CSE185
Introduction to Computer Vision
Lab 03: Spatial Filters

Instructor: Daniel Leung
TA: Mohammadkazem Ebrahimpour
Xueqing Deng

Overviews

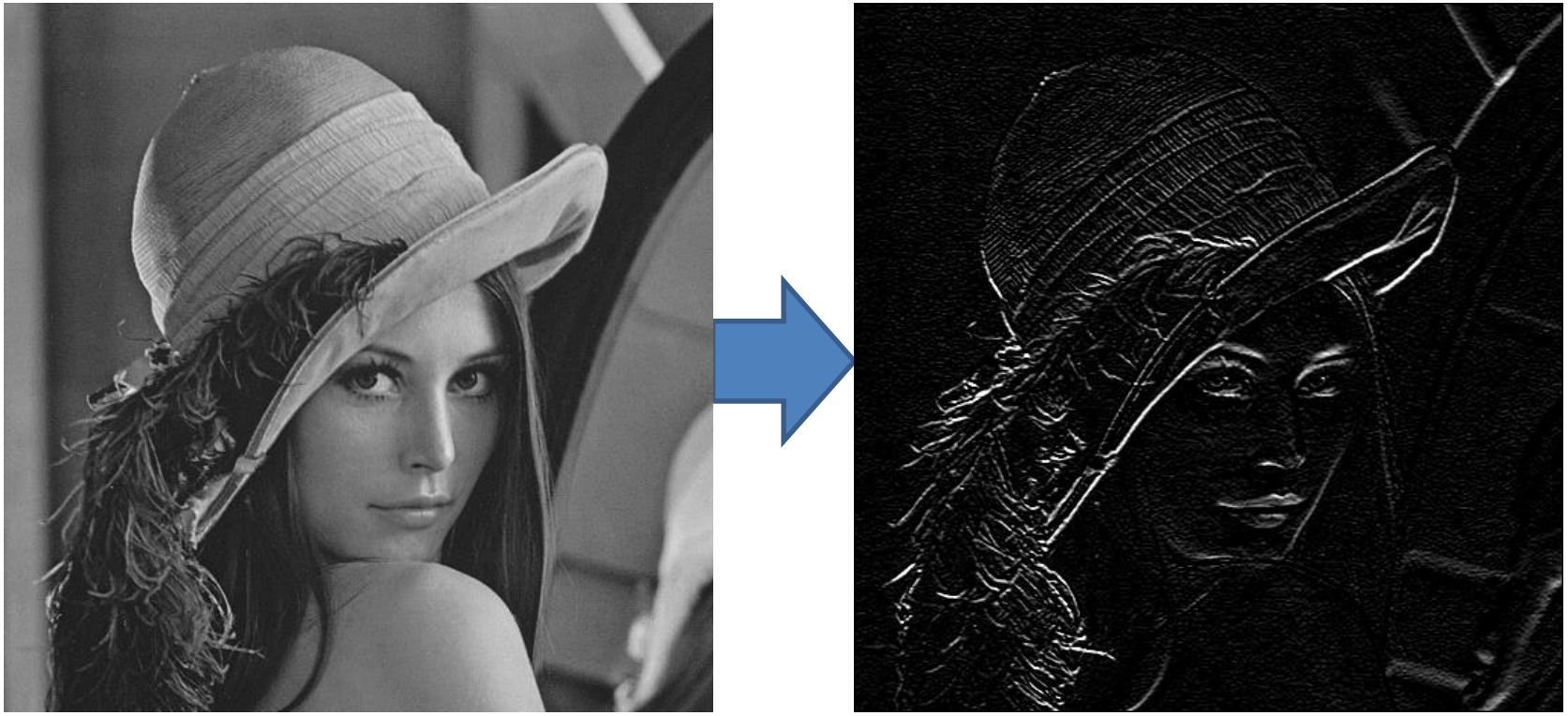
- Median filter:

Noisy input image

Median filter output

Overviews

- Sobel filter:

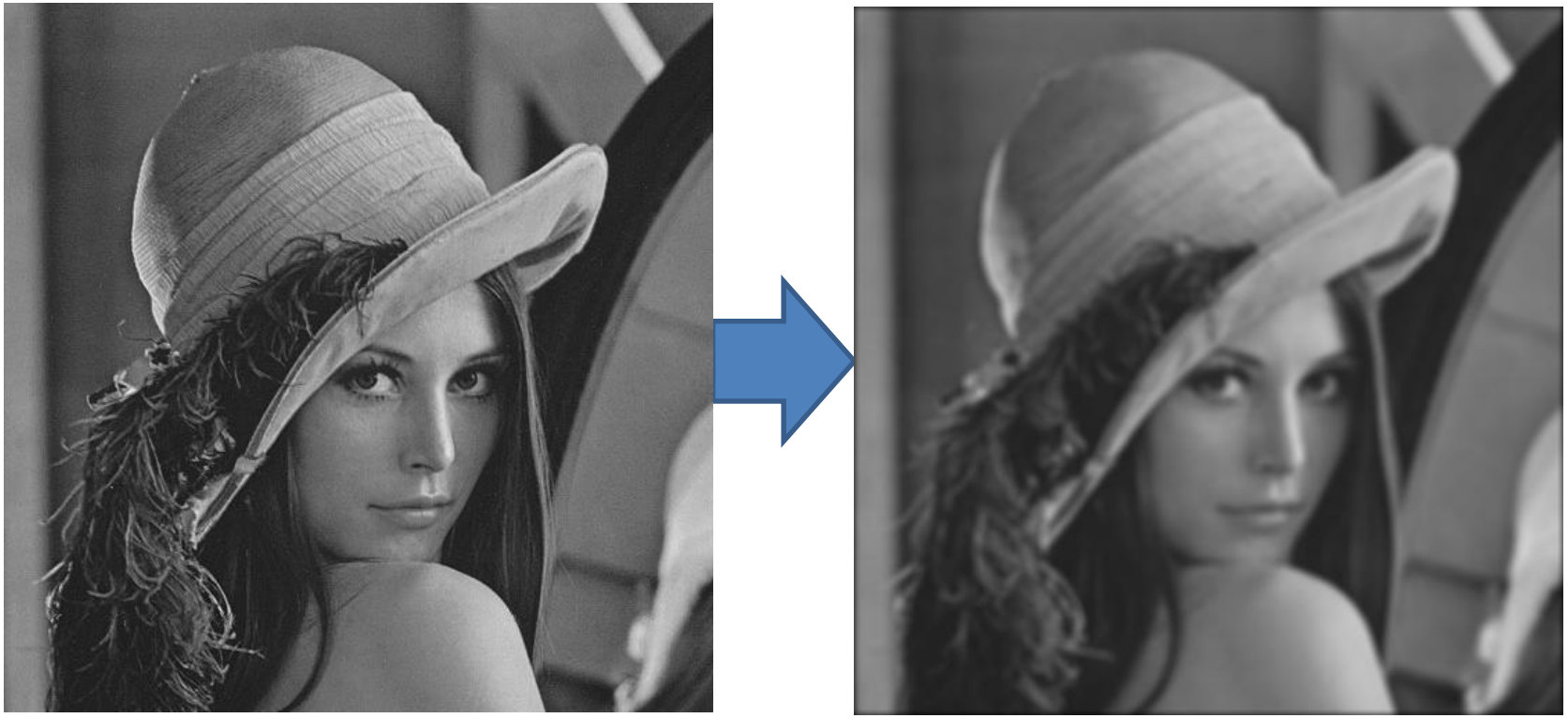


Input image

Sobel filter output

Overviews

- Gaussian filter:

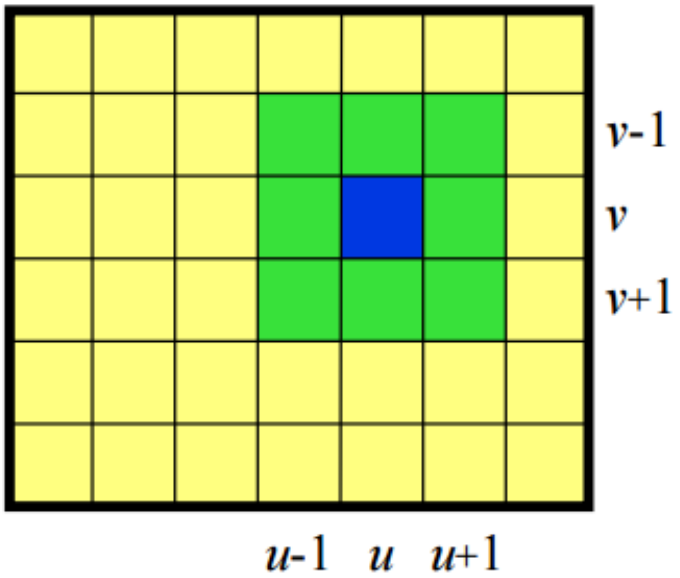


Input image

Gaussian filter output

Spatial Filter

- A **spatial filter** is an image operation where each pixel value $I(u, v)$ is changed by a function of the intensities of pixels in a neighborhood of (u, v).



