KWANGWOON UNIVERSITY

Image Convolution

2018.9.11

Seoungjun Oh(sjoh@kw.ac.kr)
Wooju Lee (krosea@kw.ac.kr)

Multimedia LAB

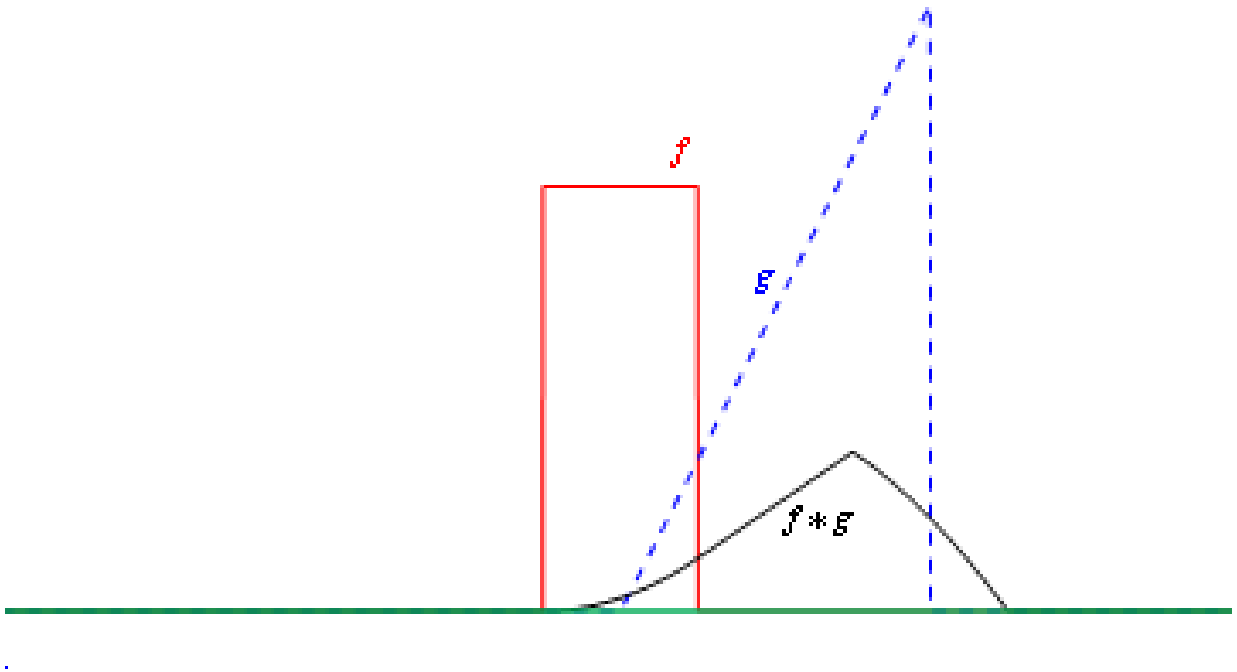
VIA-Multimedia Center, Kwangwoon University

Contents

- Convolution
- Various Masks
- 2-D Gaussian Function
- 2-D Gaussian Filtering
- Assignment

