Nama : Fadia Ramadhana
NIM : G64170026

PRAKTIKUM 12 : MORFOLOGI

PENGOLAHAN CITRA DIGITAL (KOM324)

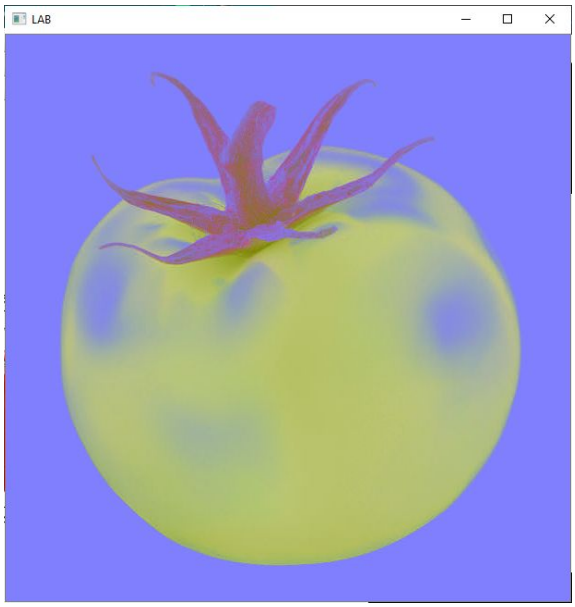
Gambar 1. Citra tomato.jpg

Konversi Color Space

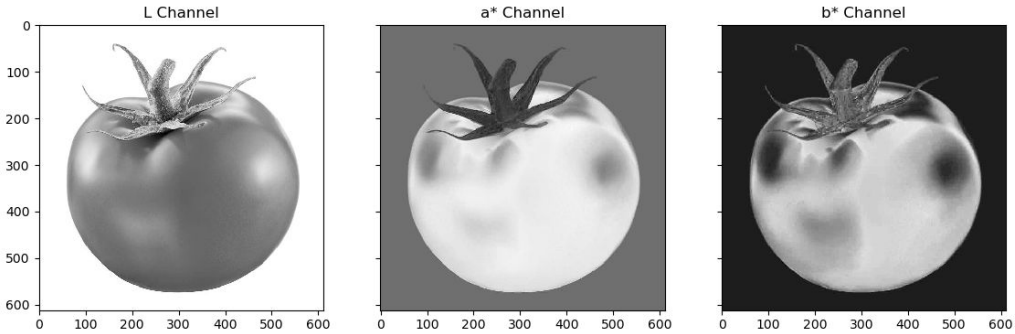
Pada tahapan pertama ini, citra tomato.jpg akan diterapkan konversi colorspace dari RGB menjadi L*a*b*. Colorspace ini dipilih karena dapat memberikan perbedaan antara buah tomat yang merah dengan bagian sepal yang berwarna hijau, yang dimana fokus disini adalah untuk melakukan segmentasi pada buah tomat yang merah saja. Colorspace L*a*b* memiliki 3 buah channel yaitu L* (luminance), a* (green to red), dan b* (blue to yellow).

```
7     #lakukan konversi colorspace yang sesuai
8     imageLAB = cv2.cvtColor(img,cv2.COLOR_BGR2LAB)
9     cv2.imshow("LAB", imageLAB)
10
11    L,a,b = cv2.split(imageLAB)
```

Output :



Gambar 2. Hasil konversi colorspace citra dari RGB ke L*a*b*



Gambar 3. Hasil split channel menjadi L*(1), a*(2), dan b*(3)

Dari hasil splitting channel pada colorspace, dapat dilihat bahwa channel a* lebih cenderung dapat memberikan kita perbedaan antara bagian sepal dengan buah tomat yang berwarna merah. Sehingga, untuk selanjutnya hasil citra dari channel a* yang akan digunakan untuk tahapan selanjutnya.

Thresholding

Pada OpenCV, range value untuk L*a*b* untuk 8 bit-image adalah :

- 0 < **L*** < 255
- 0 < **a*** < 255
- 0 < **b*** < 255

Karena untuk tahapan ini citra yang akan digunakan adalah citra dengan channel a* saja, maka

low threshold dan high threshold akan diatur mengikuti range channel tersebut. Pada kesempatan kali ini, saya mengatur low threshold dengan nilai 136 dan high threshold dengan nilai 255.

