Practice: Image Processing II

COMPUTER VISION (COURSE-HY24011)

Q YOUN HONG

Content

- Linear Filtering
- Non-linear filtering
  - Adding random noise to an image
  - Median filtering
  - Bilateral filtering
- Integral Image
- Image Thresholding
  - Global Thresholding
  - Adaptive Thresholding
  - Otsu’s Algorithm

Example0. Convolution Filtering

- Convolution: computing a pixel value from a collection of neighboring pixels (area-based operation)
  - Convolution applies the flipped window, whereas correlation does not
  - A window u is also called a kernel or a filter

Window u =

2	4	3

Correlation

	0	1	2	3	4	5	6	7	8	9
$f=$	0	0	1	0	2	2	4	3	1	0
×			×	×	×					
			2	4	3					
+				+						
$g=$	-	3	4	8	14	24	29	23	10	-

Convolution

	0	1	2	3	4	5	6	7	8	9
$f=$	0	0	1	0	2	2	4	3	1	0
×			×	×	×					
			3	4	2					
+				+						
$g=$	-	2	4	7	12	22	28	26	13	-

Input $f=$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Window u =

	-1	0	1
-1	1	2	3
0	4	5	6
1	7	8	9

Correlation window:

1	2	3
4	5	6
7	8	9

Convolution window:

9	8	7
6	5	4
3	2	1

Output $g=$

Correlation

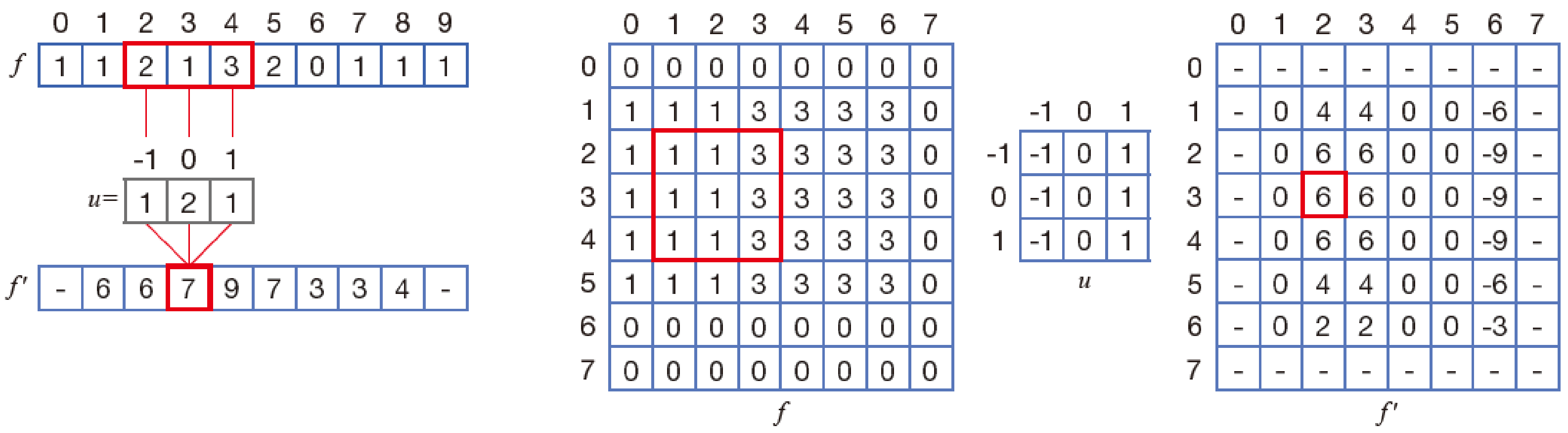
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	9	8	7	0	0	0	0
0	6	5	4	0	0	0	0
0	3	2	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Convolution

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	2	3	0	0	0	0
0	4	5	6	0	0	0	0
0	7	8	9	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

2D Convolution in OpenCV

- Some well-known filters are already implemented as built-in functions in OpenCV, e.g.) Gaussian filter
- Custom filters can be also applied
- Handling boundary pixels – 0 padding or copy padding
- Cannot be implemented in-place



Example0. Convolution Filtering

```python
import cv2 as cv
import numpy as np

img=cv.imread('soccer.jpg')
img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
cv.imshow('Original',gray)

smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),
                  cv.GaussianBlur(gray,(9,9),0.0),
                  cv.GaussianBlur(gray,(15,15),0.0)))
cv.imshow('Smooth',smooth)

femboss=np.array([[-1.0, 0.0, 0.0],
                  [ 0.0, 0.0, 0.0],
                  [ 0.0, 0.0, 1.0]])

gray16=np.int16(gray)
emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
emboss_worse=cv.filter2D(gray,-1,femboss)

cv.imshow('Emboss',emboss)
cv.imshow('Emboss_bad',emboss_bad)
cv.imshow('Emboss_worse',emboss_worse)

cv.waitKey()
cv.destroyAllWindows()
```

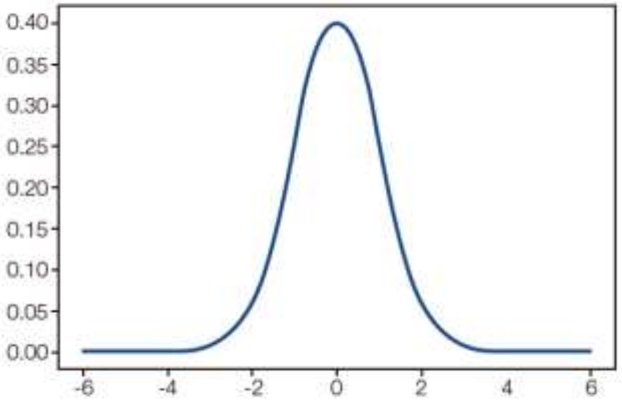
Example0. Convolution Filtering

Execution results

Example0. Convolution Filtering

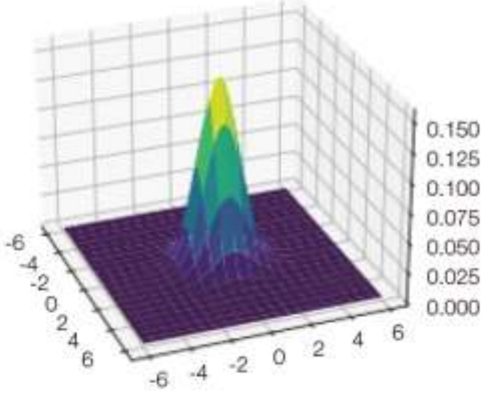
```python
1  import cv2 as cv
2  import numpy as np
3
4  img=cv.imread('soccer.jpg')
5  img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
6  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
7  cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
8  cv.imshow('Original',gray)
9
10 smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),
11                   cv.GaussianBlur(gray,(9,9),0.0),
12                   cv.GaussianBlur(gray,(15,15),0.0)))
13 cv.imshow('Smooth',smooth)
14
15 femboss=np.array([[-1.0, 0.0, 0.0],
16                   [ 0.0, 0.0, 0.0],
17                   [ 0.0, 0.0, 1.0]])
18
19 gray16=np.int16(gray)
20 emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
21 emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
22 emboss_worse=cv.filter2D(gray,-1,femboss)
23
24 cv.imshow('Emboss',emboss)
25 cv.imshow('Emboss_bad',emboss_bad)
26 cv.imshow('Emboss_worse',emboss_worse)
27
28 cv.waitKey()
29 cv.destroyAllWindows()
```

