basic_image_operations

July 25, 2019

1 Overview

In this section, we will learn about some basic operations on images which are used very extensively for any computer vision project. In particular, we will see the following: 1. Copying and creating new images 1. Cropping an image section 1. Resizing an Image 1. Creating binary masks for images

```
In [1]: // Include libraries
        #include "../resource/lib/public/includeLibraries.h"
        #include <opencv2/opencv.hpp>
        #include <opencv2/core.hpp>
        #include <opencv2/highgui.hpp>
```

```
In [2]: using namespace std;
```

```
In [3]: using namespace cv;
```

2 Create new Images

In the previous section we learnt how to load images from disk. Let us discuss some ways of creating new images or copies of images already there.

```
In [4]: // Read image
        Mat image = imread(DATA_PATH+"images/boy.jpg");
```

Let's see what image we are dealing with

|

Input Image

|

2.0.1 Create copy of an Image

```
In [5]: // Create a new image by copying the already present image using the clone operation
        Mat imageCopy = image.clone();
```

2.0.2 Create a empty matrix

Let us create an empty matrix with size (100,200).

```
In [6]: Mat emptyMatrix = Mat(100,200,CV_8UC3, Scalar(0,0,0));
        imwrite("./results/emptyMatrix.png",emptyMatrix);
```

|
Empty Matrix
|
Let's fill it with white pixels

```
In [7]: //emptyMatrix = 255*np.ones((100,200,3),dtype='uint8')
        emptyMatrix.setTo(Scalar(255,255,255));
        imwrite("./results/emptyMatrixWhite.png",emptyMatrix);
```

|
Empty Matrix with White Pixels
|

2.0.3 Create a empty matrix of the same size as original image

Let us create a matrix with the same size and data type as the original image and fill it with 100.

```
In [8]: Mat emptyOriginal = Mat(emptyMatrix.size(), emptyMatrix.type(), Scalar(100,100,100));
        imwrite("./results/emptyMatrix100.png",emptyOriginal);
```

|
Empty Matrix with 100
|

3 Cropping an Image section

Cropping, as the name suggests is cutting out a portion from the image. A good example of using image cropping is in the creation of `Memes`, where people crop someone's face and put on top of another etc. We know how to do it in photo editing softwares. Let us see how it can be done using code so that you can do it more intelligently when required.

```
In [9]: // Read image
        image = imread(DATA_PATH+"images/boy.jpg");
```

Let's see what image we are dealing with

|
Input Image
|
Let's crop his face. For that, we need to find the approximate region where the face is present. From the figure, you can see that the face lies somewhere in the region :

- between row #50 to row #200
- between col #170 to col #320

So, for cropping out this area, we need to select this region from the original matrix and assign it to another variable. This is the cropped out image.

```
In [10]: // Crop out a rectangle
         // x coordinates = 170 to 320
         // y coordinates = 40 to 200
         Mat crop = image(Range(40,200),Range(170,320));
```

```
In [11]: imwrite("./results/crop.png",crop);
```

|
Cropped Image
|

4 Copying a Region to another

Once you have cropped a part of image, you can paste it to another image by choosing where you want to paste it.

`You just need to make sure that the size of rectangle that you cropped is same as the area you want to paste on`

```
In [12]: // First let us create a copy of the original image
         Mat copiedImage = image.clone();
```

Let's copy his face to the left and right.

```
In [13]: Mat copyRoi = image(Range(40,200),Range(180,320));

         // Find height and width of the ROI
         int roiHeight = copyRoi.size().height;
         int roiWidth = copyRoi.size().width;

         // Copy to left of Face
         copyRoi.copyTo(copiedImage(Range(40,40+roiHeight),Range(10,10+roiWidth)));
         // Copy to right of Face
         copyRoi.copyTo(copiedImage(Range(40,40+roiHeight),Range(330,330+roiWidth)));

         imwrite("./results/copiedRegions.png",copiedImage);
```

—

—
—

|

Input Image

|

Output Image

|

5 Resizing an Image

We will use `cv::resize` function for resizing an image.

5.0.1 Function Syntax

```
void cv::resize (   InputArray  src,
OutputArray     dst,
Size    dsize,
double  fx = 0,
double  fy = 0,
int     interpolation = INTER_LINEAR
)
```

Parameters - **src** - input image - **dst** - output resized image - **dsize** - output image size - **fx** - scale factor along the horizontal axis; - **fy** - scale factor along the vertical axis; Either dsize or both fx and fy must be non-zero. - **interpolation** - interpolation method (Bilinear / Bicubic etc).

