Libcrn, an Open-Source Document Image Processing Library

Yann Leydier, Jean Duong, Stéphane Brès, Véronique Eglin, Frank Lebourgeois, Martial Tola

libcrn, an Open-Source Document Image Processing Library

Yann LEYDIER, Jean DUONG
CoReNum
F-69006, France
yann@leydier.info, jean.duong@corenum.com

Stéphane BRÈS, Véronique ÉGLIN, Frank LeBOURGEOIS, Martial TOLA
Université de Lyon, CNRS
INSA-Lyon, LIRIS, UMR5205,
F-69621, France
firstname.lastname@liris-cnrs.fr

***Abstract*—In this paper we introduce *libcrn*, a multiplatform open-source document image processing library aimed at researchers and companies. It is written in C++11 and has a non-contaminating license that makes it available for use in any project without legal constraints.**

The features include low-level image processing (color format conversion, binarization, convolution, PDE…), document images specific tools (connected components extraction, recursive block description, PDF export…), maths (matrix arithmetics, linear algebra, GMMs, equation solvers…), classification and clustering (kNN, k-means, HMMs…).

The API is comprehensively documented and *libcrn*'s architecture follows modern C++ guidelines to facilitate the handling of the library and enforce its safe usage.

A sample OCR, which is only 30 lines long, is described to illustrate *libcrn*'s scope of possibilities.

***Keywords*-document image processing; open source; library; toolbox**

I. Introduction

Implementing the most basic document image processing algorithm may be a good exercise for students. However, when focusing on high-level processing chains or complex methods, researchers as well as manufacturers rely on toolboxes and software libraries for low-level tasks.

Commercial software libraries are available (Intel IPP, Lead Tools…) and used in industry. They generally offer the basic tools needed to perform simple image manipulation and are well suited for non-specialist engineers. Specialists often prefer Open Source libraries as it is possible to check the details of the algorithms and modify and fine tune them when needed.

The most widely used image processing library is OpenCV[1]. Whereas is contains a great amount of algorithms, it is not originally meant for document images and lacks features and services that make it inconvenient.

Qgar[2] [1] is an Open Source document image processing library created in the early 2000s. It features the most elementary tools to create document analysis software but also lacks some crucial features such as RGB images. Although Qgar can be easily extended, its development has been stalled since 2008.

[1]http://opencv.org/

[2]http://www.qgar.org/

We introduce *libcrn*[3], licensed in LGPL (a non-contaminating Open Source license). Its aim is to allow both researchers and engineers to implement document image processing chains and algorithms. *libcrn* is available for Windows (Visual C++ 2015), Linux, MacOS and Android. It is written in C++11 using the "modern" guidelines issued by the C++ committee so that users can easily and safely use it (e.g.: no memory management is required from the users and no leak can happen). We implemented many image processing algorithm but also the mathematical tools needed to process the data that can be extracted from images.

II. Features

A. General points

In order to facilitate the storage of data, most of the objects in *libcrn* can be serialized in XML files. Multi-platform utilities are packaged so that no overwork weights on the user to make applications run on any OS (e.g.: automatic file path format conversion, file manipulation, character set conversion, dynamically loaded modules…).

B. Image

1) Formats: We provide built-in support for numerous pixel types (see tab. I). Any other type of pixel format (such as matrices!) is supported as long as it implements the basic arithmetic operators.

Category	Subcategory	Types
Color	RGB-based	RGB, HSV, YUV (television)
	Perception-based	XYZ, L*a*b*, L*u*v*
Monochromatic	Grayscale	double, int, byte
	Binary	bool
Other	2D vectors	Cartesian and polar coords
	Angles	radian, degree and byte
	Custom	any type with arithmetic ops

Table I
Pixel types supported by *libcrn*.

All the color types are trivially convertible and multiple binarization methods are offered: Niblack, Sauvola [2], local min or max, Fisher, entropy and Otsu [3].

