Introduction to OpenCV Library

Figure 1: Color image of Lena, the most used image in digital image processing *(left)*; grayscale image of Lena *(middle)*; grayscale image of Lena with modified pixel and drawn rectangle *(right)*.

Welcome to the Digital Image Processing exercises[1]. In these exercises, we'll implement image processing algorithms that are discussed in lectures from the theoretical perspective.

Listing 7 shows the full source code of our exercise that is able to load a color image, convert it to grayscale image, read pixel values, manipulate pixel values, and draw a rectangle. Let's go through the code in more detail.

OpenCV is a C++ library for manipulating images and contains a lot of image processing and analysis algorithms. Most of the OpenCV's functions reside in the **`cv`** namespace. Images are stored in matrices using **`cv::Mat`** data type. To open an image, we use **`cv::imread`** function that returns an image that we can store in a variable. Images can be color or grayscale. Most of our algorithms will use grayscale images. The following code reads image in color and grayscale variant and stores them in two different variables.

```
1 // load color image from file system to cv::Mat variable, this will be loaded using 8
      bits (uchar) per channel
2 cv::Mat src_8uc3_img = cv::imread( "images/lena.png", cv::IMREAD_COLOR );
3
4 // load color image from file system to cv::Mat variable, this will be loaded using 8
      bits (uchar) per channel
5 cv::Mat src_8uc1_img = cv::imread( "images/lena.png", cv::IMREAD_GRAYSCALE );
```

Code Listing 1: Reading an image.

As you can see, we now have the color version of Lena (Figure 1 (left)) and gray scale Lena (Figure 1 (middle)) loaded into variables **`src_8uc3_img`** and **`src_8uc1_img`**.

However, we can also convert images between different color spaces. We can easily convert a RGB image to grayscale one using **`cv::cvtColor`** function as shown in listing 2.

```
1 // convert input color image to grayscale one, CV_BGR2GRAY specifies direction of
      conversion
2 cv::cvtColor( src_8uc3_img, gray_8uc1_img, cv::COLOR_BGR2GRAY );
```

Code Listing 2: Converting a color image to a grascale one.

We just converted color image **`src_8uc3_img`** to empty image **`gray_8uc1_img`**.

When we load an image from a file, each pixel is represented using 8 bits of information (**`unsigned char`** in C++ that we can refer as **`uchar`**[2]). Pixel values are in range 0 - 255. In grayscale image, we have one **`uchar`** per pixel. In color image, we have three **`uchar`**s per pixel forming traditional **RGB** pixels. However, for some image processing operations, it is better to

[1] `http://mrl.cs.vsb.cz/people/gaura/dzo_course`

[2] This is simplified, but OK for our needs

represent image using values in range 0.0 - 1.0. Such values have to stored in real value data type. For our use, it'll be completely sufficient to use **float** data type for such representation. To convert a grayscale image from 8 bits (**uchar**) representation to 32 bits (**float**) representation we use **convertTo** method of **cv::Mat** variable. The parameters of the method are: output image, output data type, and conversion scale and the method usage is described below.

```
// convert grayscale image from 8 bits to 32 bits, resulting values will be in the
    interval 0.0 - 1.0
gray_8uc1_img.convertTo( gray_32fc1_img, CV_32FC1, 1.0 / 255.0 );
```

Code Listing 3: Converting an image represented using **uchar**s to an image represented by **float**s.

Since the main aim of this course is to implement image processing algorithms, we'll need to access pixels of an image. To do so, we'll us **at** method of the **cv::Mat** data type. This method is templated (it needs a type specifier in angle brackets before arguments. Method signature is as follows.

at<image_type>(int y, int x)

```
// read grayscale value of a pixel, image represented using 8 bits
uchar p1 = gray_8uc1_img.at<uchar>( y, x );

// read grayscale value of a pixel, image represented using 32 bits
float p2 = gray_32fc1_img.at<float>( y, x );

// read color value of a pixel, image represented using 8 bits per color channel
cv::Vec3b p3 = src_8uc3_img.at<cv::Vec3b>( y, x );

// print values of pixels
printf( "p1 = %d\n", p1 );
printf( "p2 = %f\n", p2 );
printf( "p3[ 0 ] = %d, p3[ 1 ] = %d, p3[ 2 ] = %d\n", p3[ 0 ], p3[ 1 ], p3[ 2 ] );
```

Code Listing 4: Accessing pixels.

We're assigning a grayscale value (brightness) to the **uchar** variable **p1**. **gray_8uc1_img** has brightness values represented using 8 bits. As you can see, type specifier of **at** method is set to **uchar**. It's followed by **y**, and **x** variables that specify, at which position to read pixel value. The same procedure is done in the case of **gray_32fc1_img** image, which uses 32 bits representation of brightness values. The only difference is that we use **float** instead of **uchar** data type. To access color pixels in **src_8uc3_img**, we need to use **cv::Vec3b**. This type holds three values (RGB) at once. To access each color value, we use **[]** notation as is used in the above example.

