

project-1

1. (Image denoising) Please implement the followings and analyze the results.

1) Generate the following noises, and add them to the ground-truth images.
a) impulse noise b) gaussian noise c) uniform noise

2) Implement the following denoising filters, and applied them for denoising.

a) gaussian filter b) bileteral filter c) median filter

3) Please compare and analyze the results with subjective and objective (PSNR) measurements.

a. impulse noise

-code

```

6      %impulse noise 생성
7 -    p= 0.1; %0~1사이의 확률값
8 -    cnt=int32(round(p*height*width)); %noise 가 생성될 픽셀 수
9
10 -    s=zeros(2,cnt);
11 -    s=randi([1,width],2,cnt); %noise의 위치를 s에 저장
12
13      %s에 저장된 위치에 noise 생성
14 -    for x= 1:cnt/2
15 -        out(s(2,x),s(1,x)) = 0;
16 -    end
17
18 -    for y=cnt/2+1:cnt
19 -        out(s(2,y),s(1,y))= 255;
20 -    end

```

1.randi함수를 이용해 salt&pepper noise를 생성할 이미지의 위치를 정한다.

이미지의 크기가 512x512이므로 randi 함수의 범위를 1부터 512로 설정한다.

2.noise가 생성될 픽셀의 개수는 salt , pepper 각각 확률 pxheightxwidth이다. 그러므로 배열의 random 변수를 저장할 배열의 크기는 cntx2이다.

3.for문을 이용해서 해당 위치에 salt & pepper 0과 255 값을 저장해준다.

- 결과 image



확률값 p 가 클수록 image에 noise가 많아진 것을 볼 수 있다.

b) gaussian noise

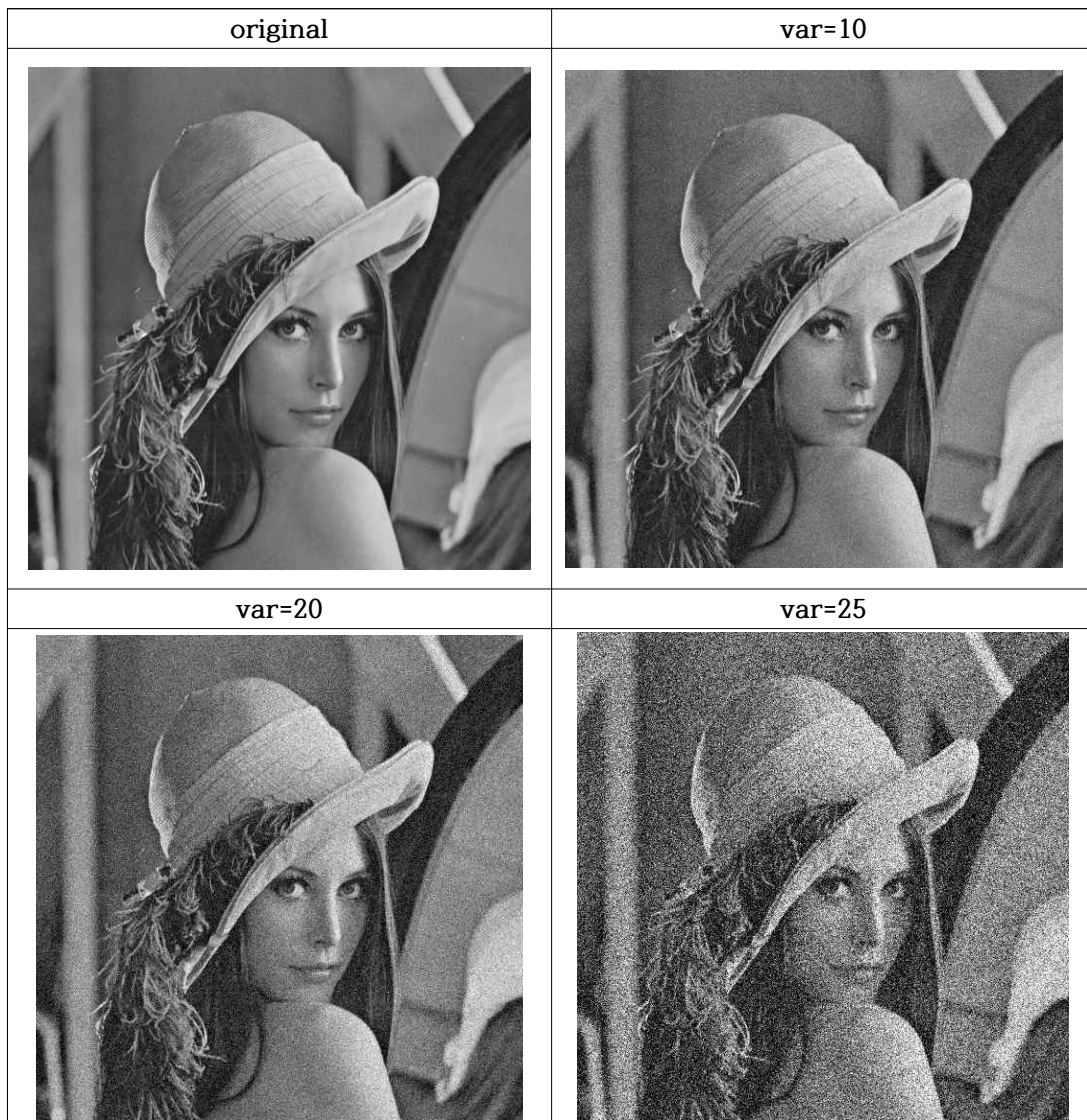
- code

```
6      %gaussian noise 생성
7      v=50; %표준편차
8      g=v*randn(512);
9      out=double(out);
10     out=out + g;
11
```

