**Morphological Transformations**

**Goal**

In this chapter,

* We will learn different morphological operations like Erosion, Dilation, Opening, Closing etc.
* We will see different functions like : **[cv.erode()](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html" \l "gaeb1e0c1033e3f6b891a25d0511362aeb" \o "Erodes an image by using a specific structuring element. )**, **[cv.dilate()](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html" \l "ga4ff0f3318642c4f469d0e11f242f3b6c" \o "Dilates an image by using a specific structuring element. )**, **[cv.morphologyEx()](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html" \l "ga67493776e3ad1a3df63883829375201f" \o "Performs advanced morphological transformations. )** etc.

**Theory**

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play. We will see them one-by-one with help of following image:



**image**

**1. Erosion**

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Here, as an example, I would use a 5x5 kernel with full of ones. Let's see it how it works:

import cv2 as cv

import numpy as np

img = [cv.imread](https://docs.opencv.org/trunk/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56)('j.png',0)

kernel = np.ones((5,5),np.uint8)

erosion = [cv.erode](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gaeb1e0c1033e3f6b891a25d0511362aeb)(img,kernel,iterations = 1)

Result:



**image**

**2. Dilation**

It is just opposite of erosion. Here, a pixel element is '1' if atleast one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

dilation = [cv.dilate](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c)(img,kernel,iterations = 1)

Result:



**image**

**3. Opening**

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, **[cv.morphologyEx()](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html" \l "ga67493776e3ad1a3df63883829375201f" \o "Performs advanced morphological transformations. )**

opening = [cv.morphologyEx](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f)(img, cv.MORPH\_OPEN, kernel)

Result:



**image**

**4. Closing**

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

closing = [cv.morphologyEx](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f)(img, cv.MORPH\_CLOSE, kernel)

Result:



**image**

**5. Morphological Gradient**

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object.

gradient = [cv.morphologyEx](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f)(img, cv.MORPH\_GRADIENT, kernel)

Result:



**image**

**6. Top Hat**

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

tophat = [cv.morphologyEx](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f)(img, cv.MORPH\_TOPHAT, kernel)

Result:



**image**

**7. Black Hat**

It is the difference between the closing of the input image and input image.

blackhat = [cv.morphologyEx](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#ga67493776e3ad1a3df63883829375201f)(img, cv.MORPH\_BLACKHAT, kernel)

Result:



**image**

**Structuring Element**

We manually created a structuring elements in the previous examples with help of Numpy. It is rectangular shape. But in some cases, you may need elliptical/circular shaped kernels. So for this purpose, OpenCV has a function, **[cv.getStructuringElement()](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html" \l "gac342a1bb6eabf6f55c803b09268e36dc" \o "Returns a structuring element of the specified size and shape for morphological operations. )**. You just pass the shape and size of the kernel, you get the desired kernel.

# Rectangular Kernel

>>> [cv.getStructuringElement](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gac342a1bb6eabf6f55c803b09268e36dc)(cv.MORPH\_RECT,(5,5))

array([[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel

>>> [cv.getStructuringElement](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gac342a1bb6eabf6f55c803b09268e36dc)(cv.MORPH\_ELLIPSE,(5,5))

array([[0, 0, 1, 0, 0],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel

>>> [cv.getStructuringElement](https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html#gac342a1bb6eabf6f55c803b09268e36dc)(cv.MORPH\_CROSS,(5,5))

array([[0, 0, 1, 0, 0],

[0, 0, 1, 0, 0],

[1, 1, 1, 1, 1],

[0, 0, 1, 0, 0],

[0, 0, 1, 0, 0]], dtype=uint8)

EnumeratorMORPH\_ERODE

It makes the kernel region black if one of them is black

Python: cv.MORPH\_ERODE

see [**erode**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#gaeb1e0c1033e3f6b891a25d0511362aeb)

MORPH\_DILATE

It makes the kernel region white if one of them is white

Python: cv.MORPH\_DILATE

see [**dilate**](https://docs.opencv.org/3.4/d4/d86/group__imgproc__filter.html#ga4ff0f3318642c4f469d0e11f242f3b6c)

MORPH\_OPEN

Python: cv.MORPH\_OPEN

It removes false positives i.e white pixels here and there

an opening operation

dst=open(src,element)=dilate(erode(src,element))

MORPH\_CLOSE

It removes false negatives i.e. black pixels here and there

Python: cv.MORPH\_CLOSE

a closing operation

dst=close(src,element)=erode(dilate(src,element))

MORPH\_GRADIENT

Python: cv.MORPH\_GRADIENT

a morphological gradient

dst=morph\_grad(src,element)=dilate(src,element)−erode(src,element)

