libcrn

Javier Izquierdo Vera y Miguel Medina Ballesteros

Data and Document Ming, Dipartimento di Ingegneria dell'Informazione

Università degli Studi di Firenze: Scuola di Ingegneria

Firenze (FI), Italia

Email: javierizquierdovera@gmail.com y maximetinu@gmail.com

***Abstract*—Di seguite è esposta la libreria c++ *libcrn*, concentrata nel Layout Analysis ed Optical Character Recognition dei immagini di documenti. Nei paragrafi che seguono si illustra la installazione della libreria, le sue caratteristiche, le classe e le funzione principale per fare un primo programma OCR e due esempi che abbiamo fatto per abituarci alla libreria ed mostrare le sue possibilità.**

I. INTRODUZIONE

libcrn è una libreria di processamento di immagini di documenti scritta in C++ (C++98 e C++11) per Linux, Windows, Mac OS X e Android. Deve essere vista come una raccolta di funzioni e classi preparati per creare software come OCRs e strumenti di analisi della struttura dei documenti con facilità. È una libreria orientata a quelli ingegneri e investigatori che cerchino di implementare algoritmi di riconoscimento di documenti nelle immagini, perciò *libcrn* confida nel Open Source e la sua licencia è LGPL[1], ciò permette la libera distribuzione e modificazione del codice.

Il primo commit del repositorio è nel Magio di 2009 e viene essendo aggiornata più o meno frequentemente. La versione più recente è sul Github della libreria, dove si trovano anche istruzioni d'installazione per ogni sistema. Pure è disponibile una documentazione online generata per *doxygen* sulla web dell'autore [3], ma è poco dettagliata, ogni classe dispone soltanto di una corta descrizione di una linea. Per ultimo ci sono qualchi piccoli articoli in una wiki sul repositorio di Sourceforge [4]. Visto che non è una documentazione ampia, la curva di apprendimento è abbastanza ripida all'inizio; e, pertanto, si deve ricorrere all'esempi che sono nella cartella "demos" della repository, con cui sono illustrate le principale classe della libreria.

II. INSTALLAZIONE

Sul Github di *libcrn* si trova una guida d'installazione della libreria e le dipendenze. In questa guida viene spiegato come fare una build della libreria per lanciare qualche esempi. Noi abbiamo seguito la guida d'installazione su Linux. Comunque, su questa guida non c'è un esempio di come compilare il nostro codice con questa libreria, pertanto da qui in poi dipende dal nostro conoscimento del compilatore *g++* e del Sistema Operativo Linux (in questo caso). Tutto quello che viene spiegato a continuazione non è sulla guida e sarebbe interessante aggiungerlo per quegli utenti novellini con *g++* e Linux.

Pur avendo fatto il *make*, la libreria non è ancora installata veramente, soltanto abbiamo fatto una build. È vero che già

abbiamo tutti i codici nella cartella di *src* e la cartella *bin* e che sarebbe possibile compilare collegando tutto manualmente, ma fare così con una libreria sarebbe una barbaria perché si può installare per facilitare i comandi [5][6], pertanto non si dovrebbe fare mai così con una libreria. Per installarla si deve lanciare il comando `make install`. Così viene creata una cartella chiamata *install_3.9.5*. Dentro di questa cartella c'è un'altra cartella chiamata *lib* in cui c'è un file chiamato *libcrn.so*. In seguito si spiegano i opzioni per compilare il nostro primo progetto:

```
g++ file.cpp path/to/libcrn.so
```

Così non c'è bisogno di usare *sudo* visto che l'installazione e locale e si fa dentro del nostro *home*. Evidentemente, possiamo decidere dove installarla.

```
g++ programma.cpp -lcrn
```

Così si può compilare da qualche parte della macchina perché la libreria verrà trovata e collegata automaticamente. Per questo è bisogno di avere la libreria installata nel sistema operativo, in */usr/local/lib* (il file *libcrn.so*) e */usr/local/include* (tutti i files che ha dentro la cartella *include*). Per farlo funzionare, prima si deve lanciare il comando `ldconfig` per trovare i nuovi files. Questa opzione si può fare direttamente copiando ed incolliando il file *libcrn.so* ed il interno della cartella *includes* a le cartelle di */usr/local/* di cui abbiamo parlato prima. A */usr/local/* si devono mettere il software installato manualmente per noi. Il software installato per i gestori di pacchetti è a */usr/*, pertanto noi non dobbiamo installare la libreria di là [6].

