SCSV3213

FUNDAMENTAL OF IMAGE PROCESSING

IMAGE ENHANCEMENT IN SPATIAL DOMAIN
(Neighborhood Processing)

Dr. Md Sah Hj Salam

Acknowledgements

- Most of the slide are taken and modified from other resources including books and slides from lectures from others universities. Mainly from O. Marques -Practical Image and Video Processing Using MATLAB, Wiley-IEEE, 2011.It is rearranged to suit the syllabus of the course.

SYNOPSIS

In this lecture, image enhancement operations in spatial domain will cover the followings

1. Neighborhood Operation
   - Introduction of Neighborhood Processing
     - Convolution
     - Correlation
   - Linear Spatial Filter
   - Non-Linear Spatial Filters

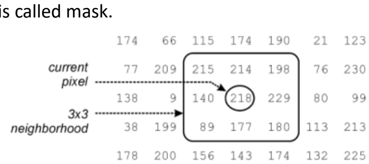
2. NEIGHBORHOOD PROCESSING

Neighborhood Processing : Intro

- Remember from previous lecture that point processing is when the operation applied on the pixels values regardless of the position !!!
- What is Neighborhood processing ? Can you guess from the named?
- After this lecture, you should
  - Understand the different and the usage of the operation.
  - Knows why some image problems need neighborhood operations to be applied while point processing cannot solved it.
  - Knows types of neighborhood processing (filter) and to which image problems the filters are suitable.

Neighborhood Processing : terminology

- Neighborhood
  - The pixels surrounding a given pixel. Most neighborhoods used in image processing algorithms are small square arrays with an odd number of pixels. This small square array is called mask.



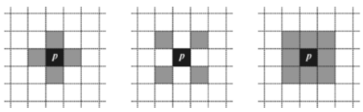
Neighborhood Processing : terminology

- Masks are normally 3×3.
- Each mask coefficient can be interpreted as a weight.

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

Neighborhood Processing : terminology

- Neighborhood
  - In the context of image topology, neighborhood has a different meaning:
    - 4-neighborhood
    - Diagonal neighborhood
    - 8-neighborhood



Neighborhood Processing

- Neighborhood-oriented processing consist of determining the resulting pixel value at coordinates (x,y) as a function of its original value and the value of (some of) its neighbors, using a *convolution* operation.
- The convolution of a source image with a small 2D array (mask or kernel) produces a destination image in which each pixel value depends on its original value and the value of (some of) its neighbors.
- The convolution mask determines which neighbors are used as well as the relative weight of their original values.

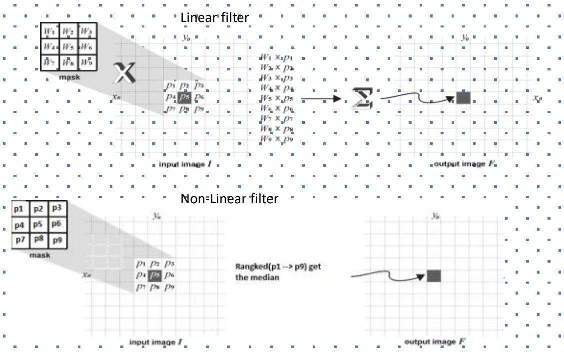
Neighborhood Processing

- Main steps:
  - Define a reference point in the input image, $f(x_0, y_0)$.
  - Perform an operation that involves only pixels within a neighborhood around the reference point in the input image.
  - Apply the result of that operation to the pixel of same coordinates in the output image, $g(x_0, y_0)$.
  - Repeat the process for every pixel in the input image.