MORPH\_TOPHAT

Python: cv.MORPH\_TOPHAT

"top hat"

dst=tophat(src,element)=src−open(src,element)

MORPH\_BLACKHAT

Python: cv.MORPH\_BLACKHAT

"black hat"

dst=blackhat(src,element)=close(src,element)−src

MORPH\_HITMISS

Python: cv.MORPH\_HITMISS

"hit or miss" .- Only supported for CV\_8UC1 binary images. A tutorial can be found in the documentation

## dilate[¶](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "dilate" \o "Permalink to this headline)

**Python:**cv2.**dilate**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

**C:**void **cvDilate**(const CvArr\* **src**, CvArr\* **dst**, IplConvKernel\* **element**=NULL, int **iterations**=1 )

**Python:**cv.**Dilate**(src, dst, element=None, iterations=1) → None

|  |  |
| --- | --- |
| **Parameters:** | * **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV\_8U, CV\_16U, CV\_16S, CV\_32F` or ``CV\_64F. * **dst** – output image of the same size and type as src. * **element** – structuring element used for dilation; if element=Mat() , a 3 x 3 rectangular structuring element is used. * **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center. * **iterations** – number of times dilation is applied. * **borderType** – pixel extrapolation method (see **[borderInterpolate()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "int%20borderInterpolate(int%20p,%20int%20len,%20int%20borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details). * **borderValue** – border value in case of a constant border (see **[createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "Ptr%3CFilterEngine%3E%20createMorphologyFilter(int%20op,%20int%20type,%20InputArray%20kernel,%20Point%20anchor,%20int%20rowBorderType,%20int%20columnBorderType,%20const%20Scalar&%20borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))** for details). |

The function dilates the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum is taken:

\texttt{dst} (x,y) =  \max _{(x',y'):  \, \texttt{element} (x',y') \ne0 } \texttt{src} (x+x',y+y')

The function supports the in-place mode. Dilation can be applied several ( iterations ) times. In case of multi-channel images, each channel is processed independently.

**See also**

[**erode()**](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate#void%20erode(InputArray%20src,%20OutputArray%20dst,%20InputArray%20kernel,%20Point%20anchor,%20int%20iterations,%20int%20borderType,%20const%20Scalar&%20borderValue)), **[morphologyEx()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "void%20morphologyEx(InputArray%20src,%20OutputArray%20dst,%20int%20op,%20InputArray%20kernel,%20Point%20anchor,%20int%20iterations,%20int%20borderType,%20const%20Scalar&%20borderValue)" \o "void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor, int iterations, int borderType, const Scalar& borderValue))**, **[createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "Ptr%3CFilterEngine%3E%20createMorphologyFilter(int%20op,%20int%20type,%20InputArray%20kernel,%20Point%20anchor,%20int%20rowBorderType,%20int%20columnBorderType,%20const%20Scalar&%20borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))**

**Note**

* An example using the morphological dilate operation can be found at opencv\_source\_code/samples/cpp/morphology2.cpp

erode

Erodes an image by using a specific structuring element.

**C++:**void **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER\_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue() )

**Python:**cv2.**erode**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

**C:**void **cvErode**(const CvArr\* **src**, CvArr\* **dst**, IplConvKernel\* **element**=NULL, int **iterations**=1)

**Python:**cv.**Erode**(src, dst, element=None, iterations=1) → None

|  |  |
| --- | --- |
| **Parameters:** | * **src** – input image; the number of channels can be arbitrary, but the depth should be one of CV\_8U, CV\_16U, CV\_16S, CV\_32F` or ``CV\_64F. * **dst** – output image of the same size and type as src. * **element** – structuring element used for erosion; if element=Mat() , a 3 x 3 rectangular structuring element is used. * **anchor** – position of the anchor within the element; default value (-1, -1) means that the anchor is at the element center. * **iterations** – number of times erosion is applied. * **borderType** – pixel extrapolation method (see **[borderInterpolate()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "int%20borderInterpolate(int%20p,%20int%20len,%20int%20borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details). * **borderValue** – border value in case of a constant border (see **[createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "Ptr%3CFilterEngine%3E%20createMorphologyFilter(int%20op,%20int%20type,%20InputArray%20kernel,%20Point%20anchor,%20int%20rowBorderType,%20int%20columnBorderType,%20const%20Scalar&%20borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))** for details). |

The function erodes the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum is taken:

\texttt{dst} (x,y) =  \min _{(x',y'):  \, \texttt{element} (x',y') \ne0 } \texttt{src} (x+x',y+y')

The function supports the in-place mode. Erosion can be applied several ( iterations ) times. In case of multi-channel images, each channel is processed independently.

morphologyEx

Performs advanced morphological transformations.