A titolo di confronto, la complessità di installazione di *libcrn* si può comparare con l'installazione della libreria *tesseract-ocr* [7], un'altra libreria *c++* che ha lo stesso proposito di *libcrn* ma è molto più famosa. Nei commandi prossimi vengono illustrati i comandi necessari per installare la libreria ed preparare il sistema e *g++* per dopo potere compilare direttamente usando `g++ program.cpp -ltesseract` [8].

```
 cd tesseract-ocr
./autogen.sh
./configure
make
sudo make install
sudo make install-langs
sudo ldconfig
```

III. CARATTERISTICHE

A. Formato

C'è supporto per diversi formati di pixel: RGB, HSV, YUV, coordinate cartesiane, coordinate polare, gradi, bytes, etc. Ma internamente si gestisce in XML rendendo più facile il stoccaggio e la portabilità multipiattaforma. [2]

B. Trattamento delle immagini

libcrn permette di immagazzinare le immagini sul piano interno come blocchi, facendo possibili suddividere successivamente questi blocchi in altri, in modo da ogni blocco corrisponde a una parte dell'immagine, seguendo una struttura ad albero. È possibile fare riferimento ai blocchi che costituiscono ogni blocco e trattarli individualmente oppure di gruppo. In questo modo,

sarebbe possibile trarre i blocchi di un documento relative alle linee, e dopo i blocchi di ogni linea relative di ogni carattere (mediante un BlockTreeExtractor), può riconoscere il testo se si confronta con immagini di ogni carattere precedentemente immagazzinato in una database (FeatureSet). Non è un metodo troppo effettivo di fare un OCR, ma è possibile ed è semplice. [2, 3]

Ogni blocco è composto di un buffer RGB, altro di grigi, e altro con il bianco e nero. Questi si possono trarre, eliminare o sostituire, permettendo così cambiare il colore dell'immagine. [3]

IV. ANALISI DI IMMAGINE DI DOCUMENTI: PROGRAMMING WORKFLOW

Al momento di sviluppare una prima applicazione, come abbiamo detto nella introduzione, la curva di apprendimento è ripida ed è necessario capire prima i esempi. Nella presente sezione si intende raccogliere tutto quello che li esempi trattano di illustrare, come vengono essendo, per esempio, le classe principale della libreria, quelle più usate, ecc; con la finalità di facilitare il inizio a chi cominciano a usare la libreria e leggere la vasta documentazione.

1. Carica di immagini files

Ci sono due classe per la gestione di files.

- La classe Path, che ripresenta il percorso completo del file e ci permette fare astrazione del formato di ogni sistema (Unix, Windows, URI…) [9]

- E la classe Image, che ripresenta a un file *.jpg* o *.png,* quelli più conosciuti. Un oggetto della classe Image viene creato con un oggetto Path come argomento del costruttore. La classe Image ha una funzionalità basica, soltanto può misurare la risoluzione e scalarla [10].

La classe Image ancora non è pronta per essere computata por *libcrn*. A tal fine, si deve ricorrere alla classe principale per trattamento di immagini di *libcrn*: la classe Block [11].

2. Immagini trattabili per la libreria

Per fare un'immagine di tipo Image un oggetto trattabile per la libreria deve essere convertita alla classe Block mediante il costruttore, visto che Block può ricevere come argomento un oggetto della classe Image.

La classe Block ripresenta una parte, oppure la totalità, di una immagine. Block anche fa parte di una gerarchia padre-figlio, in modo da dividere un'immagine di un paragrafo di un documento in righe [11]. Ha molte più funzione, tra le più importante:

- Creare un'immagine soltanto RGB, B&W, Gray o Gradient, oppure modificare o eliminare questi buffers .

- Computi e lavori con la gerarchia: navigare tra fili, mescolarli, ecc.

- Filtrare fili, per esempio secondo le dimensione, per ignorare falsi positivi come rifiuti al momento di cercare caratteri manoscritti. Questa tecnica viene fatta accanto alle funzione StrokeWidth e StrokeHeigth della libreria.

Un Block è possibile salvarlo [12]. Si può serializzare l'oggetto su un file XML per

caricarlo dopo o comunicare la libreria con altri programmi.

Secondo la documentazione, la classe Block normalmente rappresenterà un'immagine di una pagina di un documento

3. Analisi di struttura di documenti

Per analizzare la struttura di un documento, *libcrn* ci offre una forma veloce ed efficace di farlo con qualche classe già preparate.

Per esempio, libcrn ha una classe specifica per estrarre le righe di un blocco, è BlockTreeExtractorTextLinesFromProjection [18], Così. un blocco viene diviso tra altri nuovi blocchi che saranno, in questo caso, le righe. Anche possiamo sviluppare nostri BlockTreeExtractor [17], implementando l'Interfaccia (come anche fa questa classe di cui parliamo).

A questo punto, anche abbiamo altre forme di estrarre struttura. C'e anche un BlockTreeExtractor per parole, e la classe Block anche implementa la funzione ExtractCC per estrarre componenti collegati, che potrebbero essere, per esempio, caratteri; ma soltanto nel caso specifico di essere lettere maiuscole bene spaziate [19].

