# Обработка изображений в OpenCV

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

24 сентября 2018 г.

Версия: 0.10

**Аннотация**

Растровое изображение. Пиксели. Сглаживание. Обработка видео.

## 1 Обработка изображений

В данной заметке будет показано как обрабатывать растровые изображения. В первом подразделе как по-пиксельно обрабатывать изображение.

### 1.1 Растровое изображение

Как было указано в прошлой заметке растровое изображение представляется в виде матрицы. Матрица хранится в памяти построчно, т.е. строчка за строчкой. Последние означает, что данные одной строчки идут вслед за данными предыдущей. Данные каждой строчки соответствуют пикселям её составляющие, при этом каналы хранятся последовательно для каждого из пикселей.

В согласии с вышеописанным представим простейшую программу, которая создает негатив по изображению, т.е. значение каждого из каналов заменяется на "противоположный а именно – вычитается из значения 255. Сначала покажем как это можно осуществить на чистом Си, далее будем видоизменять программу, в том числе, совершенствовать.

**Си-шный способ** Обход всех пикселов в общем случае технически сложен так как необходимо учитывать тип элементов матрицы. Рассмотрим случай, когда компоненты вектора элемента матрицы является беззнаковыми 8-битными целыми числами (CV_8U).

```
1  //I -- входное изображение.
2  //В случае успеха возвращаем 0, иначе -1.
3  int makeNegative(Mat& I)
4  {
5    //Сначала проверим, что у изображения нужный тип.
6    if ( I.depth() != CV_8U)
7    {
8      //Если не подходит тип данных, то выходим
9      //из функции с ошибкой.
10     printf("Тип не поддерживается!\n");
11     return -1;
12   }
13
14   const int channels = I.channels();//Количество каналов.
15   const int rows = I.rows;//Количество строк.
16   const int cols = I.cols;//Количество столбцов.
17
18   int row;//Текущий номер строки.
19   for( row = 0; row < rows; row++)//Обходим все строки.
20   {
21     //Указатель на начало i строки матрицы.
22     //нужного типа
23     uchar *p = I.ptr<uchar>( row );
24     int col;//Текущий номер столбца.
25     //Обходим все столбцы:
26     for( col = 0; col < cols; col++)
27     {
28       int channel;//Текущий номер канала.
29       //Обходим все каналы:
30       for( channel = 0; channel < channels; channel++)
31         //Значение каждого канала каждого пикселя
32         p[col*channels + channel] = 255 - //изменяем на
33           p[col*channels + channel];//противоположный.
34     }
35   }
36   //В случае успеха возвращаем 0.
37   return 0;
38 }
```

Обычно функции расчитаны не на все типы данных, иначе бы пришлось писать

много проверок, да и кода было бы много. Поэтому, стандартно, когда функция прежде чем выполнять какие-либо действия сначала проверяет соответствие входных данных требуемым. В данном случае, функция умеет обрабатывать только матрицы элементы которой являются векторами из 8-битных беззнаковых чисел.

С точки зрения эффективности память лучше обходить последовательно (связано с устройством кэша), поэтому строчки матрицы как это и показано в программе лучше обходить построчно.

Ввиду того, что каналы в памяти идут подряд, как и сами пикселы, в случае идентичной операции для всех каналов можно избавится от внутреннего цикла, заменив цикл по колонкам на:

```
1 int col;//Текущий номер компоненты.
2 //Обходим все компоненты пикселов:
3 for( col = 0; col < cols * channels; col++)
4 {
5   //Значение каждого канала каждого пикселя
6   //изменяем на противоположный:
7   p[ col ] = 255 - p[ col ];
8 }
```

Подчеркну, что это возможно только в том случае, если все каналы обрабатываются единообразно. В противном случае, пришлось бы как ранее и было показано все-таки каждый из пикселов обрабатывать по отдельности.

