Recent Progress on the OCRopus OCR System

Thomas Breuel
U. Kaiserslautern and DFKI

ABSTRACT

The OCRopus system is an open source OCR system developed for book capture and digital library applications. It is designed to be a multilingual system in which all components are easily pluggable and replaceable. In this paper, I describe recent progress, on-going work, and preliminary results in the development of the OCRopus system, including the new component model, a new line recognizer, a new set of decoders, and language modeling tools.

1. INTRODUCTION

Due to decreasing costs of scanning, image processing, storage, and transmission, there has been renewed interest in creating high performance OCR systems for digital library applications. Google's book search application aims to scan the world's books and make them available on-line in searchable and browseable form. The Internet Archive similarly digitizes and publishes on the Internet large numbers of books and other content.

Converting these documents into electronic format remains a challenge. In addition to converting character shapes into text, OCR involves many other different steps and tasks, including text/image segmentation, text line detection, layout analysis, and linguistic post-processing.

Commercial OCR engines and mature open source OCR systems like Tesseract achieve reasonably low error rates for the text recognition portion on clean input, and can be combined with each other in order to achieve lower error rates through classifier combination techniques.

Overall, our focus in the development of OCRopus over the last year has been to improve the software architecture and support experimentation with different components and parameters better through the introduction of a new component model, to replace script- and language specific components (e.g., for orientation detection) with trainable components, to implement a new line recognizer that can be adapted to different languages, to implement beam search for integrating the output of the character recognizer with the language model, and to implement simple language models for specific languages.

In this paper, I will summarize these new developments, on-going efforts, and preliminary performance results. For other components and aspects of the OCRopus systems, such as layout analysis, please see [4]. Software corresponding to this paper can be downloaded from the new OCRopus repository at `mercurial.iupr.org`; the paper describes the software as of April, 2009.

2. COMPONENT MODEL

Different languages and document types have different requirements in terms of preprocessing, cleanup, layout analysis, OCR, and language modeling. These requirements are addressed by a growing set of different components.

We had originally hoped to be able to handle this through a top-level scripting interface, but found that the ability to instantiate, load, and save components was needed in many different places within the OCRopus system. Existing, portable component models (such as Mozilla's) were both heavy-weight and did not address the needs of OCRopus very well.

We therefore introduced a simple component model for OCRopus. The model is implemented by deriving from the `IComponent` base class. This provides both a standard set of interfaces, as well as some common component functionality. Most importantly, the component model allows components to be saved to a stream, loaded from a stream, and instantiated by name.

In addition, many document analysis algorithms are dependent on parameters. Adapting and tuning those algorithms involves extensive experimentation with different parameter values. The new component model allows parameters to be set, both on a per-component basis and globally (via the environment). Parameter settings for a particular component are saved and loaded along with the component, and they can be inspected from the command line easily.

Finally, each component can optionally accept commands via a simple method that takes an array of string arguments and returns a string result. This command interface is useful for sending component-specific commands (e.g., for debugging) without having to define a complex class hierarchy with many rarely used methods. The `command` method provides a simple form of duck typing and dynamic object oriented programming.

3. NEW PREPROCESSING MODULES

Many document analysis methods are implicitly language-specific. For example, a common method for page orientation detection of English documents relies on the ratio of ascender to descenders. Text/image segmentation methods rely on the statistics of particular shapes as they occur in documents. In order to make OCRopus preprocessing less language dependent, we have been working on developing several general-purpose approaches to preprocessing methods. For a survey of other, related learning-based approaches to preprocessing tasks in document analysis, see [8].

Output-Driven Preprocessing.

The idea behind output-driven preprocessing is to choose preprocessing methods that result in "good" OCR output. For the purposes of this discussion, let us just consider the textual portion of the OCR output in the case of text line recognition. Consider the image x of a line of text. A text line recognizer will return either a string s corresponding to the maximum posterior probability $s = \arg\max P(s|x)$ or an estimate of the posterior probability distribution $P(s|x)$. This is then combined with a language model $P(s)$, a prior distribution of strings, in order to obtain a final output from the text line recognizer.

Now assume that we have a family of transformations T that we can perform on the input string; these might be skew correction, thresholding, or page rotation. The output from the recognizer is now $P(s|Tx)$. The idea behind output-drive preprocessing is to choose $T = \arg\max_T P(s_T)$, where $s_T = \arg\max_{s_T} P(s_T|Tx)$. In different words, we apply the text line recognizer to differently transformed versions of the input image, obtain the best estimate of the recognizer, and then check how plausible that string is according to our language model.

This method works well if errors of the text line recognizer are likely to give rise to "nonsense strings". This appears to be the case for many languages. As a simple example, consider an input image representing a fuzzy version of the string "dim". When this string is thresholded with higher and higher thresholds, it will turn into strings like "clıı", strings with essentially zero probability in the language model.

The advantage of output-driven preprocessing is that it requires no separate probabilistic model: if we have a text line recognizer and a language model (both of which are required for the OCR system anyway), we can directly optimize preprocessing steps. In order to make this approach practical and efficient, we generally apply it only to a sample of the input text lines and then use the optimal transformation identified on the sample to all the text lines of the input. The approach works less well if the text line recognizer already includes its own language modeling; in that case, even if the T is not well chosen, the text line recognizer itself will tend to correct the errors and many different T will yield the same output string. However, in such cases, text line recognizers often output a "quality of match" measure, which can then be used instead of $P(s_T|Tx)$ as an objective function.

Density Models of the Shape of Connected Components.

