Procesamiento de imagenes: filtros

Tusar Verma, LU: 309/22

1

a

x(n) = 010000 h(n) = 02100 —rotado 180—> 00120

```
00:010000
00 120
```

$(h * x)(0) = 2$

```
0:010000
0 0120
```

$(h * x)(1) = 1$

```
010000
00120
```

$(h * x)(2) = 0$

```
010000
 00120
```

$(h * x)(3) = 0$

```
010000:0
  0012 0
```

$(h * x)(4) = 0$

```
010000:00
   001 20
```

$(h * x)(5) = 0$

b

x(n) = 001000 h(n) = 02100 —rotado 180—> 00120

```
00:001000
00 120
```

$(h * x)(0) = 0$

```
0:001000
0 0120
```

$(h * x)(1) = 2$

```
001000
00120
```

$(h * x)(2) = 1$

```
001000
 00120
```

$(h * x)(3) = 0$

```
001000:0
  0012 0
```

$(h * x)(4) = 0$

```
001000:00
   001 20
```

$(h * x)(5) = 0$

c

x(n) = 0 2 -1 0 0 0 h(n) = 0 -1 2 1 0 —rotado 180—> 0 1 2 -1 0

```
0 0 : 0  2 -1 0 0 0
0 1   2 -1  0
```

$(h * x)(0) = -2$

```
0 : 0 2 -1 0 0 0
0   1 2 -1 0
```

$(h * x)(1) = 5$

```
0 2 -1  0 0 0
0 1  2 -1 0
```

$(h * x)(2) = 0$

```
0 2 -1 0  0 0
  0  1 2 -1 0
```

$(h * x)(3) = -1$

```
0 2 -1 0 0  0 : 0
     0 1 2 -1   0
```

$(h * x)(4) = 0$

```
0 2 -1 0 0 0 : 0 0
       0 1 2  -1 0
```

$(h * x)(5) = 0$

d

$$x(n,m) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$x_{padding}(n,m) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$h(n,m) = \begin{bmatrix} 1 & 1 \\ *1 & 1 \end{bmatrix} h_{rotado}(n,m) = \begin{bmatrix} 1 & *1 \\ 1 & 1 \end{bmatrix}$$

donde * marca el centro

$$conv(n,m) = \begin{bmatrix} 2 & 4 & 4 \\ 2 & 4 & 4 \\ 1 & 2 & 2 \end{bmatrix}$$

2

a

$$x(n,m) = \begin{bmatrix} 1 & 4 & 1 \\ 2 & 5 & 3 \end{bmatrix}$$

i

$$h(n,m) = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} h_{rotado}(n,m) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

$$x_{padding}(n,m) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 \\ 0 & 2 & 5 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$conv(n,m) = \begin{bmatrix} -2 & 11 & 2 \\ 2 & 11 & 6 \end{bmatrix}$$

ii

$$h(n,m) = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} h_{rotado}(n,m) = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$$

$$x_{padding}(n,m) = \begin{bmatrix} 0 & 1 & 4 & 1 & 0 \\ 0 & 2 & 5 & 3 & 0 \end{bmatrix}$$

$$conv(n,m) = \begin{bmatrix} 6 & 12 & 14 \\ 9 & 19 & 21 \end{bmatrix}$$

iii

$$h(n,m) = \begin{bmatrix} -2 \\ 3 \\ -1 \end{bmatrix} h_{rotado}(n,m) = \begin{bmatrix} -1 \\ 3 \\ -2 \end{bmatrix}$$

$$x_{padding}(n,m) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 4 & 1 \\ 2 & 5 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

$$conv(n,m) = \begin{bmatrix} -1 & 2 & -3 \\ 5 & 11 & 8 \end{bmatrix}$$

b

Si tenemos que hacer una convolución entre un kernel de tamaño $M_1 \times N_1$ e imagen de tamaño $M_2 \times N_2$ tenemos que agregar suficiente padding para poder realizar el calculo. La imagen resultando termina siendo de $M_1 + M_2 - 1 \times N_1 + N_2 - 1$ ya que a los bordes le agregamos tantos pixeles de padding necesarios para que, al poner el centro del kernel en un pixel del borde podemos calcular el pixel resultante. Esta cantidad para el borde superior corresponde a la mitad del alto del kernel para el padding superior, más la misma cantidad para el padding inferior (es decir, N_1 - 1 en total). Equivalentemente para el padding de los lados.