1. randn 함수를 이용해서 평균이 0이고 표준편차가 v인 정규분포를 갖는 512*512 행렬을 생성한다.

2. 입력된 image에 위의 행렬 g를 더해주어 gaussian noise를 추가한다.

-결과 image



표준편차가 클수록 noise가 심해졌다.

c) uniform noise

- code

```
5 %uniform noise 생성
6 - v=50; %표준편차
7 - g=v*rand(512);
8 - out= double(in)+g;
9
```

1. rand 함수를 이용해서 0~1 사이의 균일한 난수를 갖는 512*512 행렬을 생성한다.
2. 입력된 image에 위의 행렬 g를 더해주어 gaussian noise를 추가한다.

-결과 mage



표준편차가 클수록 noise가 심해졌다.
impulse noise는 image의 특정 위치에서 noise가 있는 반면에 gaussian noise, impulse noise는 image에 전체적으로 나타난다.

2) denoising

a) gaussian filter

noise가 영상의 일부분이 아닌 전체적으로 더해졌을 때 noise만 제거할 수 없으므로 중심값에 가중치를 둔 가우시안 분포를 갖는 필터를 사용한다.

-code

1. 영상 테두리를 0으로 padding해준다.

```
13 %zero padding
14 out= zeros(height,width);
15 in_p= zeros(height+f_size-1,width+f_size-1);
16 in_p(f_cnt+1:height+f_cnt,f_cnt+1:width+f_cnt)=in;
```

2. gaussian filter

```
18 %filter 값 설정
19 for b=1:f_size
20     for a=1:f_size
21         h(b,a)=(1/(2*pi*sigma*sigma))*exp(-((a-f_cnt-1).^2+(b-f_cnt-1).^2)/(2*(sigma.^2)));
22     end
23 end
24 end
25
26 h= h/sum(h(:)); %normalize
```





gaussian 값을 갖는 filter를 만들고 h에 저장한다. h 원소의 합이 1이 되도록 normalize해준다.