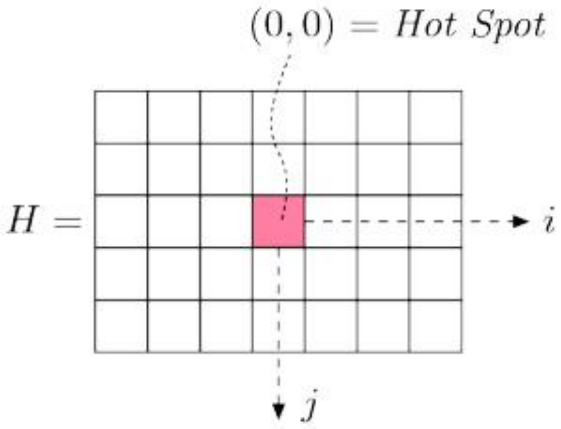
- Mean filter:

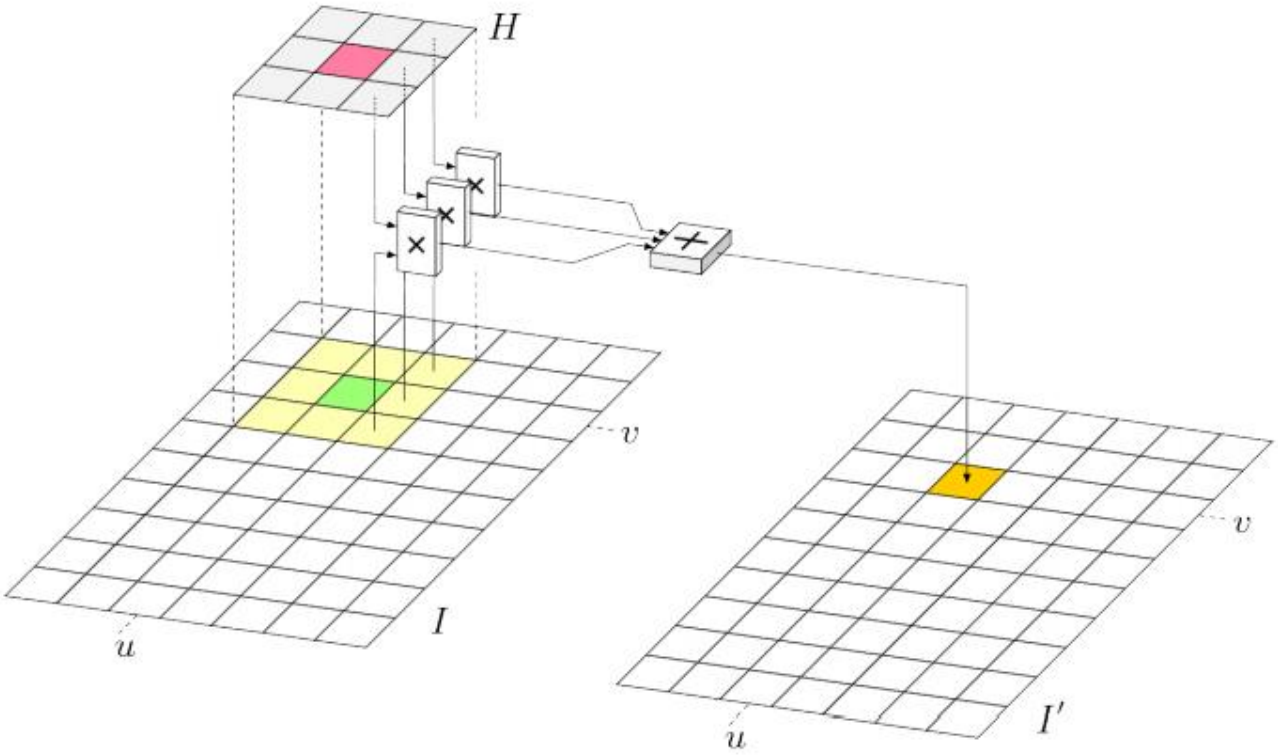
$$I'(u, v) = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u + i, v + j)$$

Linear Filter

- H is the filter matrix (kernel):

$$I'(u,v) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u+i, v+j) \cdot H(i,j)$$





Linear Filter

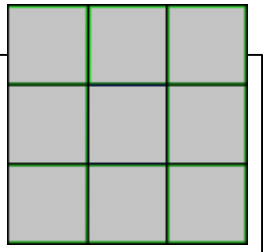
- Assume H is a 3×3 mean filter: $H = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