A second approach we have been pursuing is to build statistical models of "good" input. Using the notation from above, we build a density model $p_g(x)$. The optimal preprocessing transformation is then given by $T = \arg\max_T p_g(Tx)$.

Of course, $p_g(x)$ is difficult to model when x represents a text line or page image. We therefore usually decompose the input into a collection of connected components and then model $\log p_g(x) \approx \sum_i \log p_c(x_i)$, where the x_i represent the connected components of x. In effect, this means decomposing the page into connected components, and then for each connected component computing a score as to how well that connected component matches the prototypical image of a connected component of a character at the correct orientation.

These models $p_c(x_i)$ can be trained with minimal supervision: it is sufficient to take a collection of documents that are known to be reasonably good and mostly at the correct orientation, extract their connected components, and then compute a Gaussian mixture model to represent $p_c(x_i)$. There are some additional steps one can take to improve the performance of such models. For example, performance of the approach is improved if components are excluded from consideration for which $p_c(x_i) \approx p_c(Tx_i)$ for some or all transformations T, since such components do not contribute much information to the overall score.

Discriminative Models.

A third approach for trainable, script-independent preprocessing is a discriminative approach. In this approach, we obtain a collection of components x_i from a set of documents that have been correctly preprocessed. Then, we consider a set of inverse preprocessing transformations $R = T^{-1}$ for some T, that generate transformed variants of those components. We use these to generate a training set containing positive training samples $(x_i, 1)$ and negative training samples $(Rx_i, 0)$ (the proportion of positive to negative training samples needs to be chosen based on the prior probability of the two classes in production use). This training set is used to train a model $\hat{P}(g|x)$ that says whether the sample is "good" ($g = 1$) or "bad" ($g = 0$). In order to choose the best preprocessing method, we can then optimize $T = \arg\max_T \sum_i \log P(g|Tx_i)$.

Concretely, this means taking a training set consisting of correctly oriented pages, extracting the connected components from that page, and then training a classifier (e.g., a multilayer perceptron) on components rotate by 0 deg, 90 deg, 180 deg, and 270 deg. For the 0 deg rotation, the target output from the classifier is 1, otherwise 0. For transformations like those associated with page orientation, the usual character classifiers tend to work well, since differences between characters at different orientations (e.g., A vs. ∀) are analogous to differences that already occur in common character sets (e.g., p vs d).

4. LINE RECOGNIZER

OCRopus divides the recognition process into several (mostly) independent steps: document cleanup, layout analysis, text line recognition, and linguistic post-processing. The text line recognizer is given isolated text lines as input images, and its task is to produce either a Unicode string containing the corresponding text, or to return a recognition lattice (similar to the output of a Hidden Markov Model recognizer) representing alternative interpretations of the input

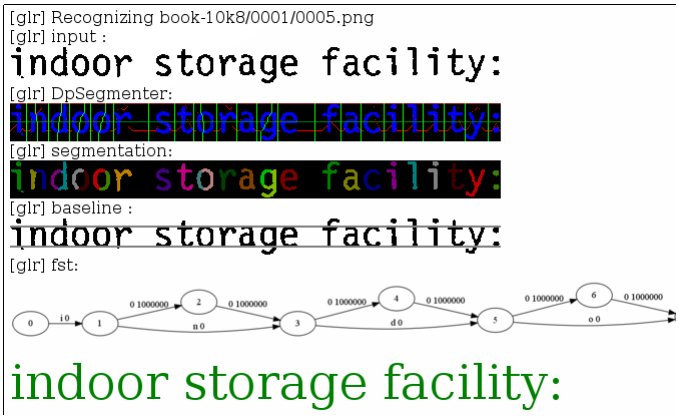

Figure 1: An illustration of the steps involved in text line recognition (this is actual debugging output obtained via the `ocropus recognize1` command). The `DpSegmenter` image shows the paths and path scores obtained by the dynamic programming text line segmenter[3]. The image below shows a color-based representation of the resulting character parts hypotheses. Below that is an illustration of the baseline and x-height estimates of the recognizer. The `fst` image below shows the output lattice from the recognizer. The final string is the output from the recognizer (without a language model).

text line.

Until recently, OCRopus did not have a built-in high performance, retrainable line recognizer. Rather, for text line recognition, it was relying on the Tesseract recognizer [13]. In addition, we had developed some prototype recognizers based on Hidden Markov Models [7].

Over the last six months, we have implemented a new, flexible text line recognizer based on oversegmentation, grouping of character segments into character hypotheses, rejection of non-character images, classification of the character hypotheses, and finally generation of a recognition lattice. All the steps of the line recognizer have been componentized using the new OCRopus component model (see above), so that it is possible to combine different segmenters, classifiers, and other components into a text line recognizer. The training parameters and other information are automatically saved along with the base classifier and can also be inspected easily. The overall operation of the text line recognizer, and its integration with language modeling is illustrated in Figure 1.

The new line recognizer is intended for source documents scanned at around 200 dpi or above, usually for alphabetic languages; for such documents, approaching text line recognition as segmentation followed character recognition works well in our experience. For lower resolution documents or other scripts, this text line recognizer may not work well, and other kinds of text line recognizers may be preferable, including segmentation-free convolutional neural networks[6, 12] and Hidden Markov Models[7] (efforts to implement such additional recognizers for OCRopus are underway).

Base Classifiers.

Perhaps the most important component of a text line recognizer is the actual character classifier itself. OCRopus provides a number of such classifiers.

