

Module 1.4.1

OpenCV Basics

Binary Image Processing Part 1

Satya Mallick, Ph.D.

June 19, 2017

Table of Contents

Binary Image Processing	2
Thresholding	2
OpenCV Thresholding Code	4
Dilation and Erosion	7
Dilation and Erosion in OpenCV	8
Code (Dilation)	9
Code (Erosion)	13
References and Further Reading	17

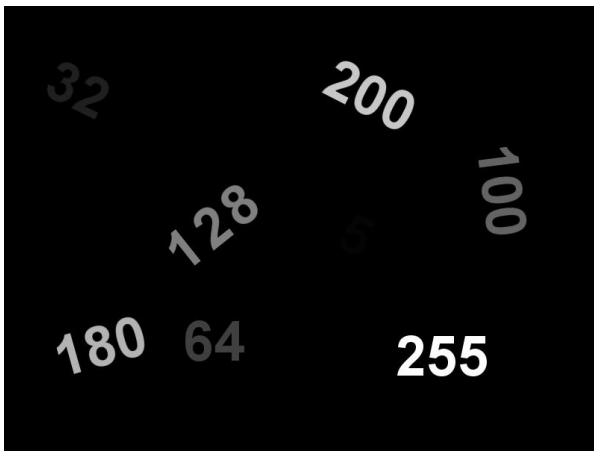
Binary Image Processing

In many computer vision tasks, we need to process a binary image that consists of pixels that are either completely black (pixel value 0) or completely white (pixel value 255). In this section, we will consider three different binary image processing algorithms

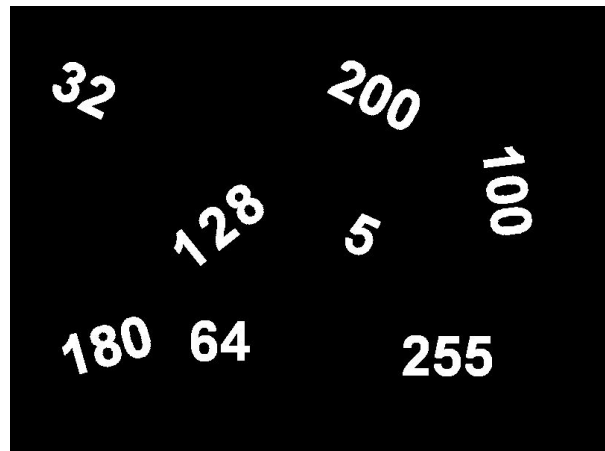
1. **Thresholding** : Used to create a binary image from a grayscale image.
2. **Erosion**: Used to shrink the size of blobs (a group of connected pixels) in a binary image.
3. **Dilation**: Used to expand the size of blobs in a binary image.
4. **Connected Component Analysis**: Used to label blobs in a binary image.

Thresholding

Thresholding is a simple operation that converts a grayscale image into a binary image based on the intensity of pixels.



Original Image



Thresholded Image

In the above, consider the image on the left. The image contains numbers written with intensity (grayscale value) equal to the number itself. E.g. the pixel values of the number 200 is 200, and that of 32 is 32.

Look hard in the left image. Do you see the number 32? How about the number 5? Now check out the thresholded image on the right. Oh yeah! All the numbers pop out nicely. Reading numbers in the thresholded image is way easier than reading numbers in the original image. Not

surprisingly a text recognition algorithm will find the thresholded image in our example much easier to process than the original image.

In Computer Vision when you make a task easier for humans, you typically make it easier for computer algorithms as well.

All thresholding algorithms take a source image (**src**) and a threshold value (**thresh**) as input and produce an output image (**dst**) by comparing the pixel value at source pixel (**x** , **y**) to the threshold. If **src (x , y) > thresh** , then **dst (x , y)** is assigned a some value. Otherwise **dst (x , y)** is assigned a different value.

