영상처리

PROJECT 1

과목	영상처리
이름	차원범
학번	2016124217
교수님	오병태 교수님
제출일	20.04.26

1. Image Denoising

1) Generate noises

a) Impulse noise


➢ 원본 이미지(좌) 와 Impulse noise를 첨부한 이미지(우)

Impulse noise의 경우는 Matlab의 rand함수를 이용하였다. 임의로 정한 threshold(0~1사이) 값에 따라 이미지와 같은 크기로 생성된 rand행렬과 비교한다. threshold값과 비교하여 더 큰 값을 갖는 경우, 255의 값을 대입하며, 1-threshold보다 작은 경우 0의 값을 대입하여 impulse noise를 생성하였다.

b) Gaussian noise


➢ 원본 이미지(좌) 와 Gaussian noise를 첨부한 이미지(우)

Gaussian noise의 경우는 Matlab의 gaussian 분포를 가지는 함수 randn을 사용하였다. 전달함수로 sigma변수값을 전달하여 noise의 정도를 조절할 수 있다.

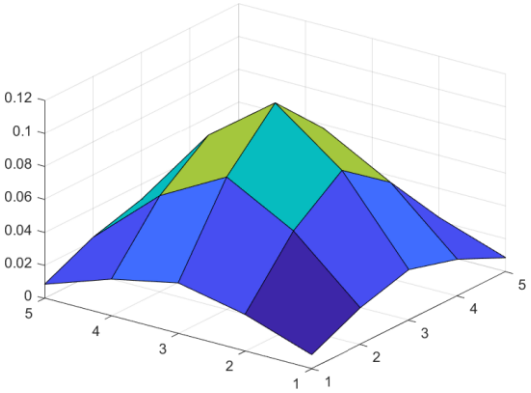
c) Uniform noise

➢ 원본 이미지(좌) 와 Uniform noise를 첨부한 이미지(우)

Uniform noise의 경우는 Matlab의 rand함수를 이용하였다. 기본적으로 0~1사이의 값을 랜덤하게 출력하기 때문에, 0.5를 빼 주어 -0.5~0.5의 값을 생성하였고, gaussian noise와 마찬가지로 sigma변수값을 전달하여 noise의 정도를 조절할 수 있다.

2) Denoising filters

a) Gaussian filter



2차원 Gaussian 함수를 이용하여 필터를 만들었다. 매개변수로는 Gaussian Sigma값과, 필터의 사이즈를 전달할 수 있다. 위의 경우는 σ = 1.25, 필터 사이즈 = 5*5로 시행한 결과이다. 이 filter를 이미지의 모든 화소에 대해 컨볼루션 연산을 진행하였다.

➢ 좌측부터 input image, impulse noise를 첨부한 이미지, gaussian filtering한 이미지

값이 0 혹은 255로 튀는 Impulse noise와 같은 경우는 노이즈의 정도가 줄어들기는 하였지만 여전히 깔끔하지 못한 결과물을 보여주었다.

- 좌측부터 input image, gaussian noise를 첨부한 이미지, gaussian filtering한 이미지 (Gaussian noise의 sigma는 20으로 설정)

Gaussian noise의 경우는 평균이 0인 노이즈이기 때문에, gaussian filtering으로 효과적으로 노이즈 제거를 할 수 있다. 하지만 경계선이 뭉개지는 현상이 발생되어 이미지의 전체적인 품질 저하도 동반되었다.

- 좌측부터 input image, uniform noise를 첨부한 이미지, gaussian filtering한 이미지 (Uniform noise의 sigma는 60으로 설정)

Gaussian noise와 시각적인 차이는 크게 나지 않으며, 마찬가지로 경계선이 뭉개지는 현상이 발생한다.

- 좌측부터 Gaussian noise를 첨부한 이미지, $\sigma = 0.8$로 설정한 Gaussian filtering이미지, $\sigma = 2.0$로 설정한 Gaussian filtering이미지 (Filter size는 5로 고정)

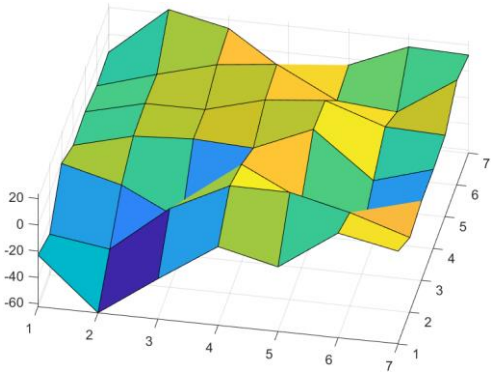
σ 값에 따른 필터링의 정도도 확인하였는데, 값이 작을수록 경계선이 뭉개지는 현상은 적게 발생하였다. 하지만 denoising의 정도가 적었으며, σ 값이 커질수록 denoising의 정도가 좋았다. 하지만 경계선이 뭉개지는 현상이 더욱 심하다는 것을 볼 수 있다.

