Eroding and Dilating

Prev Tutorial: Smoothing Images

Next Tutorial: More Morphology Transformations

Goal

In this tutorial you will learn how to:

- Apply two very common morphological operators: Erosion and Dilation. For this purpose, you will use the following OpenCV functions:
  - **cv::erode**
  - **cv::dilate**

Note

The explanation below belongs to the book **Learning OpenCV** by Bradski and Kaehler.

Morphological Operations

- In short: A set of operations that process images based on shapes. Morphological operations apply a *structuring element* to an input image and generate an output image.
- The most basic morphological operations are: Erosion and Dilation. They have a wide array of uses, i.e. :
  - Removing noise
  - Isolation of individual elements and joining disparate elements in an image.
  - Finding of intensity bumps or holes in an image
- We will explain dilation and erosion briefly, using the following image as an example:

Dilation

- This operations consists of convolving an image A with some kernel (B), which can have any shape or size, usually a square or circle.
- The kernel B has a defined *anchor point*, usually being the center of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name *dilation*).
- The dilatation operation is: $\texttt{dst}(x,y) = \max_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x',y+y')$
- Take the above image as an example. Applying dilation we can get:

- The bright area of the letter dilates around the black regions of the background.

Erosion

- This operation is the sister of dilation. It computes a local minimum over the area of given kernel.
- As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.
- The erosion operation is: $\texttt{dst}(x,y) = \min_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x',y+y')$
- Analagously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see in the result below that the bright areas of the image get thinner, whereas the dark zones gets bigger.

Code

C++ Java Python

This tutorial's code is shown below. You can also download it here

```
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

Mat src, erosion_dst, dilation_dst;

int erosion_elem = 0;
int erosion_size = 0;
int dilation_elem = 0;
int dilation_size = 0;
int const max_elem = 2;
int const max_kernel_size = 21;

void Erosion( int, void* );
void Dilation( int, void* );


int main( int argc, char** argv )
{
  CommandLineParser parser( argc, argv, "{@input | LinuxLogo.jpg | input image}" );
  src = imread( samples::findFile( parser.get<String>( "@input" ) ), IMREAD_COLOR );
  if( src.empty() )
  {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
  }

  namedWindow( "Erosion Demo", WINDOW_AUTOSIZE );
  namedWindow( "Dilation Demo", WINDOW_AUTOSIZE );
  moveWindow( "Dilation Demo", src.cols, 0 );

  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Erosion Demo",
          &erosion_elem, max_elem,
          Erosion );

  createTrackbar( "Kernel size:\n 2n +1", "Erosion Demo",
          &erosion_size, max_kernel_size,
          Erosion );

  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Dilation Demo",
          &dilation_elem, max_elem,
          Dilation );

  createTrackbar( "Kernel size:\n 2n +1", "Dilation Demo",
          &dilation_size, max_kernel_size,
          Dilation );

  Erosion( 0, 0 );
  Dilation( 0, 0 );

  waitKey(0);
  return 0;
}


void Erosion( int, void* )
{
  int erosion_type = 0;
  if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
  else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
  else if( erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }
```

```
  Mat element = getStructuringElement( erosion_type,
                       Size( 2*erosion_size + 1, 2*erosion_size+1 ),
                       Point( erosion_size, erosion_size ) );

  erode( src, erosion_dst, element );
  imshow( "Erosion Demo", erosion_dst );
}


void Dilation( int, void* )
{
  int dilation_type = 0;
  if( dilation_elem == 0 ){ dilation_type = MORPH_RECT; }
  else if( dilation_elem == 1 ){ dilation_type = MORPH_CROSS; }
  else if( dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

  Mat element = getStructuringElement( dilation_type,
                       Size( 2*dilation_size + 1, 2*dilation_size+1 ),
                       Point( dilation_size, dilation_size ) );

  dilate( src, dilation_dst, element );
  imshow( "Dilation Demo", dilation_dst );
}
```

Explanation

C++ Java Python

Most of the material shown here is trivial (if you have any doubt, please refer to the tutorials in previous sections). Let's check the general structure of the C++ program:

```
int main( int argc, char** argv )
{
  CommandLineParser parser( argc, argv, "{@input | LinuxLogo.jpg | input image}" );
  src = imread( samples::findFile( parser.get<String>( "@input" ) ), IMREAD_COLOR );
  if( src.empty() )
  {
    cout << "Could not open or find the image!\n" << endl;
    cout << "Usage: " << argv[0] << " <Input image>" << endl;
    return -1;
  }

  namedWindow( "Erosion Demo", WINDOW_AUTOSIZE );
  namedWindow( "Dilation Demo", WINDOW_AUTOSIZE );
  moveWindow( "Dilation Demo", src.cols, 0 );

  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Erosion Demo",
          &erosion_elem, max_elem,
          Erosion );

  createTrackbar( "Kernel size:\n 2n +1", "Erosion Demo",
          &erosion_size, max_kernel_size,
          Erosion );

  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Dilation Demo",
          &dilation_elem, max_elem,
          Dilation );

  createTrackbar( "Kernel size:\n 2n +1", "Dilation Demo",
          &dilation_size, max_kernel_size,
          Dilation );

  Erosion( 0, 0 );
  Dilation( 0, 0 );

  waitKey(0);
  return 0;
}
```

