fig. 3. The result of applying the standard OCRopus page layout analysis tool to documents consisting of mixed Devanagari and Latin script. Standard OCRopus line and column finders work well on Devanagari text, as well as mixed Devanagari/Latin text.

Layout analysis also appears to work well on mixed Latin/Devanagari text using the current model (see Figure 3). However, in the medium term, it will be desirable to modify OCRopus text line finding to perform a pre-classification into Devanagari and Roman characters and to create separate text line models for the two types of text. The prominent horizontal line running through most Devanagari characters may provide a convenient, if ad hoc solution to this classification problem until a more general purpose, trainable pre-classification mechanism is implemented in OCRopus.

5 Line Recognizers

Although "OCR for classical Sanskrit" might primarily be understood as recognition of Sanskrit texts written in Devanagari, the task is actually considerably broader. Some historical texts may use different writing systems, since Devanagari is not the only script in historical use for Sanskrit. Scholarly writing on Sanskrit almost always uses Latin script, and Latin script is also used for writing Sanskrit itself, including extended passages. Sanskrit written in Devanagari and Latin scripts also makes use of numerous diacritics that need to be recognized. In addition, IPA may be used for pronunciation, Greek letters may be used for classical Greek quotations, and Greek letters and other special characters may be used for indicating footnotes or other references. Text recognition for literature involving Sanskrit therefore needs to be able to cope with this wide variety of symbols, as well as mixing them within the same text and even the same line or sentence.

As indicated above, OCRopus abstracts all actual character recognition in the form of text line recognizers; that is, text line recognizers for different scripts are responsible for segmenting the text line into characters, then recognizing the characters, and then returning a data structure that represents all possible segmentation- and recognition alternatives for the input line. This approach allows the same interface to be used between the rest of the system and the character recognition engine despite the fact that different scripts require different approaches to word and character segmentation.

5.1 Built-In Line Recognizers

OCRopus already has built-in line recognizers for Latin scripts. These are important in their own right in OCR systems for recognizing scholarly texts on Sanskrit, since, in order to recognize Latin script texts, for scholarly texts in languages like English, German, or French, and for recognizing Romanized Devanagari, an OCR system needs a high performance, high accuracy text line recognizer. By the beta release, OCRopus will have four such recognizers: (1) the Tesseract recognizer, a shape matching recognizer based on character outlines (Smith, 2007), (2) an MLP-based recognizers that performs oversegmentation followed by character classification with a MLP, (3) a shape-based recognizer using a hierarchical database organization and robust shape matching under transformations, and (4) an HMM recognizer that integrates segmentation and recognition. By the 1.0 release, OCRopus will likely also incorporate a text line recognizer based on word shape.

Unlike many other approaches to OCR, these recognizers make few assumptions about the character set or fonts they are recognizing; instead, they are trained on text line input, perform alignment against ground truth data, and then train individual character shape models automatically. The feature set used by these recognizers are generic and applicable to many scripts. Language modeling is handled separately (see below). Recognition accuracy of these recognizers approaches that of commercial OCR systems.

Diacritics are currently handled by treating character+diacritic combinations as novel characters--an approach that works well for the limited character+diacritic combinations found in modern European languages. Scholarly literature has a much large number of possible combinations, for example when diacritics are used to indicate stress. In addition, some diacritics in scholarly literature span multiple characters. Increased use of OCRopus for scholarly literature may therefore require the development of diacritic recognition and removal prior to character recognition, with an integration of character recognition results and diacritics at a later stage.

5.2 External Line Recognizers

As already mentioned above, OCRopus makes it easy to incorporate external OCR and character recognition engines into the system. This will permit the use of Devanagari OCR engines that have already been developed within OCRopus for text line recognition, while still taking advantage of the layout analysis and statistical language processing facilities of OCRopus.

5.3 Devanagari

It will be desirable to take advantage of the built-in text line recognition methods in OCRopus for Devanagari recognition. Although the visual appearance of Devanagari is very different, its basic approach to writing is not all that different from Latin scripts (in fact, it has been hypothesized that the two writing systems share a common origin): Devanagari consists of a linear sequence of graphemes. Each grapheme represents a consonant or vowel, although consonants carry an implicit vowel, meaning that it is usually considered a syllabic script. Phenomena like diacritics, word spacing, and kerning are common to both writing systems.

There are some significant differences to Latin script:

- In printed Latin script, graphemes do not touch each other in clean renditions of the script; in Devanagari, touching characters are the norm.
- The text line model is different, with character “hanging from” a horizontal line running through most of the Devanagari text.
- In Latin script, grapheme order corresponds mostly to phoneme order, while in Devanagari, order is frequently violated. For example, graphemes representing vowels frequently precede graphemes representing consonants, even though the vowel phoneme follows the consonant. Unicode encoding follows phoneme order in many cases, necessitating a transformation of grapheme to Unicode order.
- Devanagari makes extensive use of ligatures for consonant clusters; this means that although the basic character set is about the same size as Latin character sets, there is a much larger set of ligatures that needs to be recognized.