3. filter 적용

```
28 %filter 적용
29 for y= f_cnt+1:height+f_cnt
30     for x=f_cnt+1:width+f_cnt
31         temp=in_p(y-f_cnt:y+f_cnt,x-f_cnt:x+f_cnt).*h;
32         out(y-f_cnt,x-f_cnt)= sum(temp(:));
33         if out(y-f_cnt,x-f_cnt)>255;
34             out(y-f_cnt,x-f_cnt)=255;
35         end
36     end
37 end
```

이 filter를 image에 원소끼리 곱하고 temp에 저장한다. temp를 1차원 행렬로 바꿔주고 전체 합을 구하고 이 값을 image에 저장한다. 모든 픽셀에 대하여 반복한다

- 결과 image

impulse noise/p=0.2	var=2.0	var=3.0
		
gaussian noise/var=25	var=2.0	var=3.0
		
impulse noise/var=25	var=2.0	var=3.0
		

gaussian noise와 uniform noise 영상이 impulse noise영상보다 개선되었다.
 impulse noise영상의 경우 개선이 거의없고 blur만 심해졌다.
 variance값이 커질수록 noise는 줄었지만 blur가 심해졌다.

b)bileteral filter

gaussian filter는 픽셀의 위치만을 가중치로 두어 연산했다. 배경과 물체가 붙어있는 경우에도 이 필터를 적용하게 된다면 오차가 매우 커질 것이다.
 즉, 엣지를 최대한 보존하기 위해 bileteral filter는 픽셀의 위치뿐만 아니라 픽셀값의 차이를 가중치에 포함시켜 연산한다.

$$\hat{I}(x) = \frac{1}{K} \sum_{y=x-N}^{x+N} e^{-\frac{(I(y)-I(x))^2}{2\sigma_d^2}} e^{-\frac{\|y-x\|^2}{2\sigma_s^2}} I(y)$$

- code

```

21 %filter 값 설정
22 for b=1:f_size
23     for a=1:f_size
24         weights_s(b,a)=(1/(2*pi*(sigma_s.^2)))*exp(-((a-f_cnt-1).^2+(b-f_cnt-1).^2)/(2*(sigma_s.^2)));
25     end
26 end
27 weights_s= weights_s/ sum(weights_s(:)); % normalize
28
29
30 %filter 적용
31 for y= f_cnt+1:height+f_cnt
32     for x=f_cnt+1:width+f_cnt
33         temp=in_p(y-f_cnt:y+f_cnt,x-f_cnt:x+f_cnt);
34
35         %range information
36         weights_r=exp(-((temp-(double(in(y-f_cnt,x-f_cnt)).*ones(f_size))).^2)/(2*sigma_r*sigma_r));
37         weights= weights_s.*weights_r+0.0001;
38         weights_f=weights/sum(weights(:));
39         result=double(in(y-f_cnt,x-f_cnt)).*weights_f;
40         out(y-f_cnt,x-f_cnt)= sum(result(:));
41     end
42 end

```

1. spatioal gaussian 계수를 구한다.
2. range gaussian 계수를 구한다.
3. 2개의 행렬을 원소끼리 각각 곱하고 전체합으로 나누어주어 1로 normalize해준다.
4. filter와 입력영상을 컨볼루션한다.

- 결과 image

impulse noise/p=0.2	var=0.2	var=0.4
gaussian noise/var=25	var=0.2	var=0.4

gaussian필터를 적용했을 때보다 엣지가 분명하게 보인다.

c) median filter

- code

```
4      %median filter
5 -    f_size=3;    %filter의 크기 f_sizexf_size
6 -    f_cnt=(f_size-1)/2;
7      %zero padding
8 -    out= zeros(height,width);
9 -    in_p= zeros(height+f_size-1,width+f_size-1);
10 -    in_p(f_cnt+1:height+f_cnt,f_cnt+1:width+f_cnt)=in;
11
12     %filter 적용
13 -    for y= f_cnt+1:height+f_cnt
14 -        for x=f_cnt+1:width+f_cnt
15 -            temp=in_p(y-f_cnt:y+f_cnt,x-f_cnt:x+f_cnt);
16 -            temp_sort=sort(temp(:)); %2D matrix를 1D matrix로 변환후 sorting
17 -            out(y-f_cnt,x-f_cnt)=temp_sort(round((f_size^2)/2)); %sorting된 값중 중간값을 적용
18 -        end
19 -    end
```

1. image를 zero padding한다.
2. filter size에 해당하는 image 값들을 temp 저장한다.
3. temp를 1D matrix로 변환 후 sorting하여 temp_sort에 저장한다.
4. sorting된 값 중 중간값을 image에 저장한다.