(L10-L12) Apply Gaussian filters of different sizes to img



1D Gaussian

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$



2D Gaussian

$$g(y,x) = \frac{1}{\sigma^2 2\pi} e^{-\frac{y^2+x^2}{2\sigma^2}}$$

Example0. Convolution Filtering

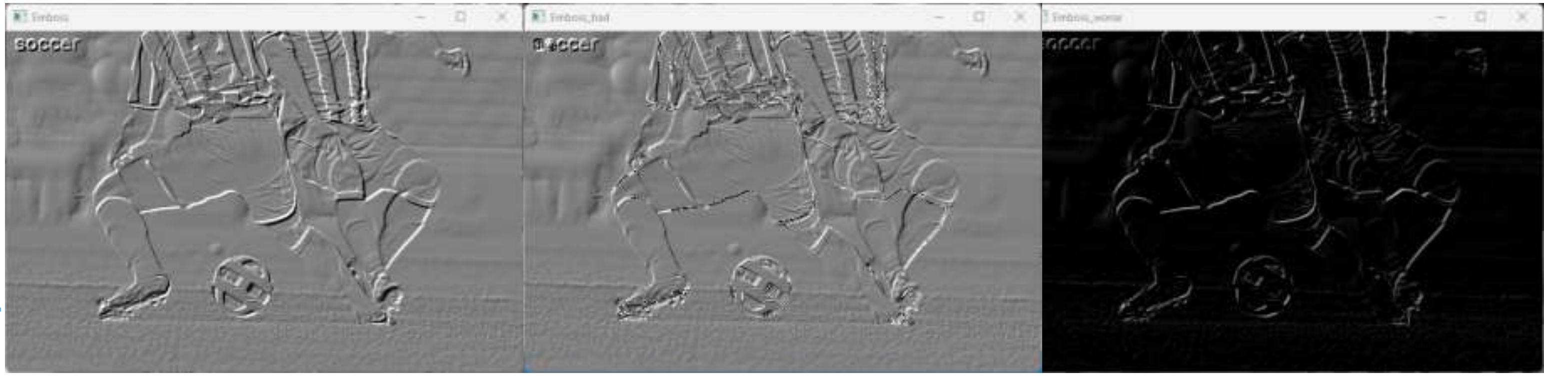

```
1  import cv2 as cv
2  import numpy as np
3
4  img=cv.imread('soccer.jpg')
5  img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
6  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
7  cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.7,(255,255,255),2)
8  cv.imshow('Original',gray)
9
10 smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),
11                   cv.GaussianBlur(gray,(9,9),0.0),
12                   cv.GaussianBlur(gray,(15,15),0.0)))
13 cv.imshow('Smooth',smooth)
14
15 femboss=np.array([[-1.0, 0.0, 0.0],
16                   [ 0.0, 0.0, 0.0],
17                   [ 0.0, 0.0, 1.0]])
18
19 gray16=np.int16(gray)
20 emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
21 emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
22 emboss_worse=cv.filter2D(gray,-1,femboss)
23
24 cv.imshow('Emboss',emboss)
25 cv.imshow('Emboss_bad',emboss_bad)
26 cv.imshow('Emboss_worse',emboss_worse)
27
28 cv.waitKey()
29 cv.destroyAllWindows()
```

(L10-L12) cv.GaussianBlur(): Apply Gaussian filters of different sizes to img

- The size of window is given as an input
- 3rd parameter: Gaussian standard deviation (σ), If 0, computed from the size of window as sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8
- Gaussian filters are used to smooth the input image (The larger the window, the smoother the resulting image)

(*) Other built-in filters: cv.blur(), cv.boxFilter(), cv.medianFilter(), cv.bilateralFilter()

Example0. Convolution Filtering




```python
1  import cv2 as cv
2  import numpy as np
3
4  img=cv.imread('soccer.jpg')
5  img=cv.resize(img,dsize=(0,0),fx=0.4,fy=0.4)
6  gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
7  cv.putText(gray,'soccer',(10,20),cv.FONT_HERSHEY_SIMPLEX,0.
8  cv.imshow('Original',gray)
9
10 smooth=np.hstack((cv.GaussianBlur(gray,(5,5),0.0),
11                   cv.GaussianBlur(gray,(9,9),0.0),
12                   cv.GaussianBlur(gray,(15,15),0.0)))
13 cv.imshow('Smooth',smooth)
14
15 femboss=np.array([[-1.0, 0.0, 0.0],
16                   [ 0.0, 0.0, 0.0],
17                   [ 0.0, 0.0, 1.0]])
18
19 gray16=np.int16(gray)
20 emboss=np.uint8(np.clip(cv.filter2D(gray16,-1,femboss)+128,0,255))
21 emboss_bad=np.uint8(cv.filter2D(gray16,-1,femboss)+128)
22 emboss_worse=cv.filter2D(gray,-1,femboss)
23
24 cv.imshow('Emboss',emboss)
25 cv.imshow('Emboss_bad',emboss_bad)
26 cv.imshow('Emboss_worse',emboss_worse)
27
28 cv.waitKey()
29 cv.destroyAllWindows()
```

(L15-L22) Design a custom emboss filter and apply to img

- Filter is defined as a 2D array
- cv.filter2D(): apply the filter to the input image

(*) Users have to take care of type conversions
(to avoid overflows)

Content

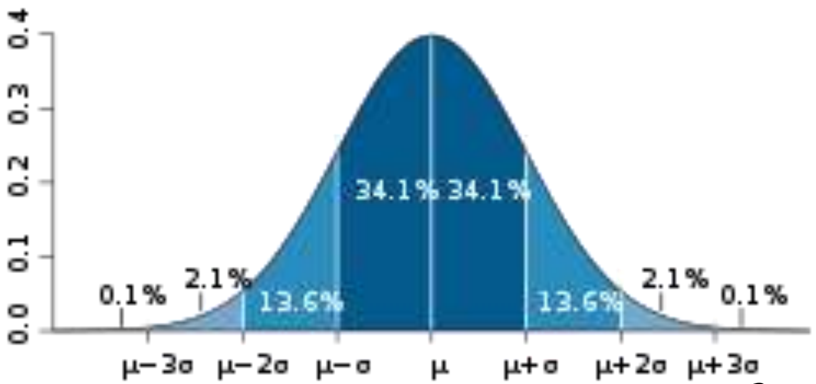
- Linear Filtering
- Non-linear filtering
  - Adding random noise to an image
  - Median filtering
  - Bilateral filtering
- Integral Image
- Image Thresholding
  - Global Thresholding
  - Adaptive Thresholding
  - Otsu’s Algorithm

Noise Model

- Noise in digital images arises during image acquisition and transmission
- Imaging sensor creates undesirable noise due to many environmental and mechanical factors
- Noise model:

$$\underbrace{f(y,x)}_{\text{Digital image}} = \underbrace{s(y,x)}_{\text{Input signal}} + \underbrace{n(y,x)}_{\text{Noise function}}$$

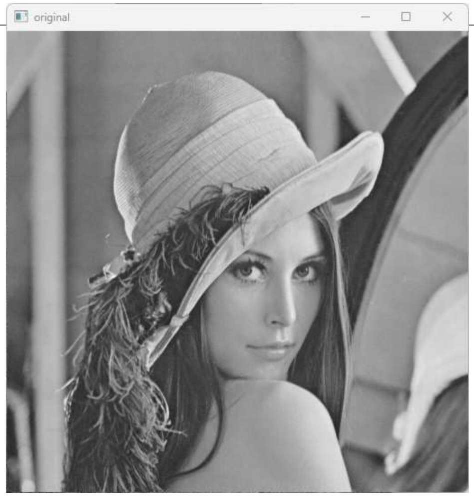
- Gaussian noise model
  - Widely used to model noise (white noise)
  - $n(y,x)$ has the normal distribution (the Gaussian distribution) pdf $\mathcal{N}(0,\sigma^2)$



Normal distribution pdf $\mathcal{N}(\mu,\sigma^2)$

Example 1. Add Gaussian Noise to Image

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

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

img32 = np.float32(img)
noise = np.zeros((img.shape[0], img.shape[1]), np.float32)

cv.randn(noise, 0, 10)
noiseimg1 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))

cv.randn(noise, 0, 20)
noiseimg2 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))

cv.randn(noise, 0, 30)
noiseimg3 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))

titles = ['original', 'stddev10', 'stddev20', 'stddev30']
images = [img, noiseimg1, noiseimg2, noiseimg3]

# Drawing images with matplotlib
#for i in range(4):
#    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
#    plt.title(titles[i])
    #plt.xticks([]),plt.yticks([])
#plt.show()

for i in range(4):
    cv.imshow(titles[i], images[i])
    cv.imwrite(titles[i] +'.jpg', images[i])
cv.waitKey()
cv.destroyAllWindows()
```



Original Image

With $\mathcal{N}(0,10^2)$ noise

With $\mathcal{N}(0,20^2)$ noise

With $\mathcal{N}(0,30^2)$ noise

Example 1. Add Gaussian Noise to Image