In its simplest form of thresholding is called **Binary Thresholding**. In addition to the source image (**src**) and threshold value (**thresh**), it takes another input parameter called maximum value (**maxValue**). At each pixel location (**x** , **y**) it compares the pixel value **src(x , y)** to **thresh**. If **src(x , y)** is greater than **thresh**, it sets the value of the destination image pixel **dst(x , y)** to **maxValue**, otherwise it sets it to zero. Here is what the pseudo code looks like

Pseudo Code [Threshold]

```
if src(x,y) > thresh
    dst(x,y) = maxValue
else
    dst(x,y) = 0
```

More generally, there are many types of thresholding based on different threshold rules applied to **src (x , y)** to get **dst (x , y)**. You can see a very detailed description of various thresholding options in this [post](#), but we have not included them in this document because they are rarely used.

OpenCV Thresholding Code

In OpenCV, thresholding is accomplished using a function called **threshold**. The code below shows the usage of Binary Thresholding. In addition, we have provided demo code for different kinds of thresholding operations in the course code base.

Let us walk through the code step by step to understand usage.

C++ [Binary Thresholding] [thresholding.cpp]

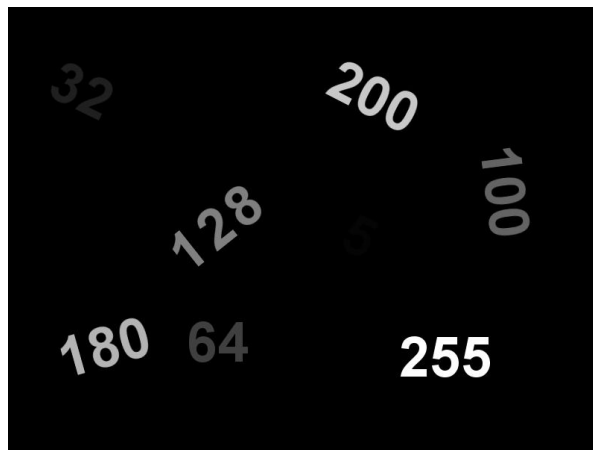
```
1 // Import OpenCV header library
2 #include<opencv2/opencv.hpp>
3 using namespace cv;
4
5 int main(void) {
6     // Read an image in grayscale
7     Mat src = imread("../data/images/threshold.png", IMREAD_GRAYSCALE);
8     Mat dst;
9     // Set threshold and maximum values
10    double thresh = 0;
11    double maxValue = 255;
```

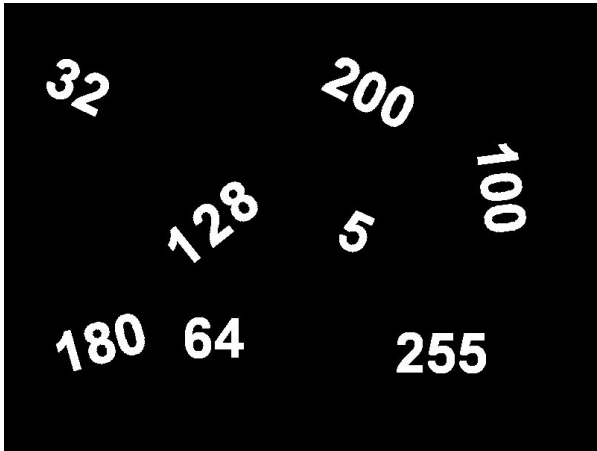
In this part of the program, we perform the Binary Thresholding operation. We have already defined the thresholding value in variable 'thresh' and also set the maximum value in variable 'maxVal'. Once we perform Binary thresholding, all pixel values greater than 'thresh' are changed to 'maxVal' and all pixel values less than 'thresh' changed to 0.

```
12 // Binary threshold
13 threshold(src,dst, thresh, maxValue, THRESH_BINARY);
```

At the end we display both the images on the screen.

```
14 // Display images
15 imshow("Original Image", src);
```



Original Image	
<pre>16 imshow("Thresholded Image", dst);</pre>	
	
Thresholded Image	
<pre>17 waitKey(); 18 // Terminate code 19 return EXIT_SUCCESS; 20 }</pre>	

Python [Binary Thresholding] [thresholding.py]

```
1 # Import opencv
2 import cv2
3 # Read an image in grayscale
4 src = cv2.imread("../data/images/threshold.png", cv2.IMREAD_GRAYSCALE)
5 # Set threshold and maximum value
6 thresh = 0
7 maxValue = 255
```

