Image Enhancement: A Lightweight, Easy-to-Use, and Efficient C++ Library

Algorithm Engineering 2026 Project

Daniel Motz
Friedrich Schiller University
Jena, Germany
daniel.motz@uni-jena.de

Leonard Teschner
Friedrich Schiller University
Jena, Germany
leonard.teschner@uni-jena.de

Mher Mnatsakanyan
Friedrich Schiller University
Jena, Germany
mher.mnatsakanyan@uni-jena.de

Abstract

TBC

Keywords

task parallelism, WebAssembly, OpenMP, image processing, C++

ACM Reference Format:

Daniel Motz, Leonard Teschner, and Mher Mnatsakanyan. 2026. Image Enhancement: A Lightweight, Easy-to-Use, and Efficient C++ Library: Algorithm Engineering 2026 Project. In *Proceedings of AEPRO 2026: Algorithm Engineering Projects (AEPRO 2026).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

1.1 Background

The digitization of documents, especially images scanned or captured with smartphone cameras, is an important step in making information available. It also forms the basis for further processing steps such as optical character recognition (OCR). Existing tools and libraries for image enhancement and document preprocessing are powerful, but often too complex, too specialized, or have many dependencies. This makes it difficult to integrate them into small, user-defined programs and leads to the bloating of these programs. There is therefore a need for a lightweight, easy-to-use, and efficient solution that focuses on image enhancement while being flexible and modular enough to be used in a variety of applications. As modern smartphone cameras capture images with ever-higher resolutions, the computing time required to process them also increases. Efficient implementation that utilizes the advantages of parallelization and other optimization techniques is therefore important for improving processing speed and ensuring user satisfaction.

1.2 Related Work

Image enhancement and document preprocessing are covered by a wide range of open-source tools and commercial software. The scope ranges from general purpose image processing libraries and OCR (optical character recognition) libraries to specialized tools for post-processing scanned documents.

One of the most widely used libraries for general image processing is OpenCV [5]. Described by Bradski in 2000 [3], OpenCV offers a wide range of functions and algorithms for image processing, such as filters, transformations, segmentation, and feature detection, as well as machine and deep learning. OpenCV is written in C++ and has a modular, cross-platform architecture that allows for easy integration into various applications. However, OpenCV is a very complex library with many functions and dependencies, which leads to overhead when only a fraction of the functions are used. OpenCV is also licensed under the BSD license. Similar to OpenCV, ImageMagick [4] is a comprehensive suite for image processing that is available as a command line tool and as a library. It offers functions such as format conversion, resizing, filtering, and color correction. However, ImageMagick is not specifically designed to enhance scanned or photographed images, but rather for general image processing. The wide range of functions and possibilities results in a large number of dependencies and a large code base. ImageMagick is licensed under the Apache License.

The open source OCR engine Tesseract is often used for document analysis. Tesseract was described by Smith in 2007 [8] and has become one of the leading OCR engines. In addition to text recognition, Tesseract also offers some image enhancement features, but the focus is on OCR. The image enhancement features are typically integrated into the OCR workflow and are not available as general image enhancement features. Tesseract is licensed under the Apache License. A similar library specializing in OCR is the C-based Leptonica library [2]. Leptonica offers image processing functions that are widely used in the backend of OCR engines such as Tesseract. It provides functions such as binarization, noise reduction, and morphological operations, which are important for preparing images for OCR recognition. Although Leptonica is relatively lightweight compared to the other libraries, it primarily serves as a low-level library and does not offer ready-to-use, configurable processing pipelines, for example. It is also written in C, which makes integration into C++ applications difficult. Leptonica is licensed under the BSD license.

A more specialized tool is Unpaper [6], which focuses on post-processing scanned documents. It offers features such as deskewing, binarization, and noise reduction that have been developed specifically for this purpose. However, Unpaper is highly specialized and cannot be flexibly integrated into custom C++ applications. Due to its configuration, Unpaper has many dependencies, which makes it difficult to use. Unpaper is licensed under the GPL license, which restricts commercial use.

1.3 Our Contribution

Unlike the existing tools and libraries described, our approach is designed to be minimalistic, extensible, and user-friendly. Instead of being a comprehensive image processing library or OCR engine, we focus on a lightweight, modular image enhancement pipeline written entirely in C++ with only one dependency (CImg [9]), which

is itself lightweight and portable. We systematically optimize and improve our intuitive baseline implementation and evaluate the performance gains through various experiments. Compared to OpenCV and ImageMagick, our work is characterized by its deliberate focus on image enhancement and compact architecture. Unlike Tesseract and Leptonica, our pipeline is not integrated into an OCR workflow and limited by low-level primitive functions, but offers a configurable standalone solution that is ready for immediate use. Our pipeline is not exclusively specialized in the post-processing of scanned documents, but is designed as a general image enhancement pipeline that can be used for a variety of applications. In addition, our licensing model (MIT-0 or CeCILL-C) ensures its use in research and commercial applications. By combining minimal dependencies, a modular architecture, portability through C++ and OpenMP, and a commercially usable license, our pipeline fills a gap between large-scale, general-purpose image processing libraries and highly specialized tools for document enhancement.