1. Load an image (can be BGR or grayscale)
2. Create two windows (one for dilation output, the other for erosion)
3. Create a set of two Trackbars for each operation:
    - The first trackbar "Element" returns either **erosion_elem** or **dilation_elem**
    - The second trackbar "Kernel size" return **erosion_size** or **dilation_size** for the corresponding operation.
4. Call once erosion and dilation to show the initial image.

Every time we move any slider, the user's function **Erosion** or **Dilation** will be called and it will update the output image based on the current trackbar values.

Let's analyze these two functions:

The erosion function

```
void Erosion( int, void* )
{
  int erosion_type = 0;
  if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
  else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
  else if( erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }

  Mat element = getStructuringElement( erosion_type,
                       Size( 2*erosion_size + 1, 2*erosion_size+1 ),
                       Point( erosion_size, erosion_size ) );

  erode( src, erosion_dst, element );
  imshow( "Erosion Demo", erosion_dst );
}
```

The function that performs the *erosion* operation is **cv::erode** . As we can see, it receives three arguments:

- *src*: The source image
- *erosion_dst*: The output image
- *element*: This is the kernel we will use to perform the operation. If we do not specify, the default is a simple `3x3` matrix. Otherwise, we can specify its shape. For this, we need to use the function **cv::getStructuringElement** :

  ```
  Mat element = getStructuringElement( erosion_type,
                       Size( 2*erosion_size + 1, 2*erosion_size+1 ),
                       Point( erosion_size, erosion_size ) );
  ```

  We can choose any of three shapes for our kernel:

  - Rectangular box: MORPH_RECT
  - Cross: MORPH_CROSS
  - Ellipse: MORPH_ELLIPSE

  Then, we just have to specify the size of our kernel and the *anchor point*. If not specified, it is assumed to be in the center.

That is all. We are ready to perform the erosion of our image.

The dilation function

The code is below. As you can see, it is completely similar to the snippet of code for **erosion**. Here we also have the option of defining our kernel, its anchor point and the size of the operator to be used.

```
void Dilation( int, void* )
{
  int dilation_type = 0;
  if( dilation_elem == 0 ){ dilation_type = MORPH_RECT; }
  else if( dilation_elem == 1 ){ dilation_type = MORPH_CROSS; }
  else if( dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

  Mat element = getStructuringElement( dilation_type,
                       Size( 2*dilation_size + 1, 2*dilation_size+1 ),
                       Point( dilation_size, dilation_size ) );

  dilate( src, dilation_dst, element );
  imshow( "Dilation Demo", dilation_dst );
}
```

Note

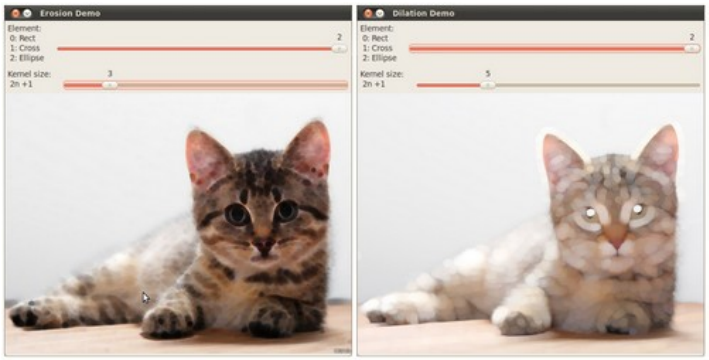
Additionally, there are further parameters that allow you to perform multiple erosions/dilations (iterations) at once and also set the border type and value. However, We haven't used those in this simple tutorial. You can check out the reference for more details.

Results

Compile the code above and execute it (or run the script if using python) with an image as argument. If you do not provide an image as argument the default sample image (LinuxLogo.jpg) will be used.

For instance, using this image:

We get the results below. Varying the indices in the Trackbars give different output images, naturally. Try them out! You can even try to add a third Trackbar to control the number of iterations.



(depending on the programming language the output might vary a little or be only 1 window)