In this part of the program, we perform the Binary Thresholding operation. We have already defined the thresholding value in variable 'thresh' and also set the maximum value in variable 'maxVal'. Once we perform Binary thresholding, all pixel values greater than 'thresh' are changed to 'maxVal' and all pixel values less than 'thresh' changed to 0.

```
8 # Binary threshold
9 th, dst = cv2.threshold(src, thresh, maxValue, cv2.THRESH_BINARY)
```

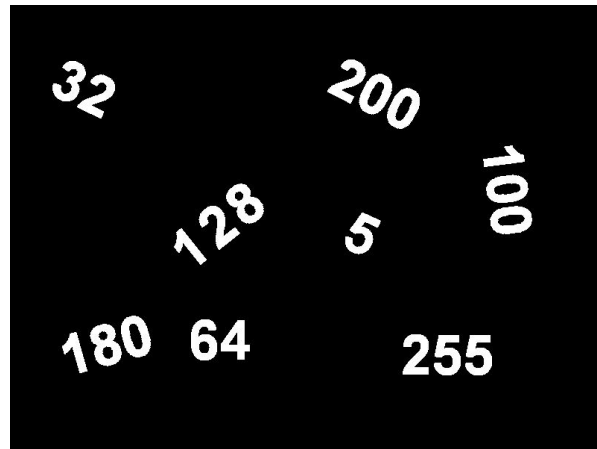
At the end we display both the images on the screen.

```
10 # Display images
11 cv2.imshow("Original Image", src);
```



Original Image

```
12 cv2.imshow("Thresholded Image", dst);
```



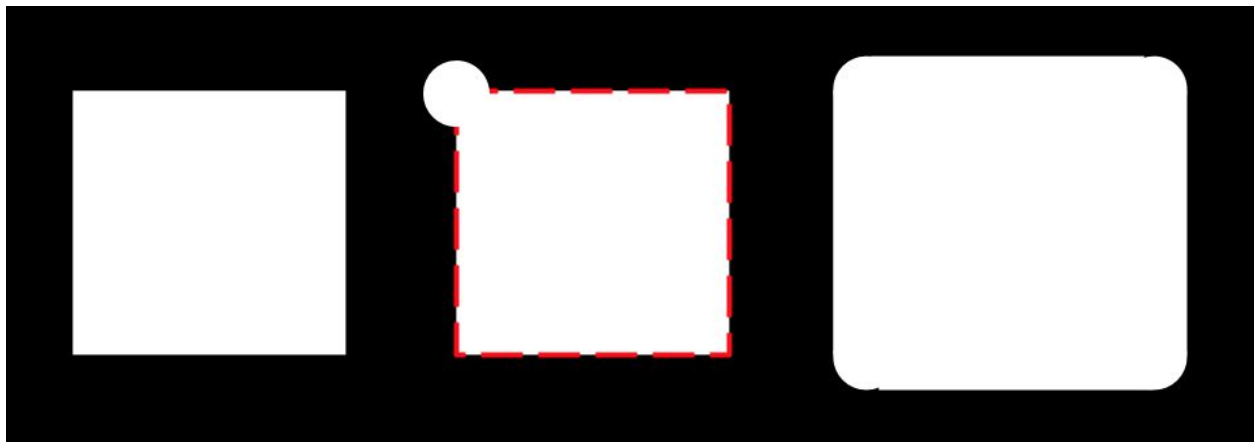
Thresholded Image

```
13 cv2.waitKey()
```