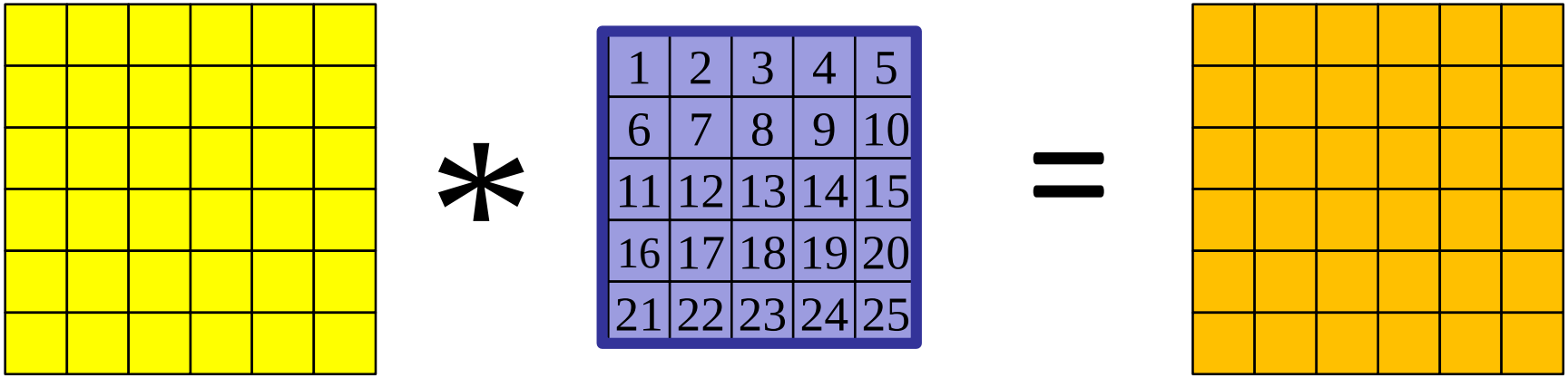
Convolution

- Convolution



Convolution

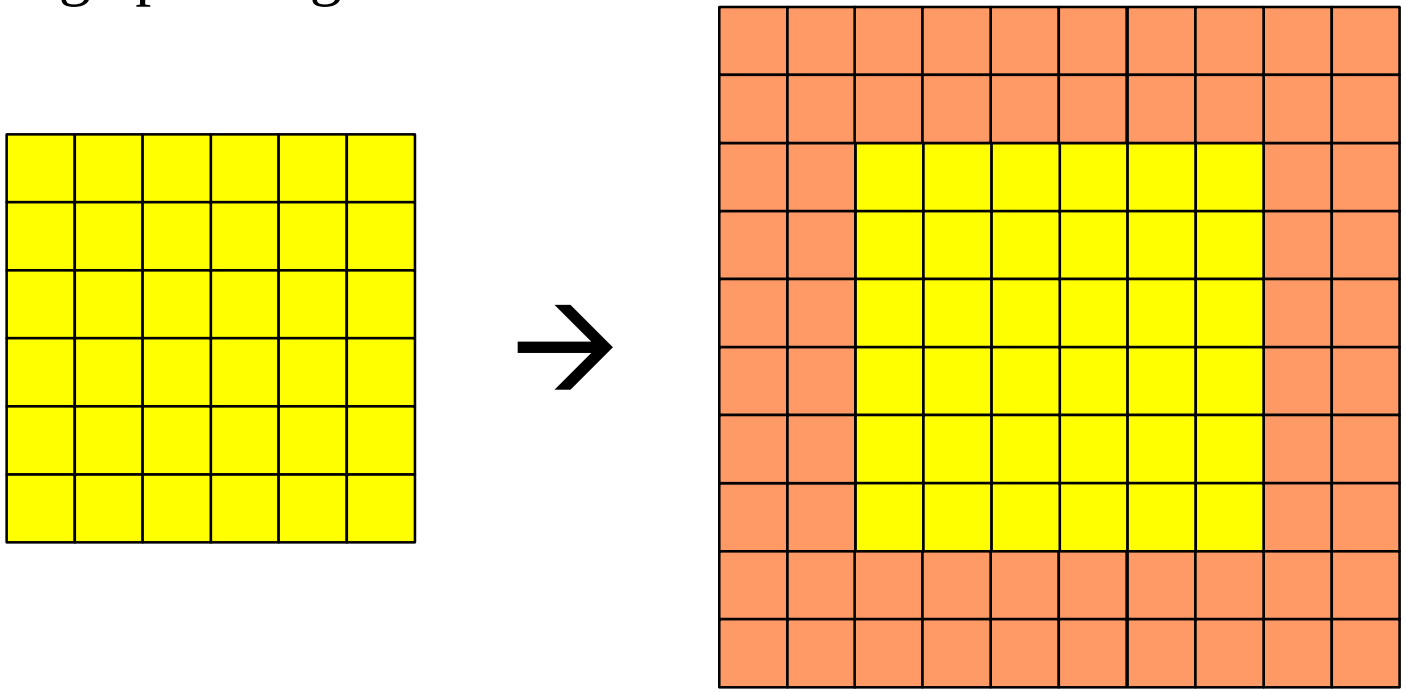
- 2-D discrete convolution



< Input image > < Mask> < Output image >

Convolution

- 2-D discrete convolution
  - Image padding



Convolution

- 2-D discrete convolution
  - Mask reverse

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

→

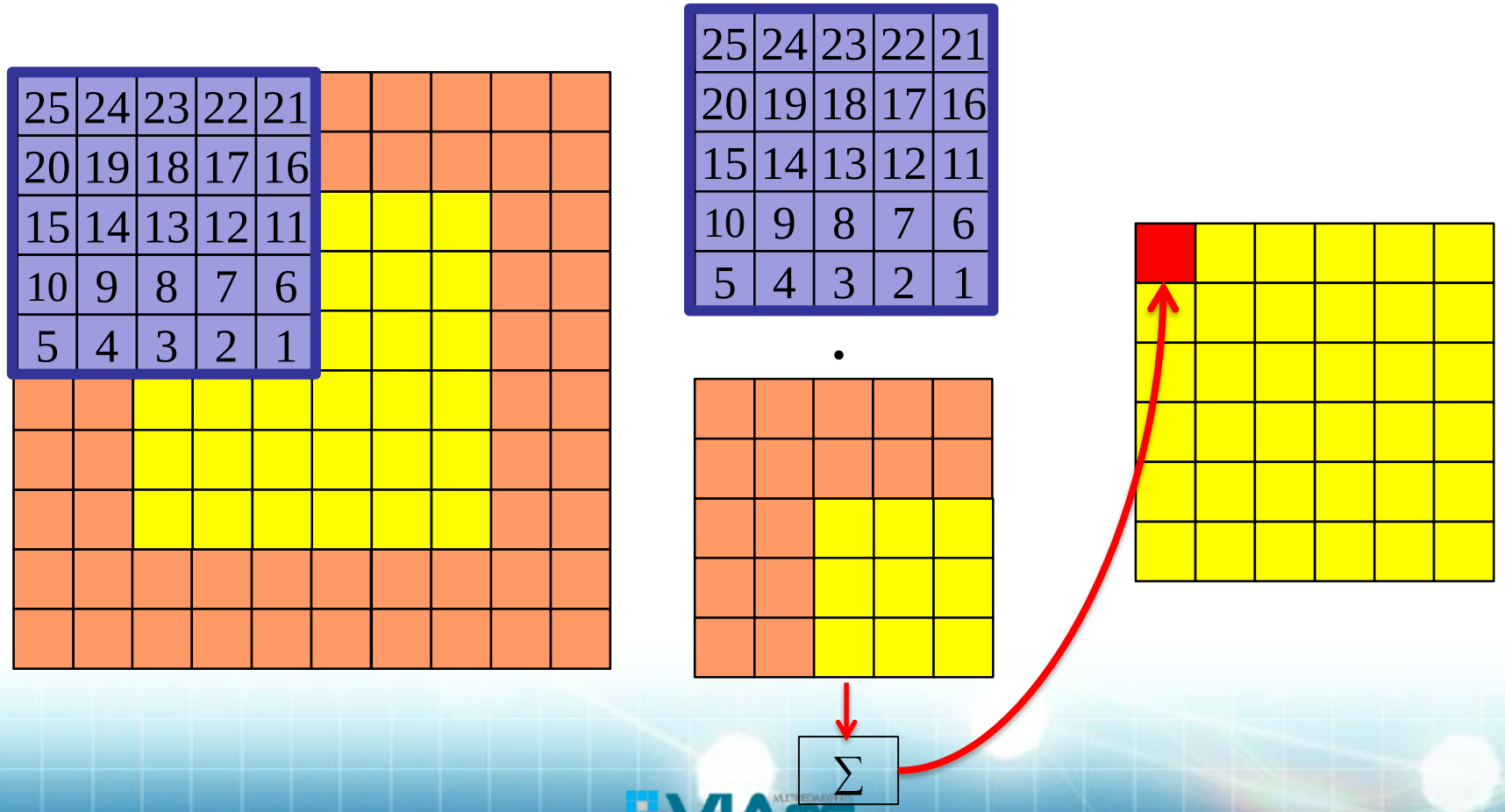
5	4	3	2	1
10	9	8	7	6
15	14	13	12	11
20	19	18	17	16
25	24	23	22	21

