

DOCLIB: a software library for document processing

Stefan Jaeger^{1a}, Guangyu Zhu^a, David Doermann^a, Kevin Chen^{2b}, Summit Sampat^b

^aInstitute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

^bBooz | Allen | Hamilton, 134 National Business Parkway, Annapolis Junction, MD 20701, USA

ABSTRACT

Most researchers would agree that research in the field of document processing can benefit tremendously from a common software library through which institutions are able to develop and share research-related software and applications across academic, business, and government domains. However, despite several attempts in the past, the research community still lacks a widely-accepted standard software library for document processing. This paper describes a new library called DOCLIB, which tries to overcome the drawbacks of earlier approaches. Many of DOCLIB's features are unique either in themselves or in their combination with others, e.g. the factory concept for support of different image types, the juxtaposition of image data and metadata, or the add-on mechanism. We cherish the hope that DOCLIB serves the needs of researchers better than previous approaches and will readily be accepted by a larger group of scientists.

Keywords: Document processing, Metadata, Software development, Software management

1. INTRODUCTION

The transfer of technology across organizational boundaries, and very often even within the same organization, typically requires a significant amount of time: Setup, configuration, and execution of research-related software all consume time that could otherwise be spent more productively. This process is often exacerbated when there is incomplete documentation, unpredictable methods required to build and execute software, inconsistent software dependencies, and/or a misuse of software causing many efforts to be done in vain. This lack of infrastructure has caused an avoidable loss of intellectual capital in the past and has hampered progress within the document processing community.

The use of a shared software library and standard development procedures can mitigate many of the current challenges and maximize productivity. A set of core research tools developed with an eye toward stability can allow valuable intellectual resources to be more focused on creative and innovative document processing solutions. This in turn can maximize return on investment for funding organizations. It would also facilitate the first steps of newcomers to the field, providing them with a stable toolbox of standard methods upon which they can build their own applications.

There are several reasons why so many software libraries have failed in becoming widely accepted standards: Most existing software packages focus on image processing rather than document processing with its specific requirements, such as manipulation of metadata. For instance, the popular OpenCV library provides high-level computer vision methods, with almost no support for document processing [11]. However, image processing is only one part of document processing, and equal weight should be attached to the representation of both the image data and the higher-level logical content of documents. While some approaches have failed because of low execution efficiency, e.g. MATLAB [14], and poor design concepts, others were too demanding in the sense that their user interface was too complex and imposed too many constraints on the developer right from the beginning. The recent approach Gamera is based on the interpreted programming language Python and uses procedural programming mainly for plug-ins where runtime performance is a priority [15]. It also provides an interface for training classifiers. For more information on existing systems, readers are referred to [12] or [13], which describe the TRUEVIZ system and in addition provide an

¹ jaeger@umiacs.umd.edu

² chen_kevin@bah.com

overview of the existing systems and their limitations when it comes to document processing and generation of metadata.

Our paper describes a systematic development approach and shared architecture, DOCLIB, which has been successfully used in collaboration across academia, business, and Government environments. It is structured as follows: Section 2 describes the basic design decisions and features our approach is based on. Section 3 provides more information about the internal representation of image data and metadata, their interaction with each other, and some of the basic image processing capabilities offered by DOCLIB. Section 4 gives a short overview of the software development process, including bug tracking and quality control, which have been essential to DOCLIB's stability, and should not be underestimated. Finally, Section 5 lists some of the current applications of DOCLIB. A summary concludes the paper.

2. APPROACH

DOCLIB has been developed with the intention of providing basic document and image-processing capabilities as well as a possibility for programmers to easily develop their own application on top of DOCLIB. The applications should be accessible through a well-documented, easy to use interface (API or command line). We deemed C++ the most appropriate programming language for this effort, especially for the support of a functional, stable, and robust programming interface, including data structures facilitating the collaborative development of research capabilities. For these reasons, we considered it important that DOCLIB supports a plug-and-play architecture that allows straightforward addition of new image types and their conversion. We planned DOCLIB as a mechanism for easily transferring and communicating software-related research ideas. Thus, DOCLIB is required to be scalable, flexible, and extendable, allowing additional functionality as needed. We therefore conceived an add-on mechanism that allows research groups or organizations to easily augment DOCLIB with new features that for various reasons do not belong to the core of DOCLIB, without modifications to the existing DOCLIB code. The solution implemented also allows research groups/organizations to “plug-in” confidential or proprietary code.