4. Estrazione di caratteristiche e comparazione con dizionario

Pur avendo *built-in* estrattori di righe, parole e componenti collegati, *libcrn* non offre un algoritmo integrato per il riconoscimento di caratteri (OCR) perché esistono tantissime possibilità.

Programmare un algoritmo OCR non è facile. Dipende dal tipo di tipografia, della lingua, della scrittura, del testo che sia scritto a mano oppure carattere digitale [19]. Per esempio, non è lo stesso riconoscere una tipografia a spaziatura fissa di questa che se legge in questo testo. OCR è una ciencia in costante investigazione e pertanto non esiste una funzione di questa libreria che risolva il problema.

Comunque, *libcrn* ci offre la possibilità di programmare i nostri *Feature Extractors* [13] e *Feature Sets* [14], in un modo che si possa individualizzare ed identificare ogni carattere usando la funzione *Extract* della classe *Feature Extractor* [13]. Così vengono estratte le caratteristiche che dopo si usano per classificare il carattere.

Anche è opportuno menzionare che *libcrn* ha qualche *Feature Sets* integrati, come *Profile* [14] o *Projection* [15], ma sono lontani del *state-of-the-art* ed in realtà appena riconoscono i caratteri.

V. ESEMPIO GENERALE

In questo source se fa un esempio delle funzioni di manipolazione di immagini.

È possibile trasformare un'immagine a bianco e nero, scala di grigi, e RGB. Pure è possibile ottenere le informazioni dell'immagine ed scalare.

Source 1.

VI. ESEMPIO OCR TWITTER

Tentativo di fare un esempio di OCR con tweets. Lo scopo era riconoscere il testo di un'immage di un tweet. [4]

Alla fine non è stato possibile perchè libcrn non è stato in grado di riconoscere, per esempio, la differenza tra "S" e "s".

Se proviamo a identificare un tweet con il testo in maiuscole funziona bene (figura 1), ma se proviamo con carattere in

minuscole e maiuscole si sbaglia (figura 2).

Come si può osservare nella figura 3 se si rotti 5 gradi non identifica un carattere di "*TEST*", e se si 30 gradi non identifica alcun carattere (figura 4).

Source 2.

VII. ESEMPIO OCR STREET NAME

Si tratta di un esempio, molto simile all'anteriore, per provare a identificare un'immage con bassa risoluzione del nome di una via. Il risultato non è quello desiderato (figura 5).

VIII. CONCLUSIONE

libcrn è una libreria utile per manipolare immagini in modo semplici ed è possibile fare un OCX semplici, ma per questo non è molto efficace. La fonte che si vuole identificare deve avere caratteri chiari e molto distinti tra se, suprattutto deve avere una differenza notabile tra maiuscole e minuscole, oppure essere unicamente per identificare uno dei due. Se il testo rotta un po o si presenta qualcosa sconosciuta non funziona bene.

IX. RIFERIMENTI

[1] LGNU Lesser General Public License - https://www.gnu.org/licenses/lgpl-3.0.en.html

[2] Yann LEYDIER, Jean DUONG "*libcrn*, an Open-Source Document Image Processing Library" in 2016 15th International Conference on Frontiers in Handwriting Recognition pp. 212-213

[3] libcrn documentation - blocks, https://liris.cnrs.fr/pleiad/doc/nightly/de/dc9/classcrn_1_1_block.html#a8e35c9089e806962b83bca4e5e288142

[4] libcrn demo - "ocr4dummies", https://github.com/Liris-Pleiad/libcrn/blob/master/src/demos/ocr4dummies.cpp

[5] Shared libraries with linux gcc, http://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html

[6] The Linux Documentation Project - Shared Libraries, http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

[7] Tesseract OCR, https://github.com/tesseract-ocr/

[8] Compiling with tesseract-ocr, https://github.com/tesseract-ocr/tesseract/wiki/Compiling#miscellaneous

[9] Path class reference https://liris.cnrs.fr/pleiad/doc/nightly/de/d1c/classcrn_1_1_path.html

[10] Image class reference, https://liris.cnrs.fr/pleiad/doc/nightly/d5/ddf/classcrn_1_1_image.html

[11] Block class reference, https://liris.cnrs.fr/pleiad/doc/nightly/de/dc9/classcrn_1_1_block.html

[12] Savable class reference, https://liris.cnrs.fr/pleiad/doc/nightly/dd/dae/classcrn_1_1_savable.html

[13] Feature Extractor class reference, https://liris.cnrs.fr/pleiad/doc/nightly/d9/d59/classcrn_1_1_feature_extractor.html

[14] Feature Set class reference, https://liris.cnrs.fr/pleiad/doc/nightly/d9/dbb/classcrn_1_1_feature_set.html