```python
1  import cv2 as cv
2  import numpy as np
3  # import matplotlib.pyplot as plt
4
5  img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
6  if img is None:
7      print('File not found')
8
9  img32 = np.float32(img)
10 noise = np.zeros((img.shape[0], img.shape[1]), np.float32)
11
12 cv.randn(noise, 0, 10)
13 noiseimg1 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
14
15 cv.randn(noise, 0, 20)
16 noiseimg2 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
17
18 cv.randn(noise, 0, 30)
19 noiseimg3 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
20
21 titles = ['original', 'stddev10', 'stddev20', 'stddev30']
22 images = [img, noiseimg1, noiseimg2, noiseimg3]
23
24 # Drawing images with matplotlib
25 #for i in range(4):
26 #    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
27 #    plt.title(titles[i])
28     #plt.xticks([]),plt.yticks([])
29 #plt.show()
30
31 for i in range(4):
32     cv.imshow(titles[i], images[i])
33     cv.imwrite(titles[i] +'.jpg', images[i])
34 cv.waitKey()
35 cv.destroyAllWindows()
```

- (L10) creates a 2D array of the same size as ‘img’ to save a noise value at each pixel
- (L12) cv.randn(): fills the array with normally distributed random numbers

- Mean(μ) = 0, the standard deviation(σ) = 10
- Since negative numbers can be created, ‘noise’ should save signed numbers (see L9,L10)

◆ randn()

```
void cv::randn ( InputOutputArray dst,
                 InputArray       mean,
                 InputArray       stddev
               )
Python:
    cv.randn( dst, mean, stddev ) -> dst
```

`#include <opencv2/core.hpp>`

Fills the array with normally distributed random numbers.

The function cv::randn fills the matrix dst with normally distributed random numbers with the specified mean vector and the standard deviation matrix. The generated random numbers are clipped to fit the value range of the output array data type.

Parameters

- **dst** output array of random numbers; the array must be pre-allocated and have 1 to 4 channels.
- **mean** mean value (expectation) of the generated random numbers.
- **stddev** standard deviation of the generated random numbers; it can be either a vector (in which case a diagonal standard deviation matrix is assumed) or a square matrix.

Example 1. Add Gaussian Noise to Image

```
1   import cv2 as cv
2   import numpy as np
3   # import matplotlib.pyplot as plt
4
5   img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
6   if img is None:
7       print('File not found')
8
9   img32 = np.float32(img)
10  noise = np.zeros((img.shape[0], img.shape[1]), np.float32)
11
12  cv.randn(noise, 0, 10)
13  noiseimg1 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
14
15  cv.randn(noise, 0, 20)
16  noiseimg2 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
17
18  cv.randn(noise, 0, 30)
19  noiseimg3 = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
20
21  titles = ['original', 'stddev10', 'stddev20', 'stddev30']
22  images = [img, noiseimg1, noiseimg2, noiseimg3]
23
24  # Drawing images with matplotlib
25  #for i in range(4):
26  #    plt.subplot(2,2,i+1), plt.imshow(images[i], 'gray')
27  #    plt.title(titles[i])
28      #plt.xticks([]),plt.yticks([])
29  #plt.show()
30
31  for i in range(4):
32      cv.imshow(titles[i], images[i])
33      cv.imwrite(titles[i] +'.jpg', images[i])
34  cv.waitKey()
35  cv.destroyAllWindows()
```

- (L13) cv.add(): adds the Gaussian noise to input image (unsigned ‘img’ has been converted to ‘img32’)
- (L13) cv.clip(): clips the range of resulting image to [0,255]

(*) Drawing images

- You can draw images either using OpenCV or Matplotlib
- (L24~L29): draw images using drawing functions in matplotlib

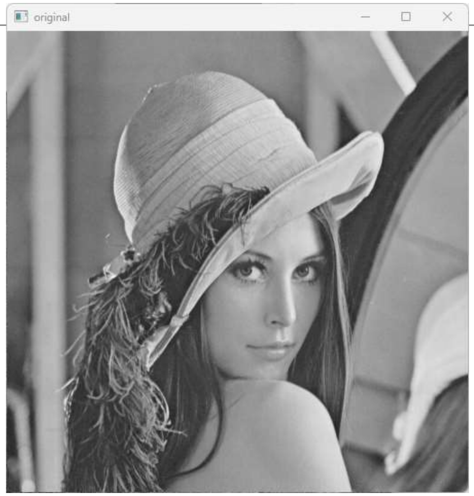
\- plt.subplot(): sets the layout of images with the given row and column numbers

Example 1. Add Gaussian Noise to Image

- Images disturb more for larger σ (Compare original and $\mathcal{N}(0,30^2)$-images)
- Consistent noises are allocated with a Gaussian noise model

(Question) Increase the standard deviation(σ) of the noise.

(Question) Change the mean(μ) of the noise.



Original Image

With $\mathcal{N}(0,10^2)$ noise

With $\mathcal{N}(0,20^2)$ noise

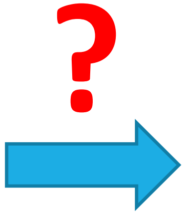
With $\mathcal{N}(0,30^2)$ noise

Example 2. Bilateral Filtering

- Can we remove noises from the image and restore the image?

Original Image

With a Gaussian noise



Example 2-1. Noise Removal by Gaussian Filter

- Let’s apply the Gaussian filter to an image with noise

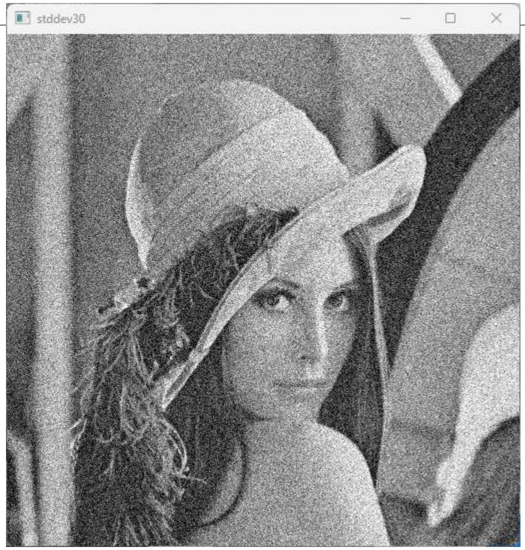
Original Image

With a Gaussian noise

```
import cv2 as cv

img = cv.imread('stddev30.jpg', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

gImg1 =
gImg2 =
gImg3 =
gImg4 =

titles = ['original', 'GaussianBlur5', 'GaussianBlur9',
          'GaussianBlur15', 'GaussianBlur27']
images = [img, gImg1, gImg2, gImg3, gImg4]

for i in range(5):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

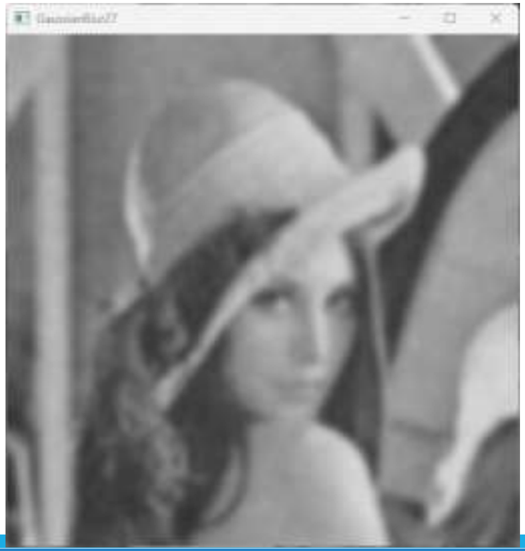
Noise Removal by Gaussian Filter

(a) Original Image



(b) Image with noise

- Gaussian filter is used in noise removal:
  - When applying to large, flat areas, noises are blurred with the surrounding pixels
  - However, the Gaussian filter also blurs edges of the image (Compare (a) with ‘GaussianBlur27’)



Applying Gaussian filters to (b) (window size k=5, 9, 15, 27)

