Report on Internship Project:

Image to text converter

Optical Character Recognition in Multilingual Text

Name: Sagar Santosh Nagarale

Abstract

This is the final report on an internship project for the Master of Science in Computational Linguistics program at the University of Washington. The project is an evaluation of existing OCR technologies in the context of recognizing text from multi-script dictionaries.

I am using python for this project and for import you will need install all requirement librabies in your system

1 Introduction

The goal of this project was to evaluate existing OCR systems with respect to recognizing mixed-script data, particularly in the context of collecting image to text

This work was completed between May 2011 and May 2012. Using Abbyy FineReader version 10.0 and Tesseract 3.02, a newly scanned corpus of some three thousand pages in Thai, Tibetan, Farsi and English has been processed.

2 Activities and Results

2.1 Building a Corpus

The first challenge for this examination, and one of the most time consuming, has been procuring a test corpus. Google Books has some multi-lingual dictionaries in their collection, but even the open domain books this writer found were not available as a download in sufficiently high resolution. According to the documentation for both OCR systems evaluated the page images need to be at least 300dpi.

Seven multi-lingual dictionaries were acquired for this project employing Farsi, Thai, Tibetan and Latin scripts. These range from a high-gloss pocket dictionary to an English onion-paper primer on Tibetan. The varying paper weights allowed me to evaluate the feasibility of using each in an automatic document feeder. The volumes were between

There are 4,627 scanned pages in this collection, with more than 19 fonts, six of which this writer had time to assemble sufficient training data for, which for this project means at least fifteen examples. More common characters have 2025 examples per font in the training data collection.

The pages were scanned using the document feeder of a desktop scanner. Dictionaries often have very thin pages, which document feeders can damage if the settings aren't calibrated properly, so it required some experimentation to find the best speed for each paper weight. Further, since the pages are typically taller than they are wide, feeding them in sideways generally resulted in less jamming. The pages later had to be digitally rotated into the correct orientation.

It took roughly 43 hours to scan in the full collection at 600 dpi. Scanning speed aside, I suspect that a higher quality document feeder could have sped the process. The one employed takes 15 pages at a time, and the thinner dictionary pages often would get pulled in together and then need to be re-scanned separately.

Furthermore, each of the volumes had to be trimmed (removing the spine) in order to use the document feeder. Nondestructive methods of books scanning would require the use of professional book scanning hardware. Several professional firms will perform this service. Quotes received for this project (seven books, 4,627 pages) ranged between $250 at Bound Book Scanning[23] and $3,000

at Crowley, Inc.[3]. Kirtas' Kabis I (base model) for non-destructive book scanning can scan 1600 pages an hour, but costs in the area of $70,000 [16]. In retrospect, it may have been better to hire Bound Book Scanning and have more time for coding and analysis.

2.2 Postprocssing

To prepare the scans, I wrote a batch script to straighten out the images. The script both rotates the pages that were scanned sideways and deskews the images. I later experimented with a variety of despeckling and contrast enhancing methods, but did not find that any of them improved recognition on this corpus.

The deskewing package I used was from a system called pagetools, released under GPL v2. [14]. The deskew package uses a very fast Radon transform algorithm; deskewing the entire collection took less than an hour. I also write a GIMP script to do the same with the hope of using other GIMP plugins to improve the image quality, but this proved unnecessary. Both this script and the plugin are available from the project Sourceforge code repository[4]. [1]

3 Tesseract

3.1 About

Tesseract was originally developed by Hewlett-Packard and UNLV in the 1990s. In 2005 it was released as open source, and in 2006 became sponsored by Google[26]. Today, as of version 3 (released in October of 2011), Tesseract can support multiplelanguage documents and already has language packs of varying quality for more than 30 languages, even including some resource-poor languages such as Cherokee and classic Quechua. Only six of these are officially supported, but the open source community contributes language packs. As of the time of writing, 3.02 is not available as a binary but must be compiled from the source accessible from the code repository.

Tesseract now has some rudimentary page layout analysis capabilities, which do extend to recognizing when text is in multiple columns on the page. Accuracy since 1995 has only increased around 1%, and can be expected on English text to approach 97% character-level recognition accuracy. The system was unique in its day for implementing component analysis before attempting segmentation, which enabled it to detect sections with inversecolor text. [25]

3.2 Training

Training Tesseract is a fair amount of work. For a detailed guide, see TrainingTesseract3[29]. I wrote a shell script to assist the user through each of the steps, as it is easy to forgot one or mistype. train tesseract.sh **is available from the project**

[1] Technical note for interested parties: on Ubuntu 12.04 the makefile failed due to a missing header, but installing the libnetpbm10-dev package solved that problem.