Эффективность можно поднять ещё больше, если учитывать расположение строк. Так, существует метод `Mat::isContinuous()`, который позволяет определить идут ли строчку в притык одна к другой. Если да, то матрицу можно обрабатывать как одномерный массив. Для этого, до внешнего цикла следует вызвать код:

```
1 if( I.isContinuous() )
2 {
3   cols *= rows;
4   rows = 1;
5 }
```

Для использования в предыдущем коде, следует убрать `const`.

**Объекты C++** Объектная модель C++ позволяет сократить немного код. Так, вместо того, чтобы обрабатывать матрицу по компонентно, её можно обрабатывать по пиксельно.

В случае цветных изображений, пиксели будут являться векторами длины 3 компоненты которого являются 8-битным беззнаковым числом. Для такого объекта в C++ есть обозначение: `Vec3b`. Тогда код можно свести к следующему:

```
Vec3b d(255, 255, 255);

int row;
for( row = 0; row < rows; row++)
{
  //Указатель на начало i строки матрицы.
  //нужного типа
  Vec3b *p = I.ptr<Vec3b>( row );
  int col;//Текущий номер столбца.
  //Обходим все столбцы:
  for( col = 0; col < cols; col++)
    p[ col ] = d - p[ col ];
}
```

Естественно для других типов изображений придется написать свой код.

**Итератор из C++** Циклы с явной индексацией элементов в C++ были обобщены на итераторы. В OpenCV соответствующий механизм поддерживается, так для обхода всех пикселов можно ими воспользоваться:

```
int makeNegative(Mat& I)
{
  if ( I.depth() != CV_8U)
  {
    printf("Тип не поддерживается!\n");
    return -1;
  }
  const int channels = I.channels();
  switch( channels )
  {
    case 1:
    {
      MatIterator_<uchar> it, end;
      for( it = I.begin<uchar>(), end = I.end<uchar>();
           it != end; ++it)
        *it = 255 - *it;
      break;
```

```
18      }
19      case 3:
20      {
21        MatIterator_<Vec3b> it, end;
22        for( it = I.begin<Vec3b>(), end = I.end<Vec3b>();
23             it != end; ++it)
24        {
25          (*it)[0] = 255 - (*it)[0];
26          (*it)[1] = 255 - (*it)[1];
27          (*it)[2] = 255 - (*it)[2];
28        }
29        break;
30      }
31      default:
32        printf("error in number of channels %d\n",
33            channels);
34    }
35    return 0;
36  }
```

**Напрямую** У класса `Mat` есть метод `Mat::at`, который позволяет получить доступ к произвольному элементу матрицы.

```
1  I.at<uchar>(row, col) = 1 - I.at<uchar>(row, col);
```

или в случае 3х компонентной матрицы

```
1  I.at<uchar>(row, col)[0] = 255 -I.at<uchar>(row, col)[0];
2  I.at<uchar>(row, col)[1] = 255 -I.at<uchar>(row, col)[1];
3  I.at<uchar>(row, col)[2] = 255 -I.at<uchar>(row, col)[2];
```

### Современный C++

**foreach** Через функцию:

```
1  void f(Vec3b &a)
2  {
3    a[0] = 255 - a[0];
4    a[1] = 255 - a[1];
5    a[2] = 255 - a[2];
```

```
6     return;
7  }
8  ...
9
10 for_each( img.begin<Vec3b>(), img.end<Vec3b>(), f );
```

Через ламба функцию:

```
1  for_each( img.begin<Vec3b>(), img.end<Vec3b>(),
2     [](Vec3b &a)
3  {
4     a[0] = 255 - a[0];
5     a[1] = 255 - a[1];
6     a[2] = 255 - a[2];
7  });
```

**Упражнения** Упражнение. В прошлой заметке было показано как изменить контраст (умножению на константу) и яркость (добавление серого) фактически благодаря операциям над матрицами. Все тоже самое можно сделать написав цикл. Напишите его в качестве тренировки. Подсказка: значения нужно "насыщать"(`saturate_cast<uchar>`).