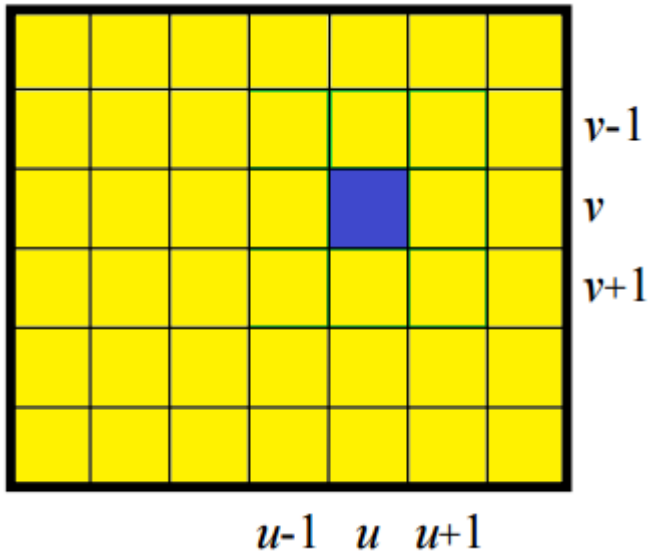
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = ???;
        I2(v, u) = value;

    end
end
```





Linear Filter

$$I'(u,v) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u+i, v+j) \cdot H(i,j)$$

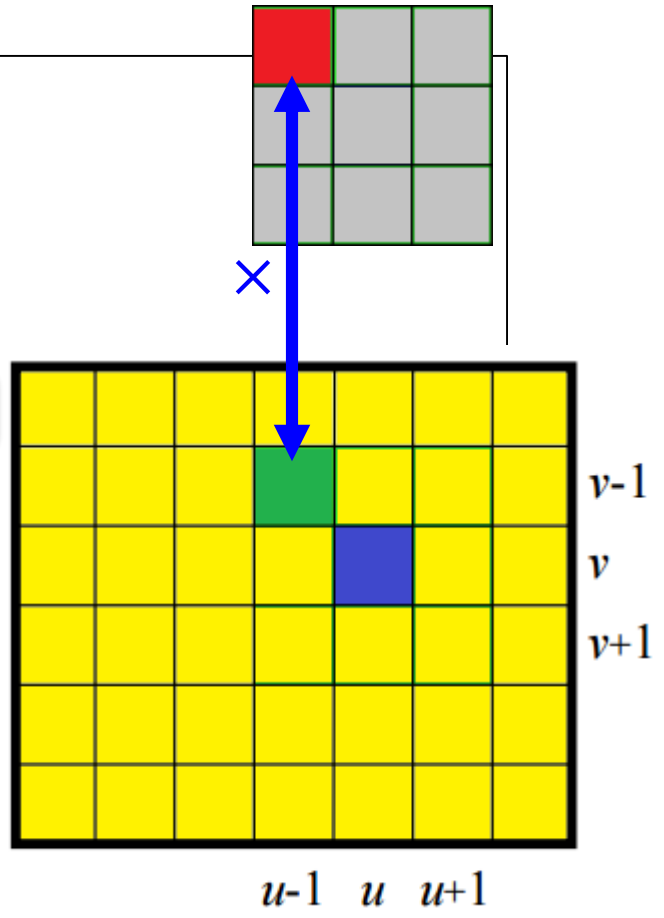
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = 0;
        for i = -1:1
            for j = -1:1
                value = value + ???
            end
        end
        I2(v, u) = value;

    end
end
```

Select neighborhood



Linear Filter

$$I'(u,v) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u+i, v+j) \cdot H(i,j)$$

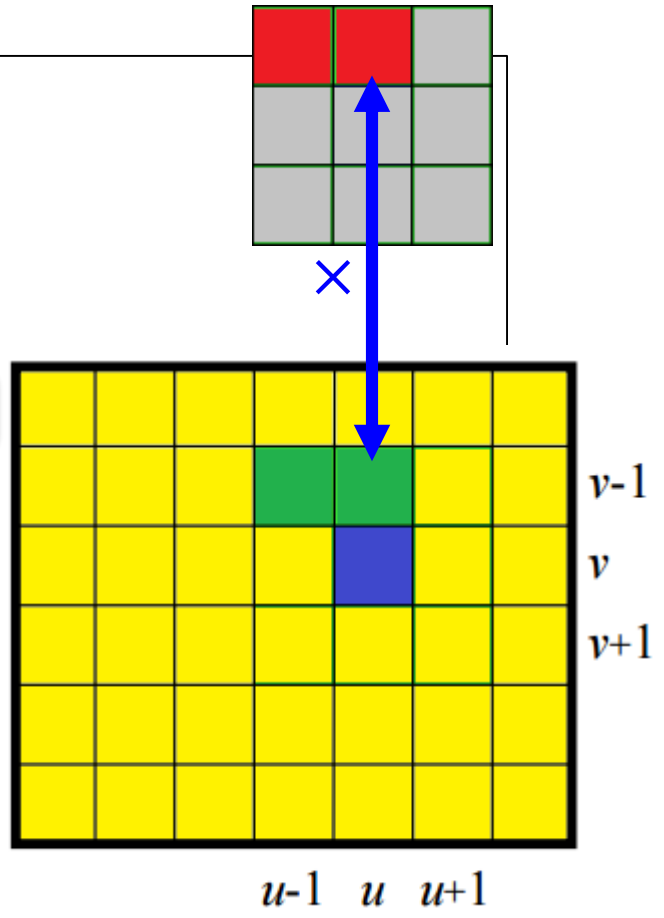
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = 0;
        for i = -1:1
            for j = -1:1
                value = value + ???
            end
        end
        I2(v, u) = value;