The cornerstone of the DOCLIB design is the juxtaposition of image data and document data, which is stored in the DLImage and DLDocument class, respectively. This design paradigm is also what sets DOCLIB apart from most previous approaches, which did not emphasize this dualism to the extent DOCLIB does. While DLImage is a container for the basic image data and operations implemented in DOCLIB, DLDocument offers data structures and methods for manipulation of metadata, i.e. data that is on a higher cognitive level than just the image pixels, such as ground-truth data for classifier training, page segmentation, etc. The central idea is that developers perform all image operations via the DLImage class and all operations on metadata through the DLDocument class. DLImage and DLDocument are two independent modules standing side by side and interacting with each other, as illustrated in Figure 1. In particular, metadata objects in the DLDocument class all have a reference to their corresponding DLImage object and can thus request their own byte image or sub image from the DLImage class.

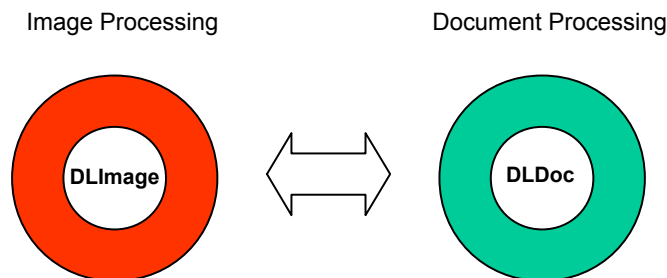


Figure 1: DLImage and DLDocument

3. IMAGE AND DOCUMENT REPRESENTATION

This section contains more information about the implementation details of DLImage and DLDocument.

3.1 DLImage

DLImage is the centralized image object class used to store image data and header information when loading an image, and also visualize metadata information such as bounding boxes or ground-truth data. The DLImage object is a generic image format that is independent of the input image types (i.e. TIFF, JPEG, GIF, etc.). DOCLIB's Image processing algorithms are designed as independent classes that can be added or deleted without affecting other core-DOCLIB features (See Figure 2).

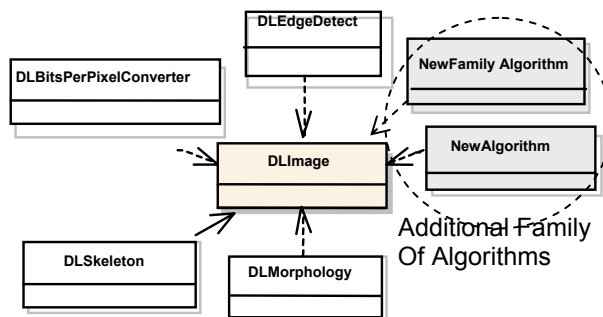


Figure 2: DOCLIB algorithm hierarchy

DOCLIB's core currently supports the following basic image operations:

<ul style="list-style-type: none"> • Resize an image • Rotate an image • Subimage • Flip an image • Contour an image • Reverse black/white pixels on an image • Paste an image • Project • Calculate connected components • Draw Line, Box, Pixels onto an image • Dilate an image • Erode an image • Sharpen an image • Blur an image 	<ul style="list-style-type: none"> • Conversions: <ul style="list-style-type: none"> - 256 Color quantized - Percent Thresh hold Binarization - Threshold Binarization - Color To Gray - Threshold Gray To Binary - NIBlackBinarization - Binary To Color - Gray To Color - Binary To Gray - YCrCb Binarization • Skeletonize • Deskew • Centroid
--	--

Closely connected with the DLImage class is the factory concept, which allows users to 'plug-in' proprietary image types without making modifications to the core DOCLIB library. DOCLIB's Image Factory class inherits the standard Factory Design Pattern concept and extends the factory idea even further for the sake of easy usability and

expandability. It enables a higher level of abstraction from the user and encapsulates the implementation of all image objects (Figure 3). The Image Factory provides the intelligence to determine an image's file format without having the user specifying it.

