영상처리 PROJECT 1

지도교수	오 병 태 교수님
제출일	2021-05-02
이름	2016124103 박상준

Project1

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

1) Generate the following noises, and add them to the ground-truth images.

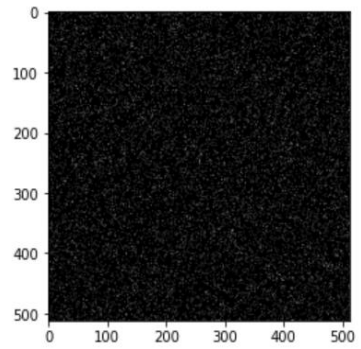
a) Impulse noise b) Gaussian noise

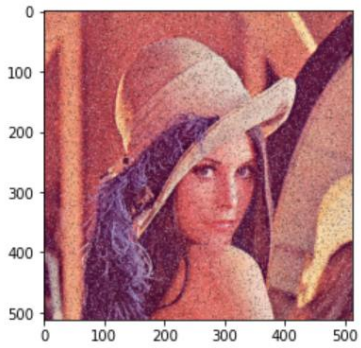
2) Compute PSNR between ground-truth and noisy image.

(Use the various noise energy by noise scaling.)

Impulse noise

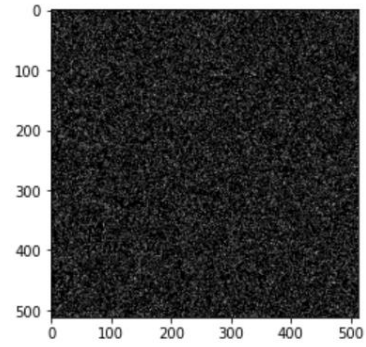
a-1) p=0.1

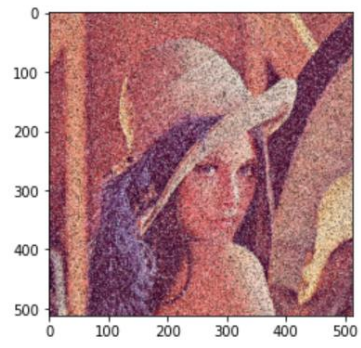




PSNR= 15.331167593921847

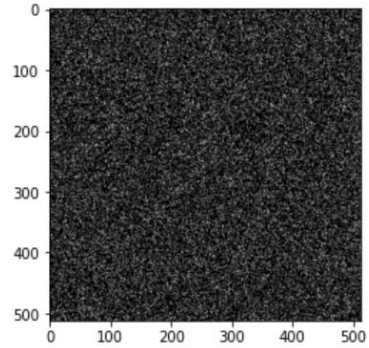
a-2) p=0.3

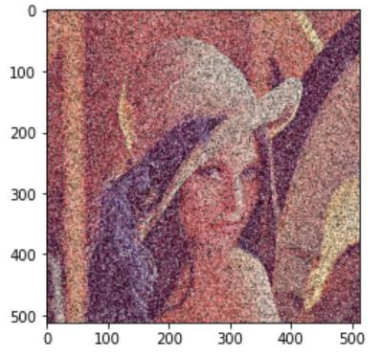




PSNR= 10.984496545253478

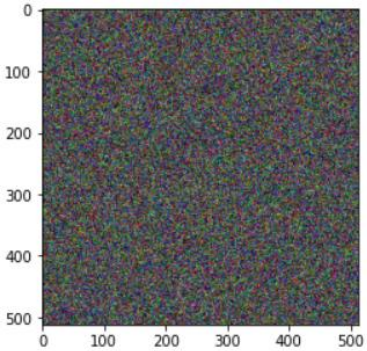
a-3) p=0.5



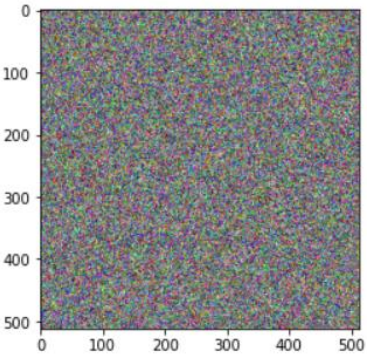


PSNR= 9.164852254918241

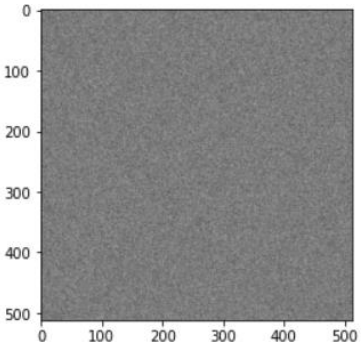
b-1) std=1

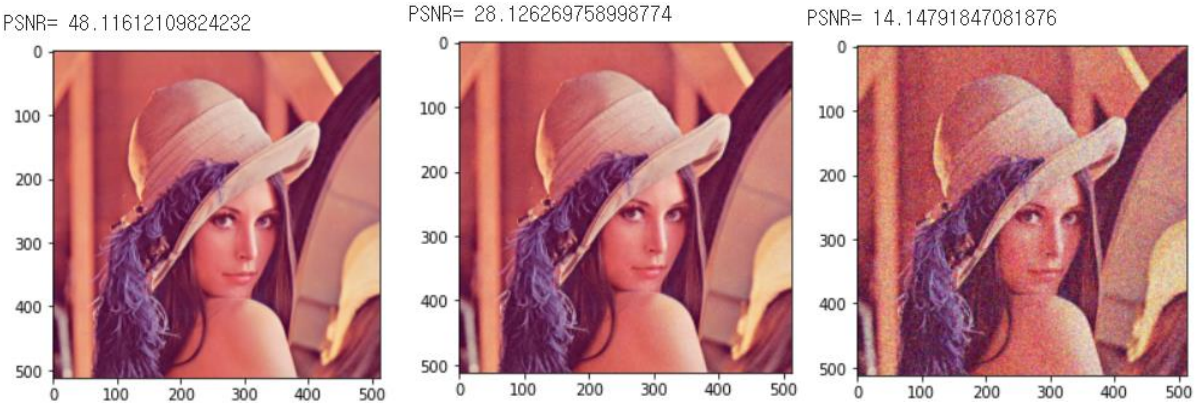


b-2) std=10



b-3) std=50





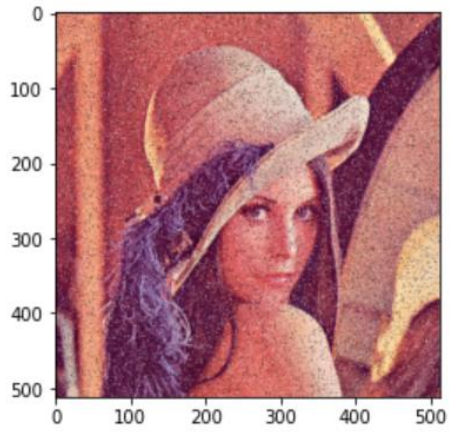
3) Implement the following denoising filters, and apply them for denoising.

a) Gaussian filter b) Bilateral filter

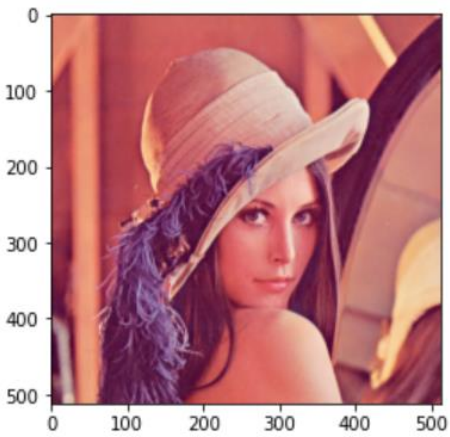
4) Please compare and analyze the results with subjective and objective (PSNR) measurements.
비교는 아래와 같이 진행함.