    end
end
```

Select neighborhood



Linear Filter

$$I'(u,v) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u+i, v+j) \cdot H(i,j)$$

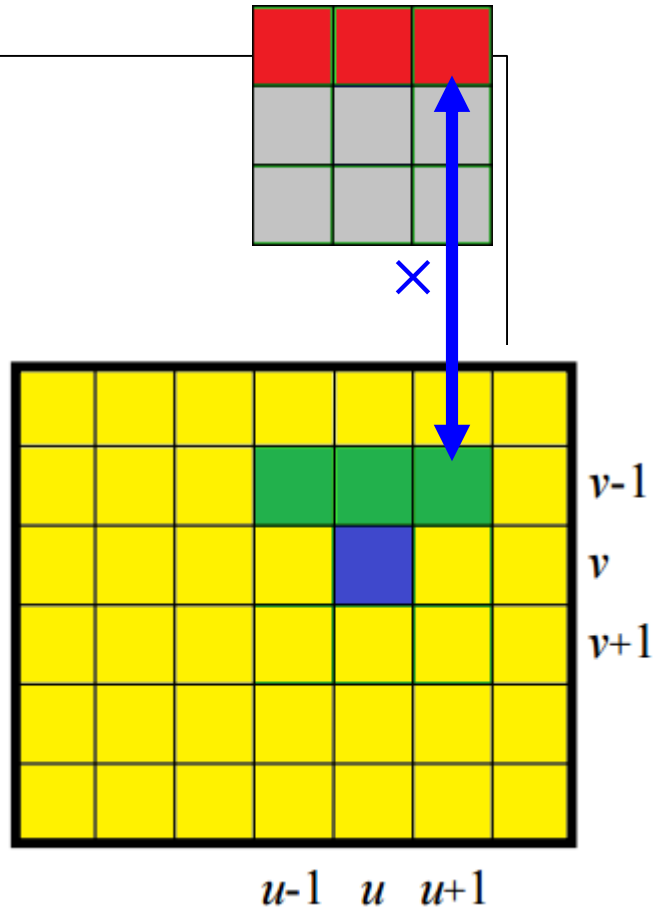
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = 0;
        for i = -1:1
            for j = -1:1
                value = value + ???
            end
        end
        I2(v, u) = value;

    end
end
```

Select neighborhood



Linear Filter

$$I'(u,v) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(u+i, v+j) \cdot H(i,j)$$

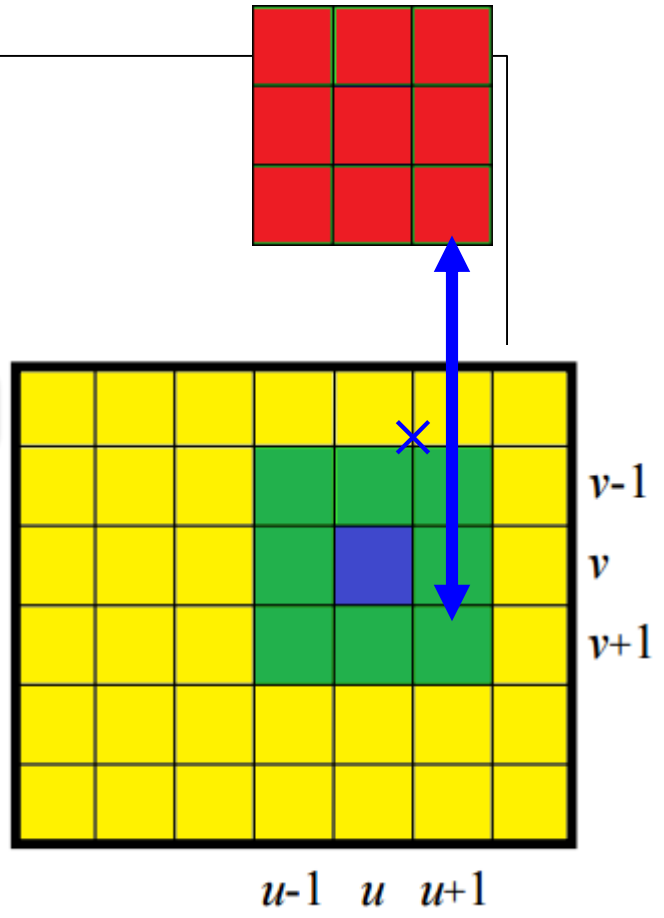
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = 0;
        for i = -1:1
            for j = -1:1
                value = value + ???
            end
        end
        I2(v, u) = value;