Sourceforge code repository[4]. The details for each font and character set must be specified, but most of the work is in generating "boxfiles," or text files that specify the coordinates of the boundaries of each character on an associated page. The documentation for Tesseract[29] specifies that characters should have between 10 and 20 training examples. The examples are processed into feature vectors used by the classifier. As described in Smith 2009[24]:

> In training, a 4-dimensional feature vector of (x, y-position, direction, length) is derived from each element of the polygonal approximation, and clustered to form prototypical feature vectors. (Hence the name: Tesseract.)

The instructions for generating training data stress that it is vital to have characters spaced out on the training page.

Overlapping boxes result in features being created or dropped inappropriately in the training stages. The suggested method is to print out the desired characters on paper, so that you have an appropriate number of characters and so that they are spaced easily. That is not possible with this dictionary project, as many of the fonts used are either proprietary or reproductions of handwritten characters.

In order to generate valid boxfile/image sets, I created pages with the desired characters essentially copy/pasted out of the corpus, cleaned up so that features from overlapping characters are removed. This made it possible to generate clean boxes, but means any baseline data (in the sense of physical spacing) is lost. The results were not impressive compared to the default language pack which was not trained on the given fonts, and so I suspect that either the importance of the baseline data or the sheer number of training examples far outweighs the benefit of having fontspecific training data. Additionally, creating training data this way was very timeconsuming.

Creating boxfiles took over 60 hours for the seven fonts I prepared, and several of the fonts had characters not presented frequently enough to provide the recommended 15 examples. As a result I do not recommend attempting to collect fontspecific training data for use in Tesseract unless the digital font is available. In addition to being incredibly tedious, it did not result in improved accuracy for any of the tested fonts. Bootstrapping a new character set would probably be better done by finding similar fonts and printing suitable training pages. (See: Future Work.)

There are multiple boxfile editing GUIs available; I tried several of them. I found the moshpytt[18] interface to be the best of the installable ones, though it suffers from not allowing the user to drag box boundaries, instead you have to use the directional arrow keys to move or resize them pixel by pixel. This slows down work considerably. The AJAX based web interface Tesseract OCR Chopper by Dino Beslagic[27] does not suffer from that problem. It allows resizing boxes with the

mouse. Unfortunately it doesn't always process the entire page, and will sometimes skip over entire columns or paragraphs. It would be useful to this community to have a multi-platform boxfile editor with draggable box boundaries, in my opinion.

As a further note on training, when I began this project multi-language support was not in place yet and so I attempted to create a custom language that essentially just combined the resources for English and Thai. This was a complete failure. The resulting text was mostly Latin characters with a smattering of Thai diacritics superimposed, in place of the Thai script. I suspect that this is because the English resources outnumbered Thai resources by a fair margin (the system does not use grammar rules). In *Adapting the Tesseract open source OCR engine for multilingual OCR* **[24] Ryan Smith mentions Thai specifically as being particularly ambiguous and so introducing multilingual recognition difficulties.**

One can optionally include a wordlist when training Tesseract. The wordlist, if present, is one of the few linguistically informed features and will be used during word segmentation.[25]

Tesseract does not output confidence data by default, and although it can be modified to do so the numbers it uses do not appear to be very useful.

3.3 Results

Overall, I was not very impressed with Tesseract's performance. The accuracies observed in this corpus do not resemble those reported by the project. Figure 1 shows a sample of these results for Thai.

Figure 1: Tesseract Results Using Official Thai Language Pack

As you can see, character recognition is decent so long as Tesseract recognizes that language of the word. However, it frequently fails to recognize that the word is in Thai and not English; here this error rate is about 80%. It appears that Tesseract 3.02's new multi-language recognition feature is not ready for practical use just yet. Perhaps the disparity in confidence between the Latin and Thai characters could be corrected so that the system prefers Thai at a lower confidence threshold. (See Future Work.)

About

Abbyy is headquartered in Moscow, Russia. They produce a range of language products, such as digital dictionaries, translation software and services, and data capture/document processing software. One of their products, Abbyy FineReader, is a particularly powerful document processing

engine that includes their SDK. Through their representative Mark Smock I obtained a 60-day trial license for the FineReader SDK.

3.4 Training

No training was required for Thai or Farsi. Abbyy provides a GUI-based training tool to generate data files for new character sets. Unfortunately, I did not have enough time to evaluate this functionality, and have been so far unable to launch the interface due to runtime errors.