[15] Feature Extractor Profile reference, https://liris.cnrs.fr/pleiad/doc/nightly/de/df7/classcrn_1_1_feature_extractor_profile.html

[16] Feature Extractor Projection reference, https://liris.cnrs.fr/pleiad/doc/nightly/dd/d8d/classcrn_1_1_feature_extractor_projection.html

[17] Block Tree Extractor class reference, https://liris.cnrs.fr/pleiad/doc/nightly/dd/dc0/classcrn_1_1_block_tree_extractor.html

[18] Block Tree Extractor Text Lines From Projection class reference, https://liris.cnrs.fr/pleiad/doc/nightly/d8/d14/classcrn_1_1_block_tree_extractor_text_lines_from_projection.html

[19] OCR Word character segmentation, http://www.how-ocr-works.com/OCR/word-character-segmentation.html

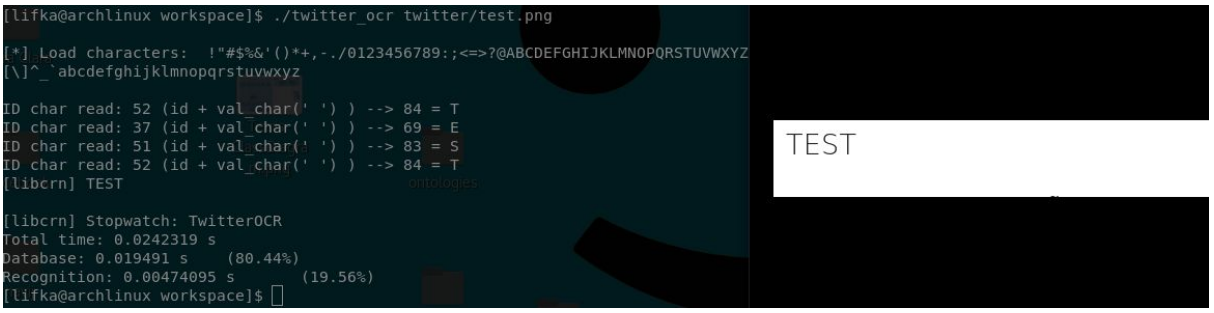

Figura 1: Esempio 1 esecuzione di “OCR for tweets” per il tweet “TEST.

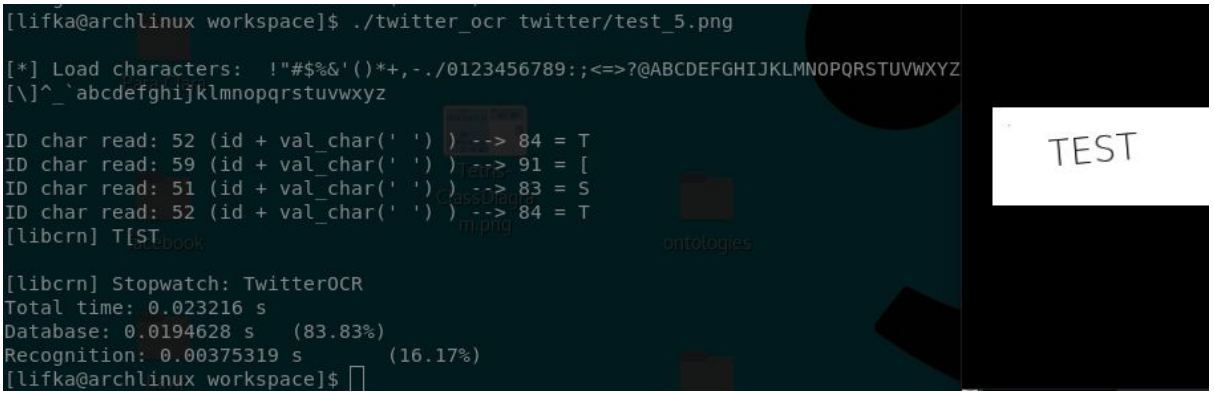

Figura 2: Esempio 1 esecuzione di “OCR for tweets” per il tweet “TEST ruotati 5 gradi.