[3]https://github.com/Liris-Pleiad/libcrn

2) Algorithms: Classical basic transformation algorithms are provided, such as rotate by shear, mathematical morphology, convolution, distance transform...

Feature extractors can be combined to allow the comparison of shapes. Sample feature extractors are provided including profile projections and gradients histograms (see fig. 1).

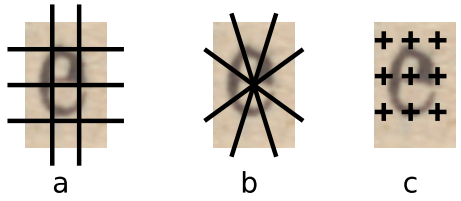

Figure 1. a,b) Angle histograms computed inside zones (rectangular or radial). c) 9 angle histograms computed with all pixels, weighted by their distance to automatically centered anchors.

Shapes can also be compared directly with an FFT-based cross-correlation or gradient matching. The gradient matching is a double inclusion measure. A mask of significant gradients is automatically computed and the angles inside a shape's mask are compared to the angles on the other shape. If the location of the pixel in the other shape is out of its dilated mask, then a penalty is applied. The sum of both inclusion measures is the dissimilarity between the two shapes (see fig. 2).

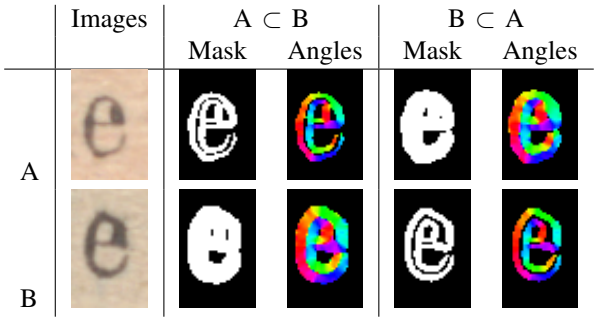

Figure 2. Two-ways inclusion measurement of two shapes.

3) PDE: Gradients can be obtained from both grayscale and color [4] images by different means (half derivatives, Gaussian kernel...) and can be isotropically diffused. From the gradients, a large number of transforms and descriptors are computed (see fig. 3):

- first and second derivatives along Cartesian axes, isophotes and flowlines;
- edge and corner;
- Hessian eigenvalues κ_1 and κ_2, Hessian corner;
- flowline, isophote, Gaussian and gradient curvature;
- divergence, Laplacian.

4) Document image processing: libcrn provides a way to describe a document's layout structure with nested lists of

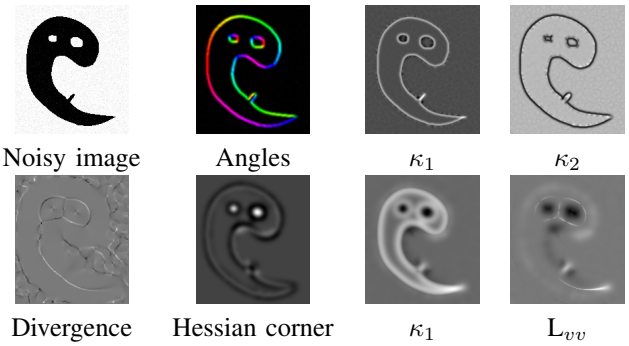

Figure 3. Sample of differential features ($\sigma = 1.5$ on the first line, $\sigma = 4.5$ on the second).

rectangular blocks. Each block refers to a part of the original image that is instantiated only when needed. This structure is of course serializable.

The fast connected components extraction [5] directly feeds the aforementioned block structure.

Simple line and word segmentation algorithms are implemented and more can be easily added to feed the block layout structure.

Other useful algorithms specific to document images such as stroke thickness estimation, text line height estimation and skew estimation are offered.

C. Maths

1) Algebra: libcrn's math toolbox includes matrices of integers, reals and complex numbers and provide all arithmetics. Two system solvers (Cramer and Gauss-Jordan) are included.