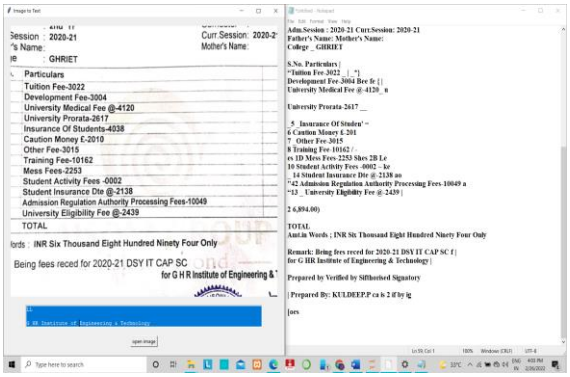
The SDK was easy to set up; it comes with a GUI installer. While FineReader 9.0 is available for Linux, Windows and Mac, the 10.0 version's Linux distribution is not yet available. As such, I used the 10.0 Windows version. The SDK provides document analysis features, grammar features, and character recognition data.

Starting with one of the SDK examples, I wrote a fairly simple Java program to analyze the page, using the languages specified through the command line, and output data in raw text and XML format. This program and accompanying bash script are available through the project Sourceforge page. However, for licensing reasons, I cannot provide the jarfile for the engine itself, which is required to run the program. The program records the uncertainty and unrecognizable characters as reported by FineReader to generate a results file. The results from sample sets can be seen in Table II.

Following in the footsteps of Doermann et all[6], I randomly selected two pages from two of the books and manually prepared ground truth data for them. I then compared them to the output of the FineReader-based program. As I am not familiar with any good tools to do text alignment, I used Microsoft Word's "Compare Document" feature to see differences in the text. Table III has the results, if all whitespace characters are considered equal (tab vs space).

Thai fared considerably better than Farsi, the problem with both Farsi texts is that the transliteration column is written in a partially cursive English script that FineReader performed poorly on.

3.5 Results



4 Future Work

As there is no Tibetan language pack for Tesseract yet, and I've gone to the trouble of learning to train Tesseract, I plan to work on that over the summer and contribute it back to the Tesseract project. Given my experience with the Thai language pack, I am going to try using existing Tibetan fonts to generate training pages in the suggested fashion (printing spaced-out characters and scanning the printed pages) rather than continuing with the attempt to collect all the examples from the existing pages. It will be interesting to see how the two sets of samples compare in terms of recognition accuracy, and what the real threshold is for sufficient character examples in this script.

It would be interesting to see if the font-specific data, when added to existing language packs, can improve Tesseract's recognition accuracy. Other work in Tesseract worth examining: finding a way to boost recognition of non-Latin characters on a page with Latin text. I suspect that the ability to give a weighted preference to one language or another could be useful in texts where one script has more ambiguous characters. IdeallyRelevant Coursework

The content of this internship, while relevant to my personal interests, has been a nearly complete departure from the coursework I had in the CLMA program. This surprises me in retrospect. One of my primary original motivations for joining the CLMA program was to better enable myself to take on a project I first attempted as an undergraduate: a machine transliteration system for cuneiform tablets. This internship might be the only directly relevant work in the program.

Advanced Statistical NLP was useful; it helped me when reading the literature and attempting to understand the algorithms used by these software packages. Particularly, the assignment on Support Vector Machines, which are discussed in *Adaptive Hindi OCR Using Generalized Hausdorff Image Comparison***[17].**

Another honourable mention goes to Deep Language Processing class, which strengthened my understanding of Hidden Markov Models, as made use of in *Use of OCR for Rapid Construction of Bilingual Lexicons***[6].**

5 Literature Review

While I will not take the time to review all of the documents that were interesting or informative in the course of this investigation, here are a few that stood out. My thanks to Jonathan Pool for several additional papers of interest: [8], [6], and [22].

5.1 Stochastic Language Models for Style-Directed Layout Analysis of Document Images

Kanungo and Mao 2003[15] experiment with a stochastic grammar describing the physical

layout of a page (headers, columns etc). Using the Viterbi algorithm, they determine the optimal state sequence for weighted automata constructed from trees representing black pixels in strips drawn on the page. The state sequence provides 1-D segmentation, hierarchical from the page down to the text lines.

They tested this algorithm on artificially noisy test images at sampling resolutions of 200-400 DPI. One version of the algorithm, Model-1, does not use explicit state duration densities, while Model-II does. They found that Model-II performed better than Model-I, especially as image noise increased.

Essentially: a projection of pixel values on the page is partitioned into strips, the darkness of the strip becomes an observation symbol in a FSA, and the optimal state transitions (representing physical boundaries) are determined a la Viterbi.