## 1.2 Дополнительно Mat

**Компоненты** Матрицу можно расслоить на разные компоненты. Для этого используется функция `split`:

```
1  Mat img;
2  img = imread(argv[1], IMREAD_COLOR);
3
4  // Размер массива должен совпасть с необходимым размером. Иначе
   падаем.
5  Mat ch[3];
6  split( img, ch);
7
8  namedWindow( "Display window", WINDOW_AUTOSIZE );
9  imshow( "Display window", ch[0] );
```

Компоненты можно обратно собрать в матрицу, в частности? в изображение:

```
1  Mat img;
```

```
2  img = imread(argv[1], IMREAD_COLOR);
3
4  vector<Mat> ch;
5  split( img, ch);
6  Mat zer = Mat::zeros( Size( img.cols, img.rows), CV_8UC1);
7  ch[0] = zer;
8  ch[2] = zer;
9
10 Mat dst;
11 // Количество каналов берется из dst.
12 merge( ch, dst);
13
14 namedWindow( "Display window", WINDOW_AUTOSIZE );
15 imshow( "Display window", dst );
```

**Больше одного измерения** Scalar::all?? .create метод.

**Печать** Можно использовать для инициализации объекта «, или печать » и для point.

Например, для отладки или дотошного вывода данных.

format Formater::FMT_??

## 1.3 Сглаживание

Одно из важны способов обработки изображений является сглаживанием. Суть сглаживания сводится к тому, что окно определенного фиксированного размера бежит по изображению. В выходное изображение заносится значения для данного окна являющиеся некой функцией от пикселов в окне.

Самым простым способом сглаживания является усреднение. Это достигается посредством функции `blur`. А англоязычной литературе этот фильтр называется обычно box filter.

Обычное усреднение имеет свои недостатки (более подробно в курсе по цифровой обработке изображений). В частности, все пикселы используются с одинаковыми весами. Это устраняет как может Гауссовский фильтр, который задается соответствующим распределением. В опессв от реализован функцией `GaussianBlur`.

Для определенного типа шума в изображении требуется свой тип фильтра. Так, шум типа соль/перец хорошо убирается медианным фильтром. Суть его заключается в том, что он заменяет центральный пиксель фильтра на медиану пикселей в окне, если между ними большая разница.

Сглаживающий фильтр сглаживает все подряд. В частности, границы между объектами. Для борьбы с этим был разработан адаптивный фильтр – билатерал фильтр. Конкретно он называется bilateralFilter.

Все данные фильтры объедены в ниже следующей программе:

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <string>
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char *argv[])
{
  Mat img;
  img = imread(argv[1], IMREAD_COLOR);

  Mat dst;

  switch( atoi( argv[2] ) )
  {
  case 1:
    blur( img, dst, Size( 10, 10), Point(-1, 1) );
  case 2:
    GaussianBlur( img, dst, Size( 9, 5), 0, 0);
  case 3:
    medianBlur( img, dst, 5);
  case 4:
    bilateralFilter( img, dst, 21, 7, 51/2);
  default:
    dst = img;
  }

  namedWindow( "Display window", WINDOW_AUTOSIZE );
  imshow( "Display window", dst );
  waitKey(0);
  return 0;
}
```

Упражнение. Написать собственную реализацию фильтров.

## 1.4 Пирамида

При использовании обычных фильтров сглаживания (Гауссовский) можно построить (имеет смысл) пирамиду. Так после сглаживания, изображения можно уменьшить в два раза в размерах. И так далее. Получиться пирамида масштабов для изображения. В некоторых методах обработки изображения она применяется.

Для построения следующего слоя пирамиды:

```
1 pyrDown( img, dst, Size( img.cols/2, img.rows/2 ) );
```

Для выполнения обратной операции можно сделать так:

```
1 pyrUp( img, dst, Size( img.cols*2, img.rows*2 ) );
```

## 1.5 Фурье преобразование

Фурье преобразование является важным инструментом при цифровой обработке изображений. Само по себе оно задается функцией `dft`. Но чтобы её корректно вызвать необходимо проделать ряд действий.

## 1.6 Считывание видео

Как ранее было показано изображения считываются функцией `imread`. Для считывания видео нужно использовать класс `VideoCapture`.

