EEE 6512   Image Processing & Computer Vision

Assignment 04

Name: Chen Yang   Username: cyang3   UF id: 52109967

4.3

It is given that A = $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$,   and B ={ (1,1), (2,1), (1,2), (2,2), (3,2) ,(1,3), (3,3) }, after converting B to the set

representation we get B= $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$, therefore we get the result as follows:

$A \cup B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$, $A \cap B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $A_b = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$. (b=(1,1))

$B_{reflection} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $!A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$,

$A \backslash B = A \cap \ ! B$, $\because \ ! B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$,   so $A \backslash B = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

4.8

A→ isthmus connected the three fruits

B→ island

C→ channel btw the banana and the stem

D→ large bay of the apple

E→ small lake inside the apple

For the left image, opening, remove isthmus and islands to remove noise, opening is anti-extensive.

For the right image, closing, fill lake, bay and channel to remove noise, closing is extensive

4.12

	a	b
	c	d
e		

4-neighbors: include the four pixels to the left, right, top and bottom. So, N_4= a,d

8-neighbors: include the eight closest pixels. So N_8 =a, b, e, d

Diagonal neighbors: include four pixels to the four corners. So $N_{diagonal}$= b, e

4.15

For Euclidean distance:

d_E (p,q)= sqrt(2^2+2^2) = $2\sqrt{2}$

d_E (p,q)= sqrt(2^2+1^2) = $\sqrt{5}$

d_E (p,q)= sqrt(2^2+0^2) = 2

d_E (p,q)= sqrt(1^2+1^2) = $\sqrt{2}$

d_E (p,q)= sqrt1^2+0^2) = 1

The shape of Euclidean is ==concentric circle== $\rightarrow$

$2\sqrt{2}$	$\sqrt{5}$	2	$\sqrt{5}$	$2\sqrt{2}$
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
2	1	0	1	2
$\sqrt{5}$	$\sqrt{2}$	1	$\sqrt{2}$	$\sqrt{5}$
$2\sqrt{2}$	$\sqrt{5}$	2	$\sqrt{5}$	$2\sqrt{2}$

For Manhattan distance:

d_4 = |2-0| + |2-0| = 4

d_4 = |2-1| + |2-0| = 3

d_4 = |2-2| + |2-0| = 2

d_4 = |2-1| + |2-1| = 2

d_4 = |2-1| + |2-2| = 1

The shape of Manhattan is ==concentric diamond== $\rightarrow$

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

For Chessboard distance:

d_8 = max(2,2) = 2

d_8 = max(1,2) = 2

d_8 = max(2,0) = 2

d_8 = max(1,1) = 1

d_8 = max(1,0) = 1

The shape of chessboard is ==squares or boxex== $\rightarrow$

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

4.20

a) prove $A \ominus (B \oplus c) = (A \ominus B) \ominus C$

from the equation 4.28 $\neg (A \oplus B) = \neg A \ominus B$ [1]

4.29 $\neg (A \ominus B) = \neg A \oplus B$ [1]

we get $A \ominus (B \oplus C) = \neg(\neg A \oplus (B \oplus C))$

$= \neg((\neg A \oplus B) \oplus C)$

$= \neg(\neg(A \ominus B) \oplus C)$

$= (A \ominus B) \ominus C$ proved !

b) try $A \check{\oplus} (B \oplus C) = (A \check{\ominus} B) \check{\ominus} C$

from the equation 4.39 $A \check{\ominus} B = A \ominus \check{B}$ [1]

we get $A \check{\ominus} (B \oplus C) = \neg(\neg A \oplus (\check{B} \oplus \check{C}))$

$= \neg((\neg A \oplus \check{B}) \oplus \check{C})$

$= \neg(\neg(A \check{\ominus} B) \oplus \check{C})$

$= (A \check{\ominus} B) \check{\ominus} C$ proved !

c) try $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$

$A \oplus (B \cup C) = \bigcup_{a \in A} (B_a \cup C_a) = (\bigcup_{a \in A} B_a) \cup (\bigcup_{a \in A} C_a)$

from the equation 4.52 $A \oplus B = \bigcup_{b \in B} A_b$ [1]

we get $A \oplus (B \cup C) = (\bigcup_{a \in A} B_a) \cup (\bigcup_{a \in A} C_a)$

$= (A \oplus B) \cup (A \oplus C)$ proved !