– Devanagari makes extensive use of diacritics; as in Latin script, these can either be recognized as part of the character itself, resulting in a large character set, or they can be recognized as separate units and later be combined with the base character set.

Many of these issues also arise with other writing systems and therefore need to be addressed within a multi-script, multi-lingual system. For example, Arabic and Urdu both are mostly-touching writing systems and have different text line models from Latin script. And Urdu and historic Latin script both make extensive use of ligatures and diacritics. CJK languages require recognizers and classifiers that can cope with character sets containing tens of thousands of characters.

Our general approach to direct recognition of Devanagari within OCRopus is to adapt the existing recognizers. The two recognizers most easily adapted are the HMM recognizer and the MLP recognizer.

In order to enable training, the first step that will be necessary will be to decide on a set of graphemes for representing Devanagari text lines. Some of the graphemes will be treated as separate character (e.g., "i"), while others may be treated as diacritics (e.g., "ii"). Then, we need to construct an invertible Unicode-to-grapheme mapping; the mapping can be represented and computed out using finite state transducers, the same technology used for language modeling within the OCRopus system.

For the HMM recognizer, models are trained using standard methods: the forward-backward algorithm, the construction of HMM models for each character, and forced alignment with ground truth. Alignment has to be carried out against the grapheme sequence, not the original Unicode character sequence.

An oversegmenting MLP-based recognizer can likewise be trained directly against the grapheme sequence (where it is important that the graphemes chosen correspond to the "characters" identified by the segmenter).

In Figure 4, we see that the primary segmenter used within OCRopus for Latin scripts, a segmenter based on computing near-vertical cuts using a dynamic programming algorithm (Breuel, 2001), also yields reasonable grapheme sequences for Devanagari; the output of this segmenter can then be used as input for MLP-based grapheme recognition, and the resulting hypothesis graph can then be transformed into a unicode character sequences using the grapheme-to-Unicode mapping established previously.

A second segmenter within OCRopus is based on skeletal features and leads to a much higher degree of oversegmentation; it could potentially be used with other sets of graphemes.

6 Language Modeling

As already mentioned above, OCRopus relies for both the representation of segmentation and recognition alternatives, as well as for the representation of statistical language models, on *weighted finite state transducers* [?] (WFSTs).

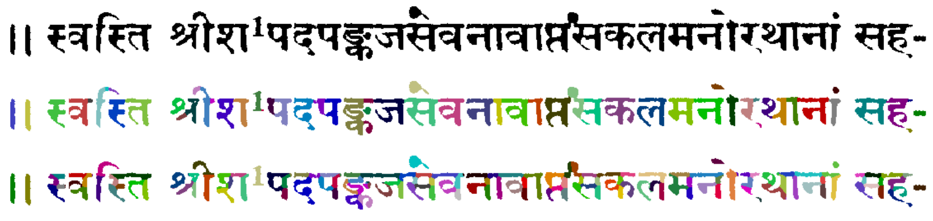


Fig. 4. Segmentations of an input string (top) into character parts. Character parts are indicated by color. Character parts are grouped together into character hypotheses, which are then recognized by the character shape recognizer. Character parts that span multiple actual characters frequently lead to recognition errors (undersegmentation). On the other hand, splitting characters into many small parts results in longer recognition times (oversegmentation). Two segmentations are shown, obtained using standard OCRopus segmenters: the first segmentation is based on the curved cut segmenter (a dynamic programming method), the second segmentation is based on the skeletal segmenter. Here, segmenters were used with their default parameter settings (performance can likely be improved further by adapting parameter settings to Devanagari)
.

WFSTs allow language models and recognition alternatives to be manipulated algebraically: they can be concatenated, unioned, intersected, composed, minimized, reversed, complemented, and transformed in a variety of other ways.

WFSTs can be thought of as directed graphs whose edges are associated with symbols and weights. The symbols are often Unicode characters, but can also represent graphemes, phonemes, ligatures, fonts and styles, and even entire strings.

WFSTs are a mature technology used, for example, in speech recognition (Mohri et al., 2000), information extraction (e.g., Kramer et al., 2007) and statistical machine translation (e.g., Kumar and Byrne, 2003). WFSTs can be constructed manually or learned from training data.

WFSTs can be used to implement the kinds of language processing problems traditionally encountered in OCR systems:

– Dictionary-based language models are very common in OCR systems; dictionaries can be represented efficiently as WFSTs by first constructing a WFST whose paths correspond to individual words and whose weights correspond to logarithmic word frequencies, then minimizing that WFST using WFST minimization algorithms, and finally computing the closure of that WFST. The resulting WFST is similar to a trie data structure with additional compression of word suffixes.
– WFSTs can be used for training by substituting the recognition language model in the OCR system with a special language model that accepts only the actual transcription of the input text; the result of recognition with this special language model is an input that has been forcibly aligned to the

ground truth. This alignment is then used to extract images for training HMM or MLP classifiers.