Kedua nilai threshold ini kemudian akan digunakan sebagai komponen dalam proses thresholding yang menghasilkan raw mask atau dalam hal ini belum diterapkan proses closing.

```
25  #lakukan thresholding pada citra channel hasil konversi yang sesuai
26  # batas bawah a*
27  min_a = 136
28  # batas atas b*
29  max_a = 255
30  maskLAB = cv2.inRange(a, min_a, max_a)
31  cv2.imshow("Thresholding result", maskLAB)
```

Output :

Gambar 4. Hasil thresholding

Dapat dilihat hasil dari proses thresholding, masih terdapat beberapa noise pada bagian sepal dan pada buah tomat terdapat bagian kecil yang bolong.

Dilasi

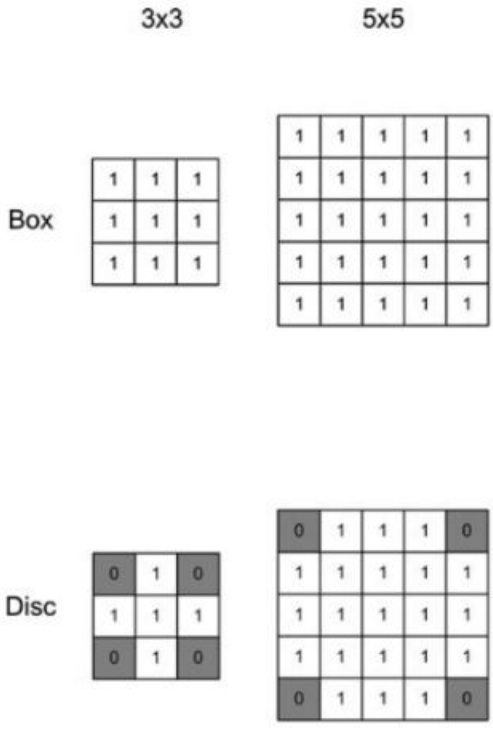
Merupakan proses penggabungan background point 0 menjadi bagian dari objek 1 berdasarkan structuring element (SE) atau kernel yang digunakan. Notasi dilasi dapat ditulis sebagai berikut :

$$D(A,S) = A \oplus S$$

Operasi ini terdiri dari mengkonvolusikan gambar A dengan beberapa kernel (S), yang dapat memiliki bentuk atau ukuran apa pun. Kernel ini memiliki titik jangkar (anchor), yang menunjukkan pusatnya.
Kernel ini melakukan scanning diatas gambar untuk menghitung nilai piksel maksimum. Setelah menghitung, gambar diganti dengan nilai anchor di bagian pusat. Dengan prosedur ini, area dengan nilai 1 (putih) semakin bertambah dan objek dengan nilai 0 (hitam) semakin berkurang.

Pada kode program dibawah, dilasi dilakukan menggunakan fungsi pada OpenCV yaitu cv2.dilate() yang memiliki tiga buah parameter yaitu, image hasil thresholding yang, structuring element (kernel), dan jumlah iterasi. Dicobakan dilasi dengan menggunakan dua jenis kernel, yaitu kernel berukuran 3x3 dan 5x5 dengan masing-masing jumlah iterasi adalah 1.

