

CSE185
Introduction to Computer Vision
Lab 03: Spatial Filters

Instructor: Prof. Ming-Hsuan Yang
TA: Taihong Xiao & Tiantian Wang

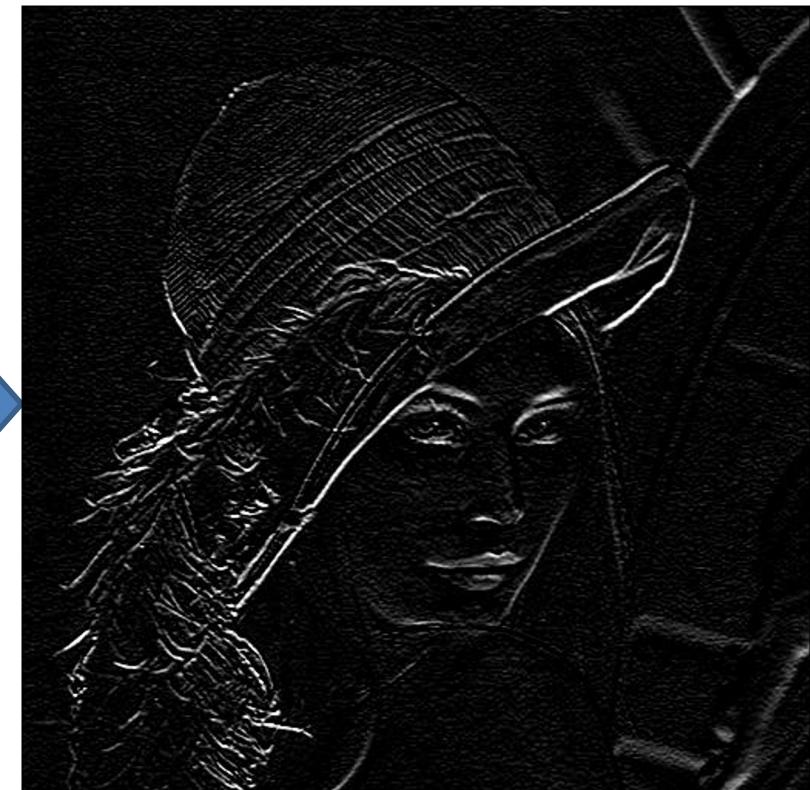
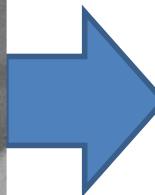
Sobel Filter

- Sobel filter is a simple edge detector

$$\bullet H_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \text{ or } H_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$



Input image



Sobel filter output

2D filter

- Submatrix Indexing:

- Code for H_x :

```
I1 = im2double(imread('lena.jpg'));

function sobel(I1)
    Hx = [-1 0 1; -2 0 2; -1 0 1]
    [rows, cols] = size(I1);
    I2 = zeros(rows+2, cols+2);
    img2(1+1:rows+1, 1+1:cols+1) = I1;

    for i=1:rows
        for j=1:cols
            S1(i,j) = sum(sum(Hx.*A(i:i+2, j:j+2)));
        end
    end
end function
```

Sobel Filter

- sobel_filter.m

```
function output = sobel_filter(img, kernel)
    % YOUR CODE HERE
end
```

- In lab03.m:

```
img = im2double(imread('lena.jpg'));

%% Sobel filter
H = [1, 2, 1; 0, 0, 0; -1, -2, -1]; % horizontal edge
%H = [1, 0, -1; 2, 0, -2; 1, 0, -1]; % vertical edge

img_sobel = sobel_filter(img, H);
figure, imshow(img_sobel);
imwrite(img_sobel, 'sobel_h.jpg');
```

- Compare your result with `I = imfilter(img, H);`

2D filter to 1D filter

$$\bullet H_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \longrightarrow H_x = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} * (1 \quad 0 \quad -1)$$

$$\bullet H_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \longrightarrow H_y = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} * (1 \quad 2 \quad 1)$$