Dilation and Erosion

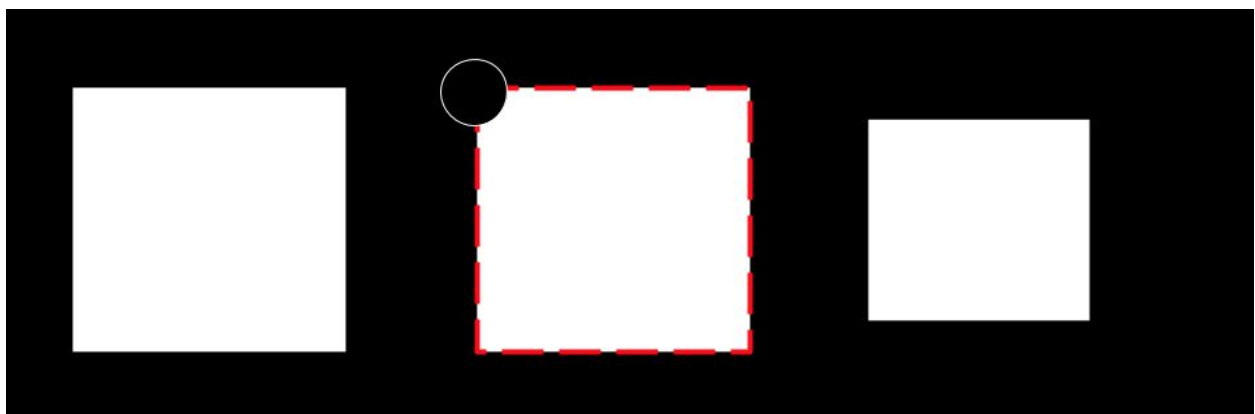
Dilation and Erosion operations on a binary image are used to either expand (dilate) or shrink (erode) an image at the boundary.

The easiest way to understand this concept is using pictures. Consider a white rectangular blob shown in the image below. We want to dilate the image using a circular **structuring element**.



To understand dilation, we first place the center of the circular structuring element on any boundary pixel of the rectangular blob as shown in the center image. We then move the structuring element along the boundary of the rectangular blob as indicated by the dotted red line. As the structuring element moves along the boundary it expands (or dilates) the rectangular blob resulting in the image on the right.

Erosion is the opposite of dilation. Instead of adding, it removes white pixels at the boundary. In the example below, the image on the right is the result of dilating a rectangular blob with a circular structuring element.



Dilation and Erosion in OpenCV

Dilation and Erosion operations are achieved by using **dilate** and **erode** functions of OpenCV.

Dilation:

C++: `void dilate(Mat initialImage, Mat DilatedImage, Mat structuringElement, int iterations=1)`

Python: `DilatedImage=cv2.dilate(initialImage, structuringElement, iterations=1)`

Erosion:

C++: `void erode(Mat initialImage, Mat ErodedImage, Mat structuringElement, int iterations=1)`

Python: `ErodedImage=cv2.erode(initialImage, structuringElement, iterations=1)`

Note: In the functions above, the parameter 'iterations' is optional and if not mentioned default is taken as 1. In case, we need to run the dilate/erode function n number of times we specify "iterations = n" in the function parameter list.

The C++ and Python implementation follows.

Code (Dilation)

C++ [Dilation] [dilation.cpp]

```
1 // Import OpenCV header library
2 #include <iostream>
3 #include <opencv2/highgui.hpp>
4 #include <opencv2/imgproc.hpp>
5 using namespace cv;
6 using namespace std;
7 int main() {
8
9     Mat image;
```

In this program, we read an image from the images directory, and store it in 'Mat image', then perform Dilation operation on the image. Finally the Dilated image is stored in 'Mat imageDilated'. At the end, both the images are displayed.

```
10 // Image read in and stored in Mat image
11 image = imread("../data/images/truth.jpg", IMREAD_COLOR);
```

We create the kernel/structuring element that is used for dilation operation.