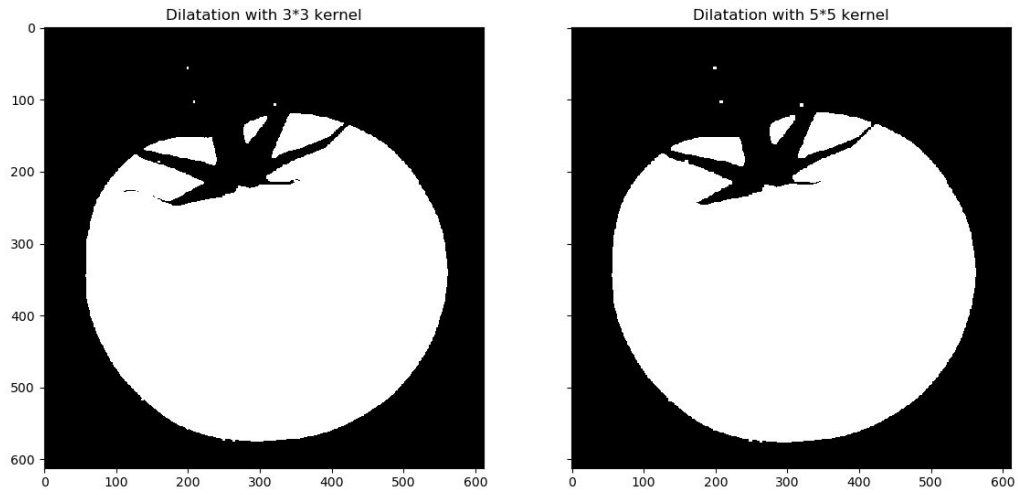
Dalam hal ini, pemilihan size kernel berukuran ganjil agar anchor point dari kernel dapat berada di pusat/tengah kernel. Kernel dengan size genap tidak dapat memiliki anchor point tepat di pusat/tengah kernel.



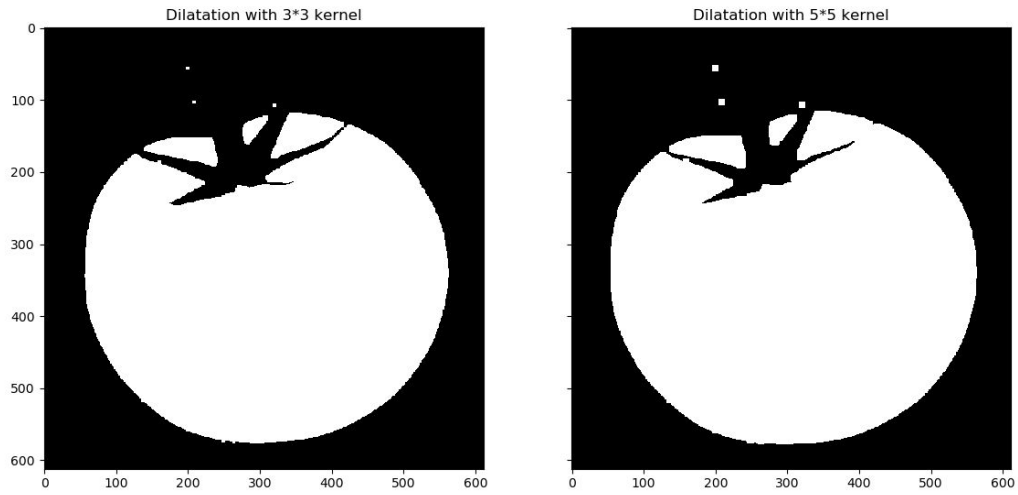
Gambar 5. Structuring Element (Kernel)

```
def kernel(size):
    k = np.ones((size,size), np.uint8)
    return k

#lakukan proses dilasi
dilatation_3 = cv2.dilate(maskLAB, kernel(3), iterations=1)
dilatation_5 = cv2.dilate(maskLAB, kernel(5), iterations=1)
fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(dilatation_3, cmap="gray")
ax[0].set_title('Dilatation with 3*3 kernel')

ax[1].imshow(dilatation_5, cmap="gray")
ax[1].set_title('Dilatation with 5*5 kernel')
```

Output :



Gambar 6. Hasil dilasi menggunakan kernel 3x3 (a) dan hasil dilasi menggunakan kernel 5x5 (b) dengan 1 iterasi



Gambar 7. Hasil dilasi menggunakan kernel 3x3 (a) dan hasil dilasi menggunakan kernel 5x5 (b) dengan 2 iterasi

Dapat dilihat dari hasil pada di atas, semakin besar size dari kernel, maka semakin besar pula wilayah dengan nilai 1 (putih) dan sebaliknya wilayah dengan nilai 0 (hitam) semakin berkurang pada gambar. Begitupun dengan jumlah iterasi dari operasi dilasi, semakin banyak iterasi maka semakin besar pula wilayah dengan nilai 1 (putih) dan sebaliknya wilayah dengan nilai 0 (hitam) semakin berkurang pada gambar.