5.7

Since the format of smoothing kernels is $\Sigma_i g(i) = 1$, the format of differentiating kernels is $\Sigma_i g(i) = 0$. Therefore,

a) 1/32 [2 4 6 8 6 4 2] is a smoothing kernel
b) 1/6 [1 2 3 2 1 0 -1 -2 -3 -2 -1] is a differentiating kernel
c) 1/9 [9 1 -1 -9] is a differentiating kernel
d) 1/11 [1 9 1] is a smoothing kernel

5.23

For figure1 $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$

Prewitt: $\frac{\partial I}{\partial x} = I \otimes$ Prewitt x $= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \otimes \frac{1}{6} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{6} * \begin{bmatrix} 2 & -2 & -2 \\ 3 & -3 & -3 \\ 2 & -2 & -2 \end{bmatrix}$

$\frac{\partial I}{\partial y} = I \otimes$ Prewitt y $= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \otimes \frac{1}{6} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \frac{1}{6} * \begin{bmatrix} 2 & 2 & 1 \\ 0 & 0 & 0 \\ -2 & -2 & -1 \end{bmatrix}$

Sobel: $\frac{\partial I}{\partial x} = I \otimes$ Sobel x $= I(x,y) \otimes \frac{1}{8} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{8} * \begin{bmatrix} 3 & -3 & -3 \\ 4 & -4 & -4 \\ 3 & -3 & -3 \end{bmatrix}$

$\frac{\partial I}{\partial y} = I \otimes$ Sobel y= $I(x,y) \otimes \frac{1}{8} * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \frac{1}{8} * \begin{bmatrix} 3 & 3 & 1 \\ 0 & 0 & 0 \\ -3 & -3 & -1 \end{bmatrix}$

Scharr: $\frac{\partial I}{\partial x} = I \otimes$ Scharr x $= I(x,y) \otimes \frac{1}{32} * \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} = \frac{1}{32} \begin{bmatrix} 13 & -13 & 13 \\ 16 & -16 & -16 \\ 13 & -13 & -13 \end{bmatrix}$

$\frac{\partial I}{\partial y} = I \otimes$ Scharr y $= I(x,y) \otimes \frac{1}{32} * \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} = \frac{1}{32} * \begin{bmatrix} 13 & 13 & 3 \\ 0 & 0 & 0 \\ -13 & -13 & -3 \end{bmatrix}$

Therefore, we can get as follows:

	Euclidean	Manhattan	Chessboard
Prewitt	0.5	0.5	0.5
Sobel	0.5	0.5	0.5
Scharr	0.5	0.5	0.5

For figure2 $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$,

Prewitt: $\frac{\partial I}{\partial x} = I(x,y) \otimes$ Prewitt x $= \frac{1}{6} * \begin{bmatrix} 1 & -2 & -1 \\ 2 & -2 & -2 \\ 2 & -1 & -2 \end{bmatrix}$

$\frac{\partial I}{\partial y} = I(x,y) \otimes$ Prewitt y $= \frac{1}{6} * \begin{bmatrix} 2 & 2 & 1 \\ 1 & 2 & 2 \\ -2 & -2 & -1 \end{bmatrix}$

Sobel: $\frac{\partial I}{\partial x} = I(x,y) \otimes$ sobel x $= \frac{1}{8} * \begin{bmatrix} 3 & 3 & 1 \\ 1 & 3 & 3 \\ -3 & -3 & -1 \end{bmatrix}$

$$\frac{\partial I}{\partial y} = I(x,y) \otimes \text{Sobel y} = \frac{1}{8} * \begin{bmatrix} 1 & -3 & -1 \\ 3 & -3 & -3 \\ 3 & -1 & -3 \end{bmatrix}$$

Scharr: $\frac{\partial I}{\partial x} = I(x,y) \otimes \text{sobel x} = \frac{1}{32} * \begin{bmatrix} 3 & -13 & -3 \\ 13 & -13 & -13 \\ 13 & -3 & -13 \end{bmatrix}$

$$\frac{\partial I}{\partial y} = I(x,y) \otimes \text{Sobel y} = \frac{1}{32} * \begin{bmatrix} 13 & 13 & 3 \\ 3 & 13 & 13 \\ -13 & -13 & -3 \end{bmatrix}$$

Therefore, we can get as follows:

	Euclidean	Manhattan	Chessboard
Prewitt	0.471	0.667	0.333
Sobel	0.530	0.750	0.375
Scharr	0.575	0.813	0.406

According to the chart we obtained above, we can conclude that the best behavior happens when we apply Prewitt and Sobel using Euclidean metric, it generates the best result because they have the least error.