    end
end
```

Select neighborhood



Linear Filter

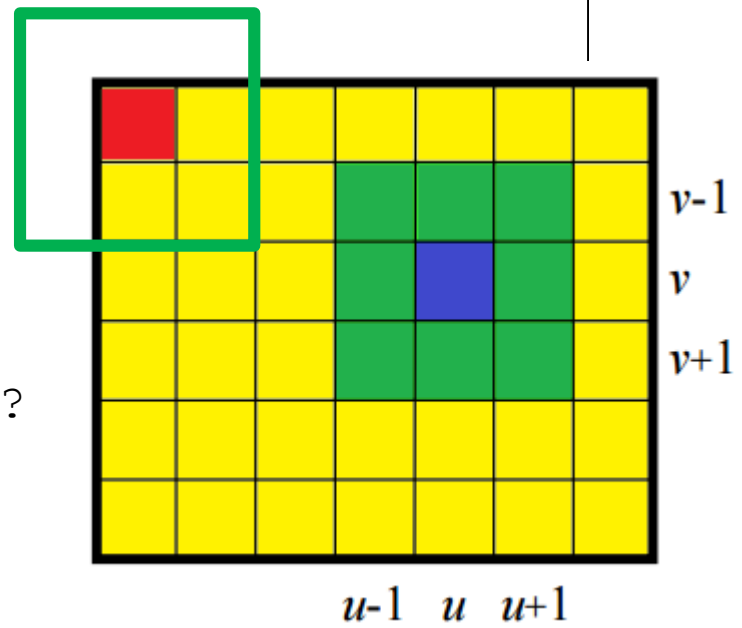
- Be careful around boundaries
  - ignore the pixels on boundaries
  - padding/extend boundaries [Optional]

```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        value = 0;
        for i = -1:1
            for j = -1:1
                value = value + ???
            end
        end
        I2(v, u) = value;

    end
end
```



Be careful about the index

Linear Filter

- Note: For Loop in MATLAB is very slow!
- Tips: use `tic` and `toc` to measure the elapsed time

```
%% Gaussian filter
hsize = 11;
sigma = 4;
tic
I = gaussian_filter_slow(img, hsize, sigma);
toc
tic
I = gaussian_filter_fast(img, hsize, sigma);
toc
```

```
Elapsed time is 3.522677 seconds.
Elapsed time is 1.608707 seconds.
>>
```

Linear Filter

- Extract patch/sub-matrix, and do matrix/vector operation.
- Again, be careful about the index around image boundaries

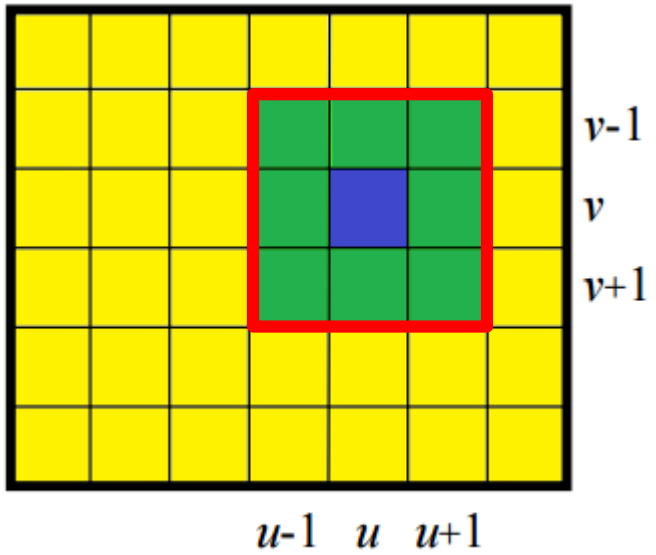
```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1 : size(I1, 2)
    for v = 1 : size(I1, 1)

        x1 = ???; x2 = ???;
        y1 = ???; y2 = ???;
        patch = I1(y1:y2, x1:x2);
        % convert matrix to vector
        % matrix/vector operations
        value = ???;
        I2(v, u) = value;

    end
end
```

Check index range



Linear Filter

- Skip boundary pixels
  - How many pixels to shift?