Neighborhood Processing

- **Linear filters**: where the resulting output pixel is computed as a sum of products of the pixel values and mask coefficients in the pixel's neighborhood in the original image.
  - Example: mean filter
- **Nonlinear filters**: where the resulting output pixel is selected from an ordered (ranked) sequence of pixel values in the pixel's neighborhood in the original image.
  - Example: median filter

Neighborhood Processing



2.1 CONVOLUTION AND CORRELATION

Convolution and correlation

- Convolution and correlation are the two fundamental mathematical operations involved in linear neighborhood-oriented image processing algorithms.
  - The two operations differ in a very subtle way.
- Convolution is a widely used mathematical operator that processes an image by computing -- for each pixel -- a weighted sum of the values of that pixel and its neighbors.
  - Depending on the choice of weights a wide variety of image processing operations can be implemented.

Convolution and correlation

- 1D convolution

The convolution between two discrete one-dimensional arrays $A(x)$ and $B(x)$, denoted by $A * B$, is mathematically described by the equation:

$$A * B = \sum_{j=-\infty}^{\infty} A(j) \cdot B(x - j) \qquad (10.1)$$

Example convolution (1D)

- A = [0 1 2 3 2 1 0]
- B = [1 3 -1]
- A*B = [1 5 8 9 4 1 -1]

Let calculate together in class how the result is achieved !!

Convolution and correlation

- 2D convolution

$$g(x,y) = \sum_{k=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} h(j,k) \cdot f(x-j, y-k)$$

- 2D correlation

$$g(x,y) = \sum_{k=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} h(j,k) \cdot f(x-j, y-k)$$

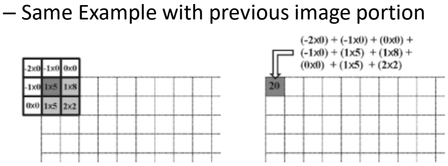
Example 2D convolution

Find the convolution of A*B given the value of A and B are shown below. B is the mask.

$$A = \begin{pmatrix} 5\,8\,3\,4\,6\,2\,3\,7 \\ 3\,2\,1\,1\,9\,5\,1\,0 \\ 0\,9\,5\,3\,0\,4\,8\,3 \\ 4\,2\,7\,2\,1\,9\,0\,6 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & -2 \end{pmatrix}$$

Convolution and correlation

- 2D convolution
  - Same Example with previous image portion



A * B = [ ]

Calculate the remaining result of A*B for the first line values..

Convolution and correlation

- Convolution with different masks
  - Convolution is a very versatile image processing method.
  - Depending on the choice of mask coefficients, entirely different results can be obtained.

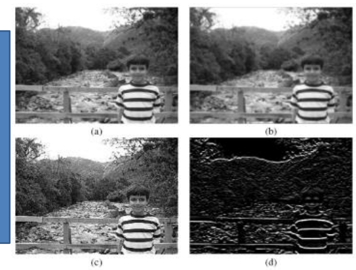
Low-pass filter	High-pass filter	Horizontal edge detection
$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Convolution and correlation

- Convolution with different masks

(a) - original
(b) - low pass filter
(c) - high pass filter
(d) - horizontal edge detection

Low pass = [1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9]
High pass = [0 -1 0
1 5 -1
0 -1 0]
HED = [1 1 1
0 0 0
-1 -1 -1]



Convolution and correlation

- Correlation
  - Simply put, correlation is the same as convolution *without* the mirroring (flipping) of the mask before the sums-of-products are computed.
  - The difference between using correlation and convolution in 2D neighborhood processing operations is often irrelevant because many popular masks used in image processing are symmetrical around the origin.

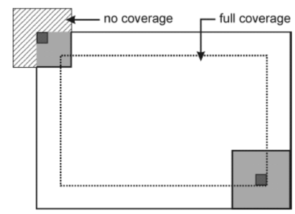
Example correlation (1D)

- A = [0 1 2 3 2 1 0]
- B = [1 3 -1]
- Calculate the correlation. This is the same example previously when we used convolution
- A*B = [1 5 8 9 4 1 -1]
- A ⌦ B = [?]

Convolution in MATLAB

- **conv2**: computes the 2D convolution between two matrices. In addition to the two matrices it takes a third parameter that specifies the size of the output.
- **filter2**: rotates the convolution mask (which is treated as a 2D FIR filter) 180° in each direction to create a convolution kernel and then calls **conv2** to perform the convolution operation.

