Морфологические трансформации

Цель

В этой главе,

- Мы изучим различные морфологические операции, такие как Erosion, Dilation, Opening, Closing и т. Д.
- Мы увидим различные функции, такие как: cv2.erode () , cv2.dilate () , cv2.morphologyEx () и т. Д.

Теория

Морфологические преобразования - это некоторые простые операции, основанные на форме изображения. Обычно он выполняется на двоичных изображениях. Он нуждается в двух входах, один из которых является нашим исходным изображением, второй - **структурирующим элементом** или **ядром**, которое определяет характер работы. Двумя основными морфологическими операциями являются Эрозия и Дилатация. Тогда его варианты формы, такие как Открытие, Закрытие, Градиент и т. Д. Также вступают в игру. Мы увидим их один за другим с помощью следующего изображения:

1. Эрозия

Основная идея эрозии - это только эрозия почв, она разрушает границы объекта переднего плана (всегда старайтесь держать передний план в белом). Так что же он делает? Ядро скользит по изображению (как в 2D свертке). Пиксель в исходном изображении (1 или 0) будет считаться 1, только если все пиксели под ядром равны 1, в противном случае он будет разрушен (сделан до нуля).

Итак, что происходит, все пиксели около границы будут отброшены в зависимости от размера ядра. Таким образом, толщина или размер объекта переднего плана уменьшается или просто уменьшается область белого изображения. Это полезно для удаления небольших белых шумов (как мы видели в главе цветового пространства), отсоединить два связанных объекта и т. Д.

Здесь, в качестве примера, я бы использовал ядро 5x5 с полными. Посмотрим, как это работает:

```
import  cv2
import  numpy  как  np

img  =  cv2 . imread ( 'j.png' , 0 )
kernel  =  np . из них (( 5 , 5 ), пр . uint8 )
эрозия  =  CV2 . erode ( img , kernel , iterations  =  1 )
```

Результат:

2. Дилатация

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 = cv2.dilate(img,kernel,iterations = 1)
```

Result:

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, **cv2.morphologyEx()**

```
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

Result:

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 = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

Result:

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 = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
```

Result:

6. Top Hat

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

```
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

Result:

7. Black Hat

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

```
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
```

Result:

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, **cv2.getStructuringElement()**. You just pass the shape and size of the kernel, you get the desired kernel.

```
# Rectangular Kernel
>>> cv2.getStructuringElement(cv2.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
>>> cv2.getStructuringElement(cv2.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
>>> cv2.getStructuringElement(cv2.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)
```

Additional Resources

1. Morphological Operations at HIPR2

Exercises

Help and Feedback

You did not find what you were looking for?

- Ask a question on the Q&A forum.
- If you think something is missing or wrong in the documentation, please file a bug report.