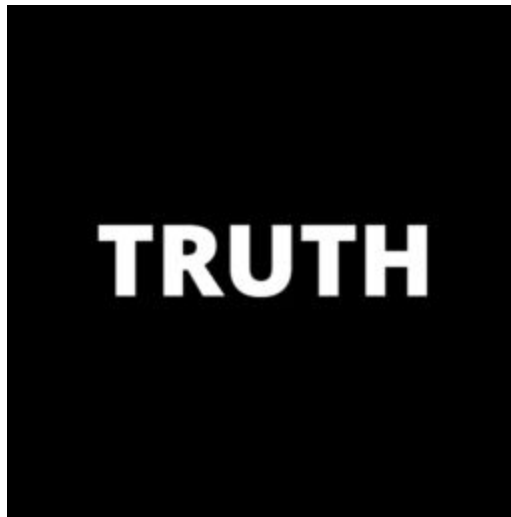
```
12 // Create a structuring element
13 int dilationSize = 6;
14 Mat element = getStructuringElement(cv::MORPH_CROSS,
15     cv::Size(2 * dilationSize + 1, 2 * dilationSize + 1),
16     cv::Point(dilationSize, dilationSize));
17 // Dilated image stored in Mat imageDilated
18 Mat imageDilated;
```

Dilation operation performed on input image and stored in 'Mat imageDilated'.

```
19 // Dilation will increase brightness.
20 dilate(image, imageDilated, element);
```

At the end we display both the images on the screen.

```
21 // Display original image
22 namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
23 imshow("Original Image", image);
```



Original Image

```
24 // Display dilated image
25 namedWindow("Dilation", CV_WINDOW_AUTOSIZE);
26 imshow("Dilation", imageDilated);
```



Dilated Image

```
27 // Wait for the user to press any key
28 waitKey(0);
29 return 0;
30 }
```

Python [Dilation] [dilation.py]

```
1 # Import OpenCV and numpy
2 import cv2
```

In this program, we read an image from the images directory, and store it in 'NumPy array image', then perform Dilation operation on the image. Finally the Dilated image is stored in 'Numpy array imageDilated'. At the end, both the images are displayed.

```
3 # Image taken as input
4 imageName = "../../data/images/truth.png"
5 image = cv2.imread(imageName, cv2.IMREAD_COLOR)
6 # Check for invalid input
7 if image is None:
8     print("Could not open or find the image")
```

We create the kernel/structuring element that is used for dilation operation.

```
9 # Get structuring element/kernel which will be used for dilation
10 dilationSize = 6
11 element = cv2.getStructuringElement(cv2.MORPH_CROSS, (2*dilationSize+1, 2*dilationSize+1),
                                     (dilationSize, dilationSize))
```

Dilation operation performed on input image and stored in 'NumPy array imageDilated'.

```
12 # Dilating the image , makes the image brighter
13 imageDilated = cv2.dilate(image, element)
```

At the end we display both the images on the screen.

```
14 # Display original image
15 cv2.namedWindow("Original Image", cv2.WINDOW_AUTOSIZE)
16 cv2.imshow("Original Image", image)
```



Original Image

```
17 # Display dilated image
18 cv2.namedWindow("Dilation", cv2.WINDOW_AUTOSIZE)
19 cv2.imshow("Dilation", imageDilated)
```



Dilated Image

```
20 # Wait for the user to press any key
21 cv2.waitKey(0)
```

Code (Erosion)

C++ [Erosion] [erosion.cpp]

```
1 // Import OpenCV header library
2 #include <iostream>
3 #include <opencv2/highgui.hpp>
4 #include <opencv2/imgproc.hpp>
5 using namespace cv;
6 using namespace std;
7 int main() {
8
9     Mat image;
```

In this program, we read an image from the images directory, and store it in 'Mat image', then perform Erosion operation on the image. Finally the Eroded image is stored in 'Mat imageEroded'. At the end, both the images are displayed.

```
10 // Image read in and stored in Mat image
11 image = imread("../data/images/truth.jpg", IMREAD_COLOR);
```

We create the kernel/structuring element that is used for erosion operation.