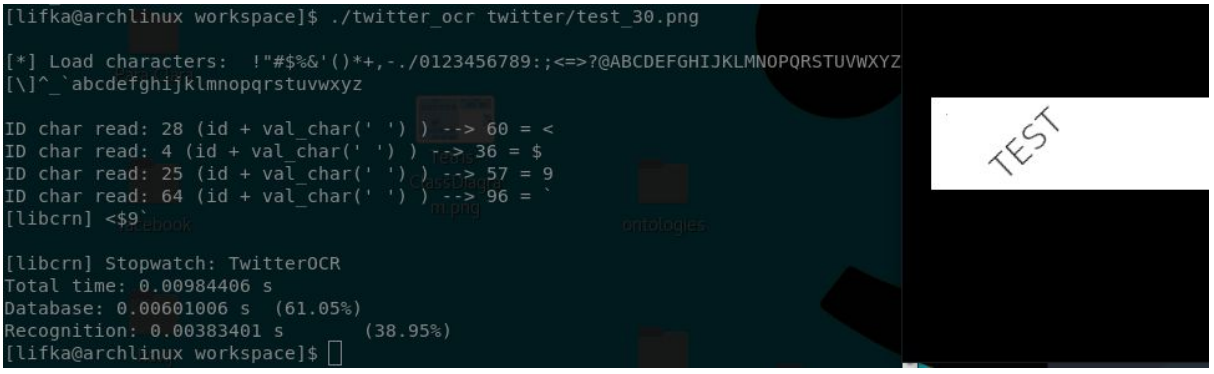

Figura 3: Esempio 1 esecuzione di “OCR for tweets” per il tweet “TEST ruotati 30 gradi.

Figura 4: Esempio 2 esecuzione di “OCR for tweets”.