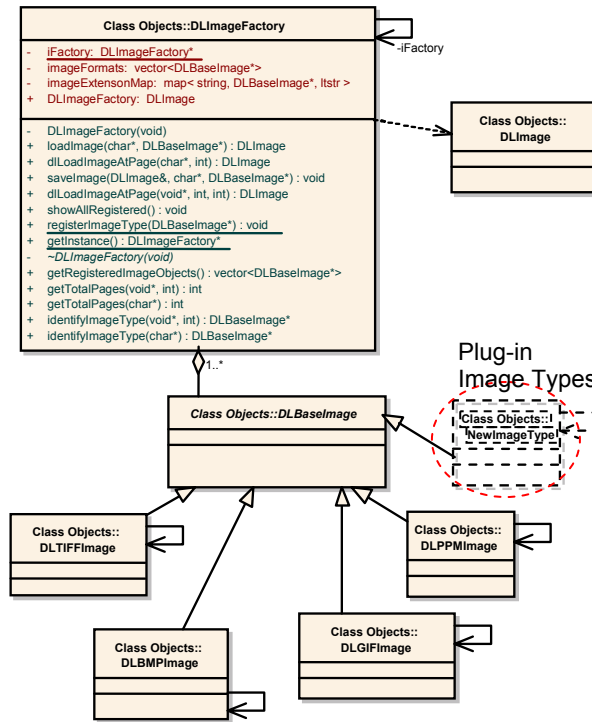


Figure 3: Image Factory class

Image type objects register their existence at the Image Factory during construction (Figure 4). Once the new image object has been registered with the Image Factory, the new image type will be considered to be one of the supported images in DOCLIB. This concept has already proven its capabilities in several practical applications, allowing users of DOCLIB to integrate and use their own proprietary image formats without code modifications.

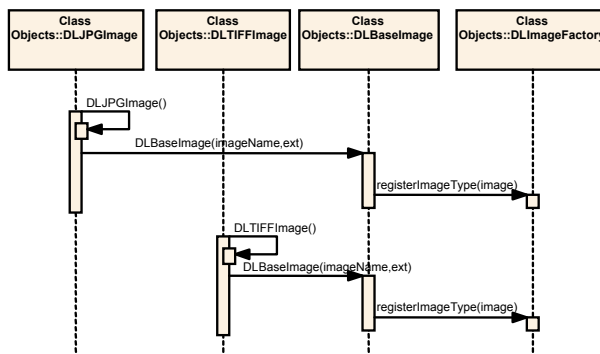


Figure 4: Image plug-in registration process

3.2 DLDocument

We devised the DLDocument class as the basic container for metadata, i.e. data describing the logical representation of a document. In addition, DLDocument provides essential methods for modifying metadata, such as assigning types or attributes. We designed DLDocument so that it is as simple as possible and yet powerful in terms of flexibility and

adaptability to new problems. DOCLIB's developer interface comprises all data structures defined within DLDocument and its subclasses, which can in turn be subclassed and thus be adapted to the specific user requirements.

The DLDocument class understands documents as hierarchical entities composed of pages containing regions of interest, which we call zones. Accordingly, DOCLIB's document hierarchy consists of the following objects:

- DLDOCUMENT
- DLPAGE
- DLZONE

The zones are hierarchical structures in themselves and can contain sub zones (see Figure 5).

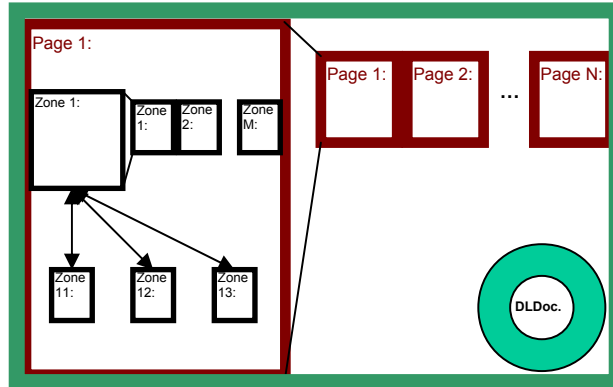


Figure 5: DOCLIB document hierarchy

The DLDocument class represents the highest level in the DOCLIB document hierarchy. It is a container for attributes that describe the layout and content of one or more pages within a document. Each DLDocument object has a unique document ID and features a list of document tags allowing developers to dynamically append or remove additional self-defined attributes. Several constructors are available, depending on the information present at the time of construction. The pages of a document are stored in a simple list structure. Developers have access to this list via a set of functions for appending, inserting, or deleting either one single page or a list of pages.