a-1) Impulse_noisy image

noisy_image: impulse_noisy (p=0.1)



original image:

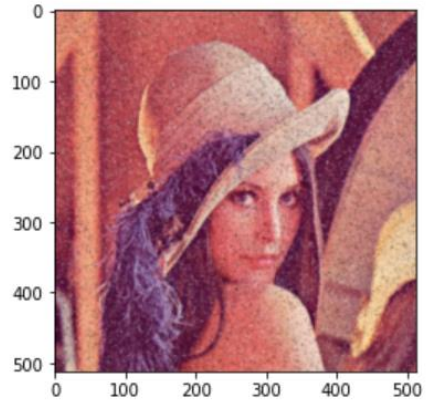


Filtered image :

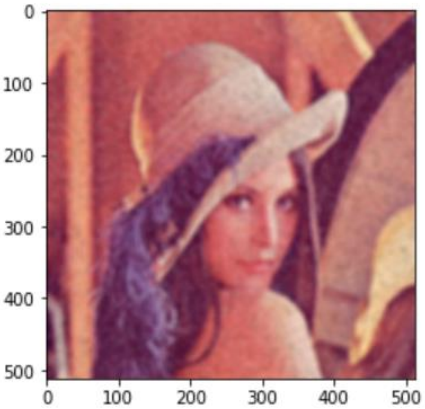
1) Gaussian filtering (size=10, std=1)

2) Gaussian filtering (size=10, std=5)

PSNR= 24.592423977691066
<matplotlib.image.AxesImage at 0x7f3ced990150>



PSNR= 24.625468229581962
<matplotlib.image.AxesImage at 0x7f3ced963e90>

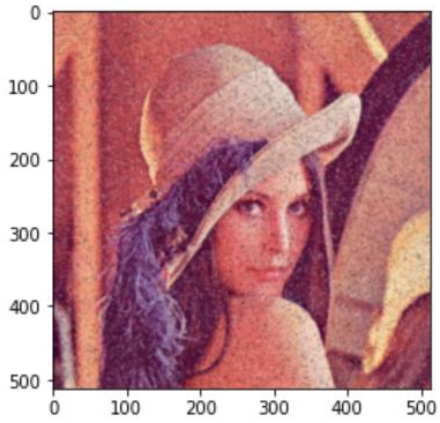


subjective하게 봤을 때, std가 작을 때가 더 우수함.

PSNR로 비교했을 때, 두 차이가 미세함.

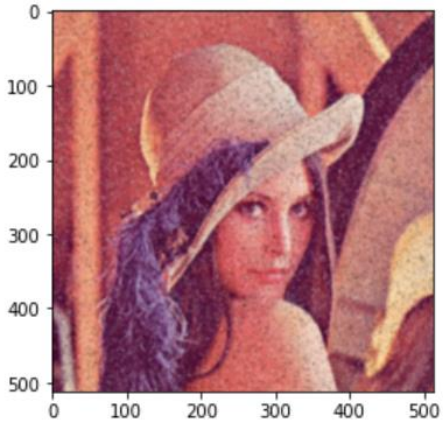
3) Gaussian filtering (size=4, std=1)

PSNR= 23.989252998882527
<matplotlib.image.AxesImage at 0x7f3ced8fafd0>



4) Gaussian filtering (size=4, std=5)

PSNR= 24.58102049958218
<matplotlib.image.AxesImage at 0x7f3ced8b8e10>

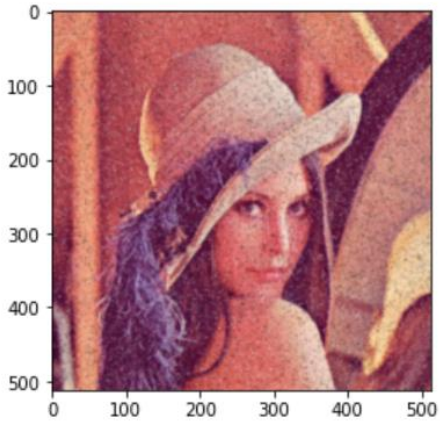


필터 사이즈를 줄였을 경우 std의 변화에 더 민감하며 std가 높을 때 PSNR로 판단하여 품질이 좋다. 육안으로 봤을 때에는 두 차이가 거의 없다.

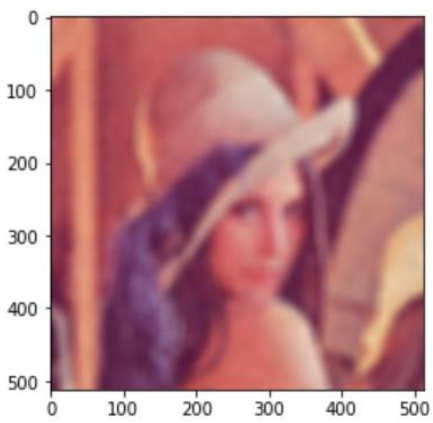
5) Gaussian filtering (size=20, std=1)

4) Gaussian filtering (size=20, std=10)

PSNR= 24.59242928342137
<matplotlib.image.AxesImage at 0x7f3ced7d1e10>



PSNR= 22.437376349900937
<matplotlib.image.AxesImage at 0x7f3ced7afd90>



필터사이즈를 크게했을 때, 비교한 결과 std가 작을 때 PSNR로 판단하여 품질이 좋고, 육안으로 판단했을 때도 std가 작은게 더 좋다.

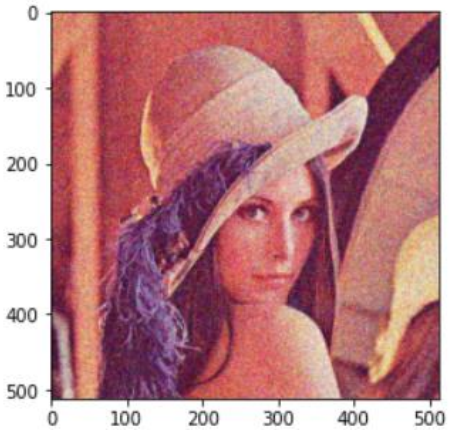
---

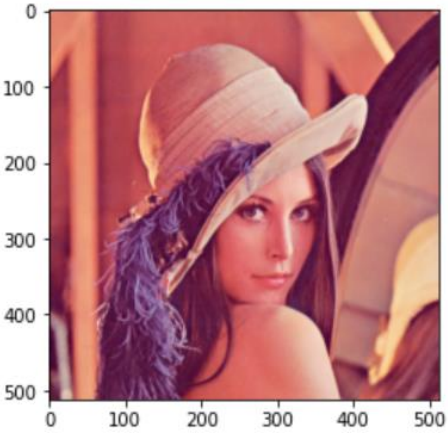
a-2) gaussian noisy image

noisy_image: impulse_noisy (std=50)

original image:

PSNR= 14.14791847081876

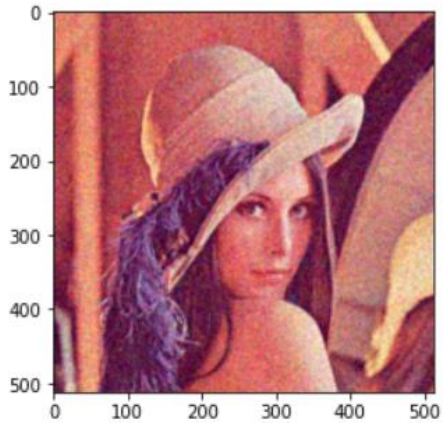




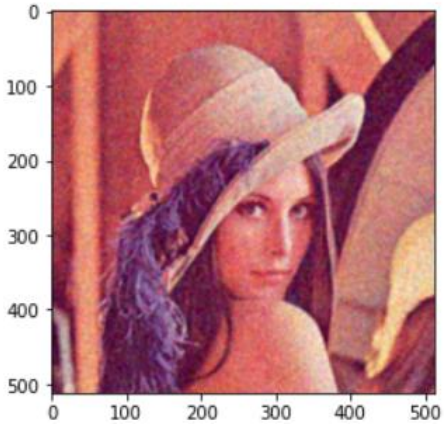
1) Gaussian filtering (size=4, std=1)

2) Gaussian filtering (size=4, std=10)

PSNR= 23.53862091851919
<matplotlib.image.AxesImage at 0x7f3ce2066ad0>



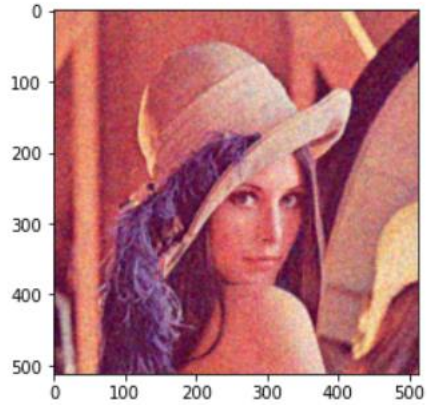
PSNR= 24.461235636546593
<matplotlib.image.AxesImage at 0x7f3ce1f5b710>



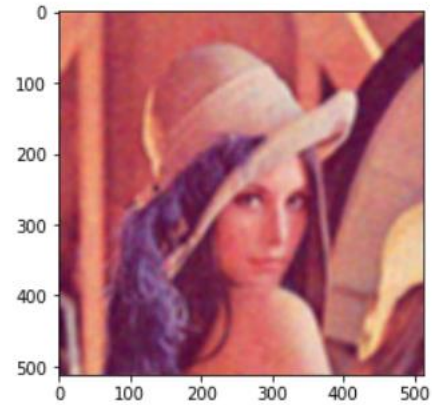
3) Gaussian filtering (size=10, std=1)

4) Gaussian filtering (size=10, std=10)

PSNR= 24.34771616732242
<matplotlib.image.AxesImage at 0x7f3ce1ec2550>



PSNR= 25.12270820491269
<matplotlib.image.AxesImage at 0x7f3ce1ea7410>



필터링을 했을 때 기존 noisy 영상보다 PSNR이 커져 품질이 좋아졌음을 발견하였음.

가우시안 노이즈와 임펄스 노이즈 두 경우에서 가우시안 필터링을 취했을 때 비슷한 결과를 냈다. 필터 사이즈가 커지면 블러링이 심해져 육안으로 확인하는 영상의 품질이 안좋아졌다. Std 역시 클 때 블러링이 심해진 것을 볼 수 있었다. 하지만 PSNR으로 판단하였을 때, std와 filter size가 적 적당 컸을 때 품질이 좋게 판단되었다.

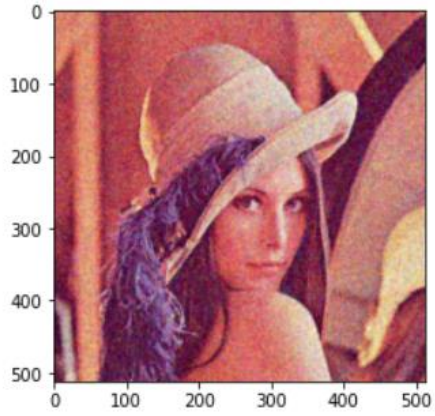
---

b-2) Gaussian noisy image

1) bilateral filtering (size=4, std1=10, std2=20)

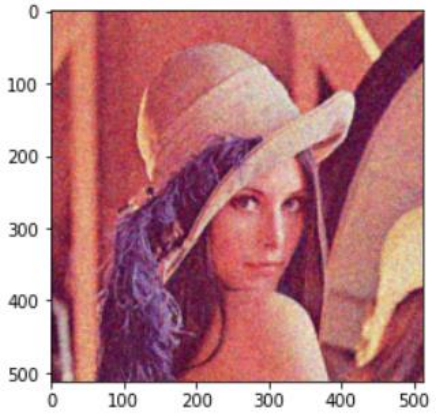
2) bilateral filtering (size=4, std1=10, std2=10)

PSNR= 17.77069421818672
<matplotlib.image.AxesImage at 0x7f3ce212df10>



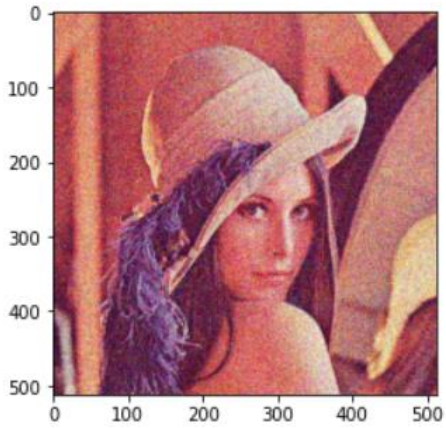
3) bilateral filtering (size=4, std1=1, std2=1)

PSNR= 16.633235885461232
<matplotlib.image.AxesImage at 0x7f3ce2b340d0>

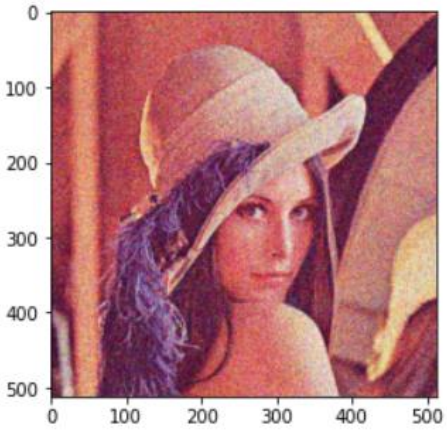


4) bilateral filtering (size=8, std1=1, std2=1)

PSNR= 14.371102675502915
<matplotlib.image.AxesImage at 0x7f3ce21f2dd0>



PSNR= 14.420472728710136
<matplotlib.image.AxesImage at 0x7f3ce1e3af10>



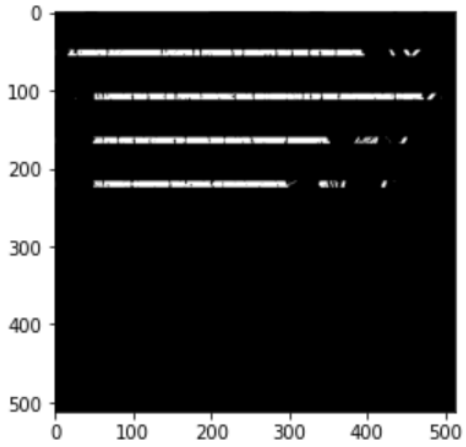
Gaussian noisy 영상을 토대로 bilateral filtering을 진행한 결과이다. Gaussian filtering을 했을 때보다 훨씬 경계에 대해 blurring이 줄어 뚜렷한 이미지를 볼 수 있었다. 하지만 PSNR을 계산했을 때, Gaussian filtering 때보다 훨씬 작은 값을 가졌다. Bilateral filtering의 parameter에 대해 분석하자면, std1, std2가 클 때 PSNR값이 커지고 filtersize가 클 때 PSNR이 큰 것을 확인 할 수 있었다. Bilateral filter 역시 std값과 filtersize가 적당히 컸을 때 성능이 좋다는 것을 알 수 있었다.