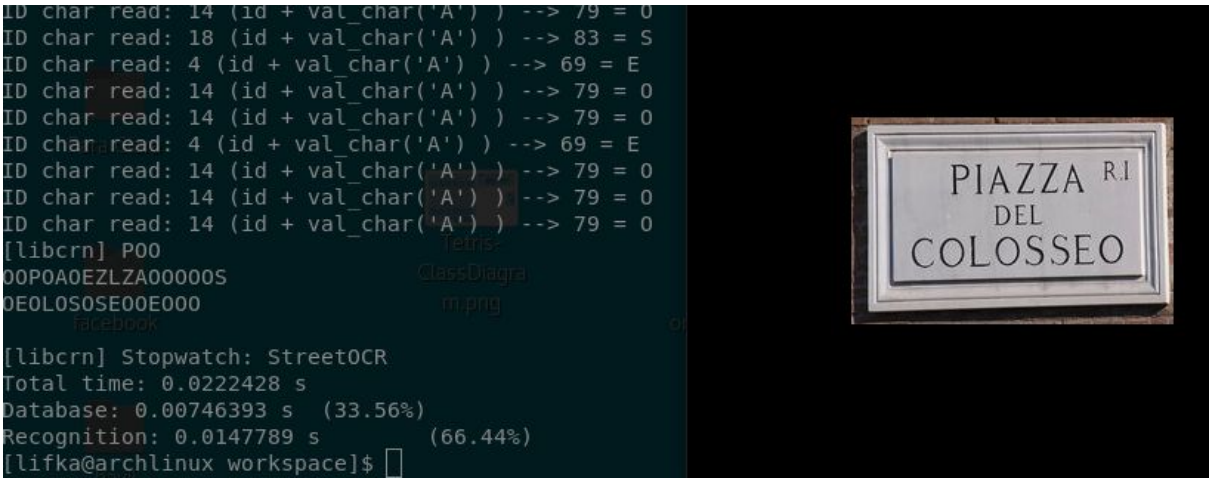

Figura 5: Esempio 1 esecuzione di “OCR street name”.

```
/* Copyright 2017
 *
 *
 * Compile with:
g++ lifka.cpp -lcrn -o lifka
 *
 * \Author:
 *      - Javier Izquierdo Vera
 */

#include <CRNBlock.h>
#include <CRNImage/CRNImageGray.h>
#include <CRNFeature/CRNFeatureSet.h>
#include <CRNFeature/CRNFeatureExtractorProfile.h>
#include <CRNFeature/CRNFeatureExtractorProjection.h>
#include <CRNFeature/CRNBlockTreeExtractorTextLinesFromProjection.h>
#include <CRNAI/CRNBasicClassify.h>
#include <CRNUtils/CRNTimer.h>
#include <CRNIO/CRNIO.h>

#define VERSION "1.0"
#define NAME "lifka"


void savePNG(crn::SImage img, std::string name){

        name.append(".png");
        img->SavePNG(name);
}

void scaleToSize(crn::SImage img, size_t width, size_t height){
        img->ScaleToSize(width, height);
}

bool scale(crn::SImage img, char* img_path, size_t width, size_t height){

        bool error = false;

        std::string name = img_path;
        name.append("_");
        name.append(std::to_string(width));
        name.append("x");
        name.append(std::to_string(height));

        scaleToSize(img, width, height);
```

```
        savePNG(img, name);

        return error;
}

crn::SBlock getBlockFromImg(crn::SImage img){
        return crn::Block::New(img);
}

crn::SImage getImgFromPath(char* img_path){
        return crn::NewImageFromFile(img_path);
}

bool getBuffer(crn::SImage img, int opt, char* img_path){
        bool error = false;

        std::string name = img_path;

        crn::SBlock block = getBlockFromImg(img);

        crn::SImage img_result = crn::SImage{};

        if (opt == 0){
                name.append("_gray");
                img_result = block->GetGray();

        } else if(opt == 1){
                name.append("_RGB");
                img_result = block->GetRGB();

        } else if(opt == 2){
                name.append("_BW");
                img_result = block->GetBW();

        }

        savePNG(img_result, name);

        return error;
}

void usage(char* prog_name){
  std::cout << std::endl << prog_name << " [option] [img]" << std::endl <<
       "Options:" << std::endl <<
       "-h    | --help                        Print this help" << std::endl <<
       "-v    | --version                     Print the script version" << std::endl <<
       "-i    | --information                 Print information about image" << std::endl
<<
       "-GRAY | --grayscale                   Get the gray scale image" << std::endl <<
       "-RGB  | --RGB                         Get the RGB image" << std::endl <<
       "-BW   | --blackwhite                  Get the black and white image" << std::endl
<<
       "-s    | --scale  [width] [height]    Scale image" << std::endl;
}

void option (int argc, char *argv[]){

        bool error = false;
        char* prog_name = argv[0];

        if (argc >= 2){

                char* option = argv[1];

                if (!strcmp(option, "-h") || !strcmp(option,"--help") ) {
```

```
                usage(prog_name);
        } else if (!strcmp(option, "-v") || !strcmp(option,"--version") ) {
                std::cout << NAME << " " << VERSION << std::endl;
        } else if(argc >= 3){

                char* img_path = argv[2];

                crn::SImage img = crn::SImage{};

                try{
                        img = getImgFromPath(img_path);
                } catch (std::exception &ex) {
                        CRNError("Cannot open image");
                        CRNVerbose(crn::String(U" ") + ex.what());
                        error = true;
                }

                if (!error){
                        if (!strcmp(option, "-GRAY") ||
!strcmp(option,"--grayscale") ) {
                                error = getBuffer(img, 0, img_path);
                        } else if (!strcmp(option, "-RGB") ||
!strcmp(option,"--RGB") ) {
                                error = getBuffer(img, 1, img_path);
                        } else if (!strcmp(option, "-BW") ||
!strcmp(option,"--blackwhite") ) {
                                error = getBuffer(img, 2, img_path);
                        } else if (!strcmp(option, "-i") ||
!strcmp(option,"--information") ) {
                                auto img_gray = crn::NewImageGrayFromFile(argv[2]);

                                std::cout << "width: " << img->GetWidth() <<
std::endl
                                 << "height: " << img->GetHeight() << std::endl
                                 << "pixels: " << img->Size() << std::endl
                                 << "strokes width: " << StrokesWidth(*img_gray) <<
std::endl
                                 << "strokes height: " << StrokesHeight(*img_gray)
<< std::endl
                                 << "lines height: " <<
EstimateLinesXHeight(*img_gray) << std::endl;

                        } else if (argc == 5){

                                if (!strcmp(option, "-s") ||
!strcmp(option,"--scale") ) {

                                        unsigned int width = atoi(argv[3]);
                                        unsigned int height = atoi(argv[4]);

                                        error = scale(img, img_path, width, height);
                                } else {
                                        error = true;
                                }

                        } else{
                                error = true;
                        }
                }

        } else {
                error = true;
        }

    } else {
```

```
                error = true;
        }

        if (error){
                std::perror("Error");
                usage(argv[0]);
                exit(-1);
        }

        exit(0);
}

int main(int argc, char *argv[]){
        option(argc, argv);
}
```