→

25	24	23	22	21
20	19	18	17	16
15	14	13	12	11
10	9	8	7	6
5	4	3	2	1

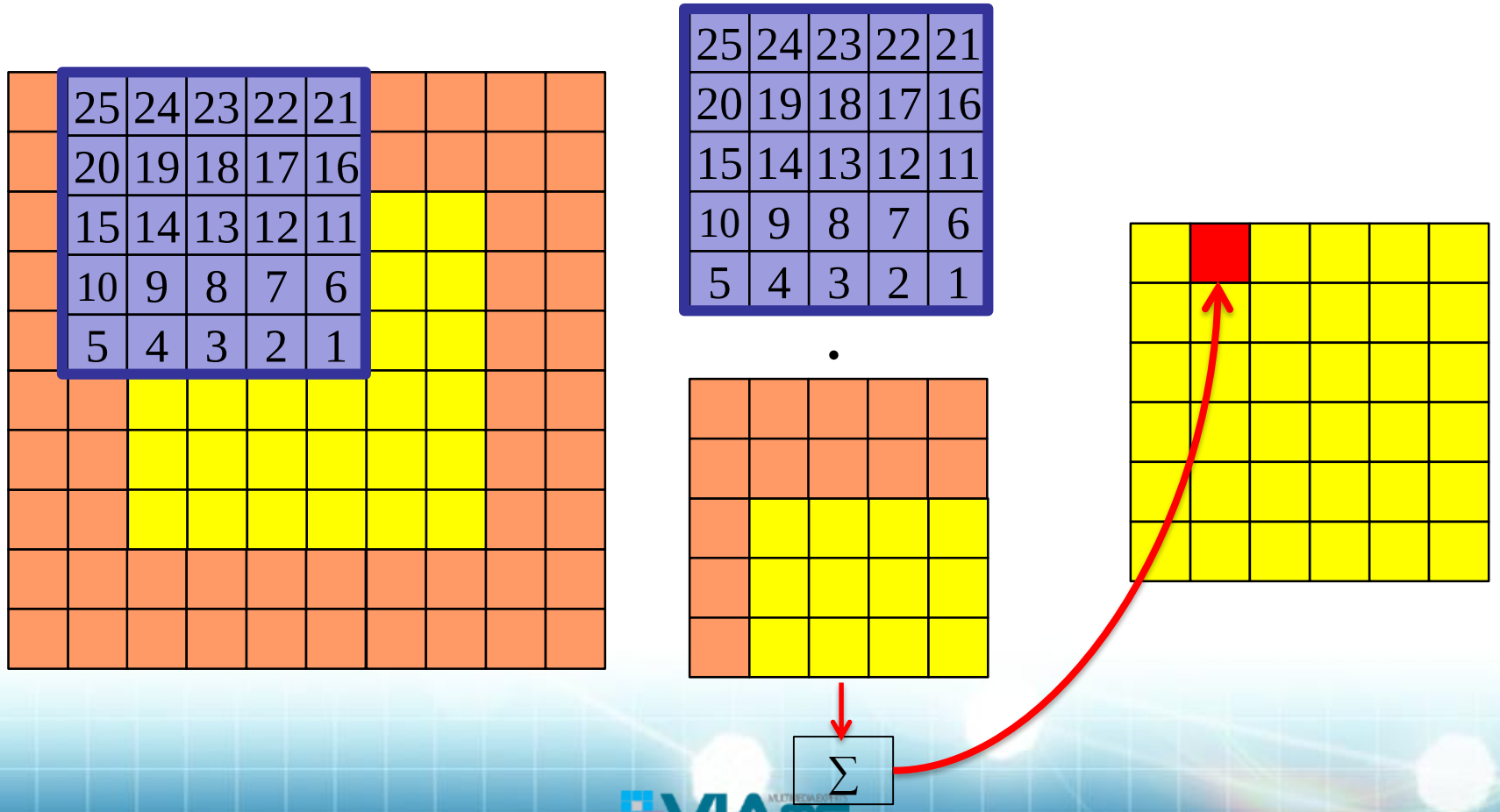
Convolution

- 2-D discrete convolution



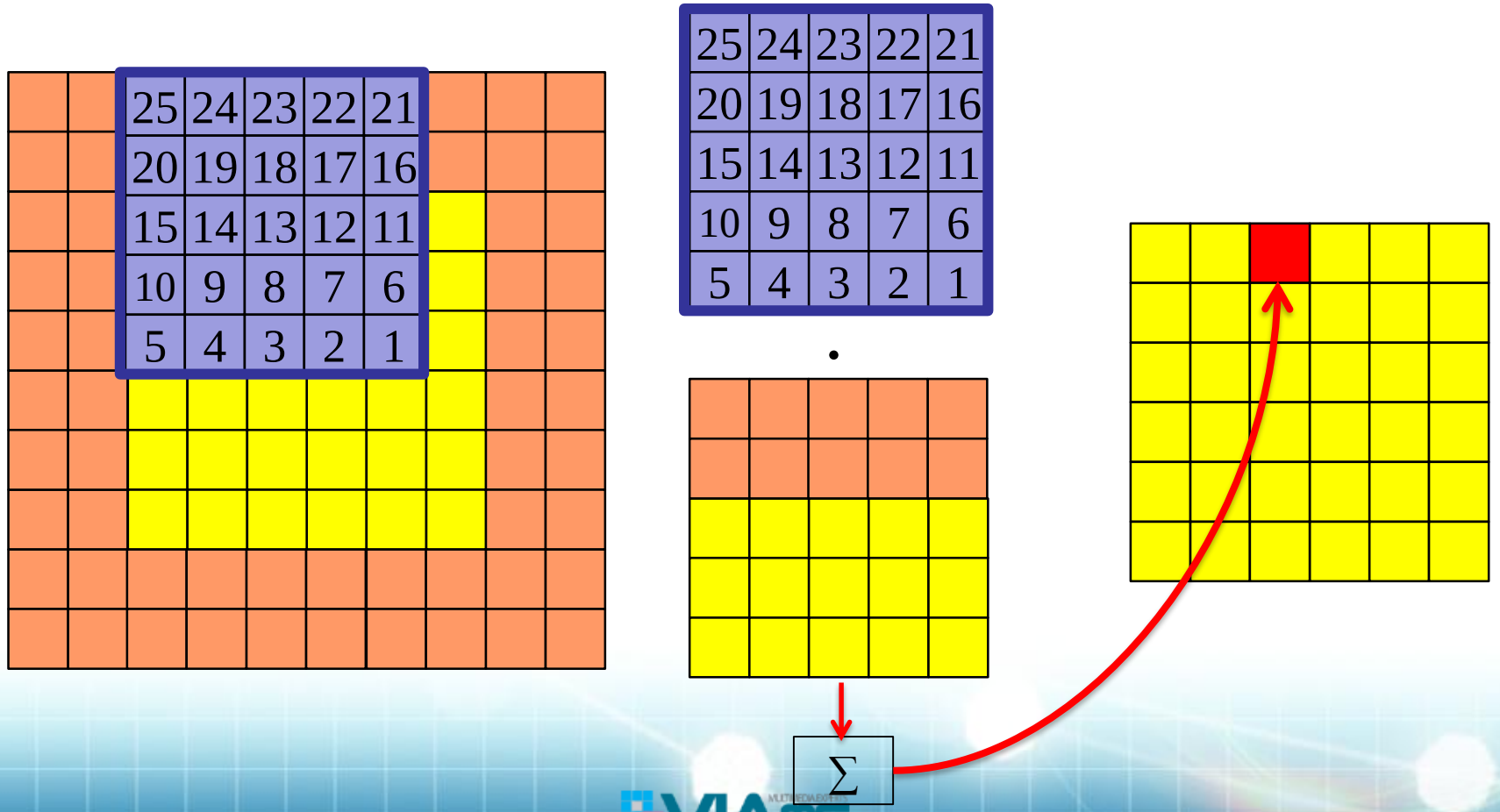
Convolution

- 2-D discrete convolution



Convolution

- 2-D discrete convolution



Various Masks

- Example

*

0	0	0
0	1	0
0	0	0

=

Original

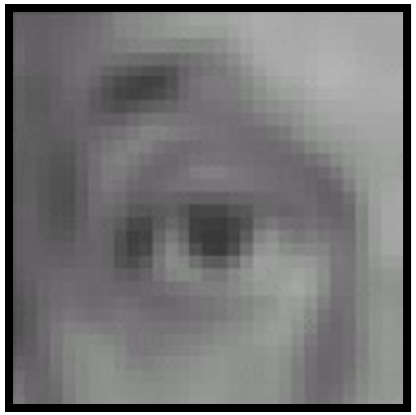
Identical image

Various Masks

- Example

Original

$$* \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a mean filter)

Various Masks

- Example (Sobel mask)

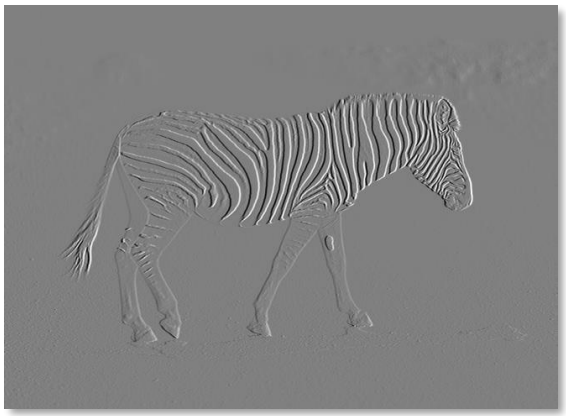