Dealing with image borders



Dealing with image borders: options

1. Ignore the borders. There are two variants of this approach:
    - Keep the pixel values that cannot be reached by the overlapping mask untouched.
    - Replace the pixel values that cannot be reached by the overlapping mask with a constant fixed value, usually zero (black).
2. Pad the input image with zeros.
3. Pad with extended values.
4. Pad with mirrored values.
5. Treat the input image as a 2D periodic function whose values repeat themselves in both horizontal and vertical directions.

- In MATLAB: check the **boundary_options** parameter for function **imfilter**.

Image smoothing (Low-Pass Filters)

- Spatial filters whose effect on the output image is equivalent to attenuating high-frequency components (i.e., fine details in the image) and preserving low-frequency components (i.e., coarser details and homogeneous areas in the image).
- Linear LPFs can be implemented using 2D convolution masks with non-negative coefficients.
- Linear LPFs are typically used to either blur an image or reduce the amount of noise present in the image.

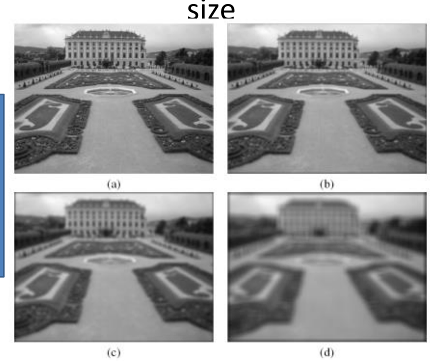
- In MATLAB: **imfilter** and **fspecial**

Mean (averaging) filter

- The simplest and most widely known spatial smoothing filter.
- It uses convolution with a mask whose coefficients have a value of 1, and divides the result by a scaling factor (the total number of elements in the mask).
- Also known as *box filter*.

Mean (averaging) filter: impact of mask size

(a) - input image (899x675)
(b) – masking 7x7
(c) – masking 15x15
(d) Masking 31x31



Mean (averaging) filter: variations

- Modified mask coefficients, e.g.:

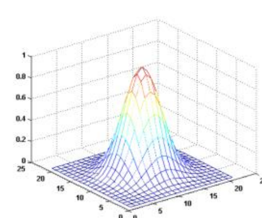
$$h(x,y) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & 0.2 & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

- Directional averaging
- Selective application of averaging calculation results
- Removal of outliers before calculating the average

Gaussian blur filter

- The best-known example of a LPF implemented with a non-uniform kernel.
- The mask coefficients for the Gaussian blur filter are samples from a 2D Gaussian function:

$$h(x,y) = \exp\left[\frac{-(x^2+y^2)}{2\sigma^2}\right]$$



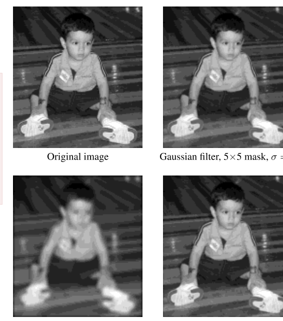
Gaussian blur filter

- Properties:
  - The kernel is symmetric w.r.t rotation, therefore there is no directional bias in the result.
  - The kernel is separable, which can lead to fast computational implementations.
  - The kernel's coefficients fall off to (almost) zero at the kernel's edges.
  - The Fourier Transform (FT) of a Gaussian filter is another Gaussian (this will be explained in Chapter 11).
  - The convolution of two Gaussians is another Gaussian.