Classifiers in OCRopus are abstracted as components that, during training, receive a stream of feature vectors and corresponding classes, and then are switched into a recognition mode, during which they need to assign classes and corresponding posterior probabilities to feature vectors. Classifiers can be trained incrementally (e.g., using stochastic gradient descent) or in batch mode. For batch mode training, OCRopus can automatically batch data into batches that are then handed to the training algorithm. In order to allow larger batches to be buffered, OCRopus can internally compress feature vectors using small, fixed-point floating point representations.

Since the ability for end users to retrain OCRopus for new fonts, scripts, and languages is important, the interface to the classifiers encourages training to be fully automated and self-contained. However, it is possible for developers to "plug in" classifiers that require manual interventions during training.

The primary classifiers in OCRopus right now are a nearest neighbor classifier, a highly optimized binary nearest neighbor classifier, and a standard multilayer perceptron (MLP) trained using stochastic gradient descent. We have also experimentally integrated a support vector machine classifier based on the standard libsvm (SVM; [5]), but have not been able to make this classifier scale well to the size of training sets and number of classes that occur in OCR problems.

Automatic Parameter Selection and Cross-Validation.

Considerable effort has gone into the development of the MLP classifier inside OCRopus, in particular its training methods. Traditionally, MLP classifiers have a reputation for being susceptible to local minima during training and requiring careful manual choice of the training parameters. We have developed and implemented a number of new training techniques that completely automate the choice of learning rates and number of hidden units. The result of this is that the OCRopus MLP classifier yields quite consistent performance for each training set. When applied to standard dataset, such as the MNIST data, the training algorithm used by OCRopus automatically yields the best reported error rates on those datasets, without any additional tuning or parameter selection.

The OCRopus MLP training algorithm is reminiscent of genetic algorithms, as well as cross-entropy-based optimization methods, and works as follows:

- The input dataset is split into a training set and a cross-validation set.
- The system generates an ensemble of MLPs with an initial distribution of learning rates and numbers of hidden units.
- Each member of the ensemble is trained on the training set for a small number of epochs and its performance evaluated on the cross-validation set.
- The ensemble members that performed worse than the median error rate are discarded.
- Each ensemble member that performed better than the median error rate is duplicated. A new number of hidden units is drawn from a log-normal distribution

centered around the current number of hidden units for each ensemble member and the number of hidden units of that member is adjusted (by deleting or adding hidden units). Similarly, a new learning rate is drawn from a log-normal distribution centered around the current learning rate.

- The ensemble training is repeated with the newly constructed ensemble.

In order to speed up training, each member of the ensemble is assigned to a separate core on a multi-core processor.

We have found this approach to be robust and reliable for a wide range of different kinds of inputs, data sets, and feature vectors. When applied to the MNIST data, the number of hidden units chosen by this algorithm tends to be significantly below that reported in the literature, at similar error rates.

Classifier Combination.

Although individual MLP recognizers yield reasonable performance on character recognition tasks, results can be considerably improved using classifier combination. A number of classifier combination methods have been implemented within OCRopus, including AdaBoost, rescored AdaBoost, and cascaded MLPs. The rescored AdaBoost classifier uses the standard AdaBoost algorithm to obtain a series of component classifiers, but then uses a least square fitting procedure to adjust the component classifier weights based on the training set. The cascaded MLP classifier is similar to a cascade correlation classifier, but is trained at the network level; that is, a first MLP is trained as usual, then a second MLP is trained that receives the output of the first MLP together with the original feature vector. In addition to these methods we have experimented (but not yet incorporated) different forms of bagging, local learning, and mixtures of experts.

The OCRopus component framework, as well as the fully automated training procedures, makes the training of such combined classifiers fully automatic; that is, a user can simply choose an rescored AdaBoost classifier, apply OCRopus to a training data set, and obtain a fully trained and cross-validated classifier that can be used interchangeably with a simple MLP classifier.

We have carried out a series of experiments on a diverse sample of 10000 text lines representing some 300'000 characters in many different styles and resolutions. The training and test set were split in such a way that the test set was statistically representative of the overall collection of input text lines, but contained many lines whose font and style were not represented in the input. In different words, from the point of view of the classifier, the training and test data sets were effectively drawn from different distributions (although, in fact, the data was correctly randomized but the number of styles and fonts is so large that even a large training sample is not representative of all styles). This difference between training and test sets is a fairly common occurrence in character recognition (see also book-adaptive recognition below).

Representative results for the different methods on different runs on a moderately sized training set (50'000 characters) are shown in Figure 2. In these experiments, we found that the MLP and cascaded MLP performed the most consistently. AdaBoost performed poorly with a weak component classifier, and had high variability when using a strong component classifier. Rescoring generally gave more consistent results, than simple AdaBoost, but the best non-rescored AdaBoost classifiers performed better than the rescored classifiers.