```
I1 = im2double(imread('lena.jpg'));
I2 = zeros(size(I1));

for u = 1+shift_u : size(I1, 2)-shift_u
    for v = 1+ shift_v : size(I1, 1)- shift_v

        x1 = ???; x2 = ???;
        y1 = ???; y2 = ???;
        patch = I1(y1:y2, x1:x2);
        % convert matrix to vector
        % matrix/vector operations
        value = ???;
        I2(v, u) = value;

    end
end
```

Shift depends on the filter size

Median Filter

- Extract patch with the same size as the filter
- Calculate the median value of the patch (Use `median()`)
- Fill in the median value to the output pixel

Noisy input image

Median filter output

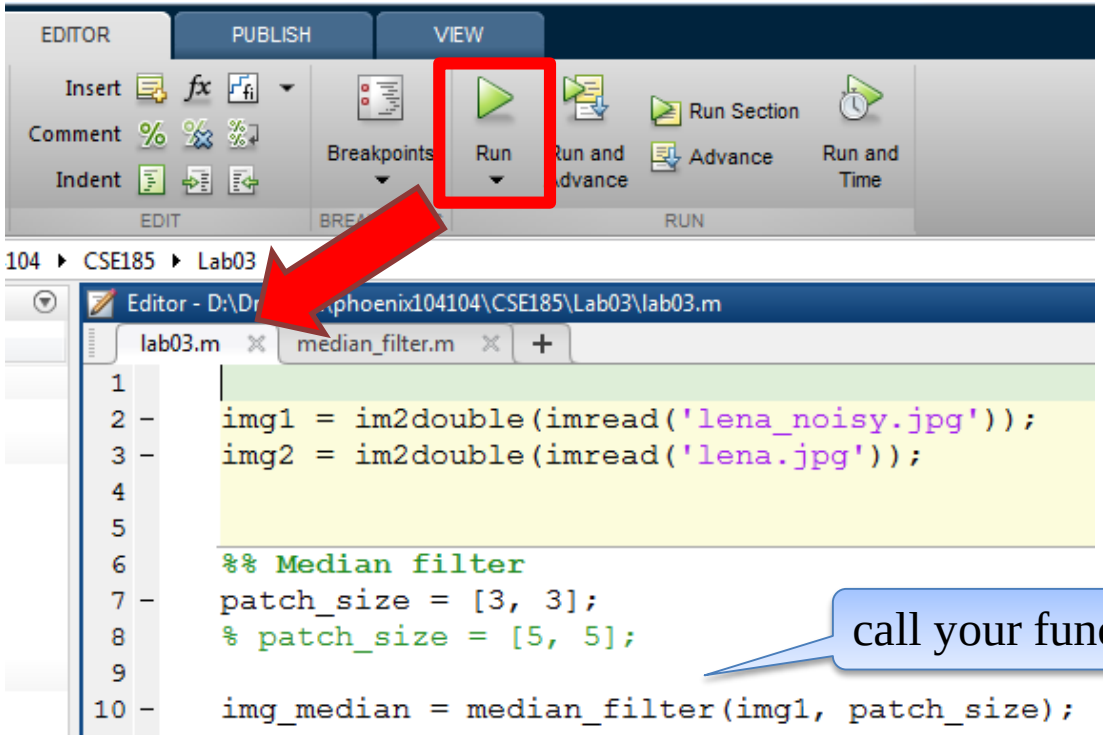
MATLAB Function

- Function template:

```
function output = median_filter(input, patch_size)
    output = input;
end
```

Assign output value, it will be automatically returned at the end of the function

- `lab03.m` is your main function, run this script instead of the function



call your function

Median Filter

- `median_filter.m`

```
function output = median_filter(img, patch_size)
    % YOUR CODE HERE
end
```

- In `lab03.m`:

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

%% Median filter
patch_size = [3, 3];
% patch_size = [5, 5];

img_median = median_filter(img, patch_size);
imwrite(img_median, 'median.jpg');
```

- Compare your result with built-in function:

```
I = medfilt2(img, patch_size);
```