Fundamental linear algebra methods such as diagonalization and inversion are implemented. Effective computation procedures will be added[4] for tensor computation [6], [7]: Higher order structure tensors [8] are proposed as extensions of classical structure tensor used in image processing [9]. Since they are tricky to compare (e.g. for clustering, feature extraction, etc.), their glyphs are to be used for analysis [10]. Thus, Zernike moments have been implemented and should be soon added to image analysis toolbox.

2) Data analysis: Multidimensional data may be processed using Principal Component Analysis (PCA). Furthermore, Gaussian mixture models are also supported for both uni and multivariate samples. In particular, the Expectation-Maximization fitting procedure (EM) is implemented.

3) Interpolation and regression: The math module contains tools for linear interpolation, cubic splines interpolation and polynomial regression.

4) Geometry: libcrn provides angle arithmetics and tools such as circular mean and variance, but also circular histograms utilities: circular earth mover's distance [11], kurtosis, trigonometric moments...

[4]Code is already available on demand. Full integration will be done for the next release of *libcrn*

5) Signal processing: FFT can be applied on complex matrices and vectors.

D. Pattern analysis

1) Clustering: libcrn provides clustering algorithms for both vectorial and metric data:

- k-means
- k-medoids (PAM and fast [12])
- Outliers: LOF and LoOP [13]
- Spectral Clustering (all formulas from [14]–[17])
- Affinity Propagation [18]

2) Classification: Many data classification problems can be addressed, using a wide variety of tools from a highly generic kNN implementation to discrete or Gaussian semi-continuous HMMs.

3) Other: Other "combinatorics oriented" algorithms are also available in *libcrn*, such as Kuhn and Munkres' "Hungarian" bipartition algorithm, the A* path finder or the disjoint set forest distribution.

E. GUI

Custom widgets are provided to create applications with Gtkmm – Gtk's C++ wrapper – versions 2 and 3. This include displaying and browsing the block structure of an image, image overlays and the automated generation of configuration panels. A Qt widget library will be available soon.

A demonstration application is included in *libcrn*. It allows to quickly test some features of the library on any image (see fig. 4). This tool is very handy to run simple algorithms over a given image without writing a new program.

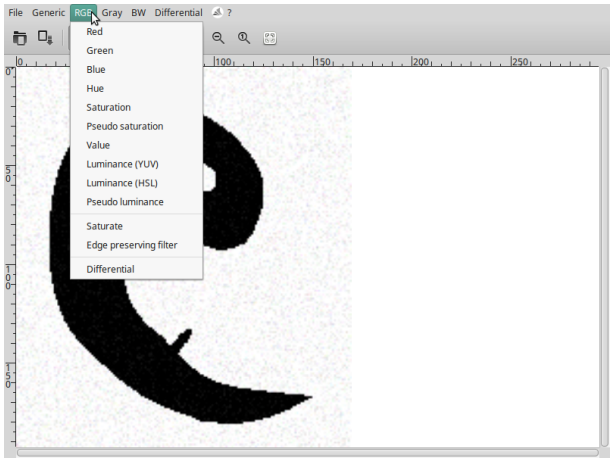

Figure 4. Titus, *libcrn*'s quick testing tool.

III. 30 lines for an OCR

To illustrate *libcrn*'s ease of use, we present a very simple OCR engine that is only 30 lines long. It works on a medieval manuscript excerpt written in capital letters with no spacing between words (see fig. 5). An occurrence of each letter in the alphabet was manually extracted and put in a folder named `data`, where each file name corresponds to the image's label. The source code is displayed in fig. 6.