3

```python
import numpy as np

def conv_discreta(A, B):
    # Obtener las dimensiones de las dos matrices
    m1, n1 = A.shape
    m2, n2 = B.shape

    # matriz resultado con padding
    result = np.zeros((m1 + m2 - 1, n1 + n2 - 1))

    # Realizar la convolución bidimensional
    for i in range(m1):
        for j in range(n1):
```

```
            for m in range(m2):
                for n in range(n2):
                    result[i + m, j + n] += A[i, j] * B[m, n]

    return result

x = np.array([[1,4,1],[2,5,3]])

i = np.array([[0, -1, 1], [-1,4,-1], [0, -1, 0]])
print(conv_discreta(x, i))

ii = np.array([[1,2,3]])
print(conv_discreta(x, ii))

iii = np.array([[-2],[3],[-1]])
print(conv_discreta(x, iii))

### verificación propiedades convolución

A = np.array([[1, 2], [3, 4]])
B = np.array([[0, 1], [2, 3]])
C = np.array([[1, 1], [1, 1]])

# conmutatividad

print(conv_discreta(A, B) == conv_discreta(B,A))

# Distributiva

print(conv_discreta(A + B, C) == conv_discreta(A, C) + conv_discreta(B, C))

# Asociatividad

print(conv_discreta(conv_discreta(A, B), C) ==conv_discreta(A,conv_discreta(B,C)))
```

4

```
import ejercicio3
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt

path_output = "./salida/"

imagen =cv2.imread("./cameraman.jpg", cv2.IMREAD_GRAYSCALE)

def filtro_media(imagen, tam_filtro):
    vector_filtro = np.ones(tam_filtro).reshape(tam_filtro, 1)

    filtro = (vector_filtro @ vector_filtro.T) / (tam_filtro * tam_filtro)

    res = ejercicio3.conv_discreta(imagen, filtro)

    cv2.imwrite(path_output + "media-" + str(tam_filtro) + ".jpg", res)
```

```
def filtro_media_ej_c(imagen):
    filtro = np.array([[1,1],[1,1],[1,1]])
    res = ejercicio3.conv_discreta(imagen, filtro)

    cv2.imwrite(path_output + "media-ej4c" + ".jpg", res)

def func_gauss_separada(x, sigma):
    return math.e**-(x**2 / 2*(sigma**2))

def filtro_gauss(imagen, tam, sigma):
    filtro_x = np.ones(tam)
    filtro_y = np.ones(tam)

    for i in range(-(tam//2), tam//2 +1):
        filtro_x[i+(tam//2)] = func_gauss_separada(i, sigma)
        filtro_y[i+(tam//2)] = filtro_x[i+(tam//2)]

    filtro_x = filtro_x.reshape(filtro_x.shape[0], 1)
    filtro_y = filtro_y.reshape(1, filtro_y.shape[0])
    mat_filtro = filtro_x @ filtro_y

    factor_normalizacion = np.sum(mat_filtro)

    ### matriz de filtro
    #mat_filtro = mat_filtro / factor_normalizacion
    #res = ejercicio3.conv_discreta(imagen, mat_filtro)


    ### vectores de filtro (por separado)
    res = ejercicio3.conv_discreta(imagen, filtro_x)
    res = ejercicio3.conv_discreta(res, filtro_y)
    res = res / factor_normalizacion

    cv2.imwrite(path_output + "gauss_tam" + str(tam) + "_sigma_" + str(sigma) + "_separado" + ".jpg", res)
    return res

# tipo
# 0: min
# 1: max
# 2: mediana

def filtro_mmm(imagen, tam, tipo):

    if tipo == 0:
        operacion = np.min
        str_salida = "minimo_"
    elif tipo == 1:
        operacion = np.max
        str_salida = "maximo_"
    else:
        operacion = np.median
        str_salida = "mediana_"

    imagen_con_padding = np.pad(imagen, [(0, tam-1),(0, tam- 1)])

    m, n = imagen.shape
```

```
    res = np.zeros(imagen.shape)

    for i in range(m):
        for j in range(n):
            ventana = imagen_con_padding[i:i+tam, j:j+tam]
            res[i, j] = operacion(ventana)

    cv2.imwrite(path_output + str_salida + "tam_" + str(tam) + ".jpg", res)


def test():
    filtro_media(imagen, 3)
    filtro_media(imagen, 21)
    filtro_media_ej_c(imagen)

    filtro_gauss(imagen, 3, 1)
    filtro_gauss(imagen, 9, 1)

    filtro_mmm(imagen, 3, 0)
    filtro_mmm(imagen, 9, 0)

    filtro_mmm(imagen, 3, 1)
    filtro_mmm(imagen, 9, 1)

    filtro_mmm(imagen, 3, 2)
    filtro_mmm(imagen, 9, 2)
```

5

```
def unsharp_mask(imagen, a, sigma):
    imagen_padding = np.pad(imagen, [(0, 2),(0, 2)])
    gauss_img = ejercicio4.filtro_gauss(imagen, 3, sigma)

    mask = imagen_padding - gauss_img

    res = imagen_padding + a * mask

    cv2.imwrite(path_output + "unsharp_masking_sigma_" + str(sigma) + "_factor_" + str(a) + ".jpg", res)

def test():
    unsharp_mask(imagen, 1, 1)
    unsharp_mask(imagen, 1, 2.5)
    unsharp_mask(imagen, 1, 10)
    unsharp_mask(imagen, 1, 20)
    unsharp_mask(imagen, 2.5, 10)
    unsharp_mask(imagen, 2.5, 15)
    unsharp_mask(imagen, 5, 2.5)
    unsharp_mask(imagen, 10, 5)
    unsharp_mask(imagen, 10, 10)
```