Operations with images

Prev Tutorial: Mask operations on matrices

Next Tutorial: Adding (blending) two images using OpenCV

Compatibility	OpenCV >= 3.0

Input/Output

Images

C++ Java Python

Load an image from a file:

```
Mat img = imread(filename);
```

If you read a jpg file, a 3 channel image is created by default. If you need a grayscale image, use:

```
Mat img = imread(filename, IMREAD_GRAYSCALE);
```

Note

Format of the file is determined by its content (first few bytes). To save an image to a file:

```
imwrite(filename, img);
```

Note

Format of the file is determined by its extension.

Use **cv::imdecode** and **cv::imencode** to read and write an image from/to memory rather than a file.

Basic operations with images

Accessing pixel intensity values

C++ Java Python

In order to get pixel intensity value, you have to know the type of an image and the number of channels. Here is an example for a single channel grey scale image (type 8UC1) and pixel coordinates x and y:

```
Scalar intensity = img.at<uchar>(y, x);
```

C++ version only: intensity.val[0] contains a value from 0 to 255. Note the ordering of x and y. Since in OpenCV images are represented by the same structure as matrices, we use the same convention for both cases - the 0-based row index (or y-coordinate) goes first and the 0-based column index (or x-coordinate) follows it. Alternatively, you can use the following notation (**C++ only**):

```
Scalar intensity = img.at<uchar>(Point(x, y));
```

Now let us consider a 3 channel image with BGR color ordering (the default format returned by imread):

C++ code

```
Vec3b intensity = img.at<Vec3b>(y, x);
uchar blue = intensity.val[0];
uchar green = intensity.val[1];
uchar red = intensity.val[2];
```

Python Python

```
_blue = img[y,x,0]
_green = img[y,x,1]
_red = img[y,x,2]
```

You can use the same method for floating-point images (for example, you can get such an image by running Sobel on a 3 channel image) (**C++ only**):

```
Vec3f intensity = img.at<Vec3f>(y, x);
float blue = intensity.val[0];
float green = intensity.val[1];
float red = intensity.val[2];
```

The same method can be used to change pixel intensities:

```
img.at<uchar>(y, x) = 128;
```

There are functions in OpenCV, especially from calib3d module, such as **cv::projectPoints**, that take an array of 2D or 3D points in the form of Mat. Matrix should contain exactly one column, each row corresponds to a point, matrix type should be 32FC2 or 32FC3 correspondingly. Such a matrix can be easily constructed from `std::vector` (**C++ only**):

```
vector<Point2f> points;
//... fill the array
Mat pointsMat = Mat(points);
```

One can access a point in this matrix using the same method `Mat::at` (**C++ only**):

```
Point2f point = pointsMat.at<Point2f>(i, 0);
```

Memory management and reference counting

C++ Java Python

Mat is a structure that keeps matrix/image characteristics (rows and columns number, data type etc) and a pointer to data. So nothing prevents us from having several instances of Mat corresponding to the same data. A Mat keeps a reference count that tells if data has to be deallocated when a particular instance of Mat is destroyed. Here is an example of creating two matrices without copying data (**C++ only**):

```
std::vector<Point3f> points;
// .. fill the array
Mat pointsMat = Mat(points).reshape(1);
```

As a result, we get a 32FC1 matrix with 3 columns instead of 32FC3 matrix with 1 column. `pointsMat` uses data from points and will not deallocate the memory when destroyed. In this particular instance, however, developer has to make sure that lifetime of `points` is longer than of `pointsMat` If we need to copy the data, this is done using, for example, **cv::Mat::copyTo** or **cv::Mat::clone**:

```
Mat img = imread("image.jpg");
Mat img1 = img.clone();
```

An empty output Mat can be supplied to each function. Each implementation calls Mat::create for a destination matrix. This method allocates data for a matrix if it is empty. If it is not empty and has the correct size and type, the method does nothing. If however, size or type are different from the input arguments, the data is deallocated (and lost) and a new data is allocated. For example:

```
Mat img = imread("image.jpg");
Mat sobelx;
Sobel(img, sobelx, CV_32F, 1, 0);
```

Primitive operations

C++ Java Python

There is a number of convenient operators defined on a matrix. For example, here is how we can make a black image from an existing greyscale image `img`

```
img = Scalar(0);
```

Selecting a region of interest:

```
Rect r(10, 10, 100, 100);
Mat smallImg = img(r);
```

Conversion from color to greyscale:

```
Mat img = imread("image.jpg"); // loading a 8UC3 image
Mat grey;
cvtColor(img, grey, COLOR_BGR2GRAY);
```

Change image type from 8UC1 to 32FC1:

```
src.convertTo(dst, CV_32F);
```

Visualizing images

C++ Java Python

It is very useful to see intermediate results of your algorithm during development process. OpenCV provides a convenient way of visualizing images. A 8U image can be shown using:

```
Mat img = imread("image.jpg");
namedWindow("image", WINDOW_AUTOSIZE);
imshow("image", img);
waitKey();
```

A call to **waitKey()** starts a message passing cycle that waits for a key stroke in the "image" window. A 32F image needs to be converted to 8U type. For example:

```
Mat img = imread("image.jpg");
Mat grey;
```

```
        cvtColor(img, grey, COLOR_BGR2GRAY);
        Mat sobelx;
        Sobel(grey, sobelx, CV_32F, 1, 0);
        double minVal, maxVal;
        minMaxLoc(sobelx, &minVal, &maxVal); //find minimum and maximum intensities
        Mat draw;
        sobelx.convertTo(draw, CV_8U, 255.0/(maxVal - minVal), -minVal * 255.0/(maxVal - minVal));
        namedWindow("image", WINDOW_AUTOSIZE);
        imshow("image", draw);
        waitKey();
```

Note

Here **cv::namedWindow** is not necessary since it is immediately followed by **cv::imshow**. Nevertheless, it can be used to change the window properties or when using **cv::createTrackbar**