f

$*$

$\frac{1}{8}$

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

$=$



$\frac{\partial f}{\partial x}$

Various Masks

- Example (Sobel mask)

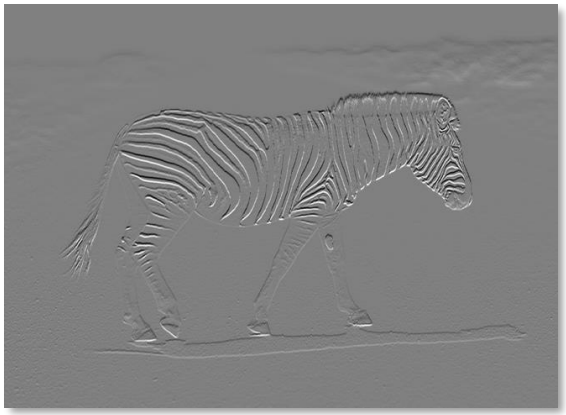
f

$*$

$\frac{1}{8}$

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

$=$



$\frac{\partial f}{\partial y}$

Various Masks

- Example

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Various Masks

- Example (Laplacian mask)

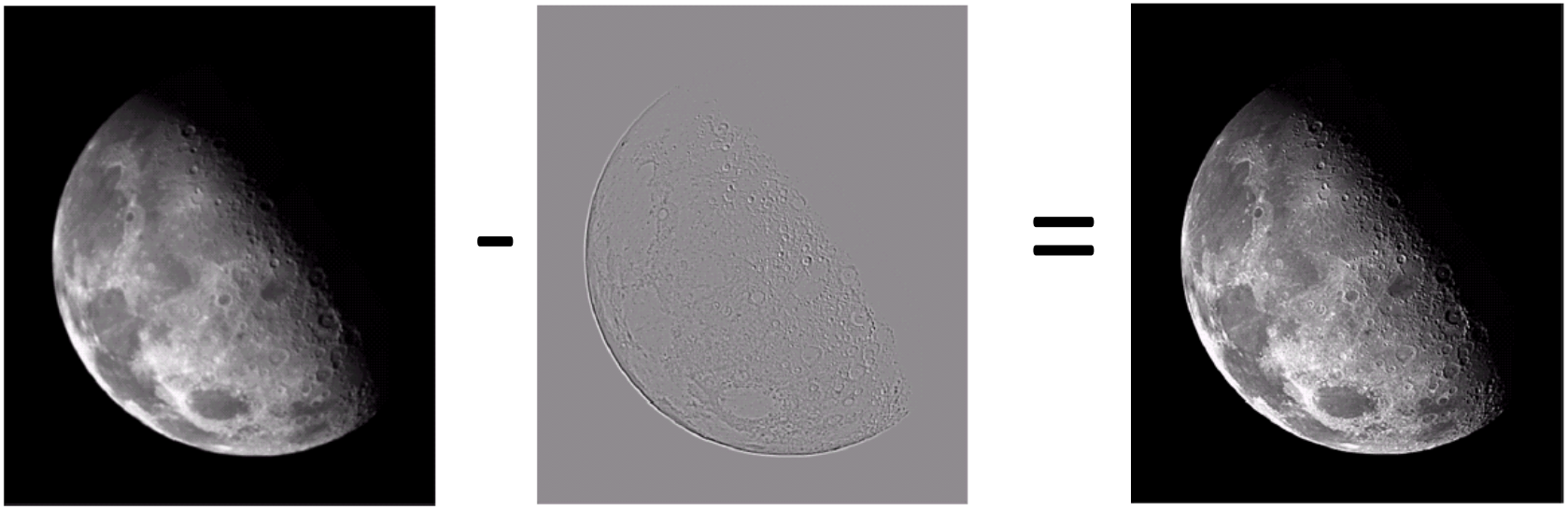
$*$

1	1	1
1	-8	1
1	1	1

$=$

Various Masks

- Example (Sharpening mask)