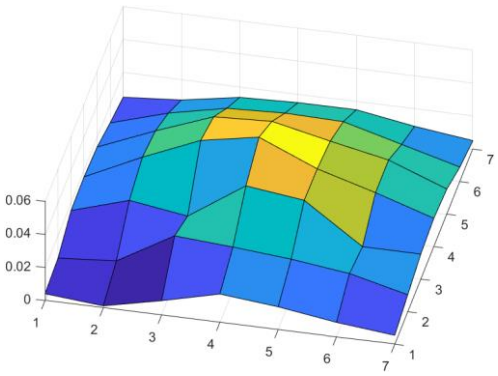


- 좌측부터 Gaussian noise를 첨부한 이미지, filter size를 3으로 설정한 Gaussian filtering이미지, filter size를 7로 설정한 Gaussian filtering이미지 ($\sigma = (filter\ size) / 4$ 로 고정)

Filter size를 바꾸어 가면서도 진행하였다. Filter size가 작으면 경계선의 뭉개짐 현상이 적지만, denoising의 정도도 약하다. Filter size가 크면 경계선의 뭉개짐 현상이 크지만, denoising의 정도도 개선되는 것을 볼 수 있다.

b) Bilateral filter





- 그림 2.b-1(좌), 2.b-2(우)

필터링하고자 하는 목표 픽셀값과 비슷하면 높은 가중치를 주며, 큰 차이를 보이면 작은 가중치를 갖도록 하는 필터를 설계하였다. 그림 2.b-1은 이미지의 픽셀 일부를 표현한 것이다. 중앙 픽셀값을 기준값 0으로 잡고, 그보다 높고 낮음을 상대적으로 표현하였다. 이 값을 이용해 가중치를 다르게 적용할 수 있다. 그림 2.b-2에서 Gaussian filter와는 다르게 픽셀값과 다른 값을 가질수록 낮은 가중치를 갖는 것을 확인할 수 있다. 기본적으로 $\sigma_s = 2, \sigma_r = 30$, 필터 사이즈=7로 두고 테스트를 진행하였다.

➢ 좌측부터 input image, impulse noise를 첨부한 이미지, bilateral filtering한 이미지

값의 차이가 크면 가중치가 급격히 떨어지는 bilateral filter의 특성 상, impluse noise에 대해서는 거의 변화를 주지 않는 것을 볼 수 있다.

➢ 좌측부터 input image, gaussian noise를 첨부한 이미지, bilateral filtering한 이미지

전반적으로 denoising이 진행됐으며, 엣지 부분의 뭉개짐 현상도 적음을 확인할 수 있다.

➢ 좌측부터 input image, uniform noise를 첨부한 이미지, bilateral filtering한 이미지

Gaussian noise와 큰 차이 없이 denosing이 진행되었다. 마찬가지로 엣지 부분의 뭉개짐 현상도 적게 관찰되었다.

➢ 그림 2.b-3(좌), 2.b-4(중간) 2.b-5(우)

Gaussian noise가 들어간 이미지에 Gaussian filtering을 적용한 이미지는 그림 2.b-3이다. 같은 noise에 Bilateral filtering을 적용한 이미지 2.b-4와 비교하였다. 확대해서 관찰하였을 때, 모서리를 포함한 경계선의 구분감이 그림 2.b-4가 더 또렷함을 볼 수 있다. 또한 2.b-5는 Matlab의 'imbilatfilt' 라이브러리를 통하여 필터링한 이미지이다. 직접 생성한 bilateral filter와 비교해서 denosing이 보다 자연스럽게 되었고, 경계선은 비슷하게 보존하는 것을 볼 수 있었다.

➢ 좌측부터 $\sigma_r = 1$, $\sigma_r = 30$, $\sigma_r = 1000$일 때의 Bilateral filtering 비교

$\sigma_s = 2$로 고정한 상태에서 σ_r값을 바꾸어 가면서 테스트하였다. $\sigma_r = 1$일 때에는 input noise 영상과 차이가 거의 없음을 통해 필터링이 제대로 되지 않음을 볼 수 있다. $\sigma_r = 30$ 일 때에는 경계선을 보존하면서 noise를 적절히 제거해 나감을 볼 수 있었고, $\sigma_r = 1000$일 때에는 noise는 제거되었으나 경계선의 보존이 이루어지지 않음을 통해 gaussian filtering과 거의 같은 필터링을 하는 것을 볼 수 있었다.