Example 2-2. Bilateral Filter

```python
import cv2 as cv
import numpy as np

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

# add noise to img
img32 = np.float32(img)
noise = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)

cv.randn(noise, 0, 5)
noiseimg = np.uint8(np.clip(cv.add(img32, noise), 0, 255))

# apply gaussian filter
gImg = cv.GaussianBlur(noiseimg, (5,5), 0.0)

# apply bilateral filter
bImg = cv.bilateralFilter(noiseimg, -1, 10, 5)

titles = ['original', 'noise', 'GaussianBlur',
          'BilateralFilter']
images = [img, noiseimg, gImg, bImg]

for i in range(4):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

Original image

Image with noise

Filtered with Gaussian filter

Filtered with bilateral filter

Example 2-2. Bilateral Filter

- Bilateral filter: removing noise while preserving edges of the image [Tomasi98]

- $g(j,i) = \frac{\sum_{k,l} f(k,l)w(j,i,k,l)}{\sum_{k,l} w(j,i,k,l)}$, where

$w(j,i,k,l) = \exp\left(-\frac{(j-k)^2+(i-l)^2}{2\sigma_d^2} - \frac{\|f(j,i)-f(k,l)\|^2}{2\sigma_r^2}\right)$

$(\Rightarrow\ w(\boldsymbol{p},\boldsymbol{q}) = \alpha \mathrm{G}_{\sigma_d}(\|\boldsymbol{p}-\boldsymbol{q}\|)\mathrm{G}_{\sigma_r}(|f(\boldsymbol{p})-f(\boldsymbol{q})|)$

- G_{σ_d}: Gaussian blurring based on pixel distance
- G_{σ_r}: Gaussian blurring based on pixel values

- If $\boldsymbol{p},\boldsymbol{q}$ are across the edge,

$\Rightarrow$ $|f(\boldsymbol{p})-f(\boldsymbol{q})|$ is large
$\Rightarrow$ $\mathrm{G}_{\sigma_r}(|f(\boldsymbol{p})-f(\boldsymbol{q})|)$ approaches to zero
$\Rightarrow$ $w(\boldsymbol{p},\boldsymbol{q})$ becomes small
$\Rightarrow$ $\boldsymbol{q}$ does not contribute to $\boldsymbol{p}$'s pixel value

Original image

Image with noise

Filtered with Gaussian filter

Filtered with bilateral filter

Example 2-2. Bilateral Filter

```python
1  import cv2 as cv
2  import numpy as np
3
4  img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
5  if img is None:
6      print('File not found')
7
8  # add noise to img
9  img32 = np.float32(img)
10 noise = np.zeros((img.shape[0], img.shape[1]), dtype=np.float32)
11
12 cv.randn(noise, 0, 5)
13 noiseimg = np.uint8(np.clip(cv.add(img32, noise), 0, 255))
14
15 # apply gaussian filter
16 gImg = cv.GaussianBlur(noiseimg, (5,5), 0.0)
17
18 # apply bilateral filter
19 bImg = cv.bilateralFilter(noiseimg, -1, 10, 5)
20
21 titles = ['original', 'noise', 'GaussianBlur',
22           'BilateralFilter']
23 images = [img, noiseimg, gImg, bImg]
24
25 for i in range(4):
26     cv.imshow(titles[i], images[i])
27 cv.waitKey()
28 cv.destroyAllWindows()
```

- (L19) cv.bilateralFilter(): applies the bilateral filter to an image
- Set 2 sigma values: σ_r(sigmaColor), σ_d(sigmaSpace)
- (Question) Increase σ_r and σ_d to large numbers

◆ bilateralFilter()

```
void cv::bilateralFilter ( InputArray   src,
                           OutputArray  dst,
                           int          d,
                           double       sigmaColor,
                           double       sigmaSpace,
                           int          borderType = BORDER_DEFAULT
                         )
```

Python:

cv.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) -> dst

`#include <opencv2/imgproc.hpp>`

Applies the bilateral filter to an image.

The function applies bilateral filtering to the input image, as described in http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html bilateralFilter can reduce unwanted noise very well while keeping edges fairly sharp. However, it is very slow compared to most filters.

Sigma values: For simplicity, you can set the 2 sigma values to be the same. If they are small (< 10), the filter will not have much effect, whereas if they are large (> 150), they will have a very strong effect, making the image look "cartoonish".

Filter size: Large filters (d > 5) are very slow, so it is recommended to use d=5 for real-time applications, and perhaps d=9 for offline applications that need heavy noise filtering.

This filter does not work inplace.

Parameters

src	Source 8-bit or floating-point, 1-channel or 3-channel image.
dst	Destination image of the same size and type as src .
d	Diameter of each pixel neighborhood that is used during filtering. If it is non-positive, it is computed from sigmaSpace.
sigmaColor	Filter sigma in the color space. A larger value of the parameter means that farther colors within the pixel neighborhood (see sigmaSpace) will be mixed together, resulting in larger areas of semi-equal color.
sigmaSpace	Filter sigma in the coordinate space. A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigmaColor). When d>0, it specifies the neighborhood size regardless of sigmaSpace. Otherwise, d is proportional to sigmaSpace.
borderType	border mode used to extrapolate pixels outside of the image, see BorderTypes

Example 3. Median Filter

- Salt & Pepper noise (Impulse noise)
  - Caused by sharp and sudden disturbances in signals
  - Shown as random white and black pixels in the image

```python
import cv2 as cv
import random

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

# add salt and pepper noise to img
noiseNum = img.size//10

for i in range(noiseNum):
    row = random.randrange(img.shape[0])
    col = random.randrange(img.shape[1])
    img[row,col] = (i % 2) * 255

cv.imshow('Salt and pepper noise', img)

# Apply Gaussian Filter
# Apply Median Filter

cv.waitKey()
cv.destroyAllWindows()
```

Example 3. Median Filter

- (Question) Apply a 3x3 Gaussian Filter and a 3x3 median filter

to the image with salt and pepper noise and compare the filtering results

(Refer to https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html)

```python
import cv2 as cv
import random

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

# add salt and pepper noise to img
noiseNum = img.size//10

for i in range(noiseNum):
    row = random.randrange(img.shape[0])
    col = random.randrange(img.shape[1])
    img[row,col] = (i % 2) * 255

cv.imshow('Salt and pepper noise', img)

# Apply Gaussian Filter
# Apply Median Filter

cv.waitKey()
cv.destroyAllWindows()
```

Write codes here!

Example 3. Median Filter

- (Answer)

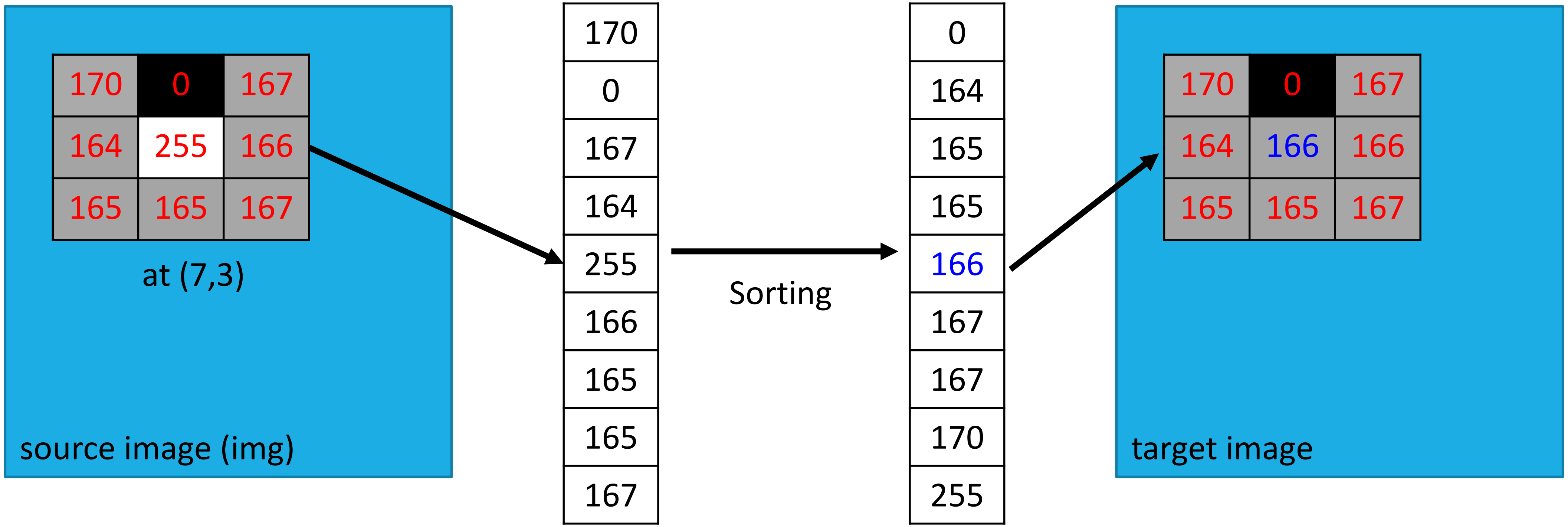
```python
import cv2 as cv
import random