Various Masks

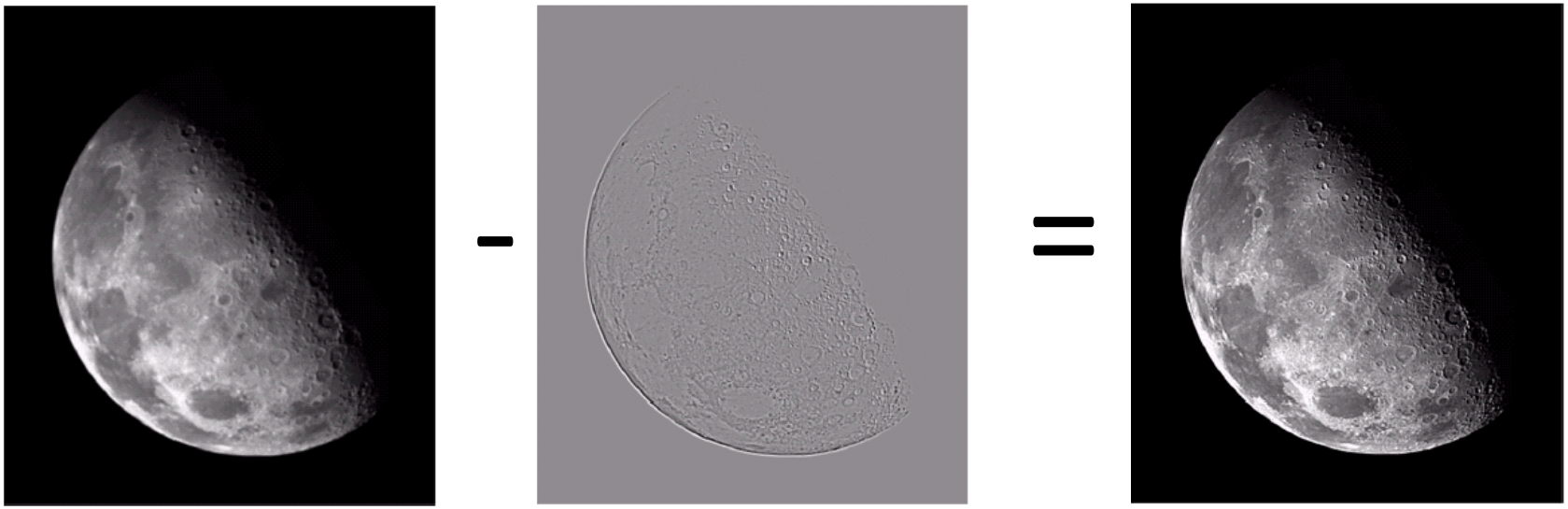
- Example (Sharpening mask)

Original

Sharpening

Various Masks

- Example (Sharpening mask)

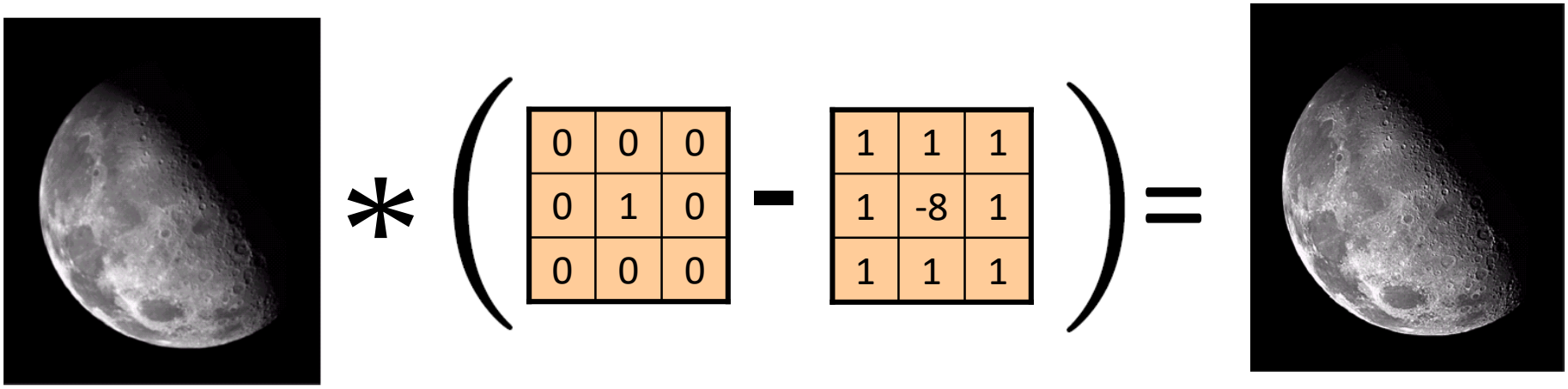


$$*\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$*\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Various Masks

- Example (Sharpening mask)



Various Masks

- Example (Sharpening mask)

*

-1	-1	-1
-1	9	-1
-1	-1	-1

=

2-D Gaussian Function

- 2-Dimensional Gaussian function

$$G(x,y)=\frac{1}{\sqrt{2\pi{\sigma_x}^2}}\frac{1}{\sqrt{2\pi{\sigma_y}^2}}e^{-\left(\frac{(x-\mu_x)^2}{2{\sigma_x}^2}+\frac{(y-\mu_y)^2}{2{\sigma_y}^2}\right)}$$

where
μ_x : expected value of X
μ_y : expected value of Y
${\sigma_x}^2$: variance of X
${\sigma_y}^2$: variance of Y

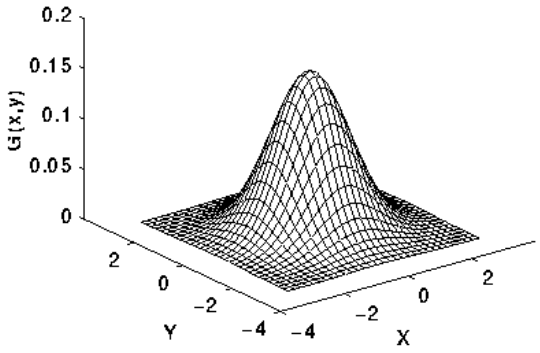
Circularly symmetric Gaussian

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{where } \mu_x=\mu_y=0,\ {\sigma_x}^2={\sigma_y}^2=\sigma^2$$

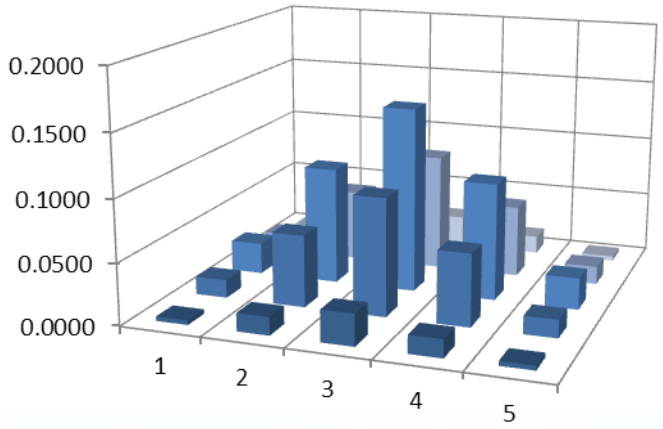
2-D Gaussian Function

- 2-Dimensional Gaussian function
  - Circularly symmetric Gaussian

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad \text{where } \mu_x = \mu_y = 0, \quad {\sigma_x}^2 = {\sigma_y}^2 = \sigma^2$$