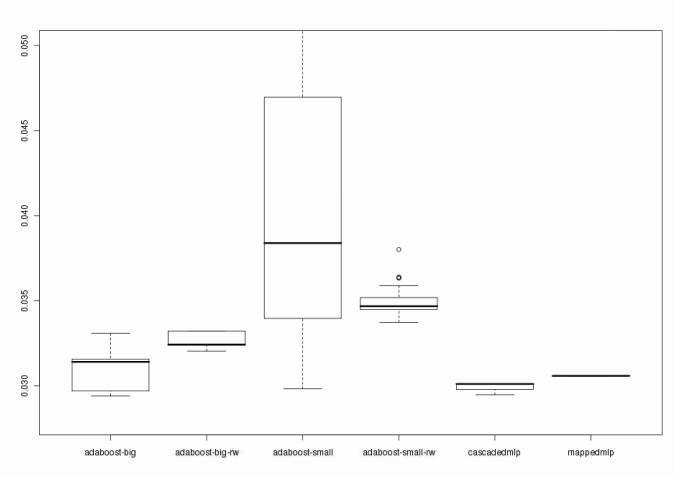

Figure 2: Performance of different classifiers and compound classifiers. The data represents 30 training runs for each classifier on 50000 training samples from our standard experimental data set, evaluated on a separate test set of 30000 samples. Statistics are shown as standard whisker-boxplots (the boxes are quartiles around the median, the whiskers indiciate the total range except for outliers). These experiments used the raw bitmap as the input feature vector. The *mappedmlp* classifier is a simple multilayer perceptron trained using the automatic training procedure described in the text; note the high reproducibility of the error rates. The *cascadedmlp* classifier is a two-stage cascading of the MLP classifier, as described in the text. The *adaboost-big* classifiers use the same multilayer perceptron as in the *mappedmlp* classifier, while the *adaboost-small* classifiers use the *mappedmlp* classifier, but with a limit of 20 on the number of hidden units. The *-rw* variants of AdaBoost use least-square reweighting, as described in the text. Overall, cascaded MLP training appears to give the best results in terms of performance and reproducibility. The performance of reweighting is inconsistent between small and large base classifiers, possibly due to overspecialization of the classifier on the training dataset. Note that in order to carry out these experiments, we used a smaller number of training samples than used for actual text line training, but experiments with larger training sets are consistent with these results.

Although these experiments are large enough to have yielded statistically meaningful results, in order to draw firm conclusions about which classifier is "best" overall, of course, we need to explore many more conditions and perform more extensive tests on large datasets. For example, the performance of AdaBoost and rescored AdaBoost may have been due to the high variability introduced by the different styles.

Nevertheless, they suggest that cascaded MLP models are, overall, preferable for current applications.

Feature Maps.

An important aspect of classifier design is designing the feature vectors that go into the classifier. Several different philosophies exist. Some approaches (e.g., many SVM-based classifiers, as shown in the MNIST results) perform no feature extraction at all and instead modify the classifier itself. Other approaches (e.g., convolutional neural networks) attempt to learn the feature extractors as part of the overall classification task. "Traditional" pattern recognition designs feature maps manually and may perform feature selection.

In OCRopus, feature extraction is abstracted in a feature map class. The feature map class is initialized with the entire text line image and then returns feature vectors for arbitrary subrectangles of the input text line. This design allows the feature map class to perform global convolutions on the input, as well as to encapsulate different approaches to rescaling and normalizing the input.

The current version of OCRopus relies on a "traditional" feature map that can be configured to extract a number of hand-crafted features for character recognition. We have performed extensive experimentation to determine which combination of features, character sizes, and normalization methods performs well in practice.

Given the large number of possible combinations, we perform these experiments by sampling from the space of possible parameter choices and then examining scatter plots of recognition errors vs. different parameter choices (Figure 3). In addition, we have semi-automated the process of determining good parameter choices by training decision trees to predict trials with low classification error given the parameter values. Using decision trees in this way makes it easy to identify both which parameter values have the biggest effect on error rates, and what parameter ranges lead to good recognition performance.

A general result from these experiments is that the incorporation of feature extractors for edges and topological features (end points, junctions, and holes) significantly improves recognition rates compared to recognition on raw bitmaps. This is perhaps not surprising, but many common approaches eschew such feature extraction steps. A second result is that recognition rates do not change much for different choices of resolution, different anti-aliasing parameters, and different approaches to character size normalization.

The current set of feature extractors only scratches the surface of what is possible. In particular, a very important class of feature extractors is those that transform the input into representations of their boundaries (e.g., plots of angle or curvature vs. pathlength along the boundary, Fourier transforms of the boundary, etc.). Such representations result in much greater data reduction of the input than the 2D feature maps used in the current line recognizer. This has the potential for reducing both recognition times and improving generalization performance.

A second set of feature extractors to be explored in the future are those that perform automated feature construction, for example based on principal component analysis (PCA), independent component analysis (ICA), and/or vector quantization (VQ).

Character/Non-Character Classification.

An important aspect of character recognition using an approach like that described above is determining whether a character hypothesis represents an actual character or a missegmentation. For example, for a character like "m", the recognizer will usually consider several segmentations corresponding approximately to hypotheses like "rn", "rıı", and "nı", and for a character like "d", the recognizer will consider a segmentation similar to "cl" as well.

Although character hypotheses like "rn" are plausible, many character hypotheses do not correspond to a valid character at all. For example, the segmenter and grouper may produce hypotheses like "Fr" or even "wm" as candidates for single character recognition.

There are several different ways of dealing with this. One way is to use a density model $p(x)$ for valid character shapes. In that case, we assign to each character hypothesis a cost of $\log p(x)$. If a character represents a valid character hypothesis, the cost will be low, while invalid segmentations yield character shapes that receive a low log likelihood according to $\log p(x)$. This kind of model is similar to the kinds of mixture models used in HMM in speech recognition and OCR.

A second approach is to add a reject class to the classifier. That is, during training, the character segmenter and grouper will generate many character hypotheses, some of which correspond to actual characters, others correspond to missegmentations. All missegmentations are assigned to a reject class, which is otherwise treated like any other class from the point of view of the classifier. This approach was used, for example, in [2]. In this case, the log posterior probability for each character hypothesis incorporates both the cost of assigning the character to the predicted class, as well as a cost measuring how probable it is that the character represents a valid character at all.

This second approach tends to work better than the first approach in our experiments, but it leads to unbalanced training sets: about half or two thirds of the characters in the training set will represent missegmented characters and be assigned to the reject class. We have therefore found a variant of this approach to lead to lower recognition errors: in this third approach, we first train a classifier to perform the accept/reject classification task (a binary classification task), and then train a separate recognizer on just the accepted characters to assign characters to classes.