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print('File not found')

# add salt and pepper noise to img
noiseNum = img.size//10

for i in range(noiseNum):
    row = random.randrange(img.shape[0])
    col = random.randrange(img.shape[1])
    img[row,col] = (i % 2) * 255

cv.imshow('Salt and pepper noise', img)

# Apply Gaussian Filter



# Apply Median Filter



cv.waitKey()
cv.destroyAllWindows()
```

Gaussian filtering result

Median filtering result

Example 3. Median Filter

- Median filter selects the median pixel value within a window



⇒ Removes salt and pepper noise easily

Content

- Non-linear filtering
  - Adding random noise to an image
  - Median filtering
  - Bilateral filtering
- Integral Image
- Image Thresholding
  - Global Thresholding
  - Adaptive Thresholding
  - Otsu’s Algorithm

Summed Area Table (Integral Image)

- Accelerate convolution when box filters of different sizes are used
- Summed area table: $s(j,i) = \sum_{k=0}^{j}\sum_{l=0}^{i} f(k,l)$ or

$$s(j,i) = s(j-1,i) + s(j,i-1) - s(j-1,i-1) + f(j,i)$$

- $s(j,i)$ is also called an integral image
- To find the summed area in $[j_0, j_1] \times [i_0, i_1]$, we need 4 samples:

$$S([j_0,j_1],[i_0,i_1]) = s(j_1,i_1) - s(j_1,i_0-1) - s(j_0-1,i_1) + s(j_0-1,i_0-1)$$

3	2	7	2	3
1	5	1	3	4
5	1	**3**	5	1
4	3	2	1	6
2	4	1	4	8

(a) S = 24

3	5	12	14	17
4	*11*	**19**	24	31
9	**17**	28	38	46
13	24	37	48	62
15	30	44	59	81

(b) s = 28

3	5	12	*14*	17
4	11	19	24	31
9	17	28	38	46
13	24	37	**48**	62
15	30	44	59	81

(c) S = 24

(a) Original image
(b) Summed area table
(c) Computation of area sum

Integral Image

- cv.integral(): computes the integral images of the source image
  - Useful for fast blurring or block correlation computation with a variable window size

The function calculates one or more integral images for the source image as follows:

$$\texttt{sum}(X,Y) = \sum_{x<X,y<Y} \texttt{image}(x,y)$$

$$\texttt{sqsum}(X,Y) = \sum_{x<X,y<Y} \texttt{image}(x,y)^2$$

$$\texttt{tilted}(X,Y) = \sum_{y<Y,abs(x-X+1)\leq Y-y-1} \texttt{image}(x,y)$$

Using these integral images, you can calculate sum, mean, and standard deviation over a specific up-right or rotated rectangular region of the image in a constant time, for example:

$$\sum_{x_1\leq x<x_2,\, y_1\leq y<y_2} \texttt{image}(x,y) = \texttt{sum}(x_2,y_2) - \texttt{sum}(x_1,y_2) - \texttt{sum}(x_2,y_1) + \texttt{sum}(x_1,y_1)$$

◆ integral() [1/3]

```
void cv::integral ( InputArray   src,
                    OutputArray  sum,
                    OutputArray  sqsum,
                    OutputArray  tilted,
                    int          sdepth = -1 ,
                    int          sqdepth = -1
                  )
```

Python:

```
cv.integral(  src[, sum[, sdepth]]                              ) -> sum
cv.integral2( src[, sum[, sqsum[, sdepth[, sqdepth]]]]          ) -> sum, sqsum
cv.integral3( src[, sum[, sqsum[, tilted[, sdepth[, sqdepth]]]]] ) -> sum, sqsum, tilted
```

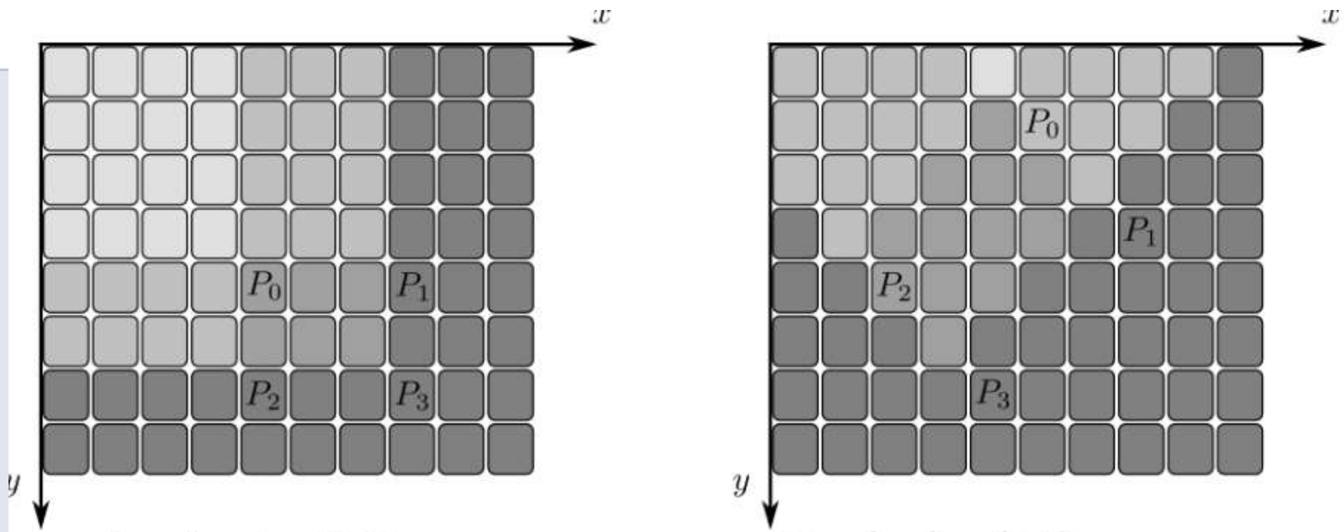
Integral Image

◆ **integral()** [1/3]

```
void cv::integral ( InputArray   src,
                    OutputArray  sum,
                    OutputArray  sqsum,
                    OutputArray  tilted,
                    int          sdepth = -1 ,
                    int          sqdepth = -1
                  )
```

Python:

```
cv.integral(  src[, sum[, sdepth]]                            ) -> sum
cv.integral2( src[, sum[, sqsum[, sdepth[, sqdepth]]]]        ) -> sum, sqsum
cv.integral3( src[, sum[, sqsum[, tilted[, sdepth[, sqdepth]]]]] ) -> sum, sqsum, tilted
```



$$P_0 = \{y, x\} = \{4, 4\}$$
$$P_1 = \{y, x + w\} = \{4, 7\}$$
$$P_2 = \{y + h, x\} = \{6, 4\}$$
$$P_3 = \{y + h, x + w\} = \{6, 7\}$$

$$P_0 = \{y, x\} = \{1, 5\}$$
$$P_1 = \{y + w, x + w\} = \{3, 7\}$$
$$P_2 = \{y + h, x - h\} = \{4, 2\}$$
$$P_3 = \{y + w + h, x + w - h\} = \{6, 4\}$$

calculation example

Parameters

- **src** input image as $W \times H$, 8-bit or floating-point (32f or 64f).
- **sum** integral image as $(W+1) \times (H+1)$, 32-bit integer or floating-point (32f or 64f).
- **sqsum** integral image for squared pixel values; it is $(W+1) \times (H+1)$, double-precision floating-point (64f) array.
- **tilted** integral for the image rotated by 45 degrees; it is $(W+1) \times (H+1)$ array with the same data type as sum.
- **sdepth** desired depth of the integral and the tilted integral images, CV_32S, CV_32F, or CV_64F.
- **sqdepth** desired depth of the integral image of squared pixel values, CV_32F or CV_64F.

Integral Image

- (Question) Let's implement a 5x5 box filtering using integral image