The DLPage class represents the logical content of a document page. Its main purpose is to accommodate information specific to pages rather than whole documents. The structure of DLPage, however, is similar to DLDocument. It also contains a page ID and a dynamic attribute list that developers can use to store self-defined attributes. The main difference is a list of zones, which are containers defining specific regions on a page, such as text blocks, images etc. DLPage provides functions for appending, inserting, or deleting zones on a page. In addition, it provides methods for accessing page-related information, e.g. its spatial dimensions or the corresponding DLImage.

A DLZone object is a general container for objects on a document page. DLZone offers a general bounding box concept that allows zones to have sub zones, so-called child zones. These child zones are stored in the same list structure used by DLDocument and DLPage to store pages and zones, respectively. A zone and its sub zones span a dynamic tree structure in which zones (nodes) can easily be appended, inserted, or deleted. In addition, DLZone provides methods for merging and splitting zones. Similar to methods in DLPage, DLZone offers methods for accessing information specific to zones. A zone object is basically defined by its origin, width and height. DOCLIB allows developers to dynamically change this information. Each DLZone object automatically performs a consistency check whenever the developer tries to do so, and denies the operation when it would lead to an inconsistent state, for instance when a zone would extend the boundaries of its parent.

In order to allow programmers to adapt DOCLIB to their own needs, and customize DOCLIB objects to the requirements of a specific application domain, developers can subclass DLZone and augment it with more specific information. Typical user-defined subclasses of DLZone could be, for example, DLTextBlock, DLTextLine, or

DLCharacter etc.

4. SOFTWARE DEVELOPMENT PROCESS

DOCLIB has shown that adapting a structured development process results in more robust research solutions that minimize technology migration issues and enhances user acceptance. Some of the main implemented key aspects that contributed to DOCLIB's success are the following:

Distribution portal: Coordinating software changes or enhancements without a distribution mechanism can be quite challenging. Given the complexity of document processing related research, it is often that code modifications are transferred to end-users. We developed a web portal that allows users to check status, download specific versions of the libraries, and seek help.

Source control: One of the most important aspects of collaborative code development is the sharing of the code itself. Without all users having access to the same code bases, each will inherently develop similar but different baselines in parallel. In order to facilitate the management of software development, we installed several open source version control software packages. These systems provide developers with the ability to obtain and modify source code for their projects. If errors occur, a configuration management (CM) tool provides methods for reverting code to a previous known stable state.

Documentation standards: Thorough and understandable documentation is the key to the marketing and use of software, but also the most often ignored. DOCLIBS's regimented documentation standards and use of auto-generating documentation packages (i.e. Doxygen [7]) allow user documentation to be quickly and easily created.

Bug tracking: Bug tracking is one of the most valuable tools in the development of DOCLIB. Bug tracking gives users and developers the ability to document bugs, request new features, and check the status of the resolutions to their issues. We use the popular Mantis bugtracker for DOCLIB [10].

Software integrity: It is difficult to guarantee software integrity within the document processing research field given the complexity of the science and the variability in data. The need to verify system behavior was evident early on during the development of DOCLIB. A regression test application was prescribed that thoroughly tested all behavior within the system. This module is often used by software developers before committing changes to existing code. This approach has also proven invaluable in identifying obscure bugs before formal core-DOCLIB software releases. The execution of regression tests also facilitates the observance of memory leaks throughout the system. They have proved very useful in minimizing the number of bugs occurring throughout DOCLIB.

5. APPLICATIONS

DOCLIB has been used to implement solutions to a wide range of problems in document processing. Many of these solutions have become part of DOCLIB's core, while others are available as add-ons, which are built on top of the DOCLIB core. A DOCLIB add-on standardizes the delivery and use of research applications. Robust test scripts, standard naming conventions, auto-generated documentation, and an organized directory hierarchy are some of the attributes that are included as part of an add-on. Conformance to the add-on conventions defined reduces the number of prerequisite assumptions and problems during installation. Add-ons allow researchers to easily swap algorithms and try out different solutions for a given problem. The use of DOCLIB add-ons has facilitated technology migration and productivity between University of Maryland and several government organizations throughout the past two years.