Beyond these traditional uses, there are a wide variety of other potential applications. For example, WFST-based language models allow us to solve the following problems:

- For source documents written in multiple languages (e.g., English and Sanskrit), we can take existing language models for English and Sanskrit and combine them. As part of the combination, we can train or specify the probable locations and frequencies of transitions between the two language models, corresponding to, for example, isolated foreign words within one language, or long quotations.
- We can represent and learn (from sample data) the correspondence and statistics of modern dictionary forms and classical spelling variants of words using WFSTs. The resulting WFST can be used in both directions: it can be used to map historical documents onto modern orthography, it can be used for aligning and comparing modern and historical texts, and it can be applied for using language models developed for one orthography in the OCR recognition of documents written using the other orthography.
- A WFSTs model can be used for transliterating Devanagari into Latin script, and the same model can then also potentially be used for the reverse direction.
- Often, ground truth data for OCR training is not perfectly aligned with the actual image data due to transcription errors, different source editions, etc. Instead of using the exact transcription of the ground truth text, we can transform it into a WFST that can be used instead of ground truth and is tolerant to misalignment, transcription errors, insertions, and deletions.
- By composing such language models, we can, for example, use a modern transliterated version of a text using modern orthography as the ground truth data for training OCRopus on a historical text printed in Devanagari.
- Some source documents, like dictionaries and catalogs, are described by small grammars. For example, a dictionary entry may have a head word in Sanskrit notated in Devanagari, followed by IPA notation in brackets, followed by a word class in italic Latin script, followed by a translation in Latin script alternating with usages examples in Sanskrit, concluding with a list of numerical references. This sequence of script, style, and language changes can be represented as a WFST and the WFST can drive not only the recognition of each part of the entry, but also mark the logical function of each substring.
- Statistical machine translation systems tend to be quite sensitive to OCR errors. However, most OCR errors are actually not strictly speaking errors, but instead places where the OCR system itself recognized an ambiguity and was forced to make a decision. OCRopus represents segmentation and recognition alternatives as WFSTs. By using the WFSTs generated by OCRopus directly as input to a statistical machine translation system (e.g., Kumar and Byrne, 2003), the error rate of the overall system can potentially be

greatly reduced. Similar considerations apply to other statistical language processing steps based on OCR output, such as named entity recognition, parts of speech tagging, and parsing.

7 Other Applications

The renewed interest in OCR systems was, in part, motivated by the widespread scanning activities by organizations like Google and the Internet Archive. These efforts tend to be large scale operations with expensive imaging and post-processing equipment.

Separately from the OCRopus effort, our research group is also developing portable, low-cost technology for document image capture applications, intended to make the ability to capture historical and scholarly literature universally available, as well as to enable handheld machine translation for both students and researchers in the field.

We have also developed non-OCR-based image alignment for historical documents, permitting the automated comparison of low-quality historical documents, as well as the recovery of annotations and differences between editions.

8 Discussion

This paper has described on-going work in the development of a multi-lingual OCR system, and how the architecture of the system is supporting multiple languages, scripts, layouts, and language models.

As a community project, the quickest way of obtaining support for Sanskrit and Devanagari is through community support. The infrastructure and tools are already in place for this, and, as we have seen, image pre-processing, layout analysis, and linguistic post-processing are already in place for this and function well for Devanagari. Simple Devanagari support will likely appear this year, both based on the Tesseract recognizer and based on the OCRopus MLP recognizer.

9 References

T. Breuel: *"Language Modeling for a Real-World Handwriting Recognition Task"*, AISB Workshop on Computational Linguistics for Speech and Handwriting Recognition, Leeds University, England, 1994.

T. Breuel: *"The OCRopus Open Source OCR System"*, Proceedings IS&T/SPIE 20th Annual Symposium 2008, 2008.

T. Breuel: *"Segmentation of handwritten letter strings using a dynamic programming algorithm."* Proc. 6th Int. Conf. on Document Analysis and Recognition (DAS), 2001.

S. Kumar, W. Byrne: "*A weighted finite state transducer implementation of the alignment template model for statistical machine translation."* Proceedings of the

Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, 2003.

M Kramer, H Kaprykowsky, D Keysers, T Breuel: *"Bibliographic Meta-Data Extraction Using Probabilistic Finite State Transducers."* International Conference on Document Analysis and Recognition, 2007.

M Mohri, F Pereira, M Riley: *"Weighted Finite-State Transducers in Speech Recognition"*. Computer Speech and Language, 2002.

S. Setlur, S. Kompalli, V. Ramanaprasad, V. Govindaraju: *"Creation of data resources and design of an evaluation test bed for Devanagari script recognition"*, in Proceedings. 13th International Workshop on Research Issues in Data Engineering: Multi-lingual Information Management, 2003.

R. Smith: *"An Overview of the Tesseract OCR Engine."* Proc. 9th Int. Conf. on Document Analysis and Recognition (ICDAR), 2007.