< 2-D Gaussian distribution with mean(0,0) and σ = 1 >

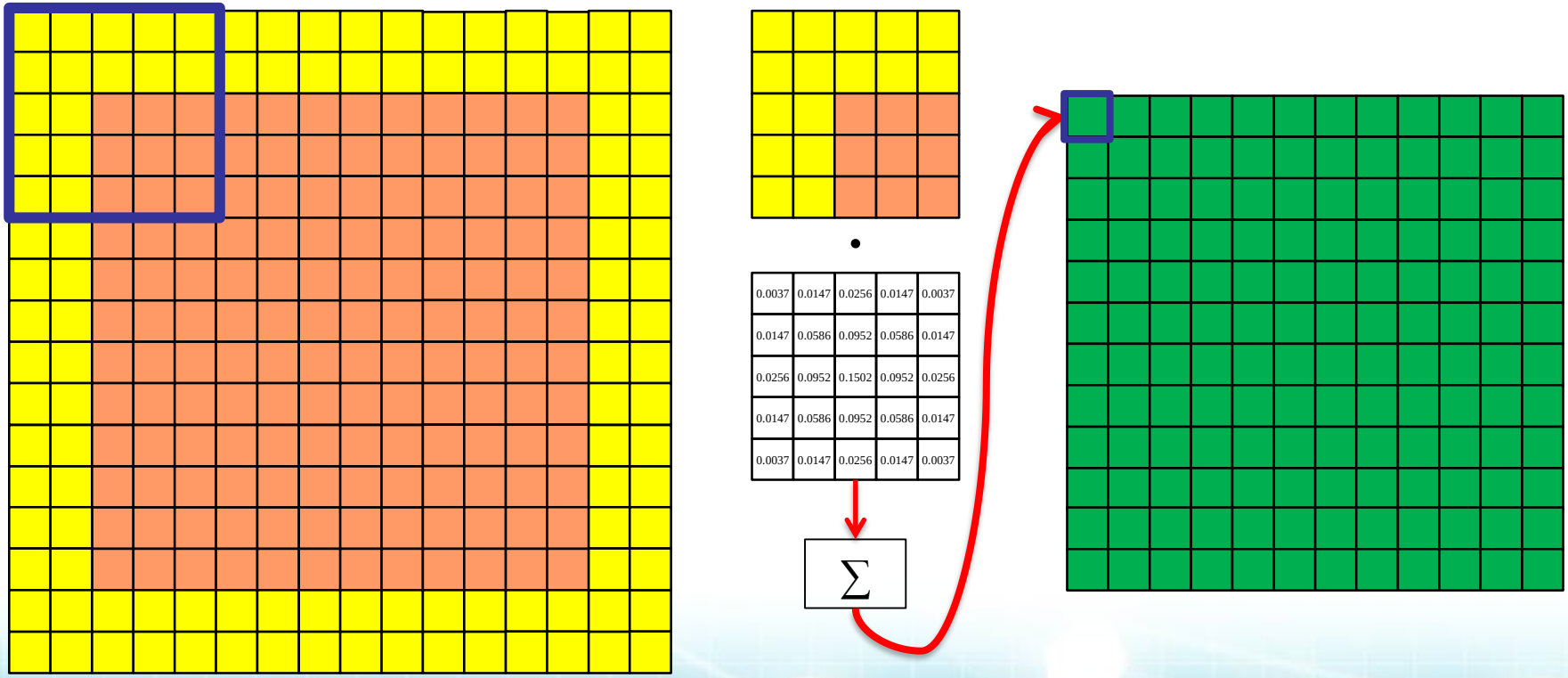


0.0037	0.0147	0.0256	0.0147	0.0037
0.0147	0.0586	0.0952	0.0586	0.0147
0.0256	0.0952	0.1502	0.0952	0.0256
0.0147	0.0586	0.0952	0.0586	0.0147
0.0037	0.0147	0.0256	0.0147	0.0037