Segmentation Errors and Contextual Recognition.

One problem with approaches to text line recognition based on segmentation and grouping is that segmentation errors constitute a serious problem for the classifier and often lead to misclassification (approaches like Hidden Markov Models and convolutional neural networks avoid this problem by integrating recognition and segmentation).

OCRopus uses a dynamic programming algorithm for character segmentation[3] that produces few errors. However, given the low overall error rate of the text line recognizer on many inputs, segmentation errors can be a significant percentage of all errors.

For example, a common segmentation error occurs in ligatures like "fi", in which the dot of the "i" ends up attached to letter "f", while the "i" itself ends up dotless. Similarly, the cuts through a ligature like "ff" may end up with nonstandard character shapes.

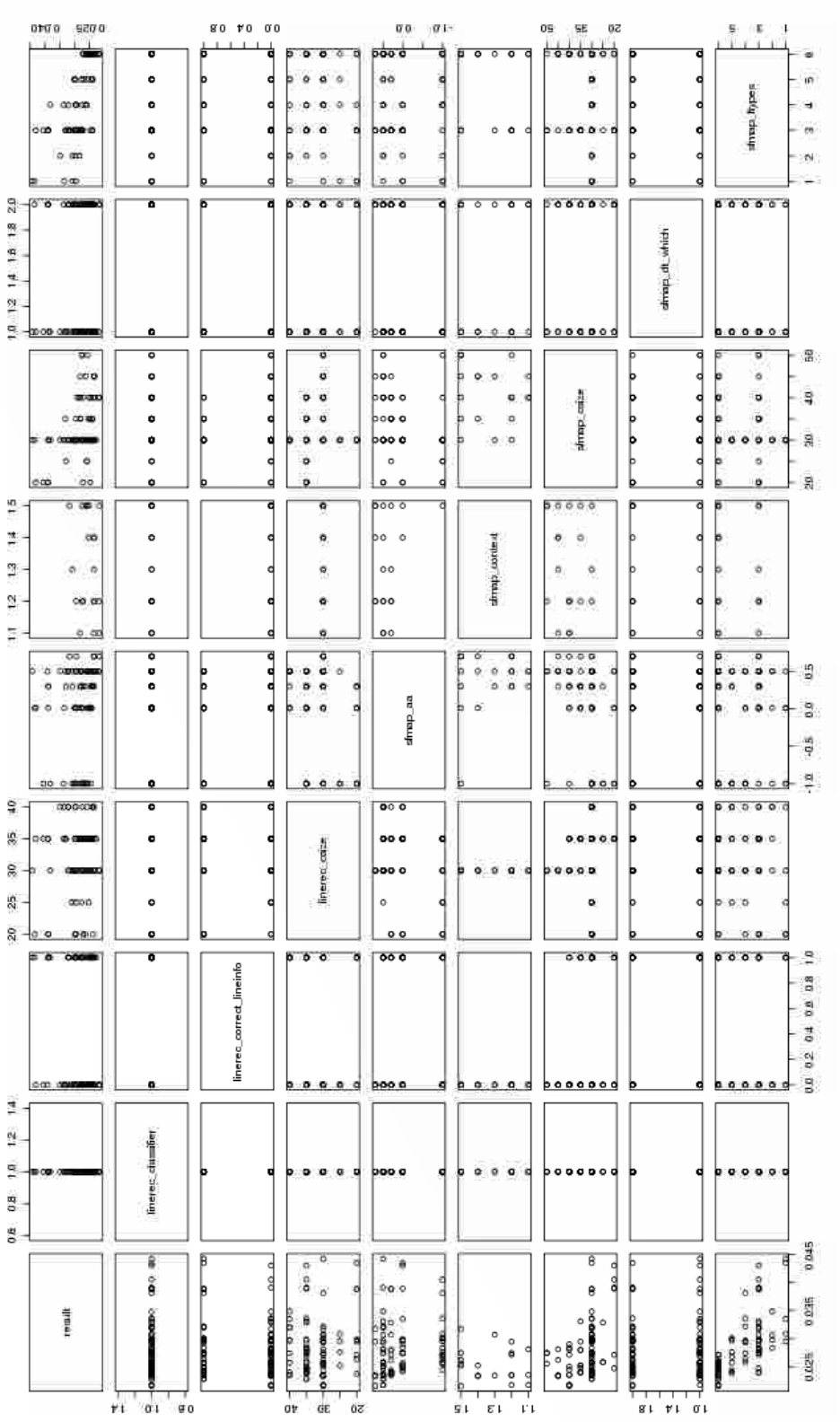

Figure 3: Scatterplot of different parameter choices vs. recognition rates summarizing multiple experiments, with a training set size of 50000 and a test set of 30000 on our standard database. Scatterplots like these help identifying good and bad parameter choices. For example, `sfmap_ftypes=6` is clearly superior to other choices for that parameter. This is an example of the kind of large scale experiments that can be carried out with OCRopus.

One simple approach for dealing with these issues is to define such ligatures as new characters; in fact, we are planning on incorporating this approach into OCRopus in the future. This also can cope with letter pairs that are not linguistically ligatures but can be difficult to segment, such as "az" or "TT".

However, in OCRopus, we have implemented a complementary approach to coping with segmentation errors. The approach works for letter pairs in which the segmenter correctly identified the existence of a "cut" between the two letters, but the cut is not necessarily in exactly the right place. The idea is to provide the classifier not just with the character hypothesis itself, but also with a representation of the neighboring characters (the context). This approach represents a compromise between scanning- or segmentation-free approaches to OCR (e.g., convolutional neural networks) and fully segmentation based methods.