The core of DOCLIB provides basic image processing methods that have been successfully tested in numerous practical applications and are considered standard by researchers and programmers alike. Add-on modules, however, are solutions to problems that have hitherto eluded practical approaches, and are still actively investigated within the

research community. Methods competing with other techniques are also implemented as add-ons, either because the research community has not yet given the final verdict on their underlying theory or they are fine-tuned to specific problem types and can thus not be considered standard for the general case. DOCLIB provides most of its document processing functionality as add-ons to the DOCLIB core. The following is a short list of applications for some of the currently available add-ons provided by DOCLIB. Several of these applications describe state-of-the-art solutions for problems that the research community has not yet found convincing solutions for (see also the literature references at the end of this paper).

- **Script Identification**

DOCLIB's script identification add-on performs script identification on machine printed document images. It is a modified version of the Hochberg algorithm described in [1], and allows classifying a document image as being printed in one of the following scripts: Amharic, Arabic, Armenian, Burmese, Chinese, Cyrillic, Devanagari, Greek, Hebrew, Japanese, Korean, Latin, or Thai. It can also be retrained to focus on different language mixes.

- **Layout Analysis**

The page layout add-on provides an automated means of training a classifier to recognize a document layout or set of layouts. The classifier can then be used to score an unknown image. The software is easily modified to include other types of object information. More details are given in [4].

- **Page Segmentation:**

For page segmentation, DOCLIB provides a module implementing the Docstrum method for structural page layout analysis [2]. The Docstrum method is based on bottom-up, nearest-neighbor clustering of page components. It detects text lines and text blocks, and is advantageous over many other methods in three main ways: independence from skew angle, independence from different text spacings, and the ability to process local regions of different text orientations within the same image.

- **Line Processing**

Text Line Detection:

Given a deskewed, oriented text image with dark machine print text on a light background, the line detection add-on will calculate text line information based on the connected component information. The image must be deskewed and oriented in order to receive meaningful results. The algorithm utilized has proven robust given text images with low resolution, broken characters, connected scripts (e.g. Arabic), multi-component character scripts (e.g. Chinese), and noise.

Line Removal:

The line removal module is used to read in a document image and locate the background horizontal lines [6]. It then creates an output file in which the background lines are removed while the text remains intact. This add-on is designed to work on handwritten documents where the user writes on lined paper. Consistent background lines with inconsistent writing are expected. Less than optimal (though still useful) output may result when this is not the case. Various parameters are defined in the module which, when modified, may improve performance depending on the data. The module does not rely on color information. Hence it is particularly handy when the background horizontal lines are similar to the color of the handwriting. The algorithm requires binary input images that are reasonably deskewed (roughly, between -9 and 9 degrees, inclusive).

- **Object Recognition**

Logo Recognition:

This module implements a logo recognition algorithm. Given a potential logo it performs a gray scale correlation against a set of candidate logos. For each candidate a score between 0 and 100 is generated corresponding to the degree of similarity. The best match is provided along with the score. In order to improve performance, the algorithm stops comparing against candidate logos when the best score is beneath a pre-defined threshold.

Stamp Detection:

The stamp detection module detects and locates elliptic stamps or seals on an input image by identifying and characterizing elliptic connected edges using both their magnitude and orientation information [8]. For a given document, it outputs an image list of detected stamps together with their corresponding confidence values. The module shows good performance for images with moderate noise and can be efficiently used in batch mode to process a large set of images.

- **Miscellaneous Applications**

Classifier Combination:

This add-on is an implementation of a new technique for information fusion and classifier combination, developed at the University of Maryland for hard and noisy environments. The underlying information-theoretical idea is to first normalize the confidence values provided by different methods for the same problem, before actually combining them. The normalization is performed in such a way that each confidence value matches its informational content. The recognition candidate providing the maximum information over all methods will then be selected as the most likely candidate for a given classification problem and presented test pattern. The method has successfully been applied to several document processing tasks, including character recognition and script identification. Several papers describing the theoretical background have been published [3].