Gaussian blur filter

- Example:

```
I = imread('Figure10_07_a.png');

h1 = fspecial('gaussian', [5 5], 1)
h2 = fspecial('gaussian', [13 13], 1);
h3 = fspecial('average', [13 13]);

J1 = imfilter(I, h1);
J2 = imfilter(I, h2);
J3 = imfilter(I, h3);
```



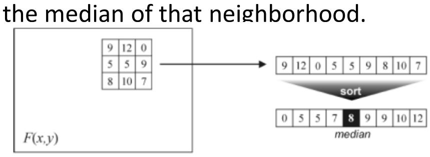
Original image | Gaussian filter, 5×5 mask, $\sigma = 1$ | Mean filter, 13×13 mask | Gaussian filter, 13×13 mask, $\sigma = 1$

Median and other nonlinear filters

- Nonlinear filters also work at a neighborhood level, but do not process the pixel values using the convolution operator.
  - Instead, they usually apply a ranking (sorting) function to the pixel values within the neighborhood and select a value from the sorted list.
  - Sometimes called *rank filters*.
  - Examples: median filter, max and min filters.

Median filter

- Works by sorting the pixel values within a neighborhood, finding the median value and replacing the original pixel value with the median of that neighborhood.



Median filter

- Example (salt-and-pepper noise reduction)

(a) Original Image
(b) with salt and pepper noise
(c) 3x3 median filter
(d) 3x3 neighborhood averaging

Image sharpening (High-Pass Filters)

- Spatial filters whose effect on the output image is equivalent to preserving or emphasizing its high-frequency components (e.g., fine details, points, lines, and edges), i.e. to highlight transitions in intensity within the image.
- Linear HPFs can be implemented using 2D convolution masks with positive and negative coefficients, which correspond to a digital approximation of the *Laplacian*, a simple, isotropic (i.e., rotation invariant) second-order derivative that is capable of responding to intensity transitions in any direction.

Image sharpening (HPF)

- The *Laplacian*

The Laplacian of an image $f(x, y)$ is defined as:

$$\nabla^2(x,y) = \frac{\partial^2(x,y)}{\partial x^2} + \frac{\partial^2(x,y)}{\partial y^2}$$

$$\nabla^2(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Image sharpening (HPF)

- Composite Laplacian mask

$$g(x,y) = f(x,y) + c\left[\nabla^2(x,y)\right]$$

- For *c*= $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

Image sharpening (HPF)

- Example:

```
I = imread('coat_of_arms_before.png');
h = fspecial('laplacian', 0);
I1 = im2double(I);
J = imfilter(I1,h);
K = I1-J;
h8 = [1 1 1; 1 -8 1; 1 1 1]
K8 = I1 - imfilter(I1,h8,'replicate');
Ja = J + 0.75;
```

Directional difference filters

- Similar to the Laplacian high-frequency filter.
  - Main difference: directional difference filters emphasize edges in a specific direction.
- Usually called *emboss filters*.
- Examples of masks that can be used to implement the emboss effect:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

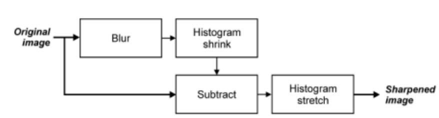
Unsharp masking

- Consists of computing the subtraction between the input image and a blurred (low-pass filtered) version of the input image.

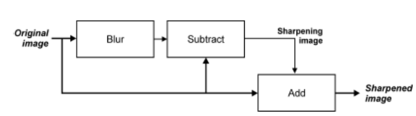
- Rationale: to "increase the amount of high-frequency (fine) detail by reducing the importance of its low-frequency contents".

Unsharp masking

- Variants (see Tutorial):
  - (1)

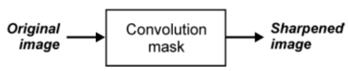


  - (2)



Unsharp masking

- Variants (see Tutorial):
  - (3)



High-boost filtering

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & c & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- where: *c* (*c* > 8) is a coefficient ("amplification factor") that controls how much weight is given to the original image and the high-pass filtered version of that image.
  - For *c*=8, the results would be equivalent to those seen earlier for the conventional isotropic Laplacian mask.
  - Greater values of *c* will cause significantly less sharpening.