2. (Image Restoration) Design a system to remove the text of the image.

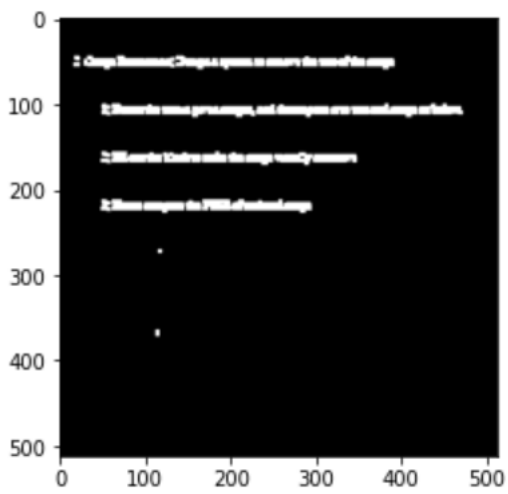
1) Detect the text in given images, and decompose them to the text and image.

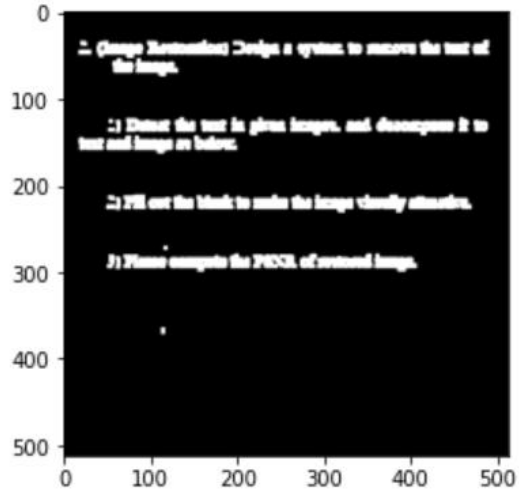
Text를 detect하기 위해 여러가지 방법을 모색하였다. 처음에는 sobel edge detector를 사용하여 검출해낸 뒤 text는 열에 평행하게 많이 위치하고 있으므로 다음과 같이 edge가 우세한 열만 남기도록 edge map을 만들었다.

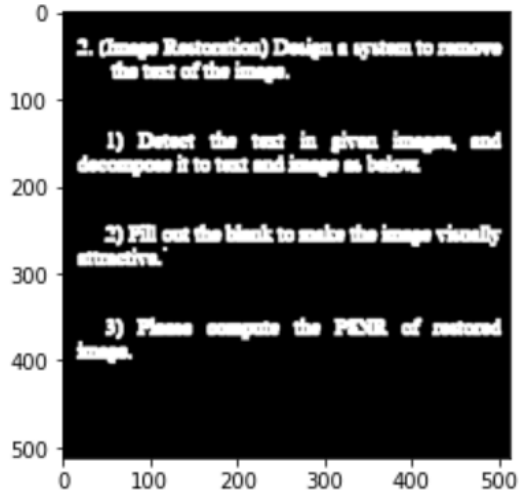
<matplotlib.image.AxesImage at 0x7f8386281190>



하지만 위의 edge map은 이후 inpainting에서 좋지않은 성능을 가졌음을 확인하였고 좀 더 명확하게 detect하기 위해 text이 가지는 특정 pixel 값을 찾아 detect하도록 알고리즘을 바꿨다. 그결과 아래와 같이 명확히 detect 되었으며 text가 아닌 위치가 detect되는 오류가 줄었다.







2) Fill out the blank to make the image visually attractive.

3) Please compute the PSNR of restored image.

Text를 image로 inpainting하기 위해 생각해낸 방법은 먼저 detect한 text가 있는 위치의 pixel 값을 0으로 해 text가 포함된 image에 구멍을 내고 필터링을 통해 text가 위치한 곳을 채우는 것이다.

이를 위하여 bilateral filtering 알고리즘을 응용하였다. 이것은 가우시안 필터 하나와 픽셀의 해당 위치 주변의 픽셀 값에 대한 필터를 곱하여 필터를 만드는 것이다. 쉽게 말하면 0으로 뚫어놓은 pixel을 났을 때 그 픽셀 주변의 pixel 값을 값으로 가지는 space KxK 필터를 생성하고 먼저 생성해 둔 gaussian filter와 곱하여 새로운 필터를 만드는 것이다. 이렇게 하면 text의 pixel값은 0으로 되어 convolution할 때 text를 제외한 image 부분만 filtering된다.

실행 결과 0인 부분이 전부일 때 divide by zero가 되어 값이 표현 안되는 경우가 생긴다. 이 때를 방지하기위해 필터사이즈를 크게 할 수 밖에 없는 한계가 생기고 inpainting한 주변이 filtersize만큼 blurring되는 단점이 생겼다. 이를 최소화 하기 위해서 가로로 써진 text의 특징에 맞춰 세로로 긴 직사각형 필터를 사용하였다. 결과 훨씬 inpainting이 더 잘 되었다. 결과는 아래와 같다.

Filter

(T=K//2)

```
for i in range(0,K):
  for j in range(0,2):
    Gs[i,j]=(np.exp(-((i-T)**2)/(2*(s**2))))
```

Gs는 가우시안 필터를 응용한 직사각형 필터

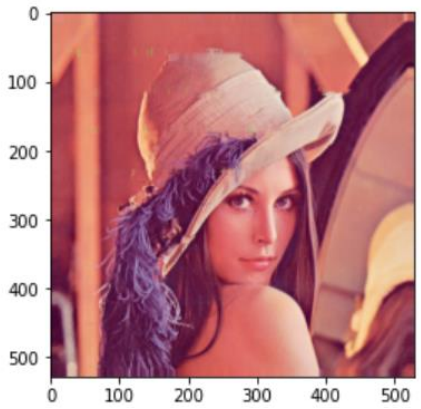
```
for c in range(channel):
  for i in range(T,row+T):
    for j in range(T,col+T):
      if(edge_map[i-T,j-T]==255):
        ix_padded[i,j,c]=0
for c in range(channel):
  for i in range(T,row+T):
    for j in range(T,col+T):
      if(edge_map[i-T,j-T]==255):
        for p in range(i-T,i+T):
          for q in range(j-1,j+1):
            t=(ix_padded[p,q,c])
            G[(p-(i-T)),(q-(j-1))]=t*Gs[(p-(i-T)),(q-(j-1))]

        G=G/np.sum(G)
        product=ix_padded[i-T:i+T,j-1:j+1,c]*G
        ix[i,j,c]=product.sum()
return ix
```

G는 bilateral filter의 원리에서 아이디어를 얻어서 글자가 들어간 pixel은 0, 글자를 제외한 pixel은 pixel값 그대로 곱하기를 진행

Lena(10)