- Pseudo code for H_x : convolve 1D filter column wise filter first, and then convolve 1D filter row wise.

```
function sobel(I1)
    HX1 = [1 1 1; 2 2 2; 1 1 1]; HX2 = [1 0 -1];
    tic
    [rows,cols] = size(I1);
    I2 = zeros(rows+2, cols+2);
    I2(1+1:rows+1, 1+1:cols+1) = I1;
    ouput = zeros(rows+2, cols+2);
    output2 = zeros(rows, cols);
    for i = 1:rows
        for j = 1:cols
            output(i,j:j+2) = sum(HX1.*I2(i:i+2,j:j+2), 1);
            output2(i,j) = sum(HX2.*output(i,j:j+2));
        end
    toc
end function
```

2D filter to 1D filter

$$\bullet H_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \rightarrow H_x = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} * (1 \quad 0 \quad -1)$$

$$\bullet H_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \rightarrow H_y = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} * (1 \quad 2 \quad 1)$$

- Pseudo code for H_x : convolve 1D filter row wise filter first, and then convolve 1D filter column wise.

```
function sobel(I1)
    HX1 = [1 0 -1; 1 0 -1; 1 0 -1]; HX2 = [1 2 1];
    tic
    [rows,cols] = size(I1);
    I2 = zeros(rows+2, cols+2);
    I2(1+1:rows+1,1+1:cols+1) = I1;
    output = zeros(rows+2, cols+2);
    output2 = zeros(rows, cols);
    for i = 1:rows
        for j = 1:cols
            output(i:i+2,j) = sum(HX1.*I2(i:i+2,j:j+2), 2);
            output2(i,j) = sum(HX2' .*output(i:i+2));
        end
    toc
end function
• Compare results and elapsed time with 2D filter and column-wise first row-
wise last
```

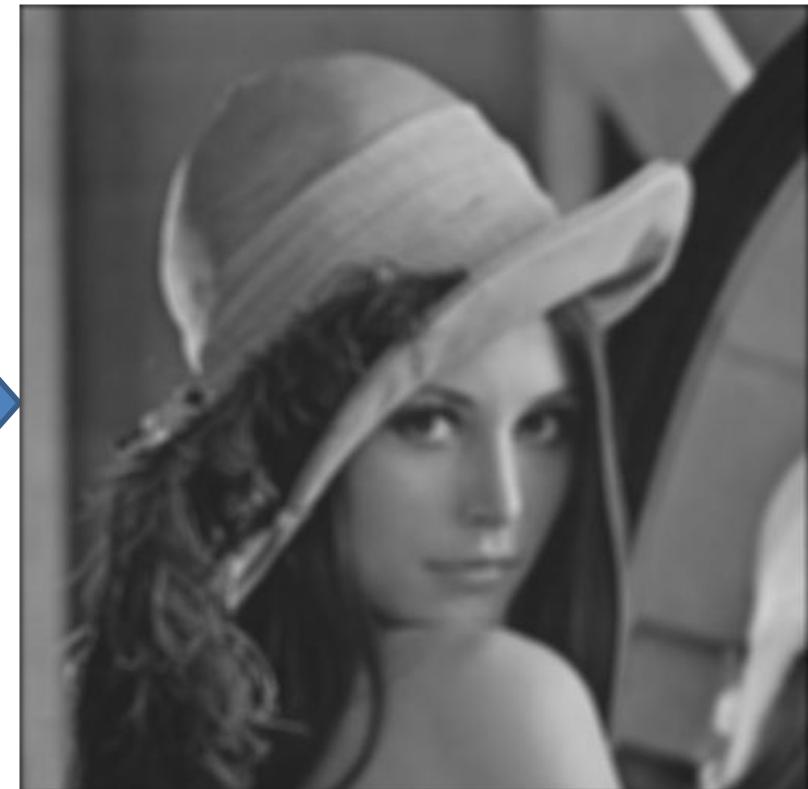
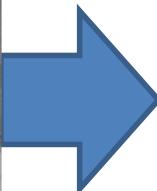
Gaussian Filter

- Gaussian filter is a low-pass/smoothing filter

$$g(\Delta x, \Delta y) = \frac{1}{Z} \exp\left(-\frac{\Delta x^2 + \Delta y^2}{2\sigma^2}\right)$$



Input image



Gaussian filter output

Gaussian Filter

- Gaussian filter is a low-pass/smoothing filter

$$g(\Delta x, \Delta y) = \frac{1}{Z} \exp\left(-\frac{\Delta x^2 + \Delta y^2}{2\sigma^2}\right)$$

- $\Delta x, \Delta y$ are offsets from the kernel center, and Z is the normalized term to make the sum of all weights equal to 1
- In MATLAB:

0.0113	0.0149	0.0176	0.0186	0.0176	0.0149	0.0113
0.0149	0.0197	0.0233	0.0246	0.0233	0.0197	0.0149
0.0176	0.0233	0.0275	0.029	0.0275	0.0233	0.0176
0.0186	0.0246	0.029	0.0307	0.029	0.0246	0.0186
0.0176	0.0233	0.0275	0.029	0.0275	0.0233	0.0176
0.0149	0.0197	0.0233	0.0246	0.0233	0.0197	0.0149
0.0113	0.0149	0.0176	0.0186	0.0176	0.0149	0.0113

Gaussian Filter

- Gaussian filter is a low-pass/smoothing filter

$$g(\Delta x, \Delta y) = \frac{1}{Z} \exp\left(-\frac{\Delta x^2 + \Delta y^2}{2\sigma^2}\right)$$

- $\Delta x, \Delta y$ are offsets from the kernel center, and Z is the normalized term to make the sum of all weights equal to 1
- In MATLAB:

0.0113	0.0149	0.0176	0.0186	0.0176	0.0149	0.0113
0.0149	0.0197	0.0233	0.0246	0.0233	0.0197	0.0149
0.0176	0.0233	0.0275	0.029	0.0275	0.0233	0.0176
0.0186	0.0246	0.029	0.0307	0.029	0.0246	0.0186
0.0176	0.0233	0.0275	0.029	0.0275	0.0233	0.0176
0.0149	0.0197	0.0233	0.0246	0.0233	0.0197	0.0149
0.0113	0.0149	0.0176	0.0186	0.0176	0.0149	0.0113

Gaussian Filter

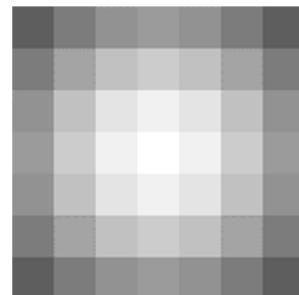
- You don't need to implement Gaussian kernel by yourself (this is bonus).

- Use `fspecial`:

```
H = fspecial('gaussian', hsize, sigma);
```

`hsize` is the size of the kernel, `sigma` = σ

- Visualization of a Gaussian kernel:



Gaussian Filter

- gaussian_filter.m

```
function output = gaussian_filter(img, hsize, sigma)
    H = fspecial('gaussian', hsize, sigma);
    % YOUR CODE HERE
end
```

- In lab03.m:

```
%% Gaussian filter
hsize = 5; sigma = 2;
% hsize = 9; sigma = 4;

img_gaussian = gaussian_filter(img, hsize, sigma);
figure, imshow(img_gaussian);
imwrite(img_gaussian, 'gaussian_5.jpg');
```

- Compare your result with $I = \text{imfilter}(img, H);$

Lab assignment 03

1. Implement `sobel_filter.m`, use H_y filter and save the image as `sobel_y.jpg`
2. Use H_x filter and save the image as `sobel_x.jpg`
3. Break 2D filter (H_y) to 1D filter, use column-wise filter first and then row-wise filter, save the image as `sobel_cr.jpg`
4. Use row-wise filter first and then column-wise filter, save the image as `sobel_rc.jpg`
5. Implement `gaussian_filter.m`, use `hsiz = 5`, `sigma = 2`, and save the image as `gaussian_5.jpg`
6. Use `hsiz = 9`, `sigma = 4`, and save the image as `gaussian_9.jpg`
7. Upload your output images and `lab03.m`, `sobel_filter.m`, and `gaussian_filter.m`

* All questions should utilize the ‘lena.jpg’

Bonus

- Implement boundary padding
 - zero padding (pixels outside the image are zero)
 - replicated/repeated padding (pixels outside the image are the same as pixels on the boundaries)
- You can try built-in function `padarray` or `wextend`, but you need to implement by yourself to get bonus points
- Implement Gaussian kernel
 - compute a matrix based on $g(\Delta x, \Delta y) = \exp\left(-\frac{\Delta x^2 + \Delta y^2}{2\sigma^2}\right)$
 - normalize the matrix to have sum equal to 1

Reference

- [Spatial Filtering](#)
- [Linear Algebra and MATLAB Tutorial](#)
- [Awesome Computer Vision](#)