**C++:**void **morphologyEx**(InputArray **src**, OutputArray **dst**, int **op**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER\_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue() )

**Python:**cv2.**morphologyEx**(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

**C:**void **cvMorphologyEx**(const CvArr\* **src**, CvArr\* **dst**, CvArr\* **temp**, IplConvKernel\* **element**, int **operation**, int **iterations**=1 )

**Python:**cv.**MorphologyEx**(src, dst, temp, element, operation, iterations=1) → None

|  |  |
| --- | --- |
| **Parameters:** | * **src** – Source image. The number of channels can be arbitrary. The depth should be one of CV\_8U, CV\_16U, CV\_16S, CV\_32F or CV\_64F. * **dst** – Destination image of the same size and type as src . * **element** – Structuring element. * **op** –   Type of a morphological operation that can be one of the following:   * + **MORPH\_OPEN** - an opening operation   + **MORPH\_CLOSE** - a closing operation   + **MORPH\_GRADIENT** - a morphological gradient   + **MORPH\_TOPHAT** - “top hat”   + **MORPH\_BLACKHAT** - “black hat”   + **MORPH\_HITMISS** - “hit and miss” * **iterations** – Number of times erosion and dilation are applied. * **borderType** – Pixel extrapolation method. See **[borderInterpolate()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "int%20borderInterpolate(int%20p,%20int%20len,%20int%20borderType)" \o "int borderInterpolate(int p, int len, int borderType))** for details. * **borderValue** – Border value in case of a constant border. The default value has a special meaning. See **[createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "Ptr%3CFilterEngine%3E%20createMorphologyFilter(int%20op,%20int%20type,%20InputArray%20kernel,%20Point%20anchor,%20int%20rowBorderType,%20int%20columnBorderType,%20const%20Scalar&%20borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))** for details. |

The function can perform advanced morphological transformations using an erosion and dilation as basic operations.

Opening operation:

\texttt{dst} = \mathrm{open} ( \texttt{src} , \texttt{element} )= \mathrm{dilate} ( \mathrm{erode} ( \texttt{src} , \texttt{element} ))

Closing operation:

\texttt{dst} = \mathrm{close} ( \texttt{src} , \texttt{element} )= \mathrm{erode} ( \mathrm{dilate} ( \texttt{src} , \texttt{element} ))

Morphological gradient:

\texttt{dst} = \mathrm{morph\_grad} ( \texttt{src} , \texttt{element} )= \mathrm{dilate} ( \texttt{src} , \texttt{element} )- \mathrm{erode} ( \texttt{src} , \texttt{element} )

“Top hat”:

\texttt{dst} = \mathrm{tophat} ( \texttt{src} , \texttt{element} )= \texttt{src} - \mathrm{open} ( \texttt{src} , \texttt{element} )

“Black hat”:

\texttt{dst} = \mathrm{blackhat} ( \texttt{src} , \texttt{element} )= \mathrm{close} ( \texttt{src} , \texttt{element} )- \texttt{src}

“Hit and Miss”: Only supported for CV\_8UC1 binary images. Tutorial can be found in this page: [https://web.archive.org/web/20160316070407/http://opencv-code.com/tutorials/hit-or-miss-transform-in-opencv/](https://web.archive.org/web/20160316070407/http:/opencv-code.com/tutorials/hit-or-miss-transform-in-opencv/)

Any of the operations can be done in-place. In case of multi-channel images, each channel is processed independently.

**See also**

[**dilate()**](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate#void%20dilate(InputArray%20src,%20OutputArray%20dst,%20InputArray%20kernel,%20Point%20anchor,%20int%20iterations,%20int%20borderType,%20const%20Scalar&%20borderValue)), [**erode()**](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate#void%20erode(InputArray%20src,%20OutputArray%20dst,%20InputArray%20kernel,%20Point%20anchor,%20int%20iterations,%20int%20borderType,%20const%20Scalar&%20borderValue)), **[createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=dilate" \l "Ptr%3CFilterEngine%3E%20createMorphologyFilter(int%20op,%20int%20type,%20InputArray%20kernel,%20Point%20anchor,%20int%20rowBorderType,%20int%20columnBorderType,%20const%20Scalar&%20borderValue)" \o "Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue))**

**Note**

* An example using the morphologyEx function for the morphological opening and closing operations can be found at opencv\_source\_code/samples/cpp/morphology2.cpp