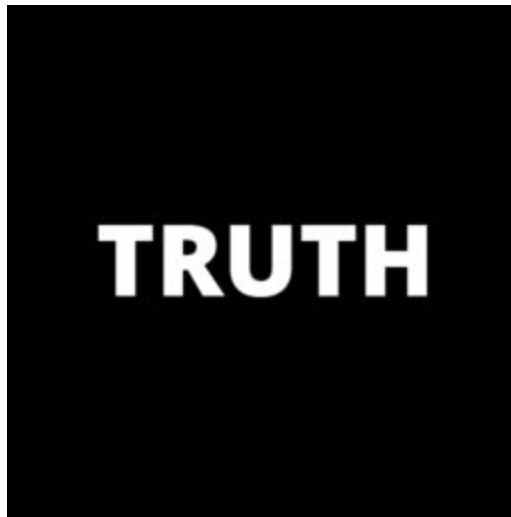
```
12 // Create a structuring element
13 int erosionSize = 6;
14 Mat element = getStructuringElement(cv::MORPH_CROSS,
15     cv::Size(2 * erosionSize + 1, 2 * erosionSize + 1),
16     cv::Point(erosionSize, erosionSize));
17 // Eroded image stored in Mat imageEroded
18 Mat imageEroded;
```

Erosion operation performed on input image and stored in 'Mat imageEroded'.

```
19 // Erosion will decrease brightness.
20 erode(image, imageEroded, element);
```

At the end we display both the images on the screen.

```
21 // Display original image
22 namedWindow("Original Image", CV_WINDOW_AUTOSIZE);
23 imshow("Original Image", image);
```



Original Image

```
24 // Display eroded image
25 namedWindow("Erosion", CV_WINDOW_AUTOSIZE);
26 imshow("Erosion", imageEroded);
```



Eroded Image

```
27 // Wait for the user to press any key
28 waitKey(0);
29 return 0;
30 }
```

Python [Erosion] [erosion.py]

```
1 # Import OpenCV and numpy
2 import cv2
```

In this program, we read an image from the images directory, and store it in 'NumPy array image', then perform Erosion operation on the image. Finally the Eroded image is stored in 'NumPy array imageEroded'. At the end, both the images are displayed.

```
3 # Image taken as input
4 imageName = "../../data/images/truth.jpg"
5 image = cv2.imread(imageName, cv2.IMREAD_COLOR)
6 # Check for invalid input
7 if image is None:
8     print("Could not open or find the image")
```

We create the kernel/structuring element that is used for erosion operation.

```
10 # Get structuring element/kernel which will be used for erosion
11 erosionSize = 6
12 element = cv2.getStructuringElement(cv2.MORPH_CROSS, (2*erosionSize+1, 2*erosionSize+1),
                                     (erosionSize, erosionSize))
```

Erosion operation performed on input image and stored in 'NumPy array imageEroded'.

```
13 # Eroding the image , decreases brightness of image
14 imageEroded = cv2.erode(image, element)
```

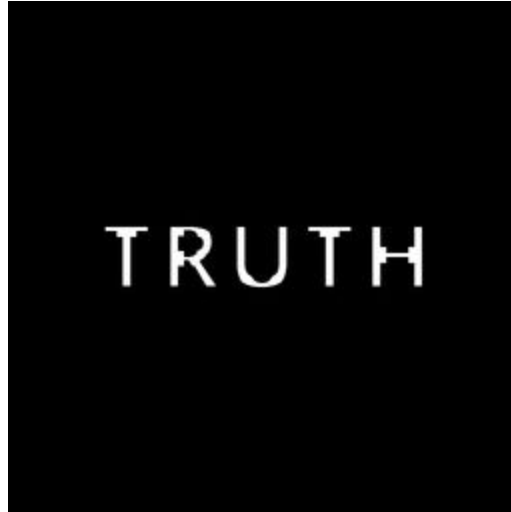
At the end we display both the images on the screen.

```
15 # Display original image
16 cv2.namedWindow("Original Image", cv2.WINDOW_AUTOSIZE)
17 cv2.imshow("Original Image", image)
```



Original Image


```
18 # Display eroded image
19 cv2.namedWindow("Erosion", cv2.WINDOW_AUTOSIZE)
20 cv2.imshow("Erosion", imageEroded)
```



Eroded Image

```
21 # Wait for the user to press any key
22 cv2.waitKey(0)
```

References and Further Reading

<http://www.learnopencv.com/opencv-threshold-python-cpp/>

http://docs.opencv.org/3.2.0/d7/d4d/tutorial_py_thresholding.html

<http://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/threshold/threshold.html>

http://docs.opencv.org/3.0-beta/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

■ ■ ■