➢ 좌측부터 $\sigma_s = 0.5$, $\sigma_s = 2$, $\sigma_s = 10$일 때의 Bilateral filtering 비교

$\sigma_r = 30$으로 고정한 상태이다. $\sigma_s = 0.5$일 때에는 Gaussian kernel의 분산이 낮아 중앙 픽셀값에

가중치가 치중된다. 이는 각 지역별 픽셀 값에 따른 가중치 차이를 주어도 효과를 보기가 어렵다. 따라서 noise의 제거와 경계선 보존 둘 다 이득을 보지 못했다. 반면 $\sigma_s = 2$일 때에는 적절한 noise제거와 모서리 보존이 이루어 졌으며, σ_s를 더 큰 값인 10으로 주었을 때에는 중앙 픽셀 값에 치중되는 정도가 적으나, 경계선의 보존도 이루어지며 noise도 제거되는 것을 볼 수 있다.

c) Median filter

Median filter는 추출해낸 mask의 픽셀 값을 오름차순 혹은 내림차순으로 정렬한 후, 그 중간값을 해당 픽셀에 대입하는 filter이다. 정렬 방법은 Bubble sort를 이용하였다.

➢ 좌측부터 input image, Impulse noise를 첨부한 이미지, Median filtering한 이미지

Noise가 0 혹은 255의 값을 가지는 impulse noise에서 Median filter는 중간값만 이용하기 때문에 효과적으로 noise제거를 할 수 있다. 하지만 해당 픽셀 값과 아주 큰 차이를 보이지 않는 Gaussian noise나, Uniform noise에서는 효과적인 noise제거를 기대하기 어렵다.

➢ 좌측부터 Impulse noise를 첨부한 이미지, filter size를 3*3으로 설정하여 Median filtering한 이미지, filter size를 5*5로 설정하여 Median filtering한 이미지

Median filter의 경우는 filter size가 커질수록 오히려 성능이 하락하는 것을 볼 수 있었다. 그 이유는 더 많은 픽셀을 고려할수록 고려하는 noise가 많아지고, 중간값에 위치하는 값이 noise일 확률이 높아지기 때문인 것으로 확인된다.

3) PSNR compare

Gaussian Filter				
	Non filtering	Filter size		
		3*3	5*5	7*7
Impulse noise	15.36	22.82	22.15	26.07
Gaussian noise	24.64	31.05	30.74	29.90
Uniform noise	24.94	31.35	30.82	30.57

$\sigma = (filter\ size)/4$

- Filter size에 따른 Gaussian Filtering성능을 PSNR로 측정하였다. Filter size가 7*7이상으로 과도하게 커지면, Impulse noise를 제외한 나머지 noise에서는 오히려 약간의 성능 하락이 동반되었다. 이는 noise의 제거보다, 이미지의 blur현상이 PSNR에 영향을 미치기 때문으로 보인다.

Gaussian Filter				
	Non filtering	σ		
		0.3	1.25	2
Impulse noise	15.36	15.55	22.15	25.84
Gaussian noise	24.64	24.81	30.74	29.79
Uniform noise	24.94	25.23	30.82	29.92

$filter\ size = 5 * 5$

- σ값에 따른 Gaussian Filtering 성능을 비교하였다. $\sigma = 0.3$과 같이 낮은 값을 주었을 때에는 분산이 너무 낮은 Gaussian kernel이 생성돼 필터링이 거의 되지 않음을 볼 수 있었다. 반면 $\sigma = 2$와 같이 높은 값을 주었을 때에는 괜찮은 성능을 보였으나, $\sigma = 1.25$일 때와 비교하면 조금 낮은 성능을 보였다. 유일하게 Impluse noise에서는 오히려 σ값이 높을수록 좋은 성능을 보였다.

Bilateral Filter				
	Non filtering	σ_s		
		0.3	2	10
Impulse noise	15.36	15.36	15.55	15.58
Gaussian noise	24.64	24.65	31.38	31.37
Uniform noise	24.94	24.94	31.60	31.55

$\sigma_r = 30,\quad filter\ size = 5 * 5$

- σ_s에 변화를 줘 가면서 Bilateral Filter의 성능을 비교하였다. Impulse noise에서는 Bilateral filter가 거의 효과가 없었으며, 나머지 noise에서는 $\sigma_s = 2$부근에서 가장 준수한 성능을 보였다. 또한 0.3과 같이 너무 낮은 값이 들어가면 필터링 자체가 거의 되지 않음을 알 수 있었다.