Source 1: Esempio manipolazione di immagini.

```
/* Copyright 2017
 *
 * based on: ocr4dummies.cpp
 *
 * A quick OCR engine to recognize letters from tweets.
 *
 * Compile with:
g++ twitter_ocr.cpp -o twitter_ocr -lcrn
*
 * \author
 * Javier Izquierdo Vera - javierizquierdovera@gmail.com
 */

#include <CRNBlock.h>
#include <CRNImage/CRNImageGray.h>
#include <CRNFeature/CRNFeatureSet.h>
#include <CRNFeature/CRNFeatureExtractorProfile.h>
#include <CRNFeature/CRNFeatureExtractorProjection.h>
#include <CRNFeature/CRNBlockTreeExtractorTextLinesFromProjection.h>
#include <CRNAI/CRNBasicClassify.h>
#include <CRNUtils/CRNTimer.h>
#include <CRNIO/CRNIO.h>

using namespace crn::literals;

int main(int argc, char *argv[]){

        std::cout << std::endl;

        // Check argument.
        if (argc < 2){
                printf("Usage: %s <image_name>\n", argv[0]);
                return -1;
        }

        // Verbose
        crn::IO::IsVerbose() = true;
        crn::IO::IsQuiet() = false;

        // Starts the quick stopwatch.
        crn::Timer::Start(U"TwitterOCR");



        /*************************************************************************/
```

```
	/* 1. Database                                                              */
	/****************************************************************************/
	// Create a feature extraction engine.
	auto featureExtractor = crn::FeatureSet{};

	// It will extract the four profiles, reduced each to 10 values in [0..100].
	featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProfile>(
				crn::Direction::LEFT | crn::Direction::RIGHT |
				crn::Direction::TOP | crn::Direction::BOTTOM, 10, 1000));

	// It will also extract the two projections under the same conditions.
	featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProjection>(
				crn::Orientation::HORIZONTAL | crn::Orientation::VERTICAL,
10, 100));

	// Create the database.
	auto database = std::vector<crn::SObject>();

	// For each character from tweet...
	std::cout << "[*] Load characters: ";

	for (auto c = ' '; c <= 'z'; ++c){

		std::cout << c;

		// Open a prototype image stored as "twetter/*.png"

		// Special character:
		auto charFileName = "twitter/fonttest/barra.png"_p;

		if (c != '/'){
			charFileName = "twitter/fonttest"_p / c + ".png"_p;
		}

		auto charimage = crn::SImage{};

		try{
			// Open image
			charimage = crn::NewImageFromFile(charFileName);
		} catch (std::exception &ex) {
			CRNError(U"Cannot open database: ");
			CRNVerbose(crn::String(U" ") + ex.what());
			return -2;
		}


		// Embed the image in a block structure
		auto charblock = crn::Block::New(charimage);

		// Extract the features and store it in the database (vector)
		database.push_back(featureExtractor.Extract(*charblock));
	}

	std::cout << std::endl << std::endl;
	crn::Timer::Split(U"TwitterOCR", U"Database");


	/****************************************************************************/
	/* 2. Document                                                              */
	/****************************************************************************/

	// Open the document image file
	auto imageFileName = crn::Path(argv[1]);
	auto pageimage = crn::SImage{};
```

```
	try {
		pageimage = crn::NewImageFromFile(imageFileName);
	} catch (...) {
		CRNError(U"Cannot open document image");
		return -3;
	}


	/*************************************************************************/
	/* 2.1 Segmentation                                                      */
	/*************************************************************************/

	// Embed the image in a block structure.
	auto pageblock = crn::Block::New(pageimage);

	// Extract text lines
	crn::BlockTreeExtractorTextLinesFromProjection{U"Lines"}.Extract(*pageblock);

	/*************************************************************************/
	/* 2.2 Recognition                                                       */
	/*************************************************************************/
	auto s = crn::String{};

	// For each line...
	for (auto nline = size_t{0}; nline < pageblock->GetNbChildren(U"Lines");
++nline){

		// Obtiene del árbol Lines la línea nline
		auto line = pageblock->GetChild(U"Lines", nline);

		// Extract connected components in the line (characters).
		// To do that, a new black and white image (that is not smeared)
		// is automatically computed.
		line->ExtractCC(U"Characters");

		/**/line->FilterMinOr(U"Characters", 1, 2);

		// Sort characters from left to right.
		line->SortTree(U"Characters", crn::Direction::LEFT);

		// For each character...
		for (auto nchar = size_t{0}; nchar < line->GetNbChildren(U"Characters");
++nchar){
			auto character = line->GetChild(U"Characters", nchar);

			// Extract the features.
			auto features = featureExtractor.Extract(*character);

			// Classify the character using the database.
			auto res = crn::BasicClassify::NearestNeighbor(features,
database.begin(), database.end());

			// Print the characters label.
			s += char32_t(U' ' + res.class_id);

			char character_fn = char32_t(U' ' + res.class_id);

			std::cout << "ID char read: "
				<< res.class_id <<
				 " (id + val_char(' ') ) --> " <<
				  char32_t(U' ' + res.class_id) <<
				  " = " << character_fn << std::endl;

		}
```

```
                // End of line
                s += U'\n';
        }


        // Verbose
        CRNVerbose(s);

        // Records time in a stopwatch. Stores a time with a name.
        // Separa el tiempo de Recognition del total
        crn::Timer::Split(U"TwitterOCR", U"Recognition");
        CRNVerbose(crn::Timer::Stats(U"TwitterOCR"));
}
```