Several different ways of representing context are possible. One is to compute two feature maps, one for the character itself and one for the context, or one for the character itself and one for the character in context. Another is to encode both the character and its context within the same feature map but use different numerical values for the character and its context. That is, if we use as the feature map just the bitmap of the character itself, if x_c is the image of the character hypothesis with the context masked, and x_s is the image of the character hypothesis with the context present, then we choose as the final input to the classifier $x = x_c + \alpha x_s$. Note that for $\alpha = 0$, the input is reduces to the segmented character hypothesis without context, and for $\alpha = 1$, the input reduces to a multiple of the character with context and no segmentation information.

We have tried different values of α and, across different conditions, found that a value of approximately $\alpha = 0.3$ assumes a minimum between $\alpha = 0.0$ (no context) and $\alpha = 1.0$ (no segmentation information). A value of $\alpha = -1$ also works, but performs worse than $\alpha = 0.3$. See Figure 4 for an exploration of these parameters.

We believe that contextual training will become increasingly important for some non-Latin scripts (e.g., Urdu, Sanskrit), as well as historical Latin script variants. Ligatures are very frequent and quite variable in those scripts; incorporation of context into the recognition process potentially allows us to perform reliable ligature recognition without treating each possible ligature as a separate glyph.

Training.

The fundamental training primitive for the OCRopus line recognizer is training on ground-truth segmented data. That is, the recognizer receives images of text lines, their transcriptions, and information about a segmentation of the text line into characters. This segmentation does not have to be perfect (e.g., approximate bounding boxes are sufficient) in order for this to succeed, since the segmented character recognizer can align imperfect segmentations with the oversegmentation used for recognition of unsegmented strings.

When only text lines and their transcriptions are available, but no corresponding ground truth character segmentation of the input line, then OCRopus can perform an EM training step to recover the missing alignment. That is, the system performs segmentation followed by classification using a pre-existing character recognizer (this can be a very simple recognizer, such as a nearest neighbor classifier constructed

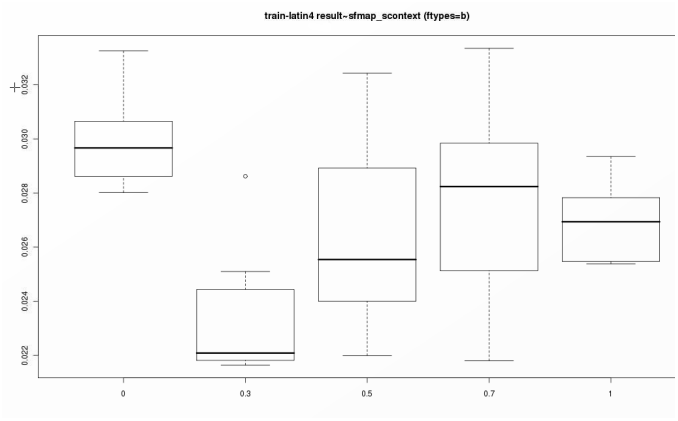

Figure 4: An experiment showing the dependence of error rates on context scale α (see the text for the meaning of this parameter). Experiments were performed with 50000 training samples and 30000 test samples. Each condition was tested in 30 experiments using randomly chosen parameters for other feature extraction parameters (see Figure 3). Note that the number of training examples used for these experiments is much smaller than the number of training examples used for training an actual line recognizer, but the results appear to generalize to much larger training sets. (Box-whisker plots as in Figure 2.)

from synthetically generated training data), then aligns the resulting recognition lattice with the ground truth text using a dynamic programming algorithm, and finally derives the most likely input segmentation from the alignment between the transcription and the input image. This then results in a segmentation of the input text line image into character hypotheses. This segmentation can then be used to train the text line recognizer on segmented characters. The overall training procedure is similar to Baum-Welch training for Hidden Markov Models.

The training procedure for the text line recognizer gives us a great deal of flexibility for adapting it to new languages. The recognizer can be trained on character shapes derived from fonts for the target language, then this simple model can be bootstrapped using text lines with transcriptions but no character segmentation. Recognizers can also be trained based on page images with unaligned text or even errorful transcriptions by aligning with general language models derived from these sources of transcription.

Overall, these different training procedures give us a great deal of flexibility in targeting OCRopus to new languages with a minimum amount of manual transcription effort. We have applied all these methods in training for Latin script and Western languages, and we have begun to explore training Russian and Fraktur using the same approach.

Performance.

The development of the text line recognizer so far has been driven by a number of training and test sets derived from the ISRI University of Washington, and Google Books databases. Error rates on our subset of ISRI data are approximately 0.77%, and 1.3% on the University of Washing-

ton dataset, using models trained on each respective dataset and then testing on a separate test set derived from the same dataset. On the ISRI data, Tesseract currently achieves an error rate of 3.8% in our experiments.

However, although these results are statistically valid and correct, considerable difficulties remain in the prediction of performance on new real-world data. For example, since the OCRopus text line recognizer has not been trained on a large number of clean LaTeX documents, it has some recognition errors on those documents for less frequent characters (e.g., "1" vs. "l"), while systems designed for and trained on more diverse sources perform slightly better on such data. We will return to the difficulties of OCR benchmarking and performance prediction in the discussion.

The focus for improving the performance of the OCRopus text line recognizer is in improving the diversity and size of the training data sets on which its classifiers are trained. Although the text line recognizer has been trained on several million characters, this represents only a small set of the possible variations in scanning and font styles that commonly occur in practice.