Untuk tahapan selanjutnya, akan digunakan citra hasil dilasi dengan ukuran kernel 3x3 dan jumlah iterasi 1 seperti pada Gambar 6(a) karena citra tersebut yang paling mendekati dengan citra yang diharapkan.

Erosi

Merupakan proses penghapusan titik-titik objek (1) menjadi bagian dari latar (0), berdasarkan structuring element (SE) atau kernel yang digunakan. Notasi erosi dapat ditulis :

$$E(A,S) = A \otimes S$$

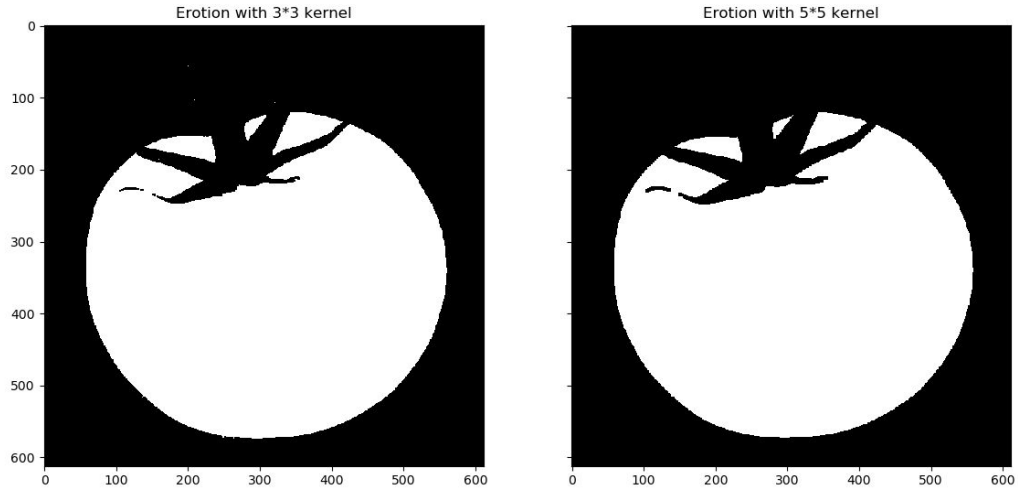
Operasi ini adalah kebalikan dari dilasi. Apa yang dilakukan adalah menghitung minimum lokal di atas area kernel. Ketika kernel S dipindai di atas gambar, operasi menghitung nilai piksel minimal yang *overlap* dengan S dan mengganti piksel gambar di bawah anchor point dengan nilai minimal itu. Sebuah piksel dalam gambar asli (baik 1 atau 0) akan dianggap 1 hanya jika semua piksel dibawah kernel adalah 1, jika tidak maka akan tererosi (dibuat menjadi nol).

Pada kode program dibawah, erosi dilakukan menggunakan fungsi pada OpenCV yaitu cv2.erode() yang memiliki tiga buah parameter yaitu, image hasil thresholding yang, structuring element (kernel), dan jumlah iterasi. Dicobakan erosi dengan menggunakan dua jenis kernel, yaitu kernel berukuran 3x3 dan 5x5 dengan masing-masing jumlah iterasi adalah 1.