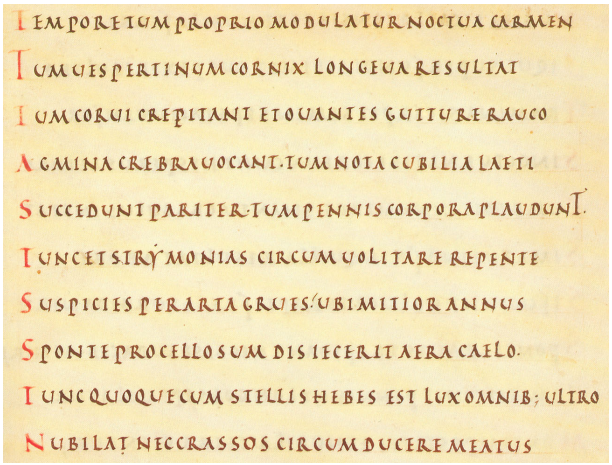

Figure 5. Sample medieval manuscript.

The first step (fig. 6.1) is to create a feature extractor that will be used to compare each character to the database. To do that, we use a `FeatureSet`, which can contain multiple elementary feature extractors. The `FeatureSet` will concatenate the feature vectors extracted by each `FeatureExtractor`. For simplicity, we use the four profile projections and the horizontal and vertical black pixels projections.

In step 2, we open each pre-labelled character image. As it is often impossible to know whether an image file contains RGB, grayscale or binary pixels, we directly store the image in a `Block` object. A `Block` automatically converts the input image to the format desired by the user (the default grayscaling and binarization methods can be changed programmatically at anytime). Is also contains named lists of sub-blocks that will be used later. The extracted features are stored in a list of shared pointers[5] to the base class `Object`.

The line segmentation is performed in 6.3. The document image is opened in the same way as the database images. We create a temporary `BlockTreeExtractor` that will extract the text lines and store them in a sub-block list named `Lines`.

Just before we actually extract the characters (fig. 6.4), we get an estimation of the mean stroke width. This will help us filter the noise. Connected components are extracted within each line. After that, each `Block` in the `Lines` sub-block list will contain a sub-block list named `Characters`. The recursive sub-block lists can be used for more complex

[5]Shared pointers are memory management objects that deletes pointers when they are not referenced anymore.

purposes and can even match an XML Alto's structure. Finally, connected components smaller than the mean stroke width are removed and the remaining ones are sorted from left to right.

The 5th and last step is the actual recognition. Each `Block` remaining in the `Characters` sub-sub-list represents a letter in the text. Its feature vector is extracted using the same `FeatureSet` as the database. We search its nearest neighbor in the database a retrieve a class number that can be used to compute the character's transcription.

Now that our homemade OCR is fully described, we shall not discuss its performance: profile projections are not known to be the best features for medieval manuscript recognition! The purpose here was to illustrate the easiness of designing applications with *libcrn*. Variety of document analysis problems can be addressed, not restricted to ancient scripts: Printed pages or manuscripts may be considered. Historic or business documents may be processed. Even more borderline tasks are feasible (e.g.: text extraction from scenes, plate recognition, mobile applications, *etc.*).

IV. Conclusion

In this paper we introduced *libcrn*, a multiplatform (non-contaminating) open-source document image processing library written in C++11 and aimed at researchers and companies available for Windows (Visual C++ 2015), Linux, MacOS and Android. Its API is comprehensively documented and *libcrn*'s architecture follows modern C++ guidelines to facilitate the handling of the library and enforce its safe usage (e.g.: no memory management is required from the users and no leak can happen).

libcrn includes low-level image processing (expandable pixel formats, binarization, PDE...), document images specific tools (connected components extraction, recursive block description...), maths helpers (algebra, data analysis, geometry, signal processing...) and pattern analysis algorithms (classification, clustering...).

We described a short code example that provides OCR capacities in only 30 lines. This illustrated the use of feature extractors, segmentation providers and classification tools. Many other applications may be designed for research or industrial needs. *libcrn* has been useful in several projects and remains constantly improving.

References

[1] K. Tombre, C. Ah-Soon, P. Dosch, A. Habed, and G. Masini, "Stable, robust and off-the-shelf methods for graphics recognition," in *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, vol. 1. IEEE, 1998, pp. 406–408.