Median Filter

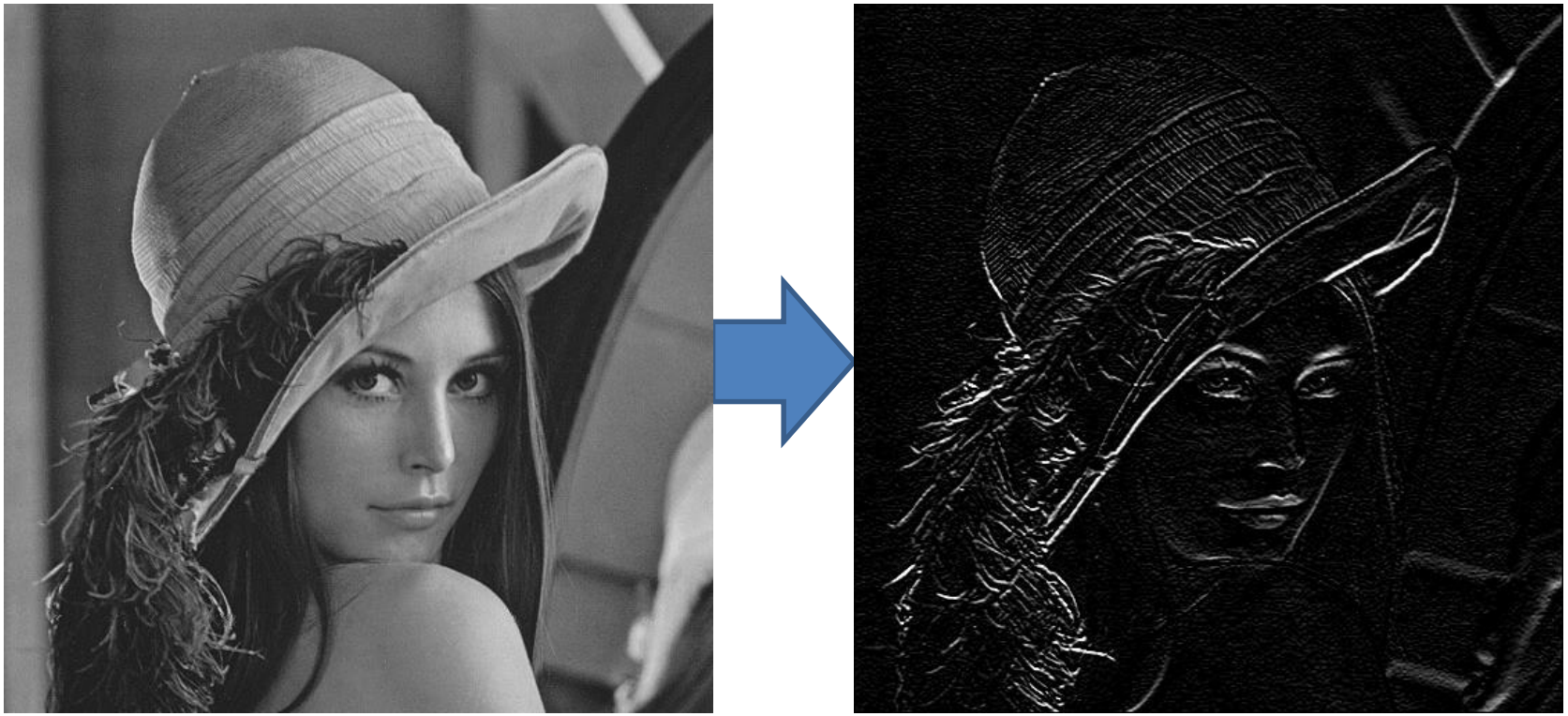
- MATLAB function `median()`:

```
>> A = [1, 2, 3;
        4, 5, 6;
        7, 8, 9];
>> median(A)
ans =
    4   5   6
```

- But we need a single value
  - convert matrix to vector first
  - or apply function twice
  - Question: are the results the same?

Sobel Filter

- Sobel filter is a simple edge detector
- $H = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$ or $H = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$



Input image

Sobel filter output

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);`

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

Δy

Δx

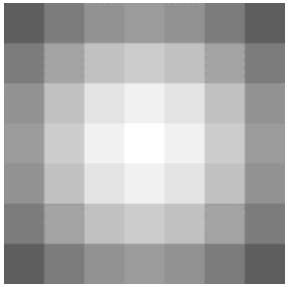
Gaussian Filter

- 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 = imfilter(img, H);`

TODO

1. Implement median_filter.m, use patch size = 3 and save the image as median_3.jpg (3pt)
2. Use patch size = 5, and save the image as median_5.jpg (3pt)
3. Implement sobel_filter.m, use horizontal filter and save the image as sobel_h.jpg (3pt)
4. Use vertical filter and save the image as sobel_v.jpg (3pt)
5. Implement gaussian_filter.m, use hsize = 5, sigma = 2, and save the image as gaussian_5.jpg (3pt)
6. Use hsize = 9, sigma = 4, and save the image as gaussian_9.jpg (3pt)
7. Upload your output images and lab03.m (2pt), sobel_filter.m, median_filter.m, and gaussian_filter.m

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