Degradation:

DOCLIB's degradation tool degrades images with different types of noise [9]. Its main purpose is to emulate noise typically introduced during the scanning of paper documents. The generation of well-defined noisy environments allows adaptation of recognizers to real-world noise usually encountered in practical documents. For instance, the following types of noise may be added to a document image: blur, jitter, horizontal and vertical lines, rotation, speckle, pixel flips, etc.

Performance Evaluation:

The F-Score module is used to compare two object sets, the putative set and the truth set. Based on this comparison, the precision, recall, and F measure values are calculated along with the F score(s) [5].

6. CONCLUSION

DOCLIB is a concerted effort to establish a new document processing library that will eventually reach wide acceptance in the document processing research community. We tried to learn from the mistakes in the past and paid a great deal of our attention to software development issues, which are crucial for DOCLIB to be accepted by the research community in the long term. Compared with other software libraries, DOCLIB offers several new features: Its architecture is based on two equally important classes for image processing and document processing. DOCLIB's image factory supports different kinds of image formats and its document hierarchy provides a user interface that is simple and yet should be powerful enough for most document processing applications. Developing document-related applications as add-ons to the core DOCLIB library has proven a useful means of sharing research capabilities among a large group of researchers during the last two years at the University of Maryland and other government organizations. While much of this paper

has concentrated on the design decisions and software-technical aspects of DOCLIB, there are already numerous state-of-the-art solutions available for different document processing tasks, as listed above.

In summary, DOCLIB has facilitated technology migration by mitigating debugging time, number of code reviews, and configuration management challenges. DOCLIB is currently not public software. However, decisions relating to sharing of the DOCLIB software, shared development environment, and add-ons are in process.

ACKNOWLEDGMENT

The partial support of this research under DOD contract MDA90402C0406 is gratefully acknowledged.

REFERENCES

- [1] J. Hochberg, P. Kelly, T. Thomas, and L. Kerns, Automatic script identification from document images using cluster-based templates, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 2, Feb. 1997, pp. 176–181.
- [2] L. O’Gorman, The Document Spectrum for page layout analysis, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 11, Nov. 1993, pp. 1162-1173.
- [3] S. Jaeger, Informational classifier fusion, *17th Int. Conf. on Pattern Recognition*, (Cambridge, UK), 2004, pp. 216-219.
- [4] L. Golebiowski, Automated layout recognition, *Symposium on Document Image Understanding Technology, Greenbelt (Maryland)*, pp. 219-228, 2003.
- [5] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques, Technical Report #00-034, Department of Computer Science and Engineering, University of Minnesota, 2000.
- [6] R. Ziemer, Removing repetitious horizontal background lines, Internal Report, Booz | Allen | Hamilton, 2004.
- [7] Doxygen, <http://www.doxygen.org>, October 2005.
- [8] G. Zhu, S. Jaeger, D. Doermann, A robust stamp detection framework on degraded documents, *SPIE Conference on Document Recognition and Retrieval*, 2006, to appear.
- [9] G. Zi, Groundtruth generation and document image degradation, Technical Report: LAMP-TR-121/CAR-TR-1008/CS-TR-4699/UMIACS-TR-2005-08, University of Maryland, College Park, May 2005
- [10] Mantis, <http://www.mantisbt.org>, October 2005.
- [11] www.intel.com/technology/computing/opencv/
- [12] C. H. Lee, T. Kanungo. The Architecture of TRUEVIZ: A groundTRUth/metadata editing and VIsualiZing toolkit. Technical Report: LAMP-TR-062/CS-TR-4212/CAR-TR-959, University of Maryland, College Park, February 2001.
- [13] T. Kanungo, C. Lee, J. Czorapinski and I. Bella, TRUEVIZ: A groundTRUth/metadata editing and VIsualiZing toolkit for OCR, *SPIE Conference on Document Recognition and Retrieval*, 2001.
- [14] MathWorks, <http://www.mathworks.com/>, October 2005.
- [15] M. Droettboom, K. MacMillan, and I. Fujinaga, The Gamera framework for building custom recognition systems, *Symposium on Document Image Understanding Technology, Greenbelt (Maryland)*, pp. 275-286, 2003.