[2] J. Sauvola, T. Seppänen, S. Haapakoski, and M. Pietikäinen, "Adaptive document binarization," in *International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1, Ulm, Germany, 1997, pp. 147–152.

[3] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[4] F. LeBourgeois, "Content based image retrieval using gradient color fields," in *International Conference on Pattern Recognition (ICPR)*, Barcelona, Spain, 2000, pp. 1027–1030.

[5] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, no. 9, pp. 1977 – 1987, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320308004573

[6] E. Nelson, *Tensor analysis*. Princeton University Press, 1967.

[7] J. G. Simmonds, *A brief on tensor analysis*. Springer-Verlag, 1994.

[8] T. Schultz, J. Weickert, and H.-P. Seidel, "A higher-order structure tensor," July 2007.

[9] S. D. Zenzo, "A note on the gradient of a multi-image," *Computer Vision, Graphics, and Image Processing*, vol. 33, pp. 116–125, 1986.

[10] T. schultz and G. Kindlmann, "A maximum enhencing higher-order tensor glyph," in *Eurographics/IEEE-VGTC Symposium on Visualization*, vol. 29, no. 3, 2010.

[11] J. Rabin, J. Delon, and Y. Gou, "Circular earth mover's distance for the comparison of local features," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE, 2008, pp. 1–4.

[12] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, pp. 3336 – 3341, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740800081X

[13] H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Loop: Local outlier probabilities," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09. New York, NY, USA: ACM, 2009, pp. 1649–1652. [Online]. Available: http://doi.acm.org/10.1145/1645953.1646195

[14] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.

[15] M. Meila and J. Shi, "A random walks view of spectral segmentation," 2001.

[16] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, Aug. 2000. [Online]. Available: http://dx.doi.org/10.1109/34.868688

[17] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering," in *Advances in neural information processing systems*, 2004, pp. 1601–1608.

[18] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

```
// 1. Feature extractor
auto featureExtractor = crn::FeatureSet{};
featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProfile>
  (crn::Direction::LEFT | crn::Direction::RIGHT | crn::Direction::TOP |
   crn::Direction::BOTTOM, 10, 100));
featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProjection>
  (crn::Orientation::HORIZONTAL | crn::Orientation::VERTICAL, 10, 100));
// 2. Database creation
auto database = std::vector<crn::SObject>{};
for (auto c = 'A'; c <= 'Z'; ++c)
{
        const auto charFileName = "data"_p / c + ".png"_p;
        auto charblock = crn::Block::New(crn::NewImageFromFile(charFileName));
        database.push_back(featureExtractor.Extract(*charblock));
}
// 3. Line segmentation
auto pageblock = crn::Block::New(crn::NewImageFromFile(imageFileName));
crn::BlockTreeExtractorTextLinesFromProjection{U"Lines"}.Extract(*pageblock);
// 4. Character segmentation
const auto sw = crn::StrokesWidth(*pageblock->GetGray());
auto s = crn::String{};
for (auto nline = size_t{0}; nline < pageblock->GetNbChildren(U"Lines"); ++nline)
{
        auto line = pageblock->GetChild(U"Lines", nline);
        line->ExtractCC(U"Characters");
        line->FilterMinOr(U"Characters", sw, sw);
        line->SortTree(U"Characters", crn::Direction::LEFT);
        for (auto nchar = size_t{0}; nchar < line->GetNbChildren(U"Characters");
          ++nchar)
        {
                auto character = line->GetChild(U"Characters", nchar);
                // 5. Recognition
                auto features = featureExtractor.Extract(*character);
                auto res = crn::BasicClassify::NearestNeighbor(features,
                  database.begin(), database.end());
                s += char32_t(U'A' + res.class_id);
        }
        s += U'\n';
}
CRNVerbose(s); // display the result
```

Figure 6. Minimalistic code for an OCR