1.4 Outline

In section 2, we introduce the pipeline that we will implement and optimize. In section 3, we describe the optimization techniques used. These are evaluated in section 4, where we compare the performance of the pipeline under different settings to the baseline implementation. Finally, we summarize the results in section 5 and provide an outlook on possible future work.

2 Pipeline

The pipeline is designed to be flexible and modular, enabling users to activate, deactivate and configure each step as required. Each step is designed to run independently of the others, except for the conversion to grayscale step 1, which forms the basis for subsequent operations. The steps in the pipeline are as follows:

(1) Grayscale Conversion: Conversion of the input image into a grayscale image to simplify processing and reduce computing time.
(2) Deskewing: Correction of skewed images to improve readability and OCR accuracy.
(3) Contrast Enhancement: Improvement of the contrast in the image to facilitate the distinction between text and background.
(4) Denoising: Removal of noise and artifacts in the image to improve clarity.
(5) Binarization: Segmentation of the grayscale image into a binary image consisting of foreground and background.
(6) Morphological Operations: Application of operations such as erosion and dilation to improve the structure of the text and remove small errors.
(7) Color preservation: The color information of the original image is preserved at the locations of the foreground (especially relevant for historical documents to preserve their original coloration).

Several algorithms are implemented in the steps 4, 5 and 6. As these have different effects on the result, users can select the algorithm best suited to their application. For instance, Bataineh's adaptive binarisation method with automatic window adjustment [1] is helpful when binarising historical, handwritten documents with uneven writing and lighting, whereas Otsu's method [7] is a good choice for binarising typewritten documents with uniform font and lighting. Standard configurations are provided for each step, enabling the pipeline to be used straight away. These configurations deliver good results for most use cases and are based on recommendations from the authors. This means that the pipeline can be used immediately, without the need for additional information. However, users can also customise the configurations to optimise the pipeline and improve their specific results. The pipeline methods 1 - 7 can be used individually by users, as a library in their own program for their specific use case.

We provide a command-line application that can be used independently of a program to access the pipeline. This enables the pipeline to be configured and used directly via the command line.

3 Pipeline Optimizations

3.1 Parallelization and Concurrency

Um auf multicore-Prozessoren eine bessere Performance zu erziehelen, werden die einzelnen Schritte mit OpenMP parallelisiert. Es werden hauptsaechlich *parallel for collapse(3)* Umgebungen fuer die Pixelverarbeitungsschleifen verwendet. Es werden aber auch andere Parallelisierungsstrategien wie die Reduzierungs clauses *reduction(min)* und *recuction(max)* bei eineigen Binarisierungsalgorithmen verwendet. Es wird darauf geachtet, dass die Parallelisierung so implementiert wird, dass keine Race-Conditions oder False-Sharing auftreten. So werden beispielsweise bei der Kontrastverbesserung thread-lokale Buffer verwendet, um False-Sharing und locking zu vermeiden. Bei der Deskewing-Logik wird die Erkennnung und die Anwendung der Korrektur getrennt. Der Winkel wird einmal erkannt und dann auf das Graustufen- und Farbbild angewendet, um unnötige Berechnungen zu vermeiden. Intuitiv wurde dies dopplet ausgefuehrt.

3.2 Memory Management and Allocation

3.3 Algorithmic Complexity Reduction

3.4 Loop level Optimizations

3.5 Architectial Refactoring

4 Experiments

5 Conclusions

References

[1] Bilal Bataineh, Siti Abdullah, Khairuddin Omar, and Mohammad Faidzul Nasrudin. 2011. Adaptive Thresholding Methods for Documents Image Binarization, Vol. 6718. 230–239. doi:10.1007/978-3-642-21587-2_25
[2] Dan Bloomberg. 2001, 2024. Leptonica Image Processing Library. http://leptonica.org
[3] Gary Bradski. 2000. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25, 11 (2000), 120–123.
[4] ImageMagick Studio LLC. 2024. *ImageMagick*. https://imagemagick.org
[5] Itseez. 2015. Open Source Computer Vision Library. https://github.com/itseez/opencv.
[6] Jens Ohrnberger. 2022. unpaper: Post-processing tool for scanned sheets. https://github.com/unpaper/unpaper
[7] Nobuyuki Otsu. 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics* 9, 1 (1979), 62–66. doi:10.1109/TSMC.1979.4310076
[8] R. Smith. 2007. An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2. 629–633. doi:10.1109/ICDAR.2007.4376991

[9] Jean-Philippe Tarel. 2024. CImg: A Simple C++ Toolkit for Image Processing. https://cimg.eu. Version 3.x.