ROI Processing

- Filtering operations are sometimes performed only in a small part of an image, known as a *region of interest* (ROI), which can be specified by defining a (usually binary) mask.
  - Image masking is the process of extracting such a subimage (or ROI) from a larger image for further processing.
- In MATLAB
  - A combination of two functions: **`roipoly`** (see Tutorial 6.2) and **`roifilt2`**

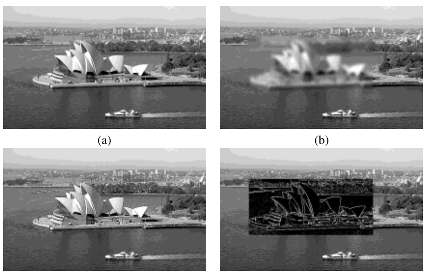
ROI Processing

- Example :

```
I = imread('Figure10_11_a.png');
r = [90 254 254 90];
c = [84 84 447 447];
BW = roipoly(I,c,r);
h = fspecial('gaussian', [15 15], 5);
J = roifilt2(h,I,BW);
h8 = [1 1 1; 1 -8 1; 1 1 1]
K = roifilt2(h8,I,BW);
h4 = [0 -1 0; -1 5 -1; 0 -1 0]
L = roifilt2(h4,I,BW);
```

ROI Processing

- Example :



(a) (b) (c) (d)

Combining spatial enhancement methods

- When faced with a practical image processing problem, a question arises:
  Which techniques should I use and in which sequence?
- There is no universal answer to this question.
  - Most image processing solutions are problem-specific and involve the application of several algorithms -- in a meaningful sequence -- to achieve the desired goal.
  - The choice of algorithms and fine-tuning of their parameters is a trial-and-error process.
  - Using the knowledge acquired so far you should be able to implement, configure, fine-tune, and combine image processing algorithms for a wide variety of real-world problems.

Matlab Functions

- You can make the mask / kernel yourself using matrix operation which we have learned or use the build in mask in Matlab, **fspecial()** and used **imfilter()** function to apply the mask.
- See next examples in the tutorial
- Understand how to use them ..

TUTORIAL ON FILTERING

TUTORIAL 1: Smoothing filter 1

- using mean/averaging filter using fpspecial() function

```
➢ I = imread('cameraman.tif');
➢ figure, subplot(1,2,1), imshow(I), title('Original Image');
➢ fn = fspecial('average')
➢ I_new = imfilter(I,fn);
➢ subplot(1,2,2), imshow(I_new), title('Filtered Image');
```

- Create and use non-uniform filter on the same image

```
➢ fn2 = [1 2 1; 2 4 2; 1 2 1]
➢ fn2 = fn2 * (1/16)
➢ I_new2 = imfilter(I,fn2);
➢ figure, subplot(1,2,1), imshow(I_new), title('Uniform Average');
➢ subplot(1,2,2), imshow(I_new2), title('Non-uniform Average');
```

TUTORIAL 2: Smoothing filter 2

- create and apply Gaussian filter

```
% create and show in bar graph
➢ fn_gau = fspecial('gaussian',9,1.5);
➢ figure, bar3(fn_gau,'b'), title('Gaussian filter as a 3D graph');

% apply on the cameran image
➢ I_new3 = imfilter(I,fn_gau);
➢ figure
➢ subplot(1,3,1), imshow(I), title('Original Image');
➢ subplot(1,3,2), imshow(I_new), title('Average Filter');
➢ subplot(1,3,3), imshow(I_new3), title('Gaussian Filter');
```

```
➢ clear all;
➢ close all;
```

TUTORIAL 3: Sharpening filter 1

- Create and use laplacian filter

```
% load the image moon
➢ I = imread('moon.tif');
➢ Id = im2double(I);
➢ figure, subplot(2,2,1), imshow(Id), title('Original Image');

% create laplacian filter
➢ f = fspecial('laplacian',0);
➢ I_filt = imfilter(Id,f);
➢ subplot(2,2,2), imshow(I_filt), title('Laplacian of Original');

% Display a scaled version of the Laplacian image for display purposes.
➢ subplot(2,2,3), imshow(I_filt,[]), title('Scaled Laplacian');

% Subtract the filtered image from the original image to create the
% sharpened image.
➢ I_sharp = imsubtract(Id,I_filt);
➢ subplot(2,2,4), imshow(I_sharp), title('Sharpened Image');
```