< Discrete approximation to 2-D Gaussian function with mean(0,0) and σ = 1 >

2-D Gaussian Filtering

- Image convolution with 2-D Gaussian filter

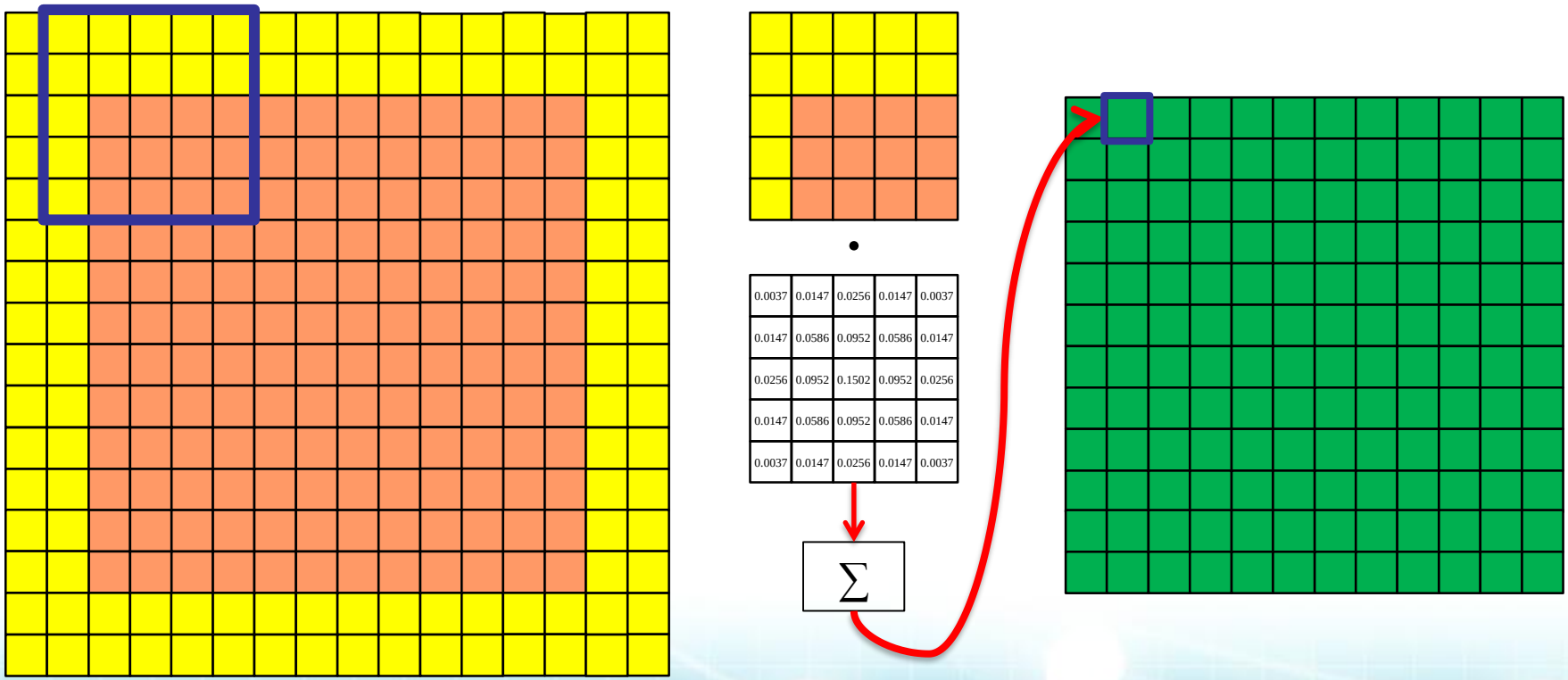


< Input image >

< Output image >

2-D Gaussian Filtering

- Image convolution with 2-D Gaussian filter

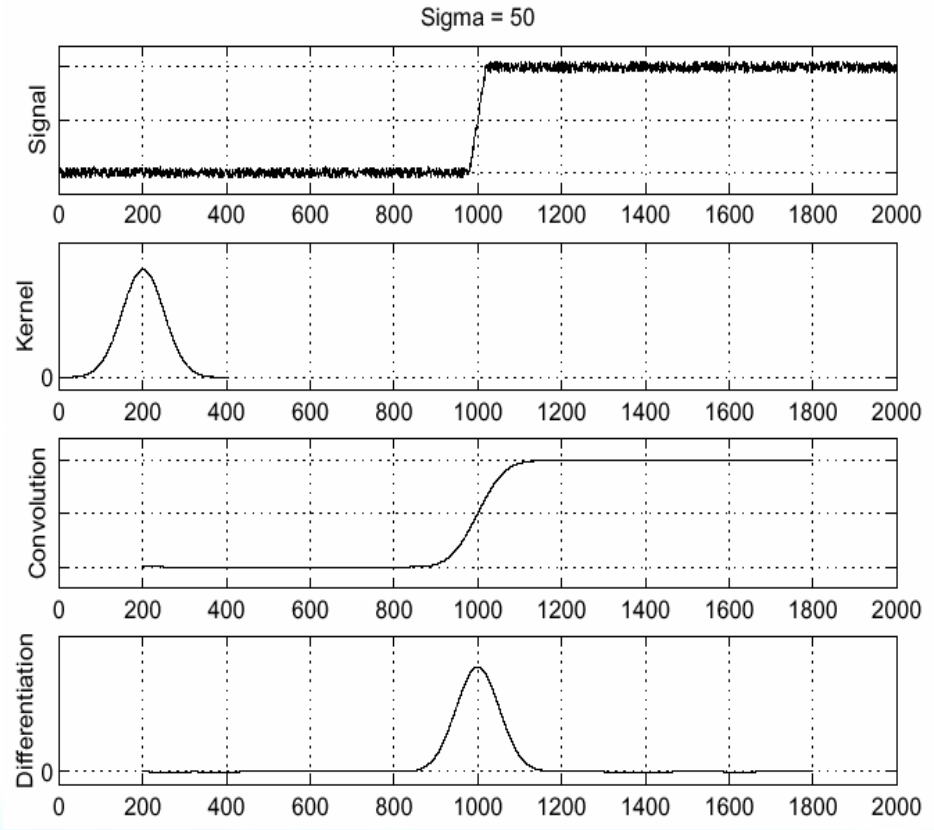


< Input image >

< Output image >

2-D Gaussian Function

- Gaussian function example



2-D Gaussian Filtering

❖ Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define WIDTH 512
#define HEIGHT 512

typedef unsigned char BYTE;

unsigned char** MemAlloc_2D(int width, int height);
void MemFree_2D(unsigned char** arr, int height); //BYTE
void GaussianFilter_5x5(unsigned char ** img_in, unsigned char** img_out, int width, int height); //BYTE
void GaussianFilter_9x9(unsigned char ** img_in, unsigned char** img_out, int width, int height);

int main()
{
    FILE *fp_in = 0, *fp_out = 0;
    BYTE **img_in = 0, **img_out = 0;
    int i = 0, n = 0;

    fp_in = fopen("Lena(512x512).raw", "rb");
    if (fp_in == NULL) {
        printf("File open failed₩n");
        return -1;
    }

    printf("2D-가우시간 필터링의 마스크 사이즈를 입력하세요.(5 OR 9) : ");
    scanf("%d", &n);

    img_in = MemAlloc_2D(WIDTH, HEIGHT);
    img_out = MemAlloc_2D(WIDTH, HEIGHT);

    for (i = 0; i < HEIGHT; i++) {
        fread(img_in[i], sizeof(BYTE), WIDTH, fp_in);
    }
```

2-D Gaussian Filtering

❖ Example

```
36      if (n==5){
37          GaussianFilter_5x5(img_in, img_out, WIDTH, HEIGHT);
38          fp_out = fopen("[Gaussian_5x5]Lena(512x512).raw", "wb");          // Output file open(.raw)
39      }
40
41      else if (n==9){
42          GaussianFilter_9x9(img_in, img_out, WIDTH, HEIGHT);
43          fp_out = fopen("[Gaussian_9x9]Lena(512x512).raw", "wb");          // Output file open(.raw)
44      }
45      else{
46          printf("Not valid mask size₩n");
47          return -1;
48      }
49
50      if(fp_out == NULL){
51          printf("File open failed₩n");
52          return -1;
53      }
54
55      for(i = 0 ; i < HEIGHT ; i++){                              // Output file write
56          fwrite(img_out[i], sizeof(BYTE), WIDTH, fp_out);
57      }
58      MemFree_2D(img_in, HEIGHT);
59      MemFree_2D(img_out, HEIGHT);
60      fclose(fp_in);                          // File close
61      fclose(fp_out);
62
63      return 0;
64  }
```

2-D Gaussian Filtering

❖ Example