```python
import cv2 as cv
import numpy as np

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print("file not found")

bImg = cv.blur(img, (5,5))

sumimg = cv.integral(img)
bImg2 = np.zeros((img.shape[0], img.shape[1]))

# write filtering with a 5x5 using sumimg
Write code here!

titles = ['Original Image','Blurred', 'With IntegralImg']
images = [img, bImg, bImg2]

for i in range(3):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

Content

- Non-linear filtering
  - Adding random noise to an image
  - Median filtering
  - Bilateral filtering
- Integral Image
- Image Thresholding
  - Global Thresholding
  - Adaptive Thresholding
  - Otsu’s Algorithm

Binarization by image thresholding

- Binarization: convert a grayscale image to a binary image with a threshold T,

$$b(j,i) = \begin{cases} 1, & \text{if } f(j,i) > T \\ 0, & \text{if } f(j,i) \le T \end{cases}$$

- In OpenCV, a binary image has either 0 or 255 as pixel values
- By binarization, an image is divided into region of interest (ROI) and non-ROI

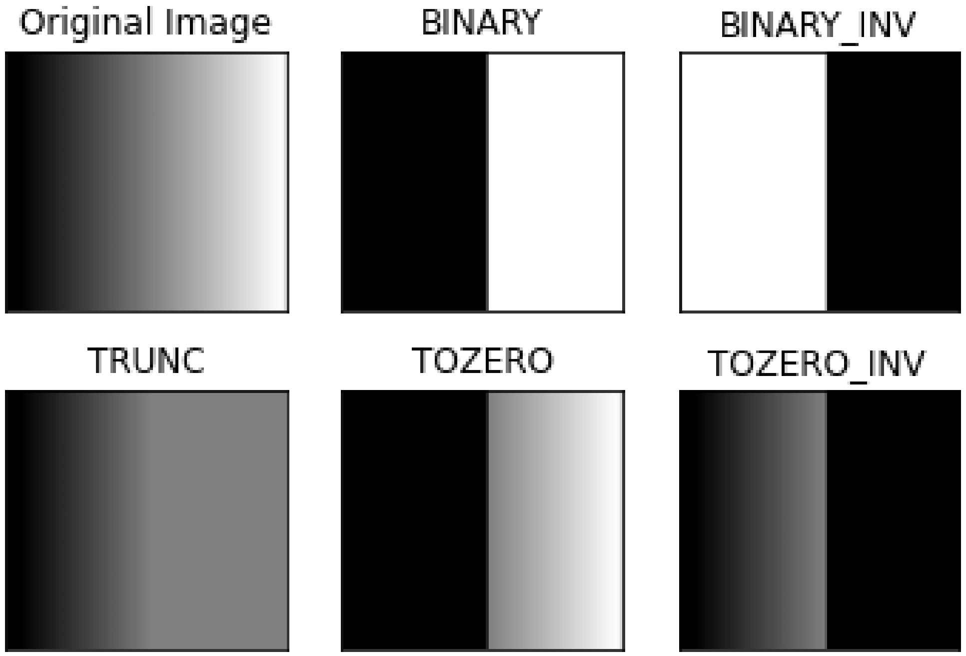
(E.g. foreground objects and background scene)






Example 5-1. Thresholding in OpenCV

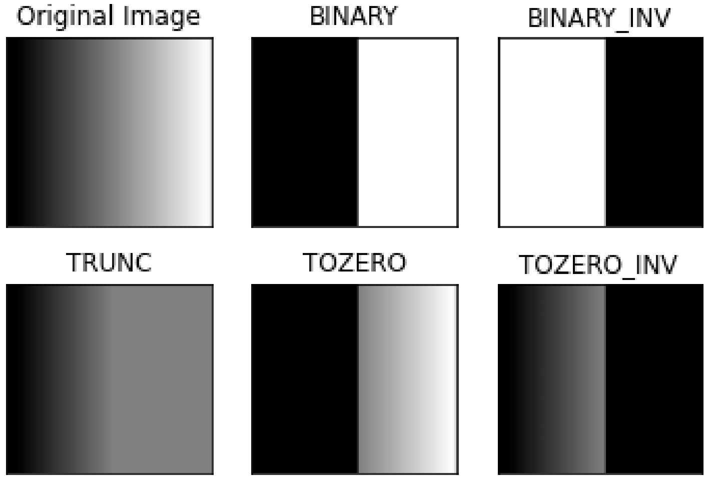
```python
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread('gradient.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print("file not found")

ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC',
          'TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),
    plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



Execution results

Example 5-1. Thresholding in OpenCV