Bilateral Filter				
	Non filtering	σ_r		
		1	30	1000
Impulse noise	15.36	15.36	15.55	25.94
Gaussian noise	24.64	24.64	31.38	29.82
Uniform noise	24.94	24.94	31.60	29.84

$\sigma_s = 2, \quad filter\ size = 5 * 5$

- σ_r에 따른 Bilateral filter의 성능을 비교하였다. σ_r이 1과 같이 너무 작으면 필터링이 되지 않았다. $\sigma_r = 30$부근에서 가장 이상적인 필터링 결과가 나왔으며, PSNR값도 가장 높았다. $\sigma_r = 1000$인 부근에서는 Gaussian filter와 거의 비슷한 결과를 얻을 수 있었다. σ_r값이 커질수록 경계선 고려를 적게 하므로 Gaussian filter와 비슷한 성능을 보일 것으로 예상된다.

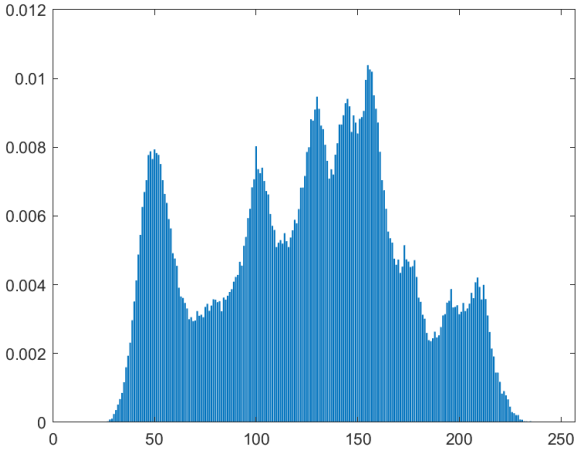
Median Filter				
	Non filtering	$filter\ size$		
		3*3	5*5	7*7
Impulse noise	18.48	34.76	23.15	18.90

- Filter size가 커질수록 그 성능이 떨어지는 것을 psnr값으로 확인할 수 있었다. 더 많은 픽셀을 바탕으로 정렬할수록 noise가 중간값에 들어갈 확률이 높기 때문으로 보인다. 가장 이상적인 필터 사이즈는 3*3이었다.

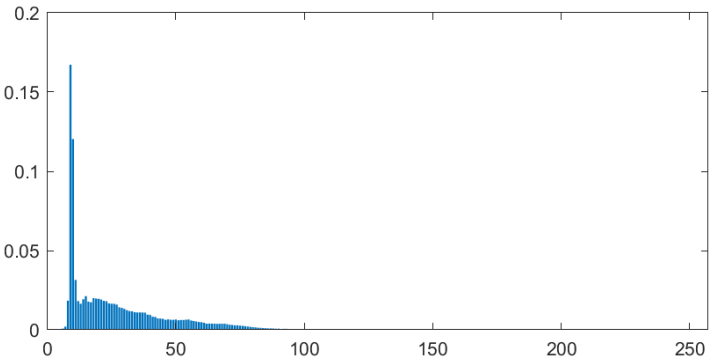
2. Image Enhancement

1) Compute histogram

히스토그램은 for문을 사용한다. 이미지의 1,1 좌표부터 가장 끝 좌표까지 돌아가면서 해당 픽셀 값에 맞는 배열의 인덱스에 +1을 시키면서 찾을 수 있다. 예시로 Lena의 이미지와, 도심 야경의 이미지를 바탕으로 히스토그램을 분석하였다.