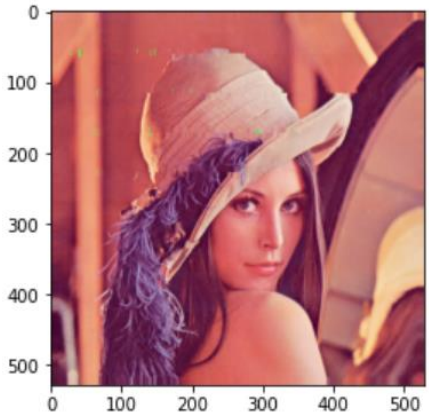
16,10

```
PSNR= 4.07811383648514
<matplotlib.image.AxesImage at 0x7f3ced523b50>
```



16,1

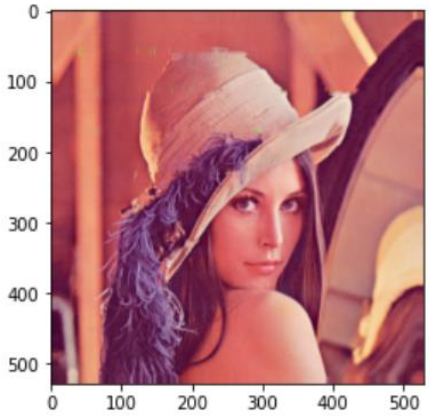
```
PSNR= 4.075444226236886
<matplotlib.image.AxesImage at 0x7f3ced76c350>
```



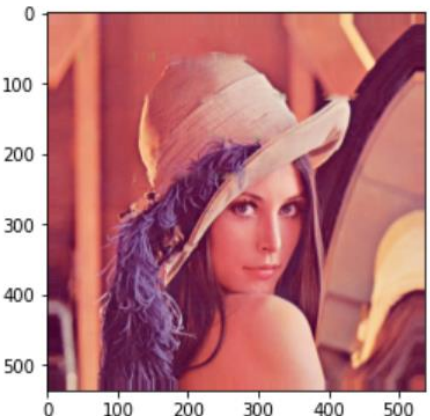
16,20

24,20

PSNR= 4.077999169188712
<matplotlib.image.AxesImage at 0x7f3ced26d650>



PSNR= 4.206614778562469
<matplotlib.image.AxesImage at 0x7f3ced2826d0>

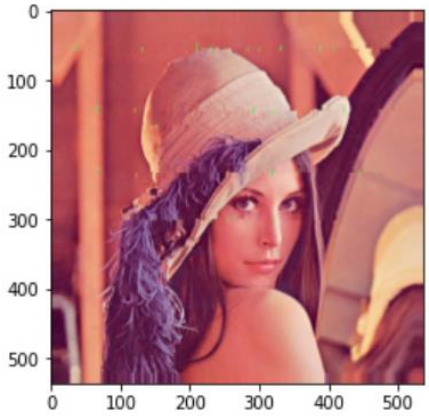


Lena(14)

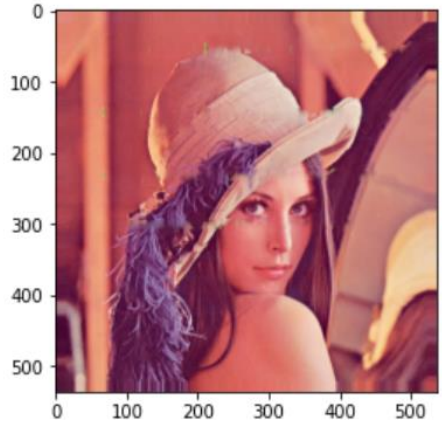
24,1

23,10

PSNR= 4.197547524785104
<matplotlib.image.AxesImage at 0x7f3cedb8e350>



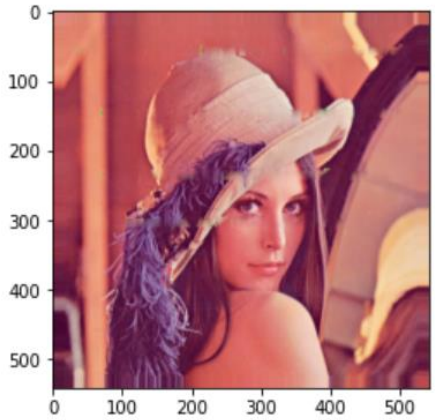
PSNR= 4.192513897822552
<matplotlib.image.AxesImage at 0x7f3ced9f2d10>



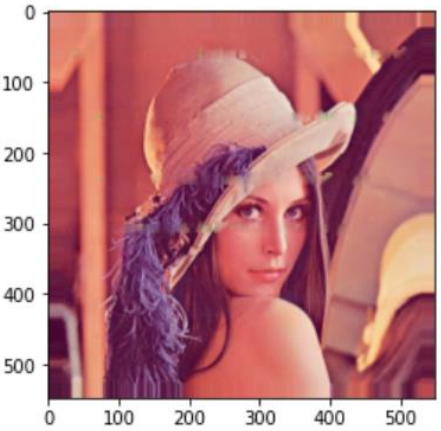
30,10

36,20

PSNR= 4.266146141558922
<matplotlib.image.AxesImage at 0x7f3ced217710>



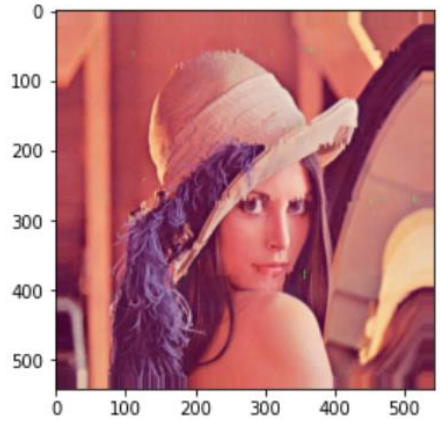
PSNR= 4.350446177766684
<matplotlib.image.AxesImage at 0x7f3ced043fd0>



Lena(18)

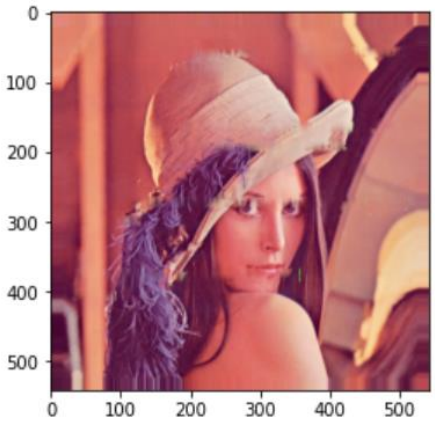
30,1

```
PSNR= 4.29711264793764
<matplotlib.image.AxesImage at 0x7f3ced0216d0>
```



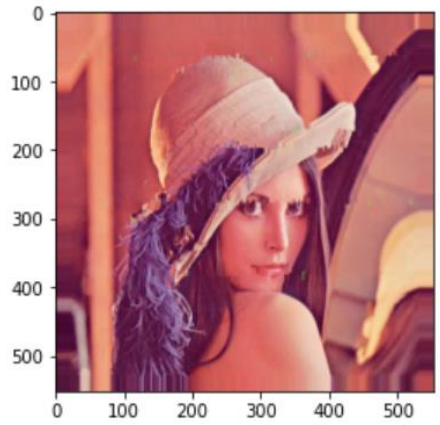
30.10

```
PSNR= 4.276086682056147
<matplotlib.image.AxesImage at 0x7f3cecf8df50>
```



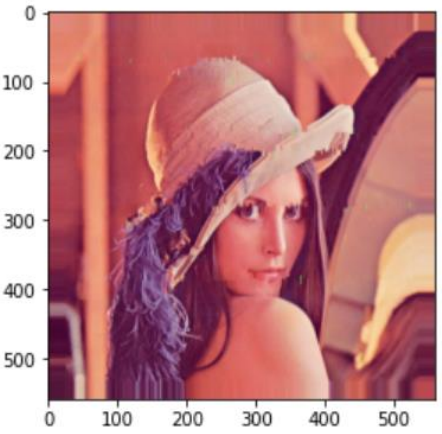
40,1

```
PSNR= 4.440688529950046
<matplotlib.image.AxesImage at 0x7f3cecf0f890>
```



46,1

```
PSNR= 4.504669447435894
<matplotlib.image.AxesImage at 0x7f3cece9c2d0>
```



3. (Image Transformation) Implement the followings and analyze and compare the results.