- 결과 image

impulse noise/p=0.3	filter size = 3x3	filter size = 9x9
gaussian noise/var=25	filter size = 3x3	filter size = 9x9

impulse noise가 많이 개선되었다. filter의 size가 클수록 blur가 심해졌다. impulse noise의 경우 noise가 있는 픽셀의 값이 0또는 255이기 때문에 주변 픽셀 값들중 중간값을 찾아서 저장하면 주변과 비슷한 값을 갖기 때문에 전체적인 화질은 개선된다. 하지만 원래의 픽셀값은 모르고 주변의 픽셀값으로 채우기 때문에 blur가 생긴다. 또한, sorting에서 복잡도가 증가한다.

3) subjective/ objective(PSNR) measurements

영상들을 각각 MOS 방식으로 점수를 매기고, 매트랩으로 PSNR을 계산하여 두 평가방법을 비교해보았다.








원본영상과 비교영상의 픽셀을 각각 비교하여 차이를 제공하여 더한 후 영상의 크기로 나누어준다.

- code

```
1- in=imread('lena_grey.bmp');
2- original=imread('lena_m_f_6.bmp');
3- [height,width]=size(in);
4- out= zeros(height,width);
5- |
6- h=double(zeros(height,width)); %filter
7-
8- %MSE
9- h= original-in;
10- h=h.*h;
11- MSE=sum(h(:))/(height*width);
12-
13- %PSNR
14- psnr= 10*log(255*255/MSE);
```








1. 원본영상을 original에 입력받고 비교할 영상을 in에 저장한다.
2. 원본영상과 비교영상의 차이를 구하고 그 값을 제공하여 모두 더하고 영상크기로 나눈값을 MSE에 저장한다.
3. PSNR 식에 MSE를 대입하여 저장한다.

- gaussian noise image에 적용

gaussian noise/ v=25	gaussian filter/ sigma=2.0	gaussian filter/ sigma=3.0
		
PSNR= 66.30/ MOS=2	PSNR=76.91 / MOS=2	PSNR= 75.41 /MOS=2
	bilateral filter/ sigma=0.2	bilateral filter/ sigma=0.4
		
	PSNR=79.20 /MOS=3	PSNR=78.50 /MOS=3
	median filter/ 3X3	median filter/ 9X9
		
	PSNR=72.26 /MOS=2	PSNR=76.18 /MOS=1

PSNR을 비교해보면 전체적으로 원래의 noise영상인 66.30보다 높은 PSNR을 갖는다. bilateral filter/ sigma=0.2에서 PSNR이 가장 높고, median filter / 3X3에서 가장 낮다. 실제 눈으로 봐서 평가해보면 bilateral filter가 적용된 영상의 화질이 가장 좋아보이고, median filter 9x9가 가장 나빠보인다.

- impulse noise image에 적용

impulse noise/ p=0.3	gaussian filter/ sigma=2.0	gaussian filter/ sigma=3.0
		
PSNR= 75.12/ MOS=1	PSNR=64.45/ MOS=2	PSNR= 64.59 /MOS=1
	bilateral filter/ sigma=0.2	bilateral filter/ sigma=0.4
		
	PSNR=64.47/ MOS=2	PSNR=64.56/MOS=1
	median filter/ 3X3	median filter/ 9X9
		
	PSNR=83.58 /MOS=3	PSNR=79.18/ MOS=4

PSNR을 비교해보면 gaussian filter와 bilateral filter가 적용된 영상은 PSNR이 더 낮아졌다. median filter를 적용한 영상은 psnr이 매우 높아졌다. median filter/3x3영상에서 가장 높고, gaussian/var=2에서 가장 낮다. 실제 눈으로 봐서 평가해보면 bilateral filter가 적용된 영상의 화질이 가장 나빠보이고, median filter 3x3이 가장 좋아보인다.