6.4
Since the Nyquist Rate is two times of the highest sampling frequency of a signal, which is the minimum frequency for a signal to sample without any undersampling. The value of Nyquist Frequency is 1/2 of the sampling frequency. If the Nyquist freq is greater than the highest freq to be sampled, the signal will get sampling without meeting any aliasing.

6.19
Since DFT of a function is always periodically which is extending horizontally and vertically to infinity. For an image, there is always a clear edge along the boundaries. Those can show the edges as great values in two dimensional DFT along the horizontal and vertical axises.

6.20
The two dimensional Discrete Fourier Transform will also rotated for 30 degrees clockwise.

Code Section for Assignment 04

Method Filter	Run time: k=3	Run time: k=203	Run time: k=403
mySpatialFilt k*k Gaussian	22.105157	945.710269	20235.509597
mySpatialFilt separable Gaussian	19.572042+20.782999	27.994485+30.291735	34.532623+39.513228
myFrequencyFilt k*k Gaussian	1.927097	1.993643	1.954818
myFrequencyFilt separable Gaussian	1.829395+1.770631	1.784427+1.751544	1.856030+1.726479
Matlab toolbox k*k Median	0.146837	8.648087	17.152545

Table. Timing Information of Each Function with Different Kernels (sec)

Note: All the time information obtained in the above table is the time consumed when the function is called, rather than timing within the function.

Note: When applying separable kernels, I choose to call function twice and time each call of function, the result is the summation of the two calls.

For example:


```
img = imread('escher.png');
filter = fspecial('gaussian',[403 403],(403/6))
tic
conv = mySpatialFilt(img,filter);
toc
```

```
%% Separable
img = imread('escher.png');
sep1 = fspecial('gaussian',[11 1],(11/6))
sep2 = fspecial('gaussian',[1 11],(11/6))
tic
conv1 = mySpatialFilt(img,sep1);
toc
figure
tic
conv2 = mySpatialFilt(conv1,sep2);
toc
```

Figure. Example of Timing Method（left:k*k, right:separable timing）

5 Images Required:

<mark>mySpatialFilt, Gaussian 11x11 kernel</mark>

```
img = imread('escher.png');
filter = fspecial('gaussian',[11 11],(11/6))
tic
conv = mySpatialFilt(img,filter);
toc
```

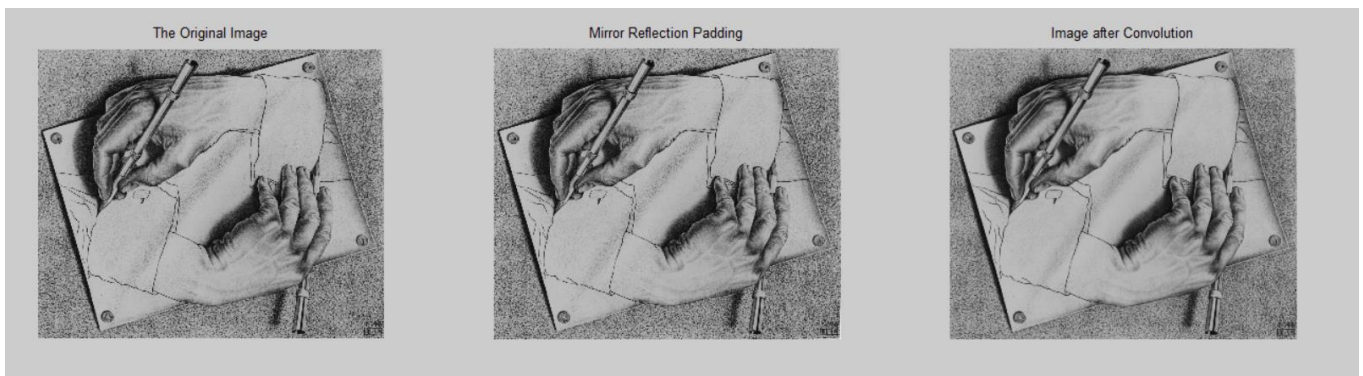**Figure. Method to Generate Result for Gaussian 11x11 kernel**

Figure. Result for Gaussian 11x11 kernel

```matlab
%% Separable
img = imread('escher.png');
sep1 = fspecial('gaussian', [11 1], (11/6))
sep2 = fspecial('gaussian', [1 11], (11/6))
tic
conv1 = mySpatialFilt(img, sep1);
toc
figure
tic
conv2 = mySpatialFilt(conv1, sep2);
toc
```