TUTORIAL 4: Sharpening filter 2

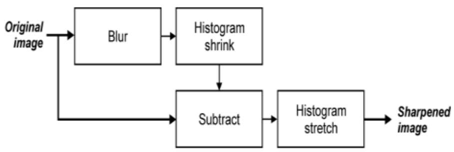
- Another way in using lapacian filter – composite mask

```
% This script assume the variable fro previous tutorial is active

➢ f2 = [0 -1 0; -1 5 -1; 0 -1 0]
➢ I_sharp2 = imfilter(Id,f2);
➢ figure, subplot(1,2,1), imshow(Id), title('Original Image');
➢ subplot(1,2,2), imshow(I_sharp2), title('Composite Laplacian');
```

```
➢ clear all;
➢ close all;
```

Sharpening filter 3

- Laplacian filter on blur image
- Process Flow



TUTORIAL 5: Sharpening filter 3

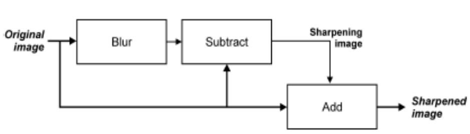
- Another way in using lapacian filter on blur image

```
➢ I = imread('moon.tif');
➢ f_blur = fspecial('average',13);
➢ I_blur = imfilter(I,f_blur);
➢ figure, subplot(1,3,1), imshow(I), title('Original Image');
➢ subplot(1,3,2), imshow(I_blur), title('Blurred Image');

% shrink the histogram of the blur image
➢ I_blur_adj = imadjust(I_blur,stretchlim(I_blur),[0 0.4]);

% Now subtract the blurred image from the original image
➢ I_sharp = imsubtract(I,I_blur_adj);

% Stretch the sharpened image histogram to the full dynamic grayscale
% range and display the final result.

➢ I_sharp_adj = imadjust(I_sharp);
➢ subplot(1,3,3), imshow(I_sharp_adj), title('Sharp Image');
```

TUTORIAL 6: Sharpening filter 4

- Another way in using lapacian filter on blur image

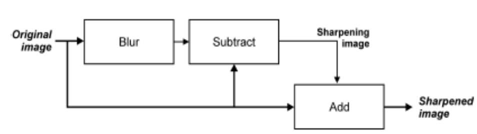
```
% Subtract the blurred image from original image to generate a sharpening  image.
➢ I_sharpening = imsubtract(I,I_blur);

% Add sharpening image to original image to produce final result.
➢ I_sharp2 = imadd(I,I_sharpening);
➢ figure, subplot(1,2,1), imshow(I), title('Original Image');
➢ subplot(1,2,2), imshow(I_sharp2), title('Sharp Image');
```



TUTORIAL 7: Sharpening filter 5

- Another way in using lapacian filter on blur image

```
% Subtract the blurred image from original image to generate a sharpening  image.
➢ I_sharpening = imsubtract(I,I_blur);

% Add sharpening image to original image to produce final result.
➢ I_sharp2 = imadd(I,I_sharpening);
➢ figure, subplot(1,2,1), imshow(I), title('Original Image');
➢ subplot(1,2,2), imshow(I_sharp2), title('Sharp Image');
```



TUTORIAL 8: Median Filter

- Sample median filter usage for noise restoration

```
➢ I = imread('eight.tif');
➢ J = imnoise(I,'salt & pepper',0.02);
➢ K = medfilt2(J);
➢ imshow(J), figure, imshow(K)
```

End Part 2:
Spatial Domain Enhancement
(Neighborhood Processing)
SCSV 3213