실제 눈으로 봤을 때 화질을 비교해보고, 그 결과와 PSNR 결과가 일치하는지 비교해보았다. 일치하는 부분도 있고, 일치하지 않는 부분도 있다. PSNR의 경우 각 픽셀값의 차이를 모두 더하여 평균낸 것이기 때문에 일부분에서는 픽셀값의 차이가 많이 나고 또 다른 부분에서 차이가 거의 없는 영상과, 전체적으로 차이가 적은 영상의 PSNR값은 비슷해질 수 있다. 사람이 직접 점수를 매겨 판단하는 MOS방식의 경우는 사람마다 관점이 다르므로 많은 사람이 필요하고 또한 회차마다 점수가 달라질 수도 있다.

2. (Edge detection) Please implement the followings and analyze the results.

1) Implement 1st order edge detector. (e.g., Sobel, Prewitt)

2) Implement 2nd order edge detector. (e.g., LoG)

3) Please compare and analyze the results.

1) sobel/ prewitt edge detector

image에서 명암의 변화량을 측정하여 변화량이 큰 곳을 엣지로 검출한다. threshold이상의 값을 1로, 이하의 값을 0으로 치환하여 이진영상으로 나타낸다.

1.1)prewitt

-1	-1	-1
0	0	0
1	1	1

m_y

-1	0	1
-1	0	1
-1	0	1

m_x

1.2)sobel

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

m_y

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

m_x

-code

1. 엣지 연산자

```
8      %prewitt
9      Fx=[-1 0 1; -1 0 1; -1 0 1];
10     Fy=[-1 -1 -1; 0 0 0; 1 1 1];
```

```

4      %sobel
5      Fx=[-1 0 1;-2 0 2; -1 0 1];
6      Fy=[-1 -2 -1;0 0 0; 1 2 1];

```

2. 마스크된 x, y값으로 엣지 강도를 구하고 이를 out에 저장한다.

```

15  for y=1:size(in,1)-2
16
17      for x=1:size(in,2)-2
18
19          %Gradient operations
20
21          Gx=sum(sum(Fx.*in(y:y+2,x:x+2)));
22          Gy=sum(sum(Fy.*in(y:y+2,x:x+2)));
23
24          %Magnitude of vector
25          out(y+1,x+1)=sqrt(Gx.^2+Gy.^2);
26      end
27  end
28

```

3. 엣지강도가 저장된 out의 픽셀값을 검사하여 threshold보다 크면 1을 저장하고 작으면 0을 저장한다.

```

35  Threshold=100;
36  for y=1:512
37      for x=1:512
38
39          if out(y,x)< Threshold;
40              tmp(y,x)=0;
41          else tmp(y,x)=1;
42          end
43      end
44  end
45

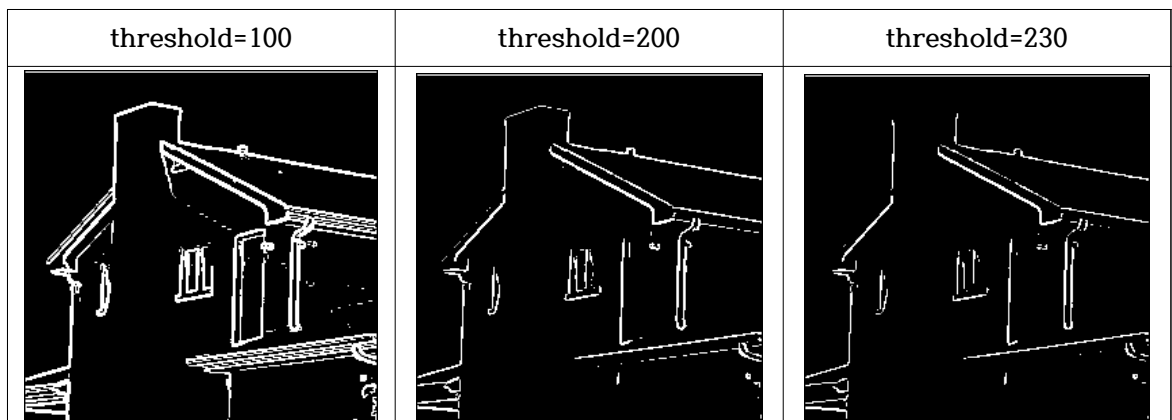
```