1) Transform the image with arbitrary angle and scale.

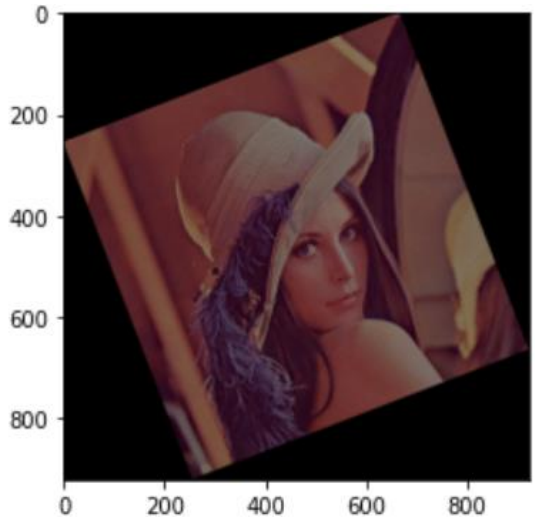
(Note that the image should be rotated with its center point)

e.g. Scale = 1.5, Angle = 30°

e.g. Scale = 0.7, Angle = 145°

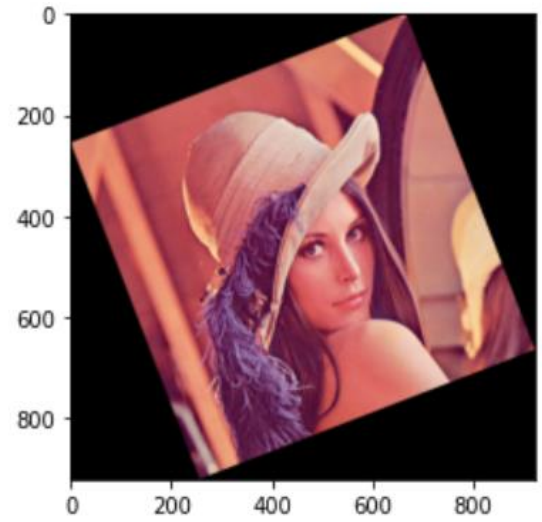
2) Use the nearest neighborhood (NN) and bilinear method for interpolation.

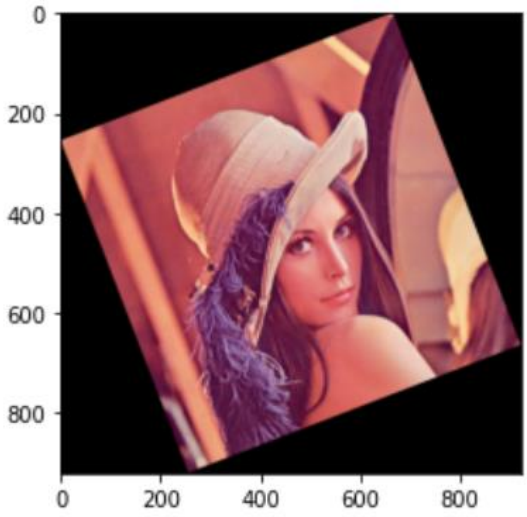
```
t_image,xmin,ymin,N=tf_system(image,1.5,30)
```



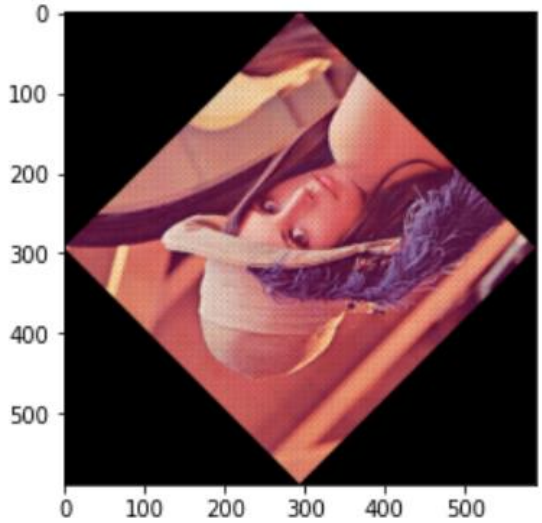
Bilinear

NN interpolation



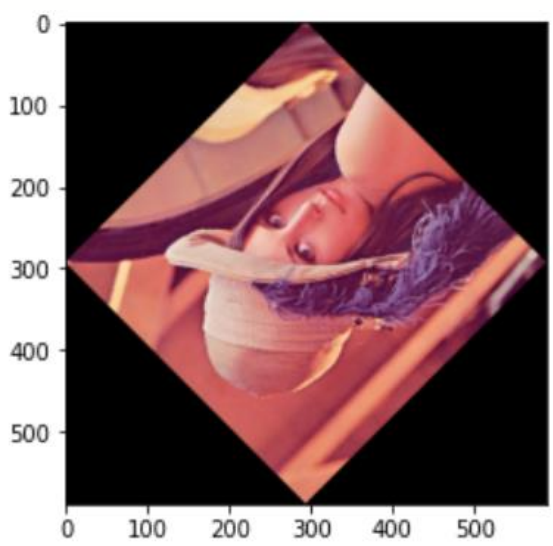
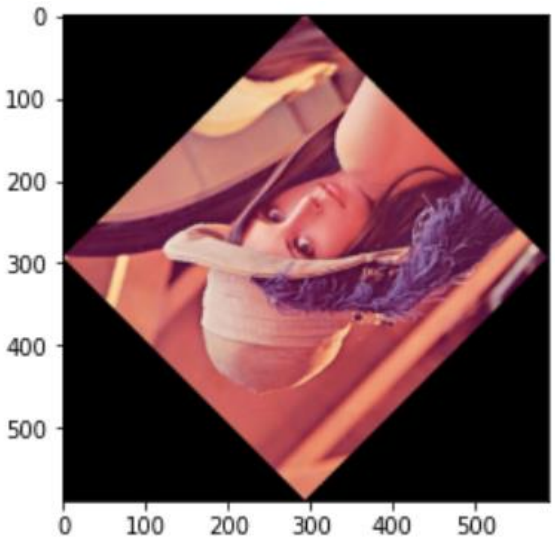



```
t_image,xmin,ymin,N=tf_system(image,0.7,145)
```