```
77  void GaussianFilter_5x5(unsigned char** img_in, unsigned char** img_out, int width, int height)
78  {
79      int i, j, m, n;
80      double temp;
81      unsigned char **img_in_pad;
82      double Gauss_5x5[5][5] =                        //  5x5 Gaussian mask
83      {
84          {0.0037, 0.0147, 0.0256, 0.0147, 0.0037 },
85          {0.0147, 0.0586, 0.0952, 0.0586, 0.0147 },
86          {0.0256, 0.0952, 0.1502, 0.0952, 0.0256 },
87          {0.0147, 0.0586, 0.0952, 0.0586, 0.0147 },
88          {0.0037, 0.0147, 0.0256, 0.0147, 0.0037 }
89      };
90
91      img_in_pad = (unsigned char**)malloc(sizeof(unsigned char*) * (height+4));      //  Memory allocation
92      for(i = 0 ; i < height+4 ; i++)
93          img_in_pad[i] = (unsigned char*)malloc(sizeof(unsigned char) * (width+4));
94
95
96      for (i = 0; i < height; i++){                //data copy
97          for (j = 0; j < width; j++){
98              img_in_pad[i+2][j+2] = img_in[i][j];
99          }
100     }
101     for(i = 2 ; i < height + 2; i++){                           // Padding
102         for(j = 0 ; j < 2 ; j++){
103             img_in_pad[i][j] = img_in_pad[i][2];
104             img_in_pad[i][width+2 + j] = img_in_pad[i][width+2 - 1];
105         }
106     }
```

2-D Gaussian Filtering

❖ Example

```
108     for(j = 2 ; j < width + 2 ; j++){
109         for(i = 0 ; i < 2 ; i++){
110             img_in_pad[i][j] = img_in_pad[2][j];
111             img_in_pad[height+2 + i][j] = img_in_pad[height+2 - 1][j];
112         }
113     }
114
115     for(i = 0 ; i < 2 ; i++){
116         for(j = 0 ; j < 2 ; j++){
117             img_in_pad[i][j] = img_in_pad[2][2];
118             img_in_pad[i][width+2 + j] = img_in_pad[2][width+2 - 1];
119             img_in_pad[height+2 + i][j] = img_in_pad[height+2 - 1][2];
120             img_in_pad[height+2 + i][width+2 + j] = img_in_pad[height+2 - 1][width+2 - 1];
121         }
122     }
123
124
125     for(i = 0; i < height; i++){                    // 2-D Gaussian filtering
126         for(j = 0; j < width; j++){
127             temp = 0;
128             for(m = 0; m < 5 ; m++){
129                 for(n = 0 ; n < 5 ; n++){
130                     temp += img_in_pad[i + m][j + n] * Gauss_5x5[m][n];
131                 }
132             }
133             img_out[i][j] = (unsigned char)floor(temp + 0.5);
134         }
135     }
136     for(i = 0 ; i < height+4 ; i++)         //memory free
137         free(img_in_pad[i]);
138     free(img_in_pad);
139 }
```

2-D Gaussian Filtering

❖ Example

```
unsigned char** MemAlloc_2D(int width, int height) //BYTE // 추가
{
    unsigned char** MemA; //BYTE
    MemA = (unsigned char**)malloc(sizeof(unsigned char*)*height); //BYTE
    for (int i = 0; i < height; i++) {
        MemA[i] = (unsigned char*)malloc(sizeof(unsigned char)*width); //BYTE
    }
    return MemA;
}

void MemFree_2D(unsigned char** arr, int height) //BYTE // 추가
{
    for (int i = 0; i < height; i++)
        free(arr[i]);
    free(arr);
}
```

Assignment

❖ Example

```
void GaussianFilter_9x9(unsigned char** img_in, unsigned char** img_out, int width, int height)
{

                                ?


}
```

2-D Gaussian Filtering

- Example

< Original image >

< 5x5 Gaussian filter >

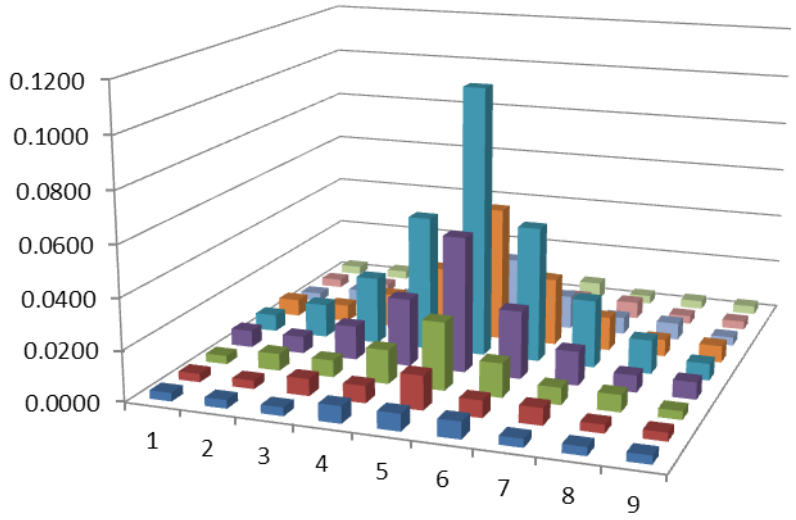
KWANGWOON UNIVERSITY

Assignment

VIAME MULTIMEDIA EXPERTS

Assignment

- Assignment #1
  - 9x9 Gaussian filter



0.0033	0.0033	0.0033	0.0067	0.0067	0.0067	0.0033	0.0033	0.0033
0.0033	0.0033	0.0067	0.0067	0.0134	0.0067	0.0067	0.0033	0.0033
0.0033	0.0067	0.0067	0.0134	0.0268	0.0134	0.0067	0.0067	0.0033
0.0067	0.0067	0.0134	0.0268	0.0535	0.0268	0.0134	0.0067	0.0067
0.0067	0.0134	0.0268	0.0535	0.1070	0.0535	0.0268	0.0134	0.0067
0.0067	0.0067	0.0134	0.0268	0.0535	0.0268	0.0134	0.0067	0.0067
0.0033	0.0067	0.0067	0.0134	0.0268	0.0134	0.0067	0.0067	0.0033
0.0033	0.0033	0.0067	0.0067	0.0134	0.0067	0.0067	0.0033	0.0033
0.0033	0.0033	0.0033	0.0067	0.0067	0.0067	0.0033	0.0033	0.0033

Assignment

- Assignment #1
  - 9x9 Gaussian filter

< Original image >

< 9x9 Gaussian filter >

Assignment

- Assignment #2
  - Sobel mask

$$* \quad \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \quad ?$$

$$* \quad \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \quad ?$$

Assignment

- Assignment #3
  - Laplacian mask and sharpening

$$*\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

$$*\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} = ?$$

Assignment

- Assignment #4
  - Laplacian of Gaussian (LOG)

$$*\left(\begin{bmatrix} 0.0037 & 0.0147 & 0.0256 & 0.0147 & 0.0037 \\ 0.0147 & 0.0586 & 0.0952 & 0.0586 & 0.0147 \\ 0.0256 & 0.0952 & 0.1502 & 0.0952 & 0.0256 \\ 0.0147 & 0.0586 & 0.0952 & 0.0586 & 0.0147 \\ 0.0037 & 0.0147 & 0.0256 & 0.0147 & 0.0037 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}\right) = ?$$