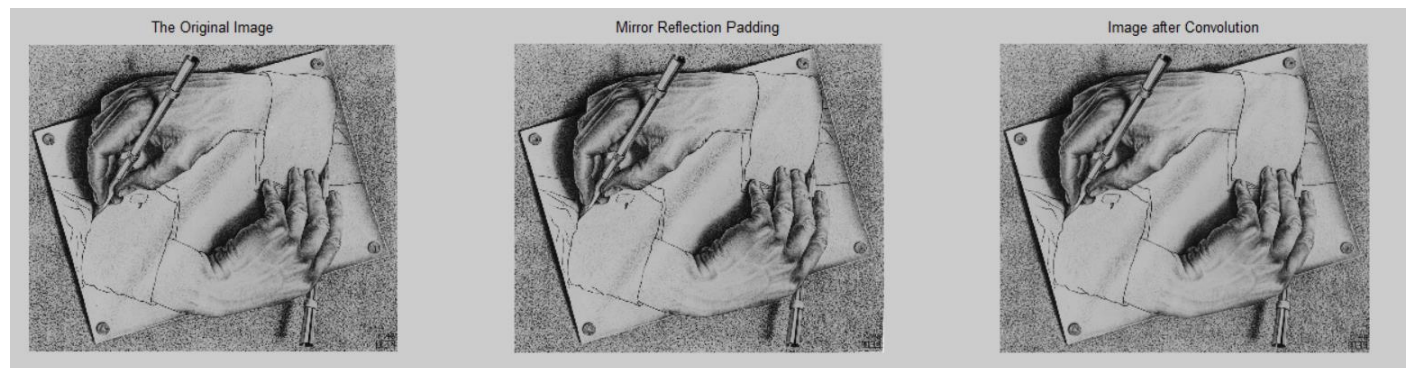**Figure. Method to Generate Result for Gaussian Separable kernels**

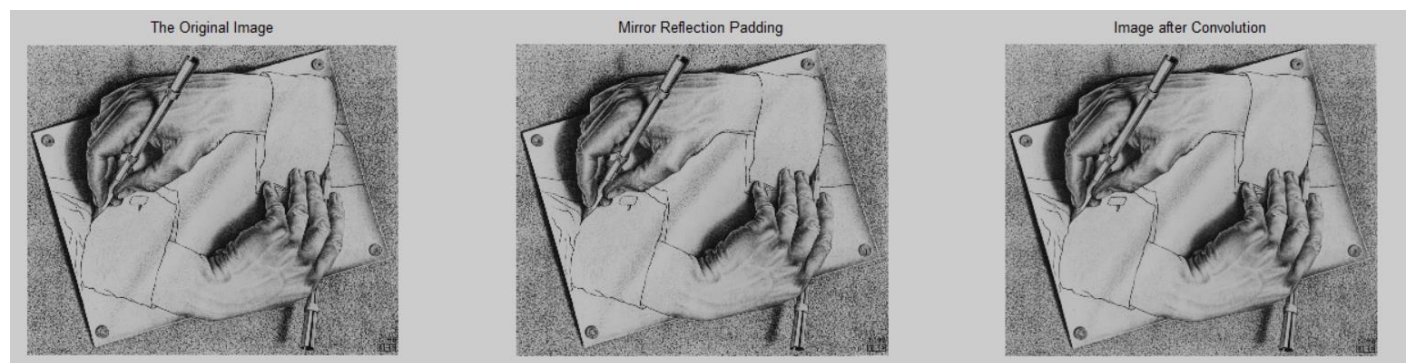**Figure. Result for conv1 (Intermediate Process)**

Figure. Result for conv2 (Final Result)

```matlab
%% Gaussian
img = imread('escher.png');
filter = fspecial('gaussian', [11 11], (11/6))
tic
conv = myFrequencyFilt(img, filter);
toc
```