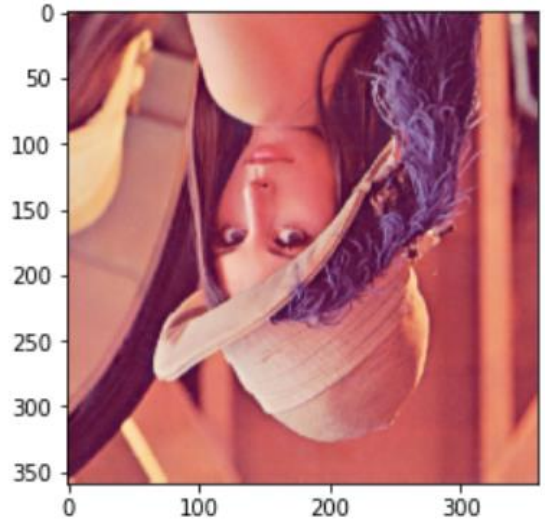


Bilinear

NN interpolation

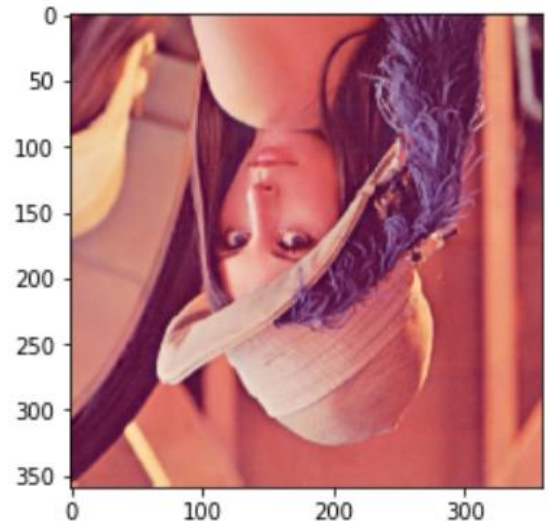
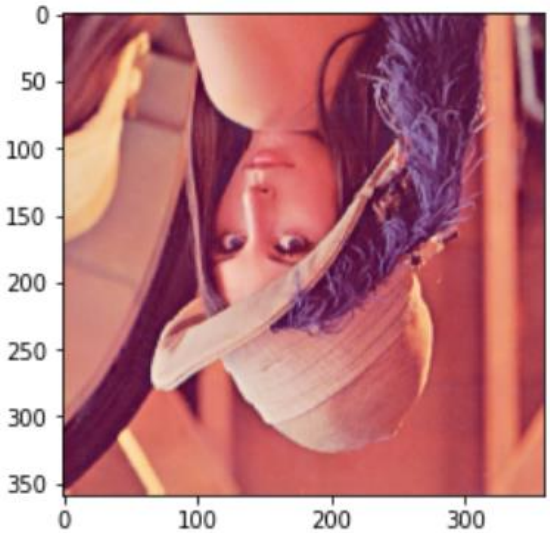



```
t_image,xmin,ymin,N=tf_system(image,0.7,180)
```

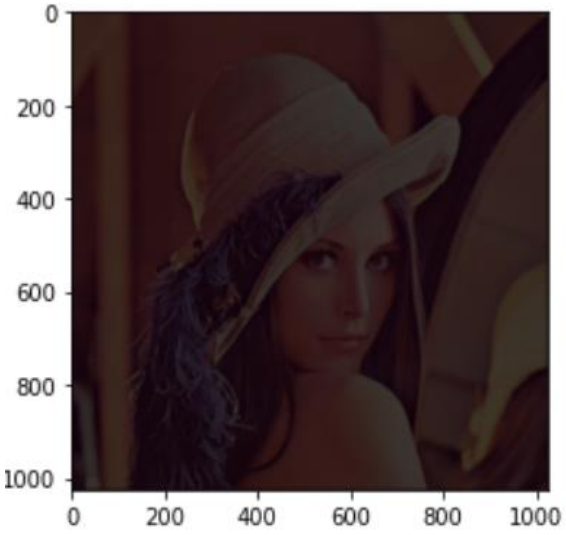


Bilinear

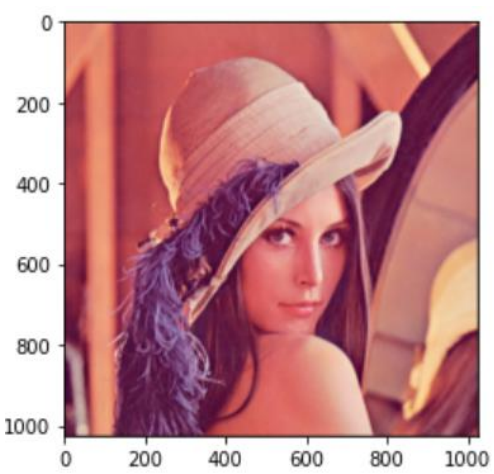
NN interpolation

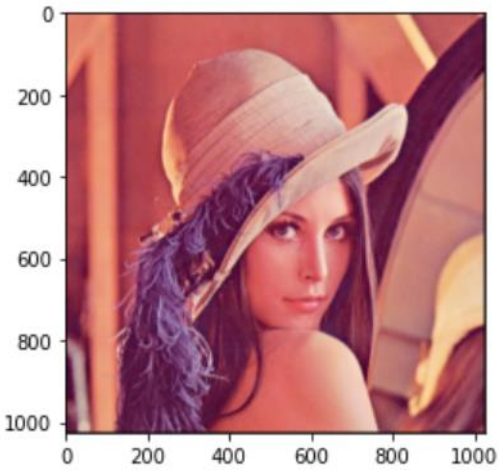



```
t_image,xmin,ymin,N=tf_system(image,2,0)
```



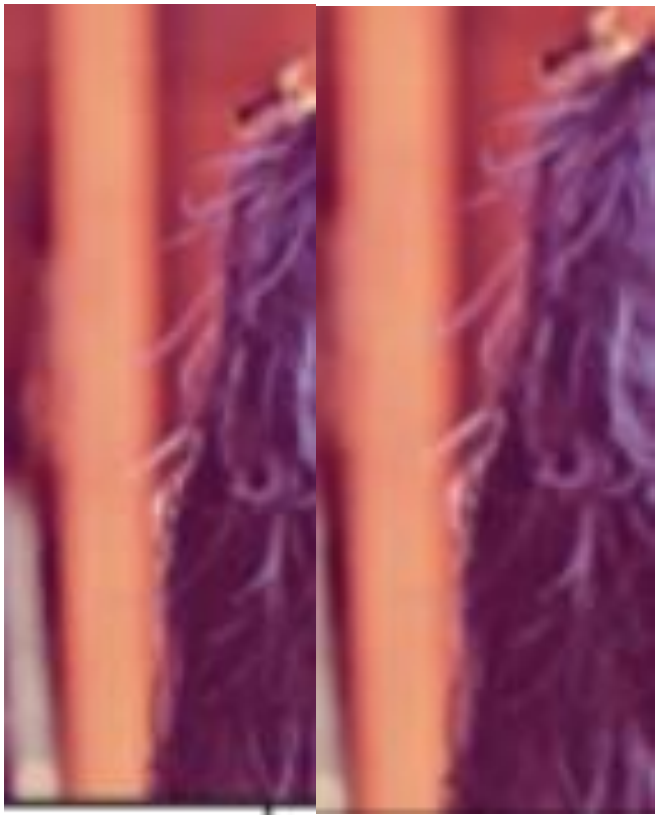
Bilinear NN interpolation





3) Compare the interpolation results.