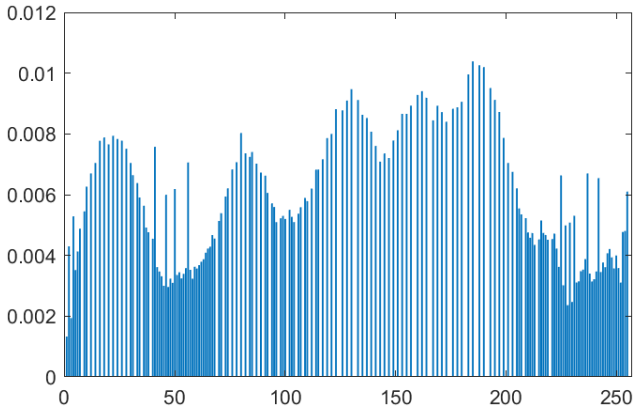
- Lena 이미지와 그 histogram



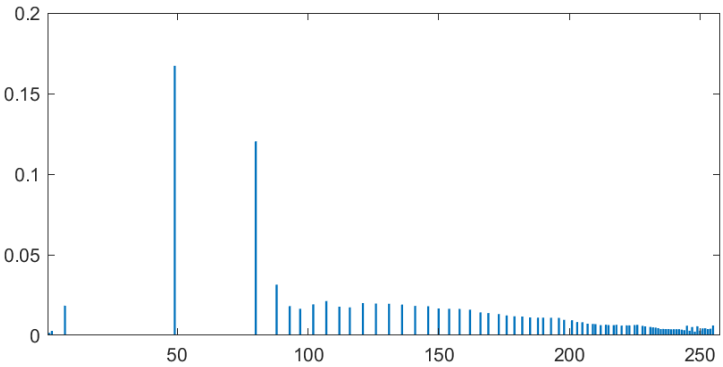
➢ 도시 야경 이미지와 그 histogram

도심 야경 이미지의 경우 lena 이미지와 달리 어두운 부분이 대부분이기 때문에 히스토그램 상에서 0~50의 성분이 대다수임을 알 수 있었다.

2) Histogram equalization



➢ Lena 이미지에 histogram equalization 한 결과

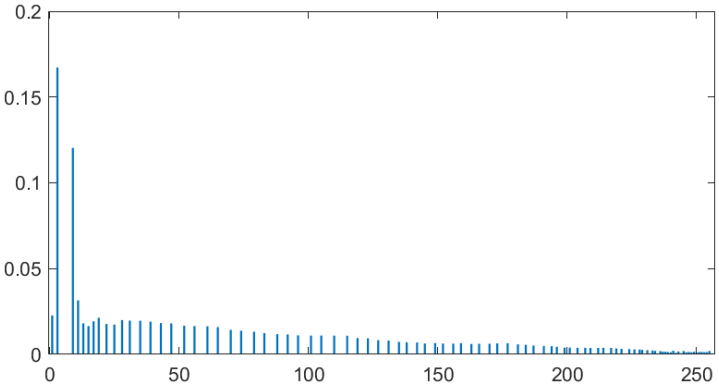


➢ 도심 야경 이미지에 histogram equalization 한 결과

Lena이미지와 비교해서, 도심 야경 이미지의 경우 히스토그램이 어두운 부분에 치중해 있었다. 따라서 보다 극적인 histogram equalization 결과를 얻을 수 있었다.

3) Histogram equalization with Weber's law

베버의 법칙(Weber's law)를 이용하여 histogram equalization을 개선하였다. 베버의 법칙에 따르면 사람의 감각기관에서, 처음에 약한 자극을 받으면 자극의 변화가 적어도 그 변화를 인지하기 쉽다. 하지만 강한 자극을 받게 되면 자극의 변화가 커야만 그 변화를 인지하기 쉽다는 것이다. 이것이 사람의 시각에도 적용되는데, 이를 응용하여 histogram equalization에 적용하였다.



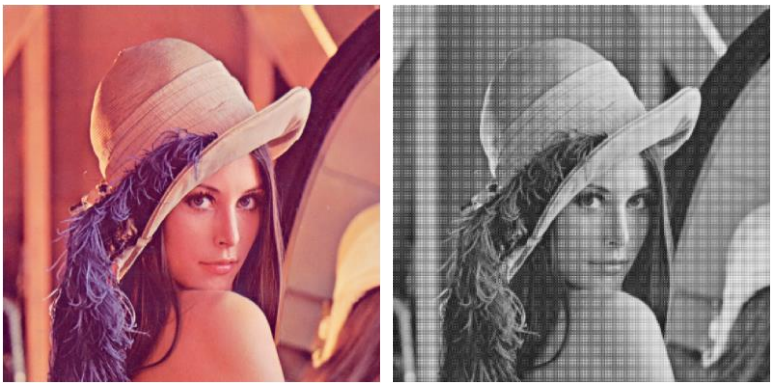
➢ Weber's law를 이용해 개선한 histogram equalization

야경 이미지와 같은 경우는 낮은 명도에서 과도한 histogram equalization이 이루어진다. 하지만 베버의 법칙에 따르면, 낮은 명도에서는 픽셀 값 사이의 변화가 적어도 인지하기 쉽다. 따라서 낮은 명도에서는 이 정도를 줄이고, 높은 명도에서는 equalization이 더 적극적으로 이루어지게 수정하였다. $y = x^{\gamma}$을 이용하여 구현할 수 있었으며, x는 해당 픽셀 값, γ는 함수 매개변수로써 3을 사용하였다. 그 결과 이미지의 구분감이 더 살아나는 듯한 효과를 볼 수 있었으며, 너무 어두운 부분에서 부자연스러운 equalization이 줄어드는 것을 볼 수 있었다.