5. LANGUAGE MODELS AND DECODING

OCRopus language modeling is based on weighted finite state transducers (WFSTs). WFSTs are capable of representing a wide variety of special purpose language models, such as n-grams, dictionary models and regular expressions. WFSTs can also address character coding and representational issues, such as Unicode representation normalization and splitting or joining ligatures. Finally, they can also be used for structural analysis of OCR output, for example for information extraction or linguistically-based reading order determination.

Earlier versions of OCRopus relied exclusively on the OpenFST[1] toolkit for language modeling and integrating the recognition lattice with the statistical language models. However, the OpenFST toolkit was not intended to provide efficient decoding of inputs; its search algorithms are intended primarily for language model construction.

We have therefore implemented special beam search and A* search algorithms for decoding. These algorithms take the recognition lattice produced by the text line recognizer as input, as well as a statistical language model, and produce the best overall decoding of the input subject to the language model. These new decoding algorithms are now capable of coping with large, realistic language models. All decoders and all internal processing of OCR results is carried out in terms of UTF-32/UCS-4 Unicode strings, strings in which each codepoint represents a Unicode character. The OpenFST toolkit is still available, and still used, for the optimization of language models.

Using this infrastructure, we have started to construct a variety of language models, and also created tools that aid in the construction of such language models.

Our currently best model is a combination of a dictionary-based model, with back-off to a simple statistical character-based model. The model also incorporates models of punctuation, spacing, and character case. Back-off parameters are estimated by searching the space of back-off parameters for those that give the overall lowest cross-validated error rate.

The use of WFSTs as a general-purpose language model representation and the consistent use of UTF-32 throughout OCRopus means that the language modeling components should work for a wide variety of languages and scripts. For the set of languages and scripts we have investigated and experimented with so far (English, German, Fraktur, Japanese, Urdu, Sanskrit), we have not identified any functional or performance issues related to our choice of WFSTs for language modeling.

For multi-script and multi-language recognition, WFST models appear particularly well-suited, because they allow us to combine language models for multiple languages in a principled way. For example, for a document containing mixed English and Latin text, we can take separate English and Latin language models and then combine them by allowing transitions from one language model to the other at word boundaries. More advanced language models can be constructed by making these transitions context dependent or learning length models for strings in each language (e.g., if Latin phrases only appear as short strings inside English text). The process by which such models can be constructed is similar to the backoff models OCRopus is already using for switching between dictionary-based and character-based language models.

6. BOOK-LEVEL RECOGNITION FORMAT

Many OCR systems are designed to recognize inputs a page at a time; OCRopus can also be used in that mode. However, OCRopus is designed to recognize whole books. In order to support such usage, OCRopus now supports an intermediate representation that contains the results of preprocessing, layout analysis, OCR, and language modeling for a whole book at a time. The new `ocropus` command line program operates on the whole book representation at a time.

The book-level representation contains images representing each page of the book, the corresponding page segmentation as a color image, and, for each page, a subdirectory containing images for all the text lines, their corresponding segmentations, the corresponding recognition lattice, and the recognized output.

The OCRopus commands operating on this representation are:

- `ocropus book2pages dir src...` – split an input image (e.g., TIFF) into a book-level representation
- `ocropus pages2lines dir` – perform layout analysis and compute the individual text line images
- `ocropus lines2fsts dir` – perform text line segmentation and recognition and output the results of recognition as recognition lattices
- `ocropus fsts2text dir` – apply a statistical language model, apply the Viterbi algorithm, and output the text corresponding to each line
- `ocropus buildhtml dir` – gather the outputs from the different processing stages and construct output in HTML/hOCR format

This approach to book-level representation has a number of advantages. It allows estimation of noise levels, fonts, resolutions, page size, and other parameters for an entire book, and then to take advantage of that information in order to

recognize each page. It also enables book-level adaptation of character and language models (see below).

The representation also makes it easy to replace built-in processing steps with processing by external tools in a simple and efficient way (on modern file systems, the overhead of file writing and opening is negligible compared to the actual processing steps). For example, an external text line recognizer can pre-load its recognition models and then iterate over all the text lines of the book, recognizing them one by one. Such a recognizer does not need to concern itself with any aspect of layout analysis, language modeling, pre-, or post-processing.

7. BOOK-LEVEL ADAPTATION

One of the original goals of the OCRopus project was to take advantage of the consistency of fonts and degradation of characters within a single book in order to increase recognition performance.

A number of methods have been proposed for achieving this in the literature. Statistically well-founded techniques are generally based on font clustering[10], statistical mixture models of styles[11], or hierarchical Bayesian models[9]. (More recently, some of these approaches have been rediscovered in the machine learning community under semi-supervised learning and transfer learning.)

Each of these methods can be implemented "directly". For example, we can try to perform a preliminary segmentation of the input into characters, then perform shape clustering, and finally try to find an optimal assignment of labels to clusters given linguistic constraints imposed on the actual sequence of characters in the original document. However, such a "direct" approach leaves many open questions, such as how to perform the segmentation and how to integrate the output of the classifier with the linguistic model.

The approach OCRopus takes to book-level adaptation is therefore the following:

- We first perform recognition using a line recognizer with a general purpose character shape model.
- Then, the language model is applied to the resulting recognition lattices for each line. The output of applying the language model is both text and a segmentation of each text line into character hypotheses.
- The text lines with the worst recognition scores from either the recognizer or the language model are rejected for the purpose of retraining.
- In a retraining step, the text line images, their segmentations, and the corresponding text are then used as if they were ground truth in the training of a new character recognizer.
- The newly trained model is then used to re-recognize the text lines and obtain the final output.