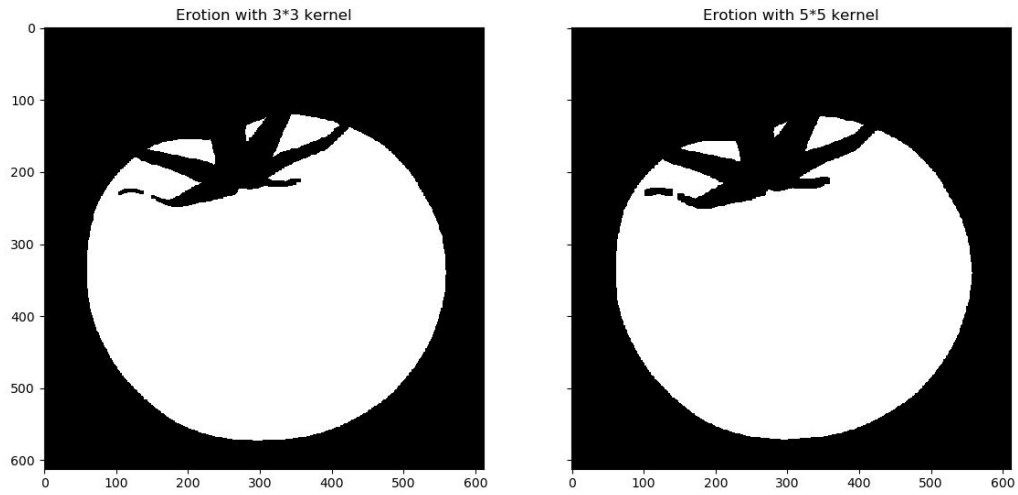
Dalam hal ini, pemilihan size kernel berukuran ganjil agar anchor point dari kernel dapat berada di pusat/tengah kernel. Kernel dengan size genap tidak dapat memiliki anchor point tepat di pusat/tengah kernel.

```
#lakukan proses erosi
erotion_3 = cv2.erode(dilatation_3, kernel(3), iterations=1)
erotion_5 = cv2.erode(dilatation_3, kernel(5), iterations=1)
fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(erotion_3, cmap="gray")
ax[0].set_title('Erotion with 3*3 kernel')

ax[1].imshow(erotion_5, cmap="gray")
ax[1].set_title('Erotion with 5*5 kernel')
```

Output :



Gambar 8. Hasil erosi menggunakan kernel 3x3 (a) dan hasil erosi menggunakan kernel 5x5 (b) dengan 1 iterasi



Gambar 9. Hasil erosi menggunakan kernel 3x3 (a) dan hasil erosi menggunakan kernel 5x5 (b) dengan 2 iterasi

Dapat dilihat dari hasil pada di atas, semakin besar size dari kernel, maka semakin besar pula wilayah dengan nilai 0 (hitam) dan sebaliknya wilayah dengan nilai 1 (putih) semakin berkurang pada gambar. Begitupun dengan jumlah iterasi dari operasi erosi, semakin banyak iterasi maka semakin besar pula wilayah dengan nilai 0 (hitam) dan sebaliknya wilayah dengan nilai 1 (putih) semakin berkurang pada gambar.

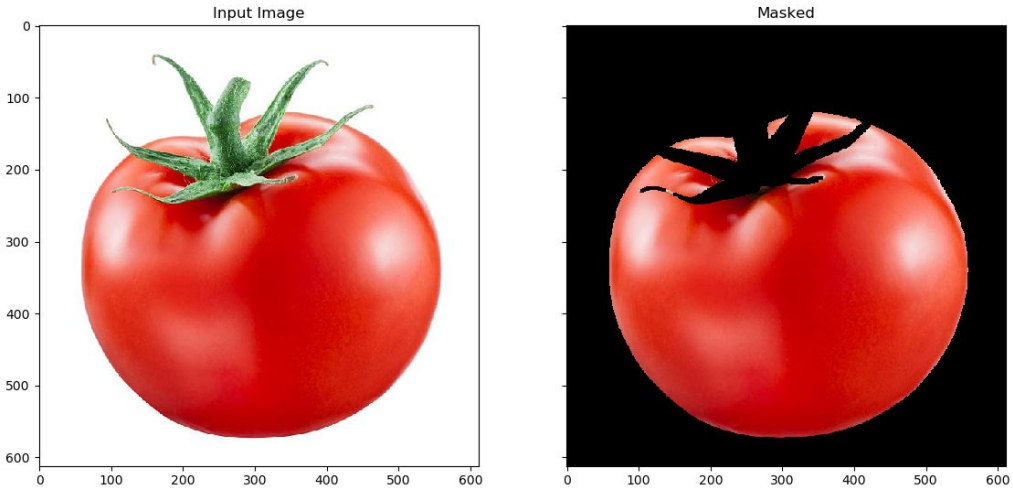
Untuk tahapan selanjutnya, akan digunakan citra hasil erosi dengan ukuran kernel 5x5 dan jumlah iterasi 1 seperti pada Gambar 8(b) karena citra tersebut yang paling mendekati dengan citra yang diharapkan.