-결과 image

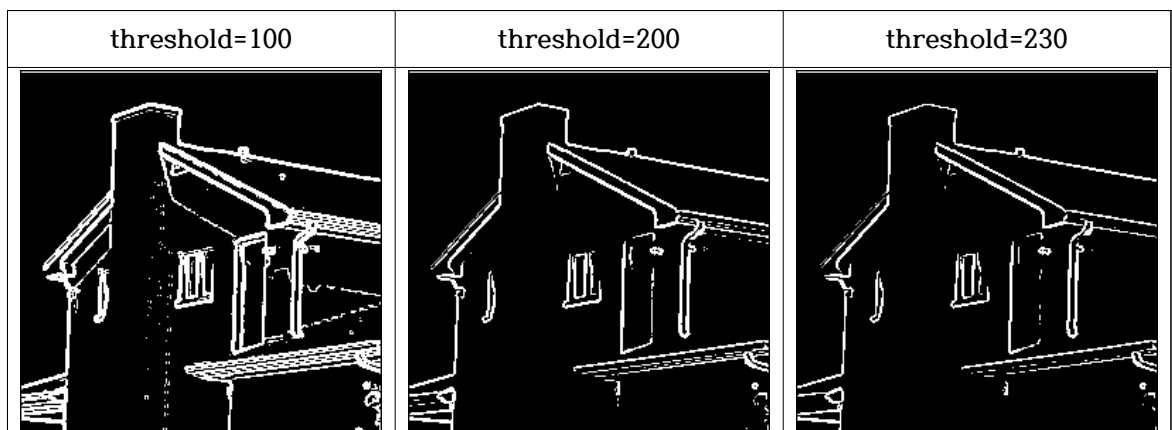


original image

- prewitt



- sobel



같은 threshold의 영상끼리 비교했을 때 sobel 영상이 엣지가 더 잘 검출됐다. prewitt의 경우 집의 굴뚝부분, 오른쪽 창문의 윗부분의 엣지가 잘 검출되지 않았다. 이는 배경의 색과 굴뚝의 색이 비슷하여 threshold가 230으로 높아졌을 때 잘 검출되지 않은것같다. 또, 오른쪽 창문의 윗부분도 빛으로 인해서 외벽과 비슷하다. 하지만 sobel의 경우 중심영상에 가중치를 2로 줬기 때문에 변화율이 더 커져서 prewitt보다는 엣지가 잘 검출되었다.

sobel의 경우 가중치가 2로 더 크기 때문에 변화량이 더 커지게 된다. 결과 이미지를 보면 threshold가 같은 경우 각각 비교해 보면 sobel이 더 많은 엣지가 검출된 것을 볼 수 있다.

2)LoG filter

LoG 필터의 경우 1차 미분과 달리 엣지의 최댓값을 찾을 수 있다. 하지만 미분은 잡음을 증폭시키므로 가우시안 필터를 이용해 영상을 스무딩시킨다.

또, variance를 조절하여 엣지의 세밀한 정도를 조절할 수 있다.

먼저 가우시안 필터를 이용해 영상을 스무딩한다. 결과 영상에 라플라시안 연산자를 적용하여 2차 미분을 구한다. 이 후 영교차를 찾아 엣지를 설정하고, 나머지는 비엣지로 설정한다. 마주보는 방향의 픽셀값을 각각 조사한다. 이 중 두 개 이상이 부호가 다르고, 그 부호가 다른 쌍의 값의 차이가 기준 T를 넘으면 엣지로 설정한다.