Figure. Method to Generate Result for Gaussian 11*11 kernel
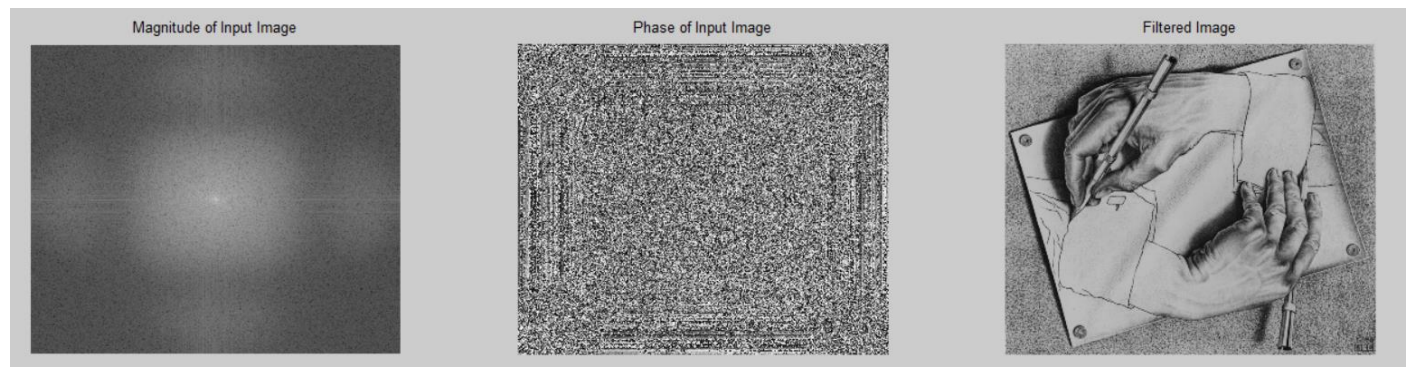

Figure. Result for Gaussian 11x11 kernel

myFrequencyFilt, Gaussian in separable form (1x11, 11x1) ⇓ include the DFT of the image before filtering as well

```matlab
%% Separable
img = imread('escher.png');
sep1 = fspecial('gaussian',[11 1],(11/6))
sep2 = fspecial('gaussian',[1 11],(11/6))
tic
conv1 = myFrequencyFilt(img,sep1);
toc
figure
tic
conv2 = myFrequencyFilt(conv1,sep2);
toc
```

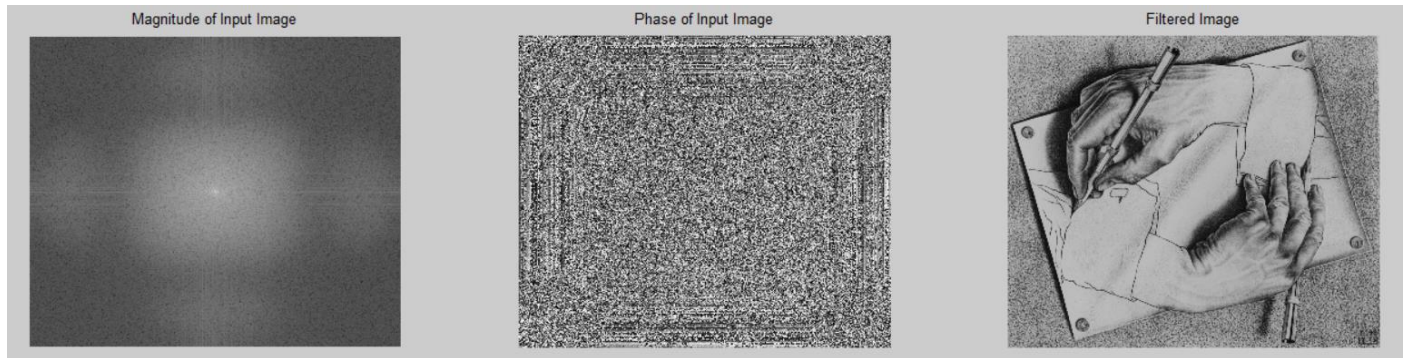**Figure. Method to Generate Result for Gaussian Separable kernels**
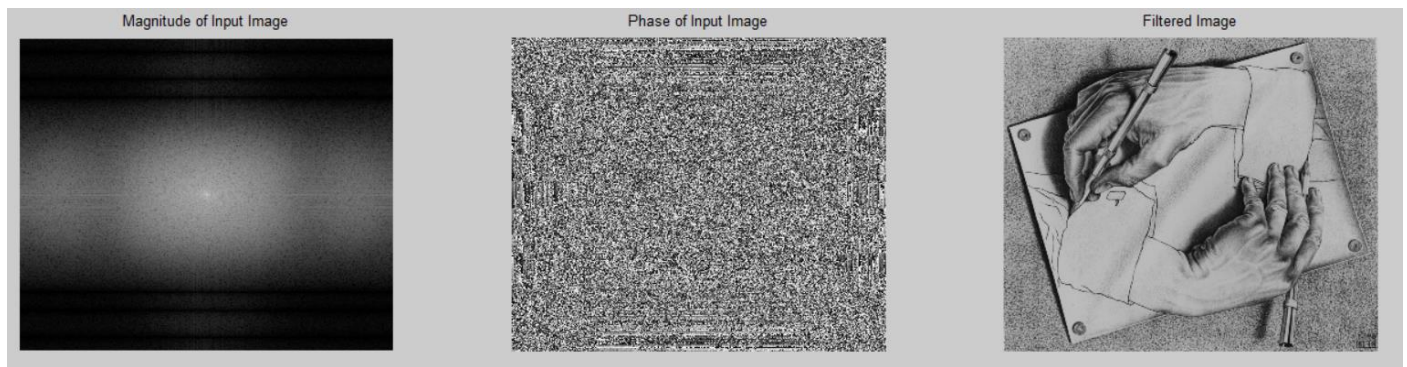