Source 2: OCR for tweets.

```
/* Copyright 2017
 *
 * based on: ocr4dummies.cpp
 *
 * A quick OCR engine to recognize letters from street names.
 *
 * Compile with:
g++ street_ocr.cpp -o street_ocr -lcrn
*
 * \author
 * Javier Izquierdo Vera - javierizquierdovera@gmail.com
 */

#include <CRNBlock.h>
#include <CRNImage/CRNImageGray.h>
#include <CRNFeature/CRNFeatureSet.h>
#include <CRNFeature/CRNFeatureExtractorProfile.h>
#include <CRNFeature/CRNFeatureExtractorProjection.h>
#include <CRNFeature/CRNBlockTreeExtractorTextLinesFromProjection.h>
#include <CRNAI/CRNBasicClassify.h>
#include <CRNUtils/CRNTimer.h>
#include <CRNIO/CRNIO.h>

using namespace crn::literals;

int main(int argc, char *argv[]){

        std::cout << std::endl;

        // Check argument.
        if (argc < 2){
                printf("Usage: %s <image_name>\n", argv[0]);
                return -1;
        }

        // Verbose
        crn::IO::IsVerbose() = true;
        crn::IO::IsQuiet() = false;

        // Starts the quick stopwatch.
        crn::Timer::Start(U"StreetOCR");



        /*************************************************************************/
        /* 1. Database                                                           */
```

```
	/*****************************************************************************/
	// Create a feature extraction engine.
	auto featureExtractor = crn::FeatureSet{};

	// It will extract the four profiles, reduced each to 10 values in [0..100].
	featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProfile>(
				crn::Direction::LEFT | crn::Direction::RIGHT |
				crn::Direction::TOP | crn::Direction::BOTTOM, 10, 1000));

	// It will also extract the two projections under the same conditions.
	featureExtractor.PushBack(std::make_shared<crn::FeatureExtractorProjection>(
				crn::Orientation::HORIZONTAL | crn::Orientation::VERTICAL,
10, 100));

	// Create the database.
	auto database = std::vector<crn::SObject>();


	// For each character...
	std::cout << "[*] Load characters: ";

	for (auto c = 'A'; c <= 'Z'; ++c){

		// Open a prototype image stored as "street/font/*.png"
		auto charFileName = "street/font/"_p / c + ".png"_p;

		auto charimage = crn::SImage{};

		try{
			// Open image
			charimage = crn::NewImageFromFile(charFileName);
		} catch (std::exception &ex) {
			CRNError(U"Cannot open database: ");
			CRNVerbose(crn::String(U" ") + ex.what());
			return -2;
		}

		// Embed the image in a block structure
		auto charblock = crn::Block::New(charimage);

		// Extract the features and store it in the database (vector)
		database.push_back(featureExtractor.Extract(*charblock));
	}

	std::cout << std::endl << std::endl;
	crn::Timer::Split(U"StreetOCR", U"Database");

	/*****************************************************************************/
	/* 2. Document                                                               */
	/*****************************************************************************/

	// Open the document image file
	auto imageFileName = crn::Path(argv[1]);
	auto pageimage = crn::SImage{};

	try {
		pageimage = crn::NewImageFromFile(imageFileName);
	} catch (...) {
		CRNError(U"Cannot open document image");
		return -3;
	}


	/*****************************************************************************/
	/* 2.1 Segmentation                                                          */
```

```
        /*****************************************************************************/

        // Embed the image in a block structure.
        auto pageblock = crn::Block::New(pageimage);

        // Extract text lines
        crn::BlockTreeExtractorTextLinesFromProjection{U"Lines"}.Extract(*pageblock);

        /*****************************************************************************/
        /* 2.2 Recognition                                                           */
        /*****************************************************************************/
        auto s = crn::String{};

        // For each line...
        for (auto nline = size_t{0}; nline < pageblock->GetNbChildren(U"Lines");
++nline){

                // Obtiene del árbol Lines la línea nline
                auto line = pageblock->GetChild(U"Lines", nline);

                // Extract connected components in the line (characters).
                // To do that, a new black and white image (that is not smeared)
                // is automatically computed.
                line->ExtractCC(U"Characters");

                /**/line->FilterMinOr(U"Characters", 2, 2);

                // Sort characters from left to right.
                line->SortTree(U"Characters", crn::Direction::LEFT);

                // For each character...
                for (auto nchar = size_t{0}; nchar < line->GetNbChildren(U"Characters");
++nchar){
                        auto character = line->GetChild(U"Characters", nchar);

                        // Extract the features.
                        auto features = featureExtractor.Extract(*character);

                        // Classify the character using the database.
                        auto res = crn::BasicClassify::NearestNeighbor(features,
database.begin(), database.end());

                        // Print the characters label.
                        s += char32_t(U'A' + res.class_id);

                        char character_fn = char32_t(U'A' + res.class_id);

                        std::cout << "ID char read: "
                                << res.class_id <<
                                 " (id + val_char('A') ) --> " <<
                                  char32_t(U'A' + res.class_id) <<
                                  " = " << character_fn << std::endl;
                }

                // End of line
                s += U'\n';
        }


        // Verbose
        CRNVerbose(s);

        // Records time in a stopwatch. Stores a time with a name.
        // Separa el tiempo de Recognition del total
        crn::Timer::Split(U"StreetOCR", U"Recognition");
```

```
        CRNVerbose(crn::Timer::Stats(U"StreetOCR"));
}
```
Source 3: OCR for street name.