-code

1. gaussian filter를 적용한다.

```
17 %filter 값 설정
18 for b=1:f_size
19     for a=1:f_size
20         h(b,a)=(1/(2*pi*(sigma.^2)))*exp(-((a-f_cnt-1).^2+(b-f_cnt-1).^2)/(2*(sigma.^2)));
21     end
22 end
23 h= h/sum(h(:)); %normalize
24
25 %gaussian_filter 적용
26 for y= f_cnt+1:height+f_cnt
27     for x=f_cnt+1:width+f_cnt
28         temp=in_p(y-f_cnt:y+f_cnt,x-f_cnt:x+f_cnt).*h;
29         out(y-f_cnt,x-f_cnt)= sum(temp(:));
30     end
31 end
32
```


2. 결과영상에 라플라시안 연산자를 적용하여 2차미분을 구한다.

```
39 %laplacian 연산자
40 L=[0 1 0; 1 -4 1; 0 1 0];
41
42 %laplacian 적용
43 for y=1:size(in,1)-2
44     for x=1:size(in,2)-2
45         out(y+1,x+1)=sum(sum(L.*in(y:y+2,x:x+2)));
46     end
47 end
```

3. 영교차를 찾아 엣지로 설정한다.

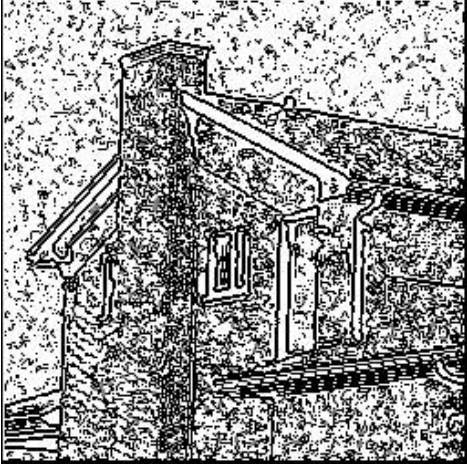
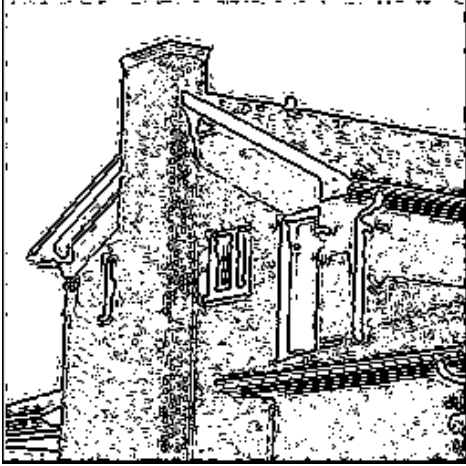
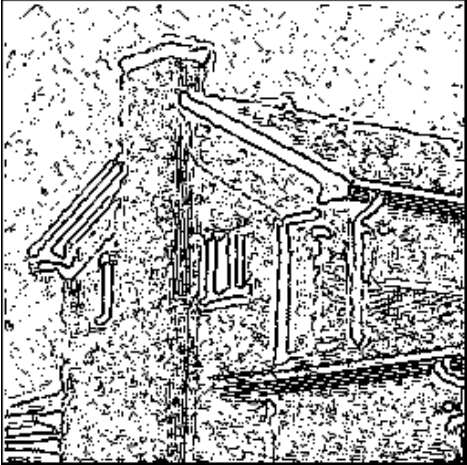
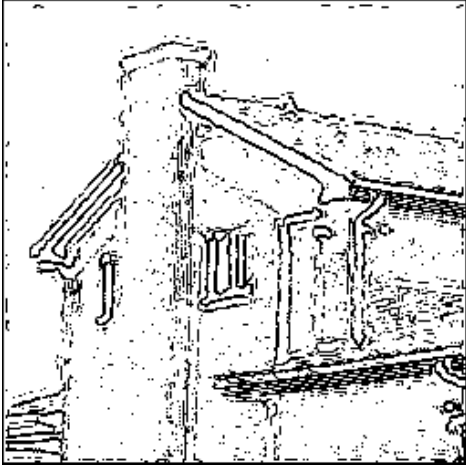
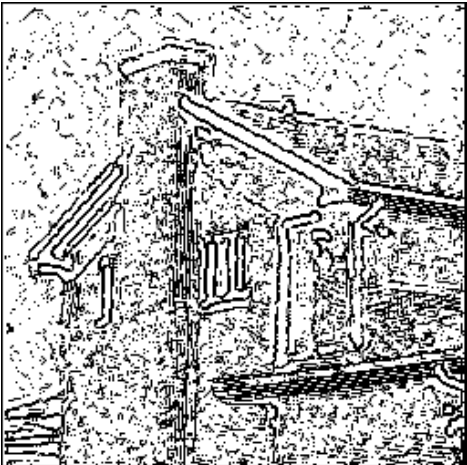
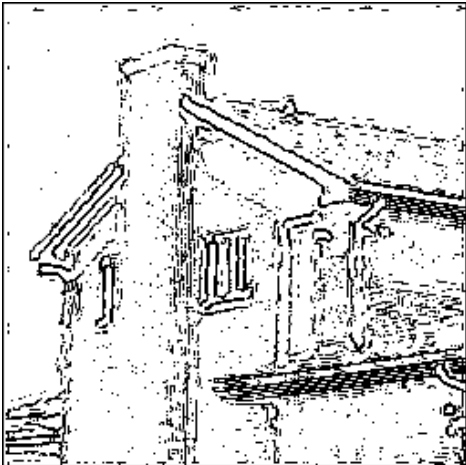
```
55 T=2.5; % threshold
56 max_t= 0;
57
58 for y=2:size(in,1)-2
59     for x=2:size(in,2)-2
60
61         cnt=0; % 부호가 다른 쌍의 갯수 저장
62
63         e_1= [out(y-1,x-1), out(y-1,x+1),out(y-1,x), out(y,x-1)];
64         e_2= [out(y+1,x+1), out(y+1,x-1),out(y+1,x), out(y,x+1)];
65
66         e_3=e_1.*e_2; %부호 확인
67         e=abs(e_1-e_2); %픽셀값의 차이
68
69         for i=1:4;
70             if e_3(1,i) < 0 && e(1,i)> T
71                 if max_t<e(1,i);
72                     max_t=e(1,i);
73                 end
74                 cnt=cnt+1;
75             end
76         end
77         if cnt>1
78             tmp(y,x)=0;
79         else tmp(y,x)=1;
80         end
81     end
82 end
83 end
```