5.2 Adaptive Hindi OCR using Generalized Hausdorff Image Comparison

Ma and Doermann 2003[17] claim to have a "rapidly retargetable" system with 8895% character level accuracy. As part of a DARPA TIDES project at the University of Maryland to acquire bilingual dictionaries, Ma and Doermann took one month to develop and train the system described.

The system scans Devangari text at 300400 DPI; the scans are then despeckled and deskewed. The system performs segmentation using methods described in O'Gormain 1993[21]. Word level script detection identifies Devengar versus Roman words.

The Roman words are fed to "a commercial English OCR" while the Hindi words are further segmented into characters, which are passed to the character classifier.

The Devangari segmenter segments characters by removing the top and bottom strips found in that script and detecting the characters and modifiers before reinserting the strip. There is some work to segment the "shadow characters", characters that do not touch other characters but cannot be separated by a vertical line.

Each character is classified using Generalized Hausdorff Image Comparison (GHIC), and algorithm which computes the Hausdorff distance, measuring the similarity between two images (assuming there is only one translation between them). Without belabouring the details of GHIC, suffice it to say that this algorithm provides a useful confidence measure.

The system was applied to the Oxford Hindi-English dictionary, a corpus of 1083 pages scanned at 400 dpi as well as the original PDFs. Accuracy was evaluated by randomly

selecting seven pages from the corpus and preparing ground truth data.

With printed-scanned images, the character-level accuracy was 87.75%, while the images taken from a pdf yielded 95% accuracy. The authors state that the classifier may be trained on small amounts of data, but do not provide specifics.

5.3 Applying the OCRopus OCR System to Scholarly Sanskrit Literature

Thomas Breuel (2009)[2] of the University of Kaiserslautern gives a high-level description of the OCRopus system (which he has led the development of). His description centres on the case of mixedscript scholarly Sanskrit literature using Latin and Devangari scripts.

Breuel discusses the system itself, briefly. It is a strictly feed-forward system (why he stresses this in the paper is a bit of a puzzlement to me as I have not heard of any OCR system with backtracking between modules) which supports multilingual and multi-script OCR. He provides a gloss of each of the modules:

1. Preprocessing - despeckling, deskewing.
2. Layout analysis - computational geometry algorithms with least square matching, Breuel claims that Voronoi methods don't perform as well.
3. Text line recognition - OCRopus employsfour recognizers here, including Tesseract. Previous to the current version of

6 Conclusions

In my opinion, while Tesseract and OCRopus are certainly promising projects, Abbyy is the most practical solution for obtaining clean data. More investigation would be required for me to comment on the practicality of training Abbyy.

References

[1] Benjawan Poomsan Becker and Chris Pirazzi. *New Thai-English, English-Thai Compact Dictionary for English Speakers with Tones and Classifiers*. Paiboon Publishing, 2009, p. 982. isbn: 1887521321. url: http://www.amazon.com/ThaiEnglish-English-Thai-DictionarySpeakers-Classifiers/dp/1887521321.

[2] Thomas Breuel. "Applying the OCRopus OCR System to Scholarly Sanskrit Literature". In: *Sanskrit Computational Linguistics* (2009), pp. 391–402. doi: 10.1007/ 978-3-642-00155-0_21. url: http: //portal.acm.org/citation.cfm?id= 1530303.1530324.

[3] The Crowley Company. *Thecrowleycompany.com*. Mar. 2012. url: http : //www.thecrowleycompany.com/.

[4] *DictionaryReader*. May 2012. url: https: / / sourceforge . net / projects / dictreader/.

[5] *DictionaryReader SourceForge Project*. May 2012. url: https://sourceforge. net/p/dictreader/.

[6] D Doermann and B Karagol-Ayan. "Use of OCR for Rapid Construction of Bilingual Lexicons". In: (2003). url: http: //onlinelibrary.wiley.com/doi/10. 1002/cbdv.200490137/abstracthttp: //www.stormingmedia.us/28/2865/ A286554.html.

[7] Utilika Foundation. *Utlika.org*. Mar. 2011. url: http://utilika.org/info/ intern2011.html.

[8] Utpal Garain and DS Doermann. "Maryland at FIRE 2011: Retrieval of OCR'd Bengali". In: *lampsrv02.umiacs.umd.edu* (2011). url: http://lampsrv02.umiacs. umd.edu/pubs/Papers/utpalgarain11b/utpalgarain-11b.pdf.

[9] Dariush Gilani. *English Persian Dictionary*. San Jose: Maple Press Inc, 1983. isbn: 0936347953.

[10] Sulayman Hayyim. *English-Persian dictionary*. 5th ed. New York: Hippocrene Books, 2006. isbn: 0781800560. url: http://books.google.com/books?hl= en\&lr=\&id=OxBMU6P-4S8C\&pgis=1.