Bilinear NN



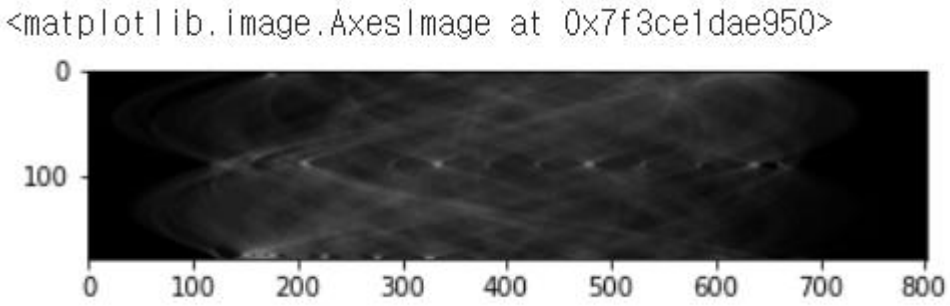
두 interpolation의 차이는 subjective하게 판별하기 어렵다. 하지만 확대해서 정교하게 살펴본 결과 bilinear interpolation을 적용한 image가 더욱 부드러운 느낌을 주는 것을 확인 할 수 있었다.

4. (Hough Transform) Please implement the followings and analyze the results.

1) Find the edge map using Sobel mask.

2) From binary edge map, apply the Hough transform.

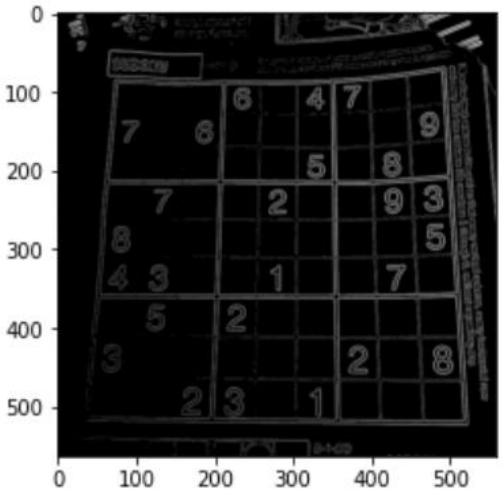
3) Analyze the result in Hough domain, and find the major lines.

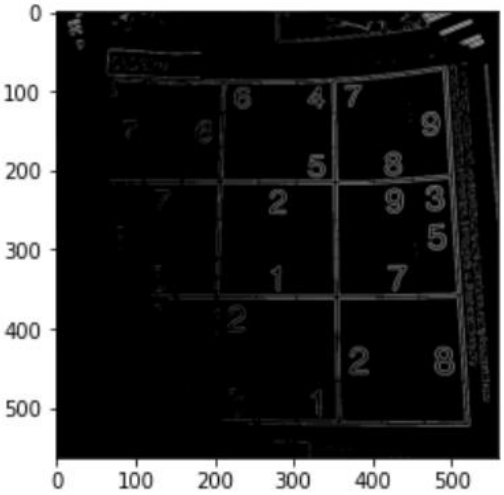


위 image는 hough transform을 진행한 뒤 hough domain에 표현한 것이다. 보면 우세한 점들이 90도 근처와 180도 근처에 있다. 이 점들은 major line을 transform 후 hough domain에서 표현한 것이다. 점을 뽑아서 역 trasform을 실행한 뒤 원래 image에 그으면 그 직선이 major line이 된다.

4) Draw the lines on top of the image as below.

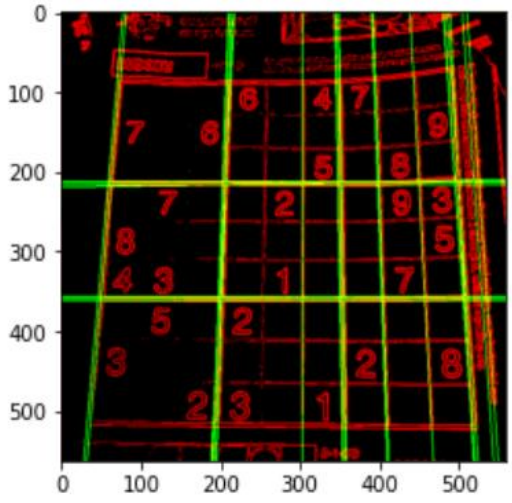
`edge_map=sobel(image,100)` `edge_map=sobel(image,200)`

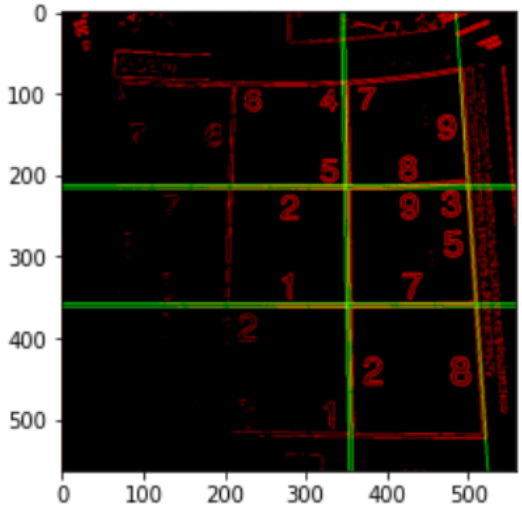




```
if Amap[theta,r]>(170):
```

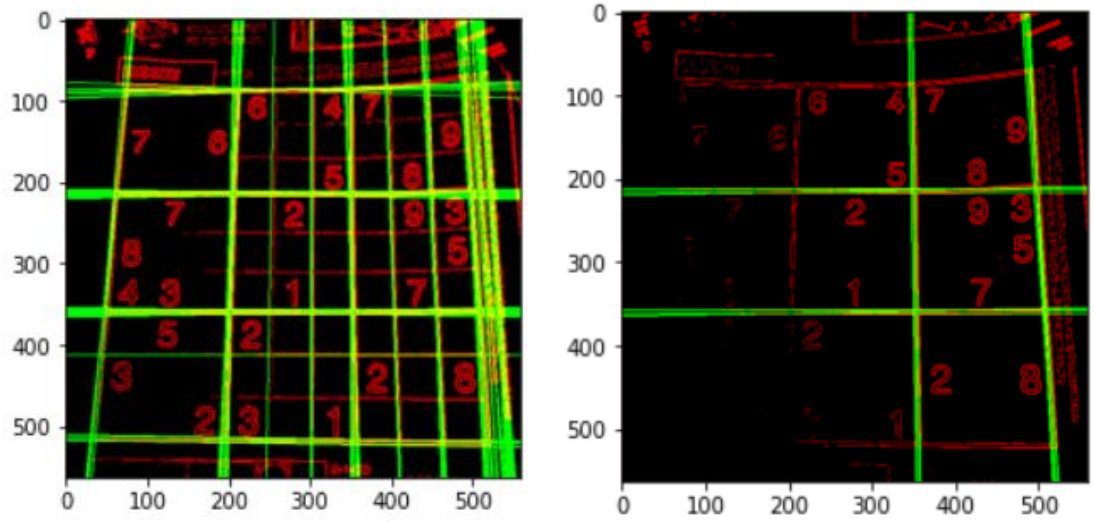
hough domain의 threshold 값 170





```
if Amap[theta,r]>(130):
```

hough domain의 threshold 값 170



다음은 open cv library를 사용하여 hough trasform을 실행하여 major line을 찾은 결과이다. 라이브러리를 사용하지 않고 한 결과와 비슷하게 나왔다.

```
img = cv2.imread(inpath,cv2.IMREAD_UNCHANGED)
edges = cv2.Canny(img,50,150,apertureSize = 3)
lines = cv2.HoughLines(edges,1,np.pi/180,150)
```