```
1  import cv2 as cv
2  import matplotlib.pyplot as plt
3
4  img = cv.imread('gradient.png', cv.IMREAD_GRAYSCALE)
5  if img is None:
6      print("file not found")
7
8  ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
9  ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
10 ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
11 ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
12 ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
13 titles = ['Original Image','BINARY','BINARY_INV','TRUNC',
14           'TOZERO','TOZERO_INV']
15 images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
16 for i in range(6):
17     plt.subplot(2,3,i+1),
18     plt.imshow(images[i],'gray',vmin=0,vmax=255)
19     plt.title(titles[i])
20     plt.xticks([]),plt.yticks([])
21 plt.show()
```

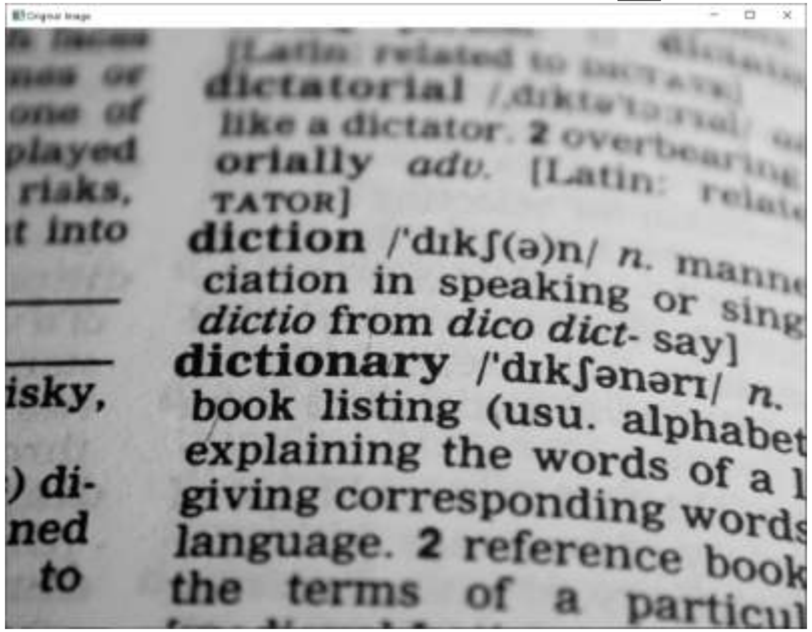


- (L8-L11) cv.threshold(): apply the thresholding to the input grayscale image
- Threshold and maximum value are passed as parameters

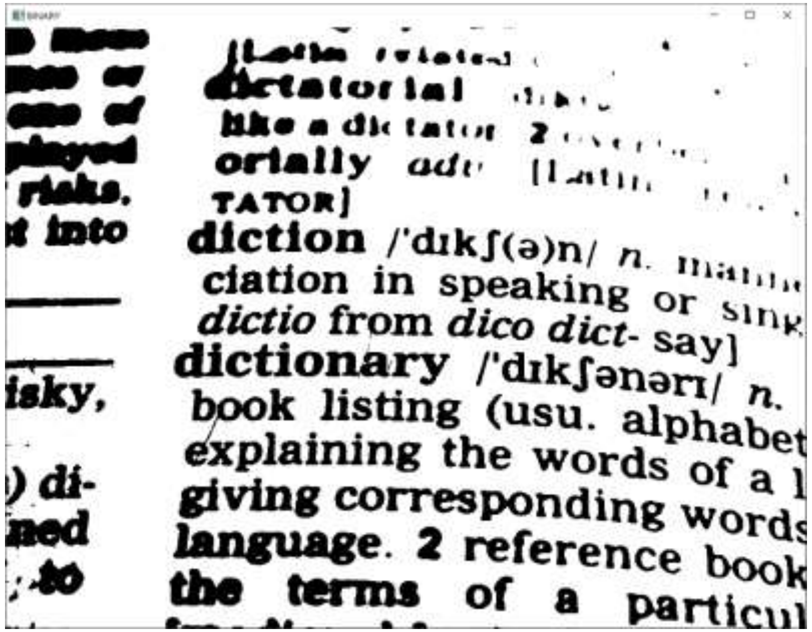
Enumerator	
cv.THRESH_BINARY	$dst(y,x) = \begin{cases} maxval & if\ src(y,x) > thresh \\ 0 & otherwise \end{cases}$
cv.THRESH_BINARY_INV	$dst(y,x) = \begin{cases} 0 & if\ src(y,x) > thresh \\ maxval & otherwise \end{cases}$
cv.THRESH_TRUNC	$dst(y,x) = \begin{cases} thresh & if\ src(y,x) > thresh \\ src(y,x) & otherwise \end{cases}$
cv.THRESH_TOZERO	$dst(y,x) = \begin{cases} src(y,x) & if\ src(y,x) > thresh \\ 0 & otherwise \end{cases}$
cv.THRESH_TOZERO_INV	$dst(y,x) = \begin{cases} 0 & if\ src(y,x) > thresh \\ src(y,x) & otherwise \end{cases}$
cv.THRESH_OTSU	Use Otsu algorithm to choose optimal T
cv.THRESH_TRIANGLE	Use Triangle algorithm to choose optimal T

Example 5-1. Thresholding in OpenCV

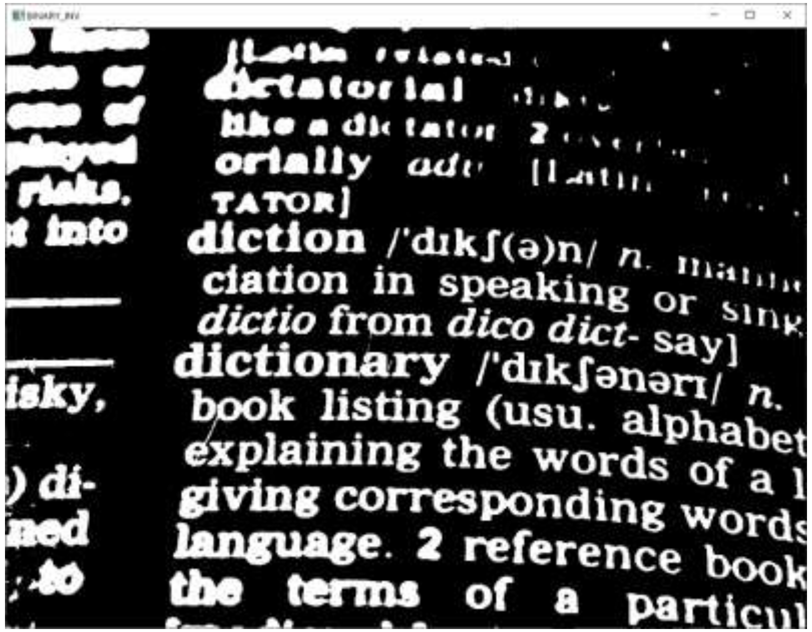
- Comparison between CV.THRESH_BINARY and CV.THRESH_BINARY_INV



Original Image



Thresholding with CV.THRESH_BINARY (T=128)



Thresholding with CV.THRESH_BINARY_INV (T=128)

⇒ In OpenCV, objects are often labelled as white, where the background as black
⇒ Use CV.THRESH_BINARY or CV.THRESH_BINARY_INV based on images or applications

Example 5-2. Adaptive Thresholding

- cv.threshold() performs global binarization
  - Use the same threshold value T for full image
  - If an image has the different lighting conditions in different areas, a single threshold T would not perform well

(Left) Original image
(Right) Binarized image with a global threshold (T = 100)

⇒ The lower left region of the right image has not been separated well as the area has small pixel values Overall

(Question) Compare with T=50

Example 5-2. Adaptive Thresholding

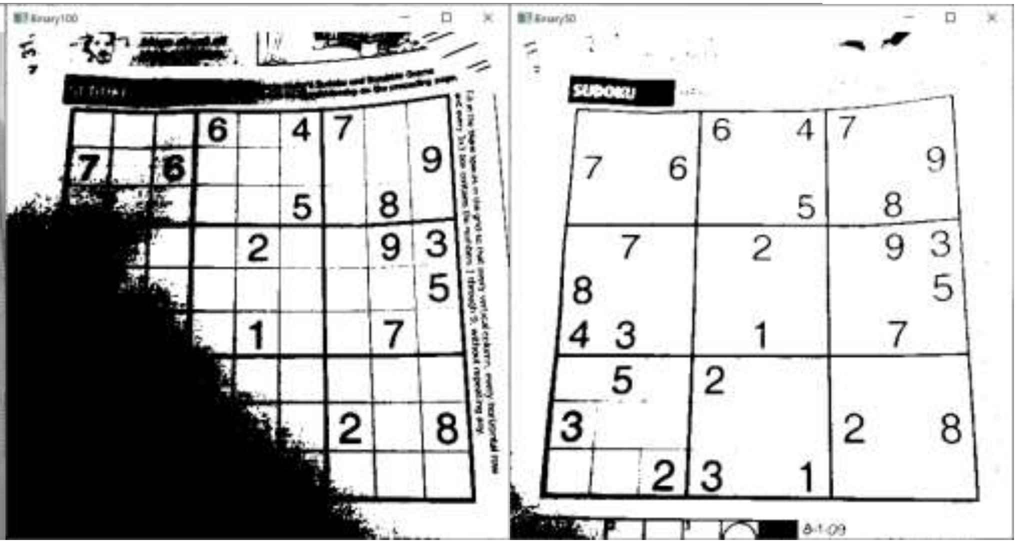
- Adaptive thresholding: apply a different threshold value for each pixel
  - Set a window around the pixel and determine T from the image histogram within the window
  - At a pixel (y, x), a threshold $T(y, x)$ is determined by:

  $T(y, x) = \mu(y, x) - C$, where $\mu(y, x)$ is the mean of the neighborhood and C is the constant
  - Performs better in binarizing an image having different lighting conditions

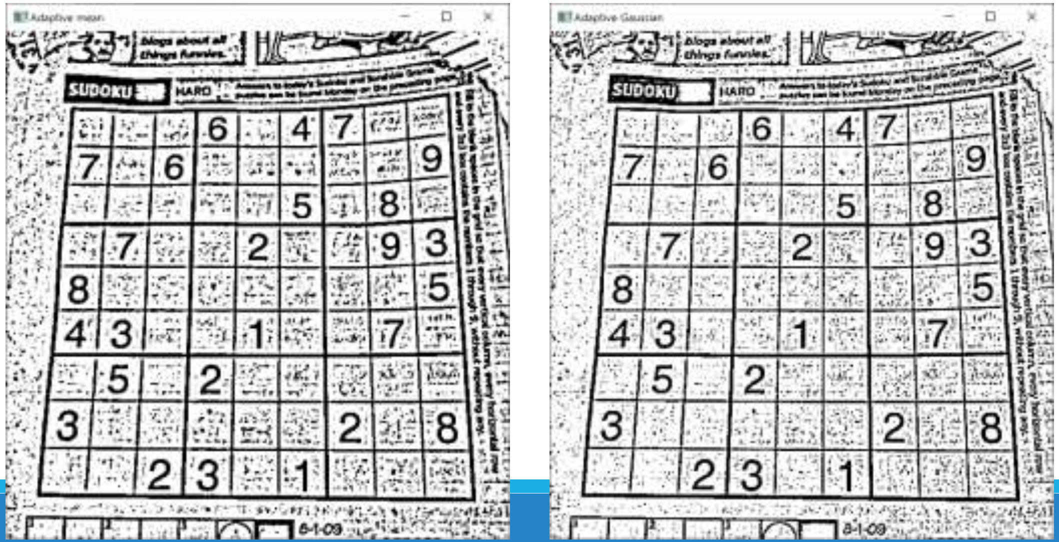
Example 5-2. Adaptive Thresholding