Check interpolation flags here

There are two ways of using the resize function. 1. Specify width and height of output image explicitly > Use the dsize argument to specify the output size in the form (width,height). 2. Specify the scaling factors for resizing (for both width and height) > Use fx and fy arguments to specify the scaling factor for horizontal and vertical axis respectively. It should be a float. The output size is calculated as `((*.), (*.))`

5.1 Method1 - Specify width and height

You have to manually keep track of the aspect ratio of the output image as this method will not preserve the aspect ratio between input and output image.

Let us take an example to see how to use the functions in OpenCV for resizing.

```
In [14]: int resizeDownWidth = 300;
         int resizeDownHeight = 200;
         Mat resizedDown;
         resize(image, resizedDown, Size(resizeDownWidth, resizeDownHeight), INTER_LINEAR);

         // Mess up with the aspect ratio
         int resizeUpWidth = 600;
         int resizeUpHeight = 900;
         Mat resizedUp;
         resize(image, resizedUp, Size(resizeUpWidth, resizeUpHeight), INTER_LINEAR);

         imwrite("./results/resizedUp.png",resizedUp);
         imwrite("./results/resizedDown.png",resizedDown);
```

|

Input Image

|

Scaled Up Image

|

Scaled Down Image

|

You can see that the output images have been scaled.

It is also evident that the aspect ratio for the scaledUp output has been totally ruined. In most cases, it is a good idea to use the scaling factor to resize images.

5.2 Method2 - Specify scaling factor

This method is useful when you want to preserve the aspect ratio of the image. For example, say you have an image of size 400x600 and you want the width to be 360. Then, instead of specifying the height, you can find the scaling factor (i.e 360/600 = 0.6) and use it to resize the image.

```
In [15]: // Scaling Down the image 1.5 times by specifying both scaling factors
         double scaleUpX = 1.5;
         double scaleUpY = 1.5;

         // Scaling Down the image 0.6 times specifying a single scale factor.
         double scaleDown = 0.6;

         Mat scaledUp, scaledDown;

         resize(image, scaledDown, Size(), scaleDown, scaleDown, INTER_LINEAR);

         resize(image, scaledUp, Size(), scaleUpX, scaleUpY, INTER_LINEAR);

         imwrite("./results/scaledUp.png", scaledUp);
         imwrite("./results/scaledDown.png", scaledDown);
```

```
In [16]: cout << "Scaled Up Image size = " << scaledUp.size() << endl;
         cout << "Scaled Down Image size = " << scaledDown.size() << endl;
```

```
Scaled Up Image size = [720 x 540]
Scaled Down Image size = [288 x 216]
```

|
Scaled Down Image
|
Scaled Up Image
|
You can see that the aspect ratio has not been disturbed.

6 Creating an Image Mask

Masking is a very important step in many image processing and computer vision algorithms. The goal is to segment out the area of interest and apply your algorithm to a specific part of the image. You can perform all operations on this segmented part of the image and put it back on the original image.

6.1 Create a mask using coordinates

```
In [17]: // Create an empty image of same size as the original
         Mat mask1 = Mat::zeros(image.size(), image.type());
         imwrite("./results/mask1.png",mask1);
```

|
Mask
|

```
In [18]: mask1(Range(50,200),Range(170,320)).setTo(255);
         imwrite("./results/mask1Revised.png",mask1);
```

|
Original Image
|
Mask
|
This is the mask for the face region of the input image

6.2 Create a mask using pixel intensity or color

Another way of creating masks is by using some logic. One example is using a simple color information. For example, Let us try to find out the pixels which are approximately red in color.

If we want to focus on `red pixels`, the simplest logic that does the trick is:

```
* The red channel should have high intensity ( keep the range of pixel values from 100 to 255 
* The other 2 channels should have low intensity ( keep the range of pixel values in Blue and 
```

There is a nice OpenCV function which can do exactly this. We will use the opencv function `inRange`

Function Syntax It finds the pixels which lie in between the specified range. It produces a binary output image in which the white pixels corresspond to those pixels in the input image which fall in the specified range. The pixel values which fall outside the specified range are black (0)

```
void cv::inRange    (   InputArray  src,
InputArray  lowerb,
InputArray  upperb,
OutputArray     dst
)
```

Parameters - **`src`** - first input array. - **`lowerb`** - inclusive lower boundary array or a scalar. - **`upperb`** - inclusive upper boundary array or a scalar. - **`dst`** - output array of the same size as src and CV_8U type.

It produces a binary image (pixels are either black or white).

```
In [19]: Mat mask2;
         inRange(image, Scalar(0,0,150), Scalar(100,100,255), mask2);
         imwrite("./results/mask2.png",mask2);
```

—

—
—

|

Input Image

|

Mask

|

As you can see the white portion in the output mask corressponds to the red pixels in the original image.

We will discuss more about color based masking in later modules.