Figure. Result for conv1 (Intermediate Process)

Figure. Result for conv2 (Final Result)

MATLAB median filter, median 11x11 kernel

```matlab
%% Median
img = imread('escher.png');
tic
Y3=medfilt2(img,[11 11]);
figure,imshow(Y3),title('median result with matlab toolbox');
toc
```

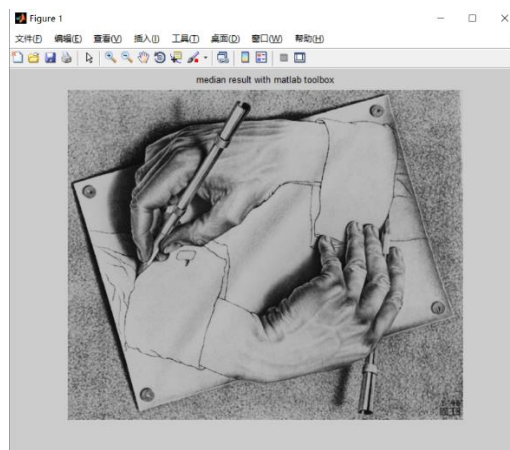**Figure. Method to Generate Result for Gaussian 11*11 kernel**

Figure. Result for Gaussian 11x11 kernel

4 Questions and Solutions:

What is the difference between gaussian versus median filtering, with respect to the output filtered image?
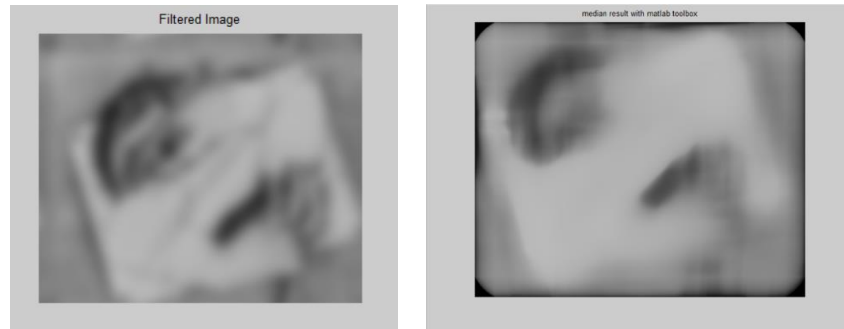

Figure. Gaussian filter median filter comparison

When the filter size k takes a relatively small value, the difference between the pictures of the two processing methods is <u>too small to be distinguished by naked eyes.</u> In order to present a better contrast effect, we choose a relatively large value of k. The above two pictures from left to right are the pictures processed by the Gaussian filter and the pictures processed by the median filter. At this time, the selected filter size is k = 403. ==The Gaussian filter seems to perform better,== just equivalent to overlaying a blur mask on the picture. Although the median filter can also blur the picture, it is not as good as the Gaussian filter in terms of edge processing and preservation of picture details. The reason is that Gaussian filter is better to deal with salty noise.

What is the difference between filtering in the spatial domain versus in the frequency domain, with respect to time?

==Filtering in Frequency domain is a lot more faster than filtering in Spatial domain.== Assume the size of filter is k*k. When filtering in the spatial domain, k square times multiplications and 2*k summations are performed every time the filter stops. The number of stops is equal to the pixels numbers of the picture. When filtering in the frequency domain, the computational complexity is greatly reduced. Excluding the time to calculate fft and ifft, only one multiplication calculation is required (assuming that we have perfectly padding the filter).

What is the difference between filtering with a kxk filter versus the separable form of the same filter, with respect to time?