```python
import cv2 as cv

img = cv.imread('sudoku.jpg', cv.IMREAD_GRAYSCALE)
if img is None:
    print("file not found")

ret,thresh1 = cv.threshold(img,100,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,50,255,cv.THRESH_BINARY)

thresh3 = cv.adaptiveThreshold(img, 255,
                               cv.ADAPTIVE_THRESH_MEAN_C,
                               cv.THRESH_BINARY, 11, 2)
thresh4 = cv.adaptiveThreshold(img, 255,
                               cv.ADAPTIVE_THRESH_GAUSSIAN_C,
                               cv.THRESH_BINARY, 11, 2)

titles = ['Original Image','Binary100','Binary50',
          'Adaptive mean','Adaptive Gaussian']
images = [img, thresh1, thresh2, thresh3, thresh4]

for i in range(5):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

Original Image



Global Thresholding



Adaptive Thresholding

Example 5-2. Adaptive Thresholding

```
1  import cv2 as cv
2  
3  img = cv.imread('sudoku.jpg', cv.IMREAD_GRAYSCALE)
4  if img is None:
5      print("file not found")
6  
7  ret,thresh1 = cv.threshold(img,100,255,cv.THRESH_BINARY)
8  ret,thresh2 = cv.threshold(img,50,255,cv.THRESH_BINARY)
9  
10 thresh3 = cv.adaptiveThreshold(img, 255,
11                                cv.ADAPTIVE_THRESH_MEAN_C,
12                                cv.THRESH_BINARY, 11, 2)
13 thresh4 = cv.adaptiveThreshold(img, 255,
14                                cv.ADAPTIVE_THRESH_GAUSSIAN_C,
15                                cv.THRESH_BINARY, 11, 2)
16 
17 titles = ['Original Image','Binary100','Binary50',
18           'Adaptive mean','Adaptive Gaussian']
19 images = [img, thresh1, thresh2, thresh3, thresh4]
20 
21 for i in range(5):
22     cv.imshow(titles[i], images[i])
23 cv.waitKey()
24 cv.destroyAllWindows()
```

- (L10-L13) cv.adaptiveThreshold(): apply adaptive thresholding
- cv.ADAPTIVE_THRESH_MEAN_C: the threshold value is the mean of neighborhood minus constant C ($T(y,x) = \mu(y,x) - C$)
- cv.ADAPTIVE_THRESH_GAUSSIAN_C: the threshold value is the Gaussian-weighted sum of neighborhood minus constant C

```
cv.adaptiveThreshold( src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst] ) -> dst
```

Parameters

src	Source 8-bit single-channel image.
dst	Destination image of the same size and the same type as src.
maxValue	Non-zero value assigned to the pixels for which the condition is satisfied
adaptiveMethod	Adaptive thresholding algorithm to use, see AdaptiveThresholdTypes. The BORDER_REPLICATE | BORDER_ISOLATED is used to process boundaries.
thresholdType	Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV, see ThresholdTypes.
blockSize	Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
C	Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.

Example 5-3. Otsu Binarization Algorithm

- In global thresholding, find the optimal threshold T that minimizes the weighted within-class variance:

$$T = \operatorname*{argmin}_{t \in \{0,1,\dots,L-1\}} v_{within}(t)$$

, where $v_{within}(t) = w_0(t) v_0(t) + w_1(t) v_1(t)$,

$w_0(t) = \sum_{i=0}^{t} \hat{h}(i)$, $\quad w_1(t) = \sum_{i=t+1}^{L-1} \hat{h}(i)$,

$\mu_0(t) = \frac{1}{w_0(t)} \sum_{i=0}^{t} i\hat{h}(i)$, $\quad \mu_1(t) = \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} i\hat{h}(i)$,

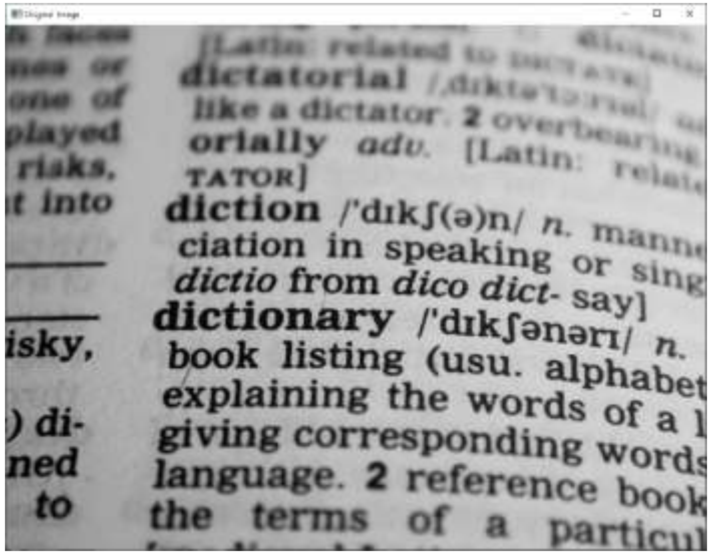
$v_0(t) = \frac{1}{w_0(t)} \sum_{i=0}^{t} \hat{h}(i)\left(i - \mu_0(t)\right)^2$, $\quad v_1(t) = \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} \hat{h}(i)\left(i - \mu_1(t)\right)^2$

Example 5-3. Otsu Binarization Algorithm

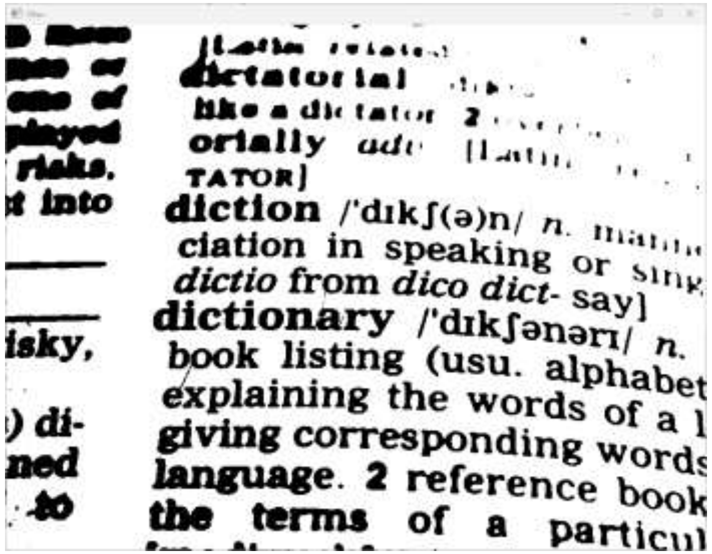
- Otsu binarization: use a flag cv.THRESH_OTSU in cv.threshold()

```python
import cv2 as cv

img = cv.imread('diction.jpg', cv.IMREAD_GRAYSCALE)
if img is None:
    print("file not found")

ret,thresh1 = cv.threshold(img,128,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,0, 255,cv.THRESH_BINARY+cv.THRESH_OTSU)

print('Otsu Threshold',ret)

titles = ['Original Image','Binary128','Otsu']
images = [img, thresh1, thresh2]

for i in range(3):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

Original image



Binarized image with Otsu Algorithm (Optimal threshold = 125.0)



ReviewTask 0321

① (Integral Image) Let's implement a 5x5 box filtering using integral image

```python
import cv2 as cv
import numpy as np

img = cv.imread('lenna.png', cv.IMREAD_GRAYSCALE)
if img is None:
    print("file not found")

bImg = cv.blur(img, (5,5))

sumimg = cv.integral(img)
bImg2 = np.zeros((img.shape[0], img.shape[1]))

# write filtering with a 5x5 using sumimg
```

Write code here!

```python
titles = ['Original Image','Blurred', 'With IntegralImg']
images = [img, bImg, bImg2]

for i in range(3):
    cv.imshow(titles[i], images[i])
cv.waitKey()
cv.destroyAllWindows()
```

ReviewTask 0321

② (Otsu Binarization) Binarize the following images using Otsu Algorithm

And print out the optimal threshold values for each image

(soccer.jpg – red channel only)

(rose.png – compare the binarization results of gray-scaled image and red channel only-image)