먼저 마주보는 4개의 쌍을 각각 e1, e2에 저장한다.

e1= [북서, 남서, 북, 서] e2=[남동, 남서, 남, 동]

부호가 서로 다른 쌍이 몇 개인지 알기위해서 e_3에 각 원소를 곱한 값을 저장하고, 그 값이 0보다 작으면 부호가 서로 다른것이므로 cnt에 1을 더해준다. 값의 차이가 threshold를 넘는지 판단하기 위해 값의 차이를 e에 저장하고 T보다 큰지 판별한다. 1)부호가 서로다른 쌍이 2개 이상이고 (cnt>1), 2)그 값의 차이가 threshol보다 크다면($e > T$) 영교차이다. 이 두 조건이 만족하면 해당하는 픽셀 값을 0으로 바꿔주고 이외의 경우는 1로 바꿔 저장한다.

-결과 image

	T=2.0	T=4.0
sigma=2.0		
sigma=4.0		
sigma=8.0		

가우시안의 표준편차와 threshold값에 따라 검출되는 엣지가 달라졌다. threshold값이 커질수록 눈으로 봤을 때의 엣지와 비슷해졌고, 검출되는 엣지가 줄어들었다. 표준편차가 작을수록 영상의 엣지가 더 얇고 세밀하다. 표준편차가 8인 영상을 보면 2인 영상보다 엣지가 두껍다.

3) 결과 분석

sobel, prewitt 필터의 경우 단순히 1차미분을 해서 픽셀값의 변화율을 threshold와 비교하여 엷지를 검출했다. sobel의 경우 중심값에 가중치를 주기 때문에 prewitt보다 엷지강도가 더 커지게 된다. 따라서 threshold가 같은 영상끼리 비교해보면 sobel의 엷지강도가 더크기 때문에 다소 모호한 부분의 엷지의 경우 검출될 수 있다.

LoG 필터는 가우시안의 표준편차에 따라 엷지의 세밀함을 조절 할 수 있었다. 표준편차가 작을수록 엷지가 더 세밀했다. 가우시안 필터를 적용하고 라플라시안 연산을 하게 되어 계산량이 많아졌다.