3. Image demosaicing

1) RGB to Bayer pattern

Bayer pattern을 만들기 위하여 이미지의 RGB성분을 각각 추출하였다. 이후 각 RGB성분에 mean pooling과정을 통해 이미지의 크기를 1/4크기로 축소하였다. 이들을 Bayer pattern에 맞는 패턴으로 다시 배열하여 Bayer pattern을 만들었다. 그 결과는 아래와 같다.



➢ 원본 RGB이미지(좌), Bayer pattern 이미지(우)

RGB이미지를 Bayer pattern으로 변환한 결과이다. 같은 픽셀이라도 R, G, B 간의 각 값의 편차가 크므로 이를 출력했을 때에는 바둑판이 겹쳐진 듯한 시각적 효과를 볼 수 있었다.

2) Demosaic Bayer patterned image using bilinear interpolation

Bilinear interpolation은 목표 지점의 픽셀 위치에 대해 주변 픽셀 위치의 값과 픽셀 값을 고려해서 interpolation하는 방법이다. 이를 구현하기 위해 Bayer pattern 이미지의 RGB를 따로 추출해 냈다. 이렇게 추출한 3개의 행렬에서 각각 빈 부분을 주변 3*3크기의 픽셀 값들과 비교하여 bilinear interpolation을 실시하였다.

➢ 원본 이미지(좌), Demosaic 한 이미지(우)

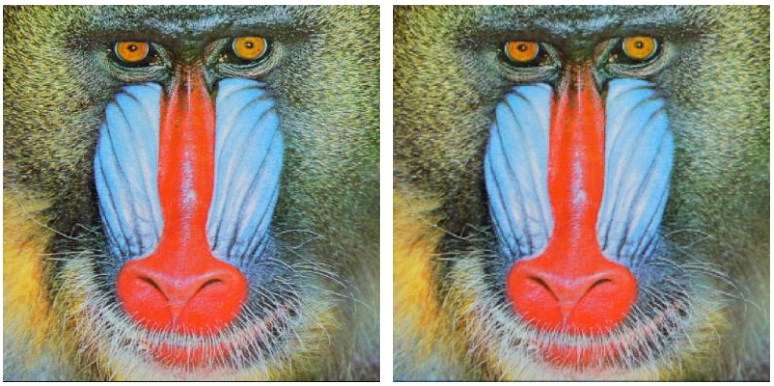
겉보기에는 원래 이미지와 큰 차이가 없어 보이지만, 가까이서 보면 blur현상이 조금 들어간 것을 볼 수 있었다. 이는 Bayer pattern을 만드는 과정에서 원본 대비 1/4수준으로 픽셀 값 손실이 일어났으며, 다시 interpolation하는 과정에서도 완벽하게 복구하지 못한 원인으로 추측된다.

3) PSNR compare

Input image	Lena	Babbon	Pepper
Bilinear interpolated	35.55	26.97	33.78
Library Used	35.71	27.39	33.38

3가지의 이미지로 PSNR을 비교한 결과, 이미지별로 PSNR값의 편차가 큰 것을 볼 수 있었다. 또한 Matlab 라이브러리인 'demosaic' 함수를 이용하여 직접 만든 demosaic함수와 비교하였다. 결과는 0.4이내로 편차를 보였는데 둘 다 비슷한 성능을 가지는 것을 확인할 수 있었다. 또한 Babbon 이미지의 경우가 가장 낮은 PSNR값을 기록하였는데, 이는 bayer pattern을 만드는 과정과 다시 demosaic 하는 과정에서 발생한 blur현상의 영향을 많이 받기 때문인 것으로 생각된다. 털과 같은 섬세한 표현이 많은데 이 부분에서 표현력이 떨어져 낮은 PSNR값을 가졌다.

4) Compare the statistics



➢ 원본 이미지(좌), Demosaic 한 이미지(우)

비교를 위해 PSNR이 가장 낮았던 'Babbon' 이미지를 통해 비교를 실시하였다.