==For the Spatial Domain:==

==When k is a relatively small number, k*k filtering is faster than the separable method.== This is because the calculation time within the function is particularly small, but the more the function is called, the more time is taken up. However, as k goes greater, the separable method is more efficient than the k*k filter. This is because both methods need to scan the entire image size. The computational complexity of k*k filtering is k*k, while the complexity of the separable method is k*2. ==Therefore, regardless of the time to call the function, when k is large, the calculation time of the separable method is less.==

==For the Frequency Domain:==

In this filtering method, Because the computational complexity is relatively low, the time to call the function is the main part, and the calculation time in the function can be ignored, so no matter how much k is taken, in myFrequencyFilt, the k*k filter is faster than the separable filter

When filtering with a Gaussian kernel, what is the effect on the output when the sigma value is changed, but the filter size remains the same? How about when sigma value remains the same, but the filter size is changed?

If the value of sigma becomes greater, the covariance of kernel will also become greater. So the difference of each unit kernel is larger than before. In this situation, if the size of kernel is still the same, the whole difference of kernel will become larger. So the influence of filter on the image will become weaker.

If the value of sigma stays the same, and the filter size becomes larger. Correspondingly, the weight of filter center become less, which means the convoluted value will get more affected by the surrounding pixel values. If the convolution value is more affected by the adjacent pixel value, the blur effect of the filter will be stronger. Besides, the corner will not be clear anymore.

Code Explanation for mySpatialFilt.m

In the following several paragraphs, I will briefly introduce the basic ideas and ideas of *mySpatialFilt* , this code file will be divided into several parts to explain in detail.

```matlab
function conv_img = mySpatialFilt(img, filter)
%% Load and Initialization
img = double(img);
[m,n] = size(img);% Read the size of image
[t,r] = size(filter); % Read the size of filter
filt_size = max(t,r);
d = floor(filt_size/2);       % Compute the thickness that we need to pad to each boundaries
padd_img = zeros(m+2*d,n+2*d);  % Initialize a new image for padding preparation
conv_img = zeros(m,n);        % Initialize a new image for convolution preparation
```

Figure. Code mySpatialFilt Part1

Part 1: Load and Initialization

In this part, the function will initialize some variables related to the input, including image size, filter size, the width of the edge of the image that needs to be mirrored, the size of the new image after mirroring and the convoluted image size.

```matlab
%% Pad the image
% We divided the section that need to be padded into 5 part
% 1. Pad the original image to the centre part of new padded image
padd_img(d+1:m+d, d+1:n+d) = img;
% 2. Pad the top boundary to the new location of paddded image
top = img(1:d,:);
padd_img(1:d, d+1:n+d)= top(end:-1:1,:) ;
% 3. Pad the bottom part
bot = img(m-d+1:m,:);
padd_img(m+d+1:m+2*d, d+1:n+d)=bot(end:-1:1,:) ;
% 4. Pad the image obtained above, reallocate its left part to the
% left boundary
left = padd_img(:, d+1:2*d);
padd_img(:,1:d)=left(:,end:-1:1);
% 5. Pad the right part to the right boundary.
right = padd_img(:, n+1:d+n);
padd_img(:,n+d+1:2*d+n)=right(:,end:-1:1);
```

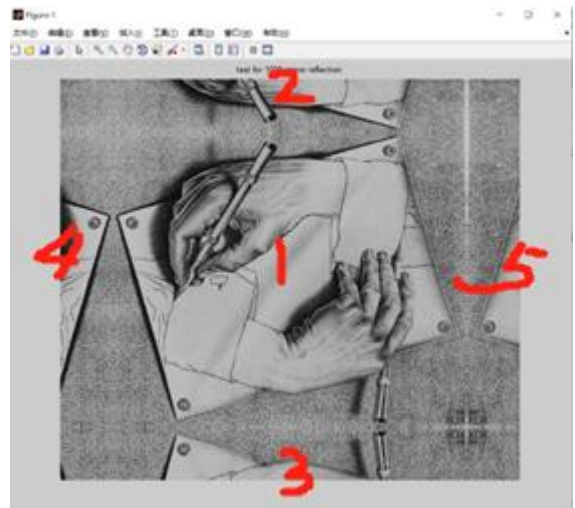**Figure. Code mySpatialFilt Part2**

Figure. Test for Padding Image

Part 2: Padding Process of the Image

The second part is the specific process of image mirroring pad. In this process, the method I use is not the built-in function of padd_array, but in the form of matrix operation. The image mirror reflection padding is divided into 5 parts according to the linear relationship of the filter size and pad thickness. First, we place the original input image in the center of the previously initialized padd_img. Second, we use the original image to flip the upper edge of the thickness up to mirror flip. Third, use the same method as the second step to mirror and flip the bottom edge.