This sequence of steps is similar to the original EM training algorithm. However, instead of a ground truth transcript for each text line, we use a more general purpose language model. From the point of view of OCRopus, the difference between ground-truth-based training and language-model based book adaptation is merely the amount of information that is conveyed from the language model to the classifier.

Clustering, style mixture, and hierarchical models of book-level adaptation can be implemented within this framework through particular choices of classifiers:

- Choosing a classifier for the retraining step that performs mixture density estimation followed by classification results in a classification procedure that implements recognition-by-clustering. Here, the first round of text line recognition is used in order to perform the segmentation step that is necessary for clustering recognition. The mixture modeling and discriminative training of the mixture classifier make precise the notions of distance and weighting of each prototype character.
- Choosing a classifier for the retraining step that is a mixture of pre-trained component classifiers and then adjusting only the mixture component weights during the retraining step yields a style mixture classifier.
- Choosing a classifier that can be trained with stochastic gradient descent for the retraining step can be used to approximate a hierarchical Bayesian algorithm. In order to perform retraining, the classifier during retraining is initialized with parameters for the non-adaptive classification case. During retraining, the learning rate and number of samples/epochs that are presented for training determine the tradeoff between the prior classifier and the adaptation to the book.

We note that there is a second form of adaptation that is useful and simpler than the methods mentioned previously. It is somewhat analogous to parametric adaptation mentioned above. Given a set of classifiers, represented by parameter settings θ_q, we perform text line recognition $s_l^{(q)} = \arg\max_{s_l} P(s_l|x_l, \theta_q)$ on a sample of text lines x_l from the book. We then pick the θ_q that maximizes the probability of the strings $s^(q)_l$ according to a language model $P(s)$. Or, in short, we pick the classifier among a collection of classifiers, that yields the "most plausible" outputs. This approach can also be interpreted as a special case of the style mixture classifier, in which the mixture parameters are restricted to have weights of 0 and 1 exclusively.

In OCRopus, we have focused so far on adaptation using stochastic gradient descent. We have demonstrated reductions in error rates by 30-50% in preliminary experiments.

Book-level adaption is not only important for generally improving the performance of recognition (in particular for scripts for which limited training data is available), it can also specifically help with the recognition of documents printed in multiple scripts by selecting and adapting combinations of classifiers to the particular collection of scripts found within a document.

8. DISCUSSION

This paper has given an overview of the on-going efforts to create a multi-script, multi-lingual OCR system for digital library applications. We have not yet performed large scale training for other scripts or languages, but we have tested individual processing steps that have given us confidence that OCRopus is on the right path for non-Latin scripts and diverse languages, in particular: training based on synthetic data, extensive semi-supervised training, training using errorful ground truth data, combinations of multiple language models, and different kinds of classifier testing.

A significant obstacle that we have observed in building and testing OCR systems is the lack of truly representative training and test data. The ISRI and UW3 datasets, for example, have been useful for both training and testing, but the fact that they are not representative of OCR problems as a whole can be inferred from the fact that, while models trained and tested on disjoint subsets of each dataset perform well, training on one dataset and testing on the other yields significantly worse performance. For building modern book recognition applications, a particularly significant limitation is the lack of large amounts of high quality scans of modern books. These issues are likely going to be more serious for other languages, where intrinsically less training data is available.

Acknowledgements

The OCRopus project would like to acknowledge the generous support of Google.

9. REFERENCES

[1] C Allauzen, M Riley, J Schalkwyk, W Skut, M Mohri OpenFst: A general and efficient weighted finite-state transducer library Lecture Notes in Computer Science, 2007 - Springer

[2] T.M. Breuel A system for the off-line recognition of handwritten text Proceedings of 12th International Conference on Pattern Recognition, pages 129-34 vol.2

[3] T.M. Breuel Segmentation of handprinted letter strings using a dynamic programming algorithm Proceedings of Sixth International Conference on Document Analysis and Recognition, pages 821-6

[4] The OCRopus Open Source OCR System Thomas M. Breuel Proceedings IS&T/SPIE 20th Annual Symposium 2008

[5] Chih-Chung Chang and Chih-Jen Lin LIBSVM: a library for support vector machines Software available at `http://www.csie.ntu.edu.tw/ cjlin/libsvm`, 2001.

[6] Y LeCun, L Bottou, Y Bengio, P Haffner Gradient-based learning applied to document recognition Proceedings of the IEEE, 1998

[7] Zhidong Lu Schwartz, R. Natarajan, P. Bazzi, I. Makhoul, J. Advances in the BBN BYBLOS OCR system ICDAR '99, pp. 337-340. '

[8] S Marinai, M Gori, G Soda Artificial neural networks for document analysis and recognition Transactions PAMI, 27(1), pp. 23-35.

[9] C. Mathis, T. M. Breuel Classification using a Hierarchical Bayesian Approach Proceedings of the International Conference on Pattern Recognition (ICPR'02), Quebec City, Quebec, Canada

[10] G. Nagy, S. Sharad, K. Einspahr, and T. Meyer Efficient algorithms to decode substitution ciphers with applications to OCR. 8th Int. Conf. on Pattern Recognition, vol. 1, pp. 352–355, 1986.

[11] P. Sarkar Style consistency in pattern fields. Ph.D. thesis, Rensselaer Polytecnic Institute.

[12] PY Simard, D Steinkraus, JC Platt Best practices for convolutional neural networks applied to visual document analysis ICDAR 2003.

[13] Smith, R. An Overview of the Tesseract OCR Engine ICDAR 2007, pp. 629-633.