Another important operation with image pixels is, of course, setting a new pixel brightness. This is done again using **at** method. The only difference from the read operation is that we assign a new value to the method. An example is shown in the listing below.

```
// set pixel value to 0 (black)
gray_8uc1_img.at<uchar>( y, x ) = 0;
```

Code Listing 5: Assigning a new value to a pixel.

When implementing image processing algorithms, you'll quite often need to go through all image pixels and perform some operation with brightness values. To access all pixels, we usually use two nested for loops to iterate over all rows and in each row to iterate over all columns. As an example (6), we'll create a gradient image. First, we crate a new image **gradient_8uc1_img** with 50 rows and 256 columns and with **CV_8UC1** type. This means that image will use 8 bits as data representation (**uchar**) and one channel, so it's essentially a grayscale image. Then we iterate over all pixels and assign brightness value according to the column number.

```
// declare variable to hold gradient image with dimensions: width= 256 pixels, height=
    50 pixels.
// Gray levels wil be represented using 8 bits (uchar)
```

```
cv::Mat gradient_8uc1_img( 50, 256, CV_8UC1 );

// For every pixel in image, assign a brightness value according to the x coordinate.
// This wil create a horizontal gradient.
for ( int y = 0; y < gradient_8uc1_img.rows; y++ ) {
    for ( int x = 0; x < gradient_8uc1_img.cols; x++ ) {
        gradient_8uc1_img.at<uchar>( y, x ) = x;
    }
}
```

Code Listing 6: Create an image of a horizontal gradient from the black color to the white using grayscale.

Figure 2: A gradient image produced by our code.

You can find the full listing of the exercise's code below.

```
#include <iostream>

#include <opencv2/opencv.hpp>

int main( int argc, char *argv[] )
{
    cv::Mat src_8uc3_img = cv::imread( "images/lena.png", cv::IMREAD_COLOR ); // load color image from file system to Mat variable, this will be loaded using 8 bits (uchar)
    //cv::imshow( "LENA", img );

    // declare variable to hold grayscale version of img variable, gray levels wil be represented using 8 bits (uchar)
    cv::Mat gray_8uc1_img;
    // declare variable to hold grayscale version of img variable, gray levels wil be represented using 32 bits (float)
    cv::Mat gray_32fc1_img;

    cv::cvtColor( src_8uc3_img, gray_8uc1_img, cv::COLOR_BGR2GRAY ); // convert input color image to grayscale one, CV_BGR2GRAY specifies direction of conversion
    gray_8uc1_img.convertTo( gray_32fc1_img, CV_32FC1, 1.0 / 255.0 ); // convert grayscale image from 8 bits to 32 bits, resulting values will be in the interval 0.0 - 1.0

    int x = 10, y = 15; // pixel coordinates

    uchar p1 = gray_8uc1_img.at<uchar>( y, x ); // read grayscale value of a pixel, image represented using 8 bits
    float p2 = gray_32fc1_img.at<float>( y, x ); // read grayscale value of a pixel, image represented using 32 bits
    cv::Vec3b p3 = src_8uc3_img.at<cv::Vec3b>( y, x ); // read color value of a pixel, image represented using 8 bits per color channel

    // print values of pixels
    printf( "p1 = %d\n", p1 );
    printf( "p2 = %f\n", p2 );
    printf( "p3[ 0 ] = %d, p3[ 1 ] = %d, p3[ 2 ] = %d\n", p3[ 0 ], p3[ 1 ], p3[ 2 ] );

    gray_8uc1_img.at<uchar>( y, x ) = 0; // set pixel value to 0 (black)

    // draw a rectangle
    cv::rectangle( gray_8uc1_img, cv::Point( 65, 84 ), cv::Point( 75, 94 ),
                   cv::Scalar( 50 ), cv::FILLED );
```

```
34
35      // declare variable to hold gradient image with dimensions: width= 256 pixels,
        height= 50 pixels.
36      // Gray levels wil be represented using 8 bits (uchar)
37      cv::Mat gradient_8uc1_img( 50, 256, CV_8UC1 );
38
39      // For every pixel in image, assign a brightness value according to the x coordinate
        .
40      // This wil create a horizontal gradient.
41      for ( int y = 0; y < gradient_8uc1_img.rows; y++ ) {
42          for ( int x = 0; x < gradient_8uc1_img.cols; x++ ) {
43              gradient_8uc1_img.at<uchar>( y, x ) = x;
44          }
45      }
46
47      // diplay images
48      cv::imshow( "Gradient 8uc1", gradient_8uc1_img );
49      cv::imshow( "Lena gray 8uc1", gray_8uc1_img );
50      cv::imshow( "Lena gray 32fc1", gray_32fc1_img );
51
52      cv::waitKey( 0 ); // wait until keypressed
53
54      return 0;
55  }
```

Code Listing 7: Full exercise code.