In the second and third steps I used [(end:-1:1,:)] to perform row vector reverse order operation on a specific matrix. The fourth step is to use the new image obtained by the above process to mirror pad its left edge according to the same linear relationship. The fifth step is the same as the fourth step, pad the right edge.

In the fourth and fifth steps we use [(:,end:-1:1)] to implement the column vector reverse operation of a specific matrix. At this point, the entire mirroring pad process is over, and the result perfectly meets the requirements in the question. The above figure on the right is the test result of padd_img. In this test, the edge size of the mirror flip is 500.

```
%% Compute Convolution
for i = 1:m
    for j = 1:n
        conv_img(i,j) = sum(sum(filter.*(padd_img(i:i+t-1, j:j+r-1))));
    end
end
%% Plot the Result
subplot(1,3,1),imshow(uint8(img)),title('The Original Image');
subplot(1,3,2),imshow(uint8(padd_img)),title('Mirror Reflection Padding');
subplot(1,3,3),imshow(uint8(conv_img)),title('Image after Convolution');
end
```

Figure. Code mySpatialFilt Part3

Part 3 : Computation of Convolution in Space Domain and Result Output

The calculations in this part are very intuitive. The filter will sweep pixel by pixel on the picture where the mirror flip is over, and each stop will multiply the pixel value of the area where the filter is located and the weight of the corresponding pixel of the filter itself. Sum all the multiplication results, and assign this sum to the corresponding position of the newly obtained convolution image. This part will traverse the entire picture and contains two for loops. This calculation of spatial convolution will be very time-consuming. Finally, in order to better verify the results, I chose to output three images in this section. Original image, padded_img and convolved image.

Code Explanation for myFrequencyFilt.m

In the following several paragraphs, I will briefly introduce the basic ideas and ideas of *myFrequencyFilt* . This code will be shown as a whole and analyzed step by step.

```matlab
function conv_img = myFrequencyFilt(img,filter)
    % Read and Load the Image
    img=double(img);
    [m,n] = size(img);
    % Compute DFT of image and filter
    fft_img=fft2(img);
    fft_filter = fft2(filter,m,n);
    % Compute convolution using multiplication in Frequency domain
    conv_img = ifft2(fft_img.*fft_filter);
    % Compute Phase of the Image
    phase=(1+angle(fft_img)/pi)*255;
    % Using the algorithm in the homework1 to shift the DFT image
    mid = floor(size(fft_img)/2);
    fft_img = fft_img([mid(1)+1:end,1:mid(1)],[mid(2)+1:end, 1:mid(2)]);
    % Computing the Image Magnitude with the Obtained shifted Imaage above
    magnitude=log(1+abs(fft_img));
    % Plot the result
    subplot(1,3,1),imshow(magnitude,[]),title('Magnitude of Input Image');
    subplot(1,3,2),imshow(phase,[]),title('Phase of Input Image');
    subplot(1,3,3),imshow(uint8(conv_img)),title('Filtered Image');
end
```

Figure. Code myFrequency

Like the previous code, first we need to read the basic information of the input image. Then we calculate the Fourier transform of the input image and the Fourier transform of the input filter respectively. Note that in this step, while calculating the Fourier transform of the filter, we expanded its size to be consistent with the image size. Next, by multiplying the Fourier form of the image obtained in the frequency domain and the Fourier form of the filter, we can obtain the Fourier form of the convolved image. After inversely transforming it, a new convolved image is obtained. Because the multiplication operation in the frequency domain is equivalent to the convolution operation in the space domain. However, the difference is that because multiplication has a much lower computational complexity compared to convolution, the time consumed in this step is much less than the previous convolution in the spatial domain.

In addition, we also need to calculate the DFT form of the original image, which includes magnitude and phase. By definition, we can easily calculate the phase of the image through a fixed angle calculation. Next we use a horizontal line and a vertical line to divide the spectrogram into four blocks, and exchange the diagonal and anti-diagonal lines for these four blocks. In this process, we used the same method as in Homework 1 to exchange the four quadrants in the spectrogram. The purpose is to symmetrical the obtained spectrogram with 0 frequency as the center. After the new spectrogram is obtained, the magnitude image after center symmetry is obtained according to the magnitude calculation formula. Finally, we draw all the required images, magnitude images, phase images and convolutional images.

Reference

[1]   B. H. Brown, R. H. Smallwood, D. C. Barber, P. V Lawford, and D. R. Hose, *Image processing and analysis*. 2004.