Masking

Setelah mendapatkan mask, maka langkah selanjutnya adalah melakukan segmentasi wajah pada gambar asli dengan menggunakan mask yang telah didapatkan pada step sebelumnya dengan memanfaatkan fungsi **cv2.bitwise_and()** yang menjaga setiap piksel dalam gambar yang diberikan jika nilai yang sesuai dalam mask adalah 1. Cara bekerja fungsi ini mirip seperti subtraction, dimana gambar output merupakan hasil gambar hasil irisan dari dua buah gambar.

```
61  #lakukan masking
62  masked = cv2.bitwise_and(img, img, mask = erotion_5)
63  cv2.imshow("masked", masked)
64  fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
65  ax = axes.ravel()
66
67  ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), cmap="gray")
68  ax[0].set_title('Input Image')
69
70  ax[1].imshow(cv2.cvtColor(masked, cv2.COLOR_BGR2RGB), cmap="gray")
71  ax[1].set_title('Masked')
```

Output :



Gambar 10. Input Image (a) dan citra hasil masking (b)

Full Code

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('tomato.jpg')

#lakukan konversi colorspace yang sesuai
imageLAB = cv2.cvtColor(img,cv2.COLOR_BGR2LAB)
cv2.imshow("LAB", imageLAB)

L,a,b = cv2.split(imageLAB)

fig, axes = plt.subplots(1, 3, figsize=(6, 6), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(L, cmap="gray")
ax[0].set_title('L Channel')

ax[1].imshow(a, cmap="gray")
ax[1].set_title('a* Channel')

ax[2].imshow(b, cmap="gray")
ax[2].set_title('b* Channel')

```

```
#lakukan thresholding pada citra channel hasil konversi yang sesuai
# batas bawah a*
min_a = 136
# batas atas b*
max_a = 255
maskLAB = cv2.inRange(a, min_a, max_a)
cv2.imshow("Thresholding result", maskLAB)

def kernel(size):
    k = np.ones((size,size), np.uint8)
    return k

#lakukan proses dilasi
dilatation_3 = cv2.dilate(maskLAB, kernel(3), iterations=1)
dilatation_5 = cv2.dilate(maskLAB, kernel(5), iterations=1)
fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(dilatation_3, cmap="gray")
ax[0].set_title('Dilatation with 3*3 kernel')

ax[1].imshow(dilatation_5, cmap="gray")
ax[1].set_title('Dilatation with 5*5 kernel')
```

```
48
49      #lakukan proses erosi
50      erotion_3 = cv2.erode(dilatation_3, kernel(3), iterations=1)
51      erotion_5 = cv2.erode(dilatation_3, kernel(5), iterations=1)
52      fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
53      ax = axes.ravel()
54
55      ax[0].imshow(erotion_3, cmap="gray")
56      ax[0].set_title('Erotion with 3*3 kernel')
57
58      ax[1].imshow(erotion_5, cmap="gray")
py      ax[1].set_title('Erotion with 5*5 kernel')
60
61      #lakukan masking
62      masked = cv2.bitwise_and(img, img, mask = erotion_5)
63      cv2.imshow("masked", masked)
64      fig, axes = plt.subplots(1, 2, figsize=(6, 6), sharex=True, sharey=True)
65      ax = axes.ravel()
66
67      ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB), cmap="gray")
68      ax[0].set_title('Input Image')
69
70      ax[1].imshow(cv2.cvtColor(masked, cv2.COLOR_BGR2RGB), cmap="gray")
71      ax[1].set_title('Masked')
72
73      plt.show()
74      cv2.waitKey()
```

Referensi

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

https://hub.packtpub.com/opencv-image-processing-using-morphological-filters/

https://medium.com/@himnickson/morphological-operations-in-image-processing-cb8045b98fcc

https://www.geeksforgeeks.org/image-segmentation-using-morphological-operation/

https://docs.opencv.org/3.3.1/d3/db4/tutorial_py_watershed.html