```
1 //Код
```

В конструкторе указывается путь к картинке.

## 1.7 Ввод/вывод видео

Помимо ввода и вывода изображений библиотека opencv позволяет вводить и выводить видео данные хоть и с некоторыми ограничениями видь в первую очередь данная библиотека предназначена для анализа визуальной информации.

**Ввод кадров** Класс `cv::VideoCapture` предназначен для считывания видео потоков, в частности, из файлов:

```
#include <iostream>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

int main(int argc, char *argv[])
{
  if( argc < 2 )
  {
    std::cout << "need file name on command line!"
        << std::endl;
    return -1;
  }

  cv::VideoCapture cap( argv[1] );
  if( !cap.isOpened() )
  {
    std::cout << "error openning video file"
        << std::endl;
    return -1;
  }

  std::cout << "Video file " << argv[1] <<
      " is opened!" << std::endl;

  cv::Mat frame;
  int cnt = 0;
  while(1)
  {
    cap >> frame;
    if( frame.empty() )
    {
      std::cout << "ahh, all frames processed!"
          << std::endl;
      break;
    }
    cnt++;
  }
```

```
39    std::cout << "frames count " << cnt << std::endl;
40    return 0;
41  }
```

Упражнение. Измени программу так, чтобы каждый $n$-ый кадр был записан на диск (по отдельности) как изображение.

Упражнение. Измени программу так, чтобы в окошке показывались кадры из видео потока.

**Считывание дополнительной информации** Видео поток помимо самих кадров – растровые изображения – характеризуется и другой информацией, например, общим количеством кадров в видео файле и частотой проигрывания. Для этого вставим следующий фрагмент кода в предыдущую программу:

```
1  int total_frames = cap.get( CV_CAP_PROP_FRAME_COUNT );
```

В переменную `total_frames` будет записано реальное общее количество кадров в видео потоке. Соответственно вывод можно поменять на:

```
1  std::cout << "frames count " << cnt << " of "
2      << total_frames << std::endl;
```

Приведу краткий список полезных параметров:

Общие параметры:

| | |
|---|---|
| `CV_CAP_PROP_FRAME_WIDTH` | Ширина видео кадра. |
| `CV_CAP_PROP_FRAME_HEIGHT` | Высота видео кадра. |
| `CV_CAP_PROP_FPS` | Количество кадров в секунду. |
| `CV_CAP_PROP_FOURCC` | Название видео кодека потока в формате fourcc. |

Для обработки видеофайлов:

| | |
|---|---|
| `CV_CAP_PROP_FRAME_COUNT` | Общее количество кадров в видео файле. |
| `CV_CAP_PROP_POS_FRAMES` | Номер текущего считываемого кадра. |
| `CV_CAP_PROP_POS_MSEC` | Текущая позиция в миллисекундах. |
| `CV_CAP_PROP_POS_AVI_RATIO` | Процент обработанных кадров. |

При взаимодействии с видеокамерой:

Параметры камеры:

| | |
|---|---|
| CV_CAP_PROP_BRIGHTNESS | Яркость картинки. |
| CV_CAP_PROP_CONTRAST | Контраст картинки. |
| CV_CAP_PROP_SATURATION | Насыщенность цвета. |
| CV_CAP_PROP_SHARPNESS | Резкость картинки. |
| CV_CAP_PROP_EXPOSURE | Величина выдержки. |
| CV_CAP_PROP_AUTO_EXPOSURE | Автоматическая подстройка выдержки. |
| CV_CAP_PROP_IRIS | Размер диафрагмы. |
| CV_CAP_PROP_ZOOM | Степень увеличения. |
| CV_CAP_PROP_FOCUS | Положение фокуса. |
| CV_CAP_PROP_ISO_SPEED | Значение чувствительности ISO. |

Подчеркну, что не все параметры обязаны быть поддержаны. Так, если он не поддерживается, то возвращается отрицательное значение. Также следует отметить, что не все параметры кросс-платформенны, т.е. результат может отличаться в зависимости от используемой операционной системы.

**Вывод**