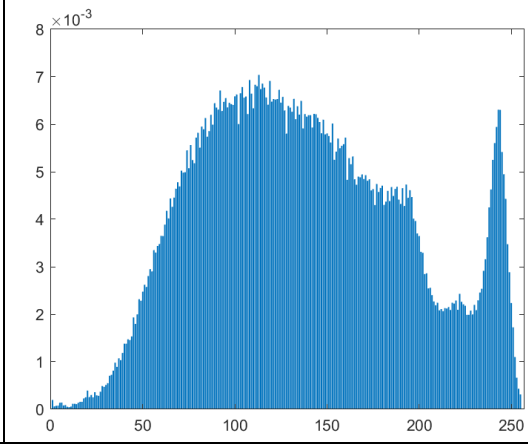
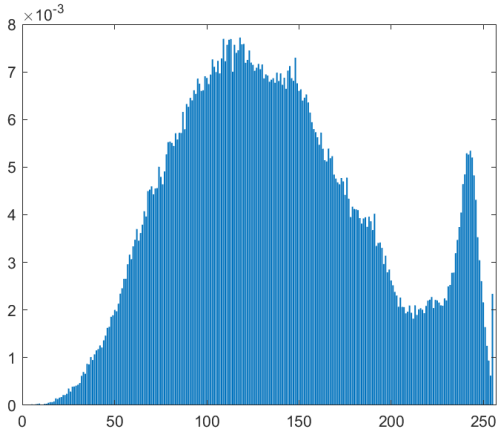
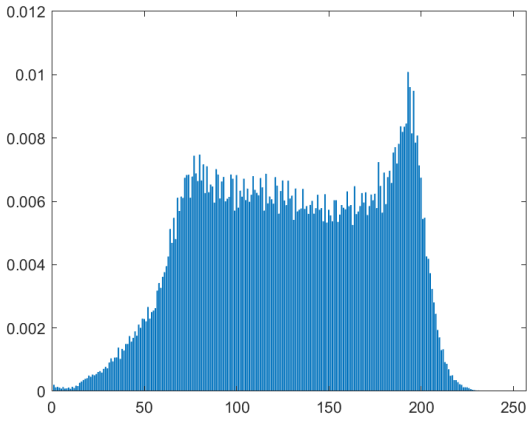
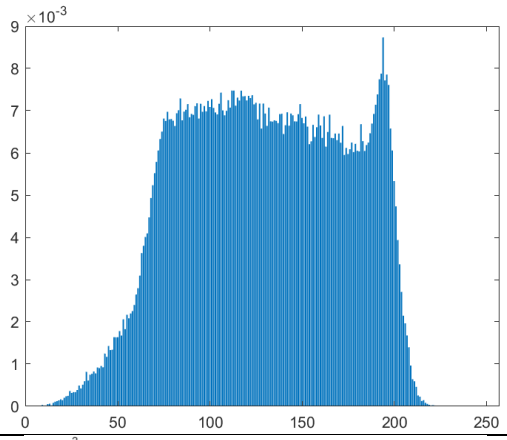
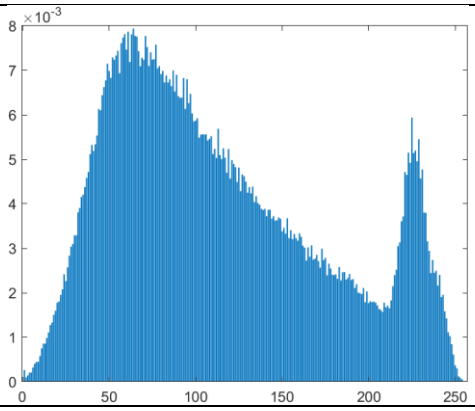
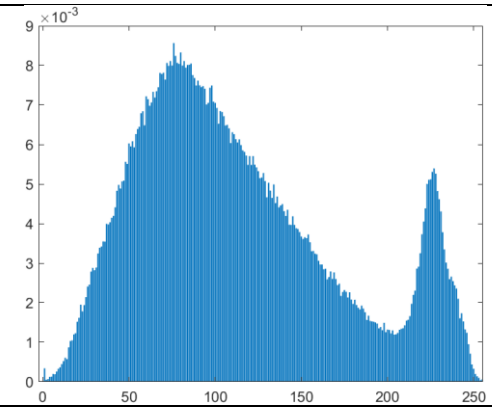
평균			
	R	G	B
Original	137.40	128.86	113.12
Bilinear interpolated	137.06	128.76	112.94

➢ 주변 값과 비교하여 상대적인 평균을 갖는 Bilinear interpolation의 경우 전체적인 평균값은 비슷하게 가지는 것을 확인할 수 있었다,

분산			
	R	G	B
Original	972	442	3017
Bilinear interpolated	1078	463	2939

단위: 1만

➢ 분산의 경우도 원래 값과 큰 차이를 보이지 않았다.

히스토그램		
	Original image	Interpolated image
Red		
Green		
Blue		

- 이미지 히스토그램을 통해 비교한 결과이다. Original image의 경우 보다 들쭉날쭉한 히스토그램을 갖고 있지만 interpolated image의 경우 들쭉날쭉한 정도가 줄어들었다. 이 또한 주변 값과 비교를 통해 빈 픽셀을 유추하는 Bilinear interpolation의 영향 때문으로 보인다.

5) Demosaicing using pixel value differences

Bilinear interpolation의 경우 근처의 값에 거리를 생각해 빈 픽셀을 채운다. 하지만 edge와 같이 픽셀 값 간의 차이가 큰 경우에도 고려하지 않고 interpolation 한다는 단점이 있다. 이는 'Babbon' 이미지와 같이 섬세한 표현이 필요한 이미지의 경우에 그 성능이 현저하게 떨어진다는 단점이 있다. 따라서 픽셀 값 간의 차이도 고려하는 demosaic을 구현하였다. 4개의 픽셀을 고려하는 상황에서, 이들의 평균을 구한다. 이후 최대값과 최솟값, 그리고 남은 2개의 값을 분리하여 평균과 비교한다. 만약 평균과 거리가 멀다면 낮은 가중치를 가지고 거리가 가깝다면 높은 가중치를 가지게 된다. 따라서 경계선 보존에 더욱 유리한 알고리즘을 만들 수 있었다.

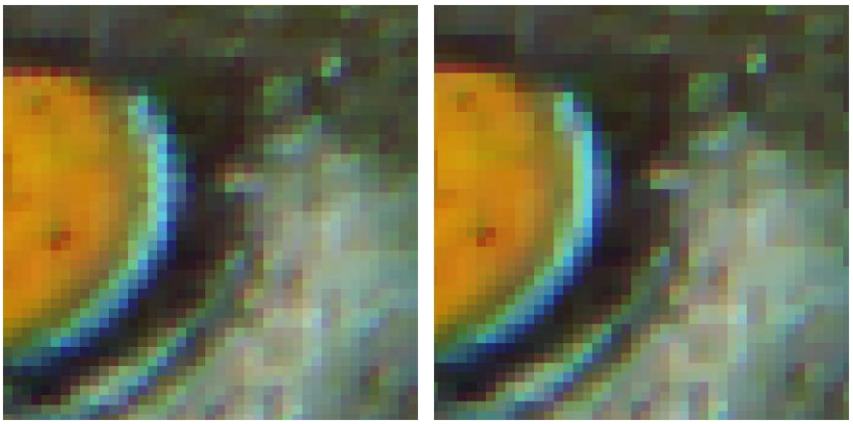
$$mi = \frac{\sigma}{(I_{mean} - I_{min})^2 + 1}$$

$$ma = \frac{\sigma}{(I_{mean} - I_{max})^2 + 1}$$

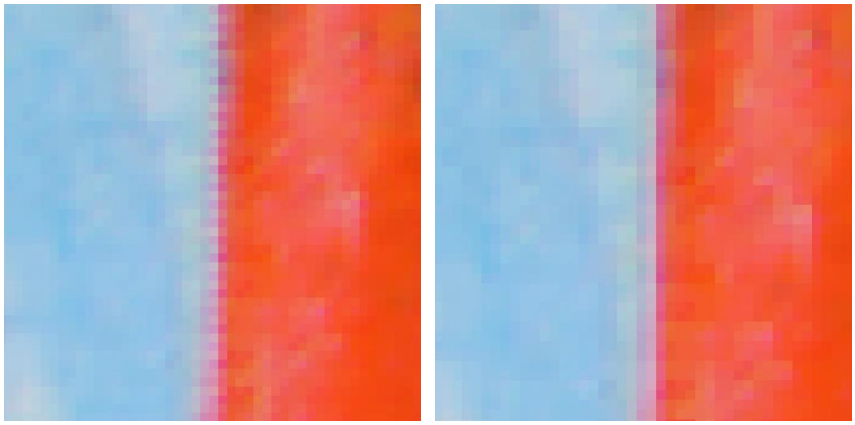
$$me = \frac{1}{(I_{mean} - (0.5I_{median1} + 0.5I_{median2}))^2 + 1}$$

$$Output = (\frac{1}{me + mi + ma})(me(0.5I_{median1} + 0.5I_{median2}) + mi(I_{min}) + ma(I_{max}))$$

알고리즘은 위와 같으며, 평균 값과 최소, 최대, 중간값2개가 같을 때 0으로 나누어지는 것을 방지하기 위하여 1을 분모에 더하였다. 평균값과 차이가 크면 더 작은 가중치를 가지고, 차이가 작으면 더 높은 가중치를 가진다. σ값은 1로 넣었으며, 값에 따른 차이가 클 것으로 예상되었으나, PSNR기준 0.002전후의 미미한 차이를 보여주었다. 본문에서는 $\sigma = 1$을 사용하여 진행하였다.



➢ Bilinear interpolation(좌), 개선된 interpolation(우)



➢ Bilinear interpolation(좌), 개선된 interpolation(우)

픽셀 값 차이가 극명한 부분을 확대 캡처하여 확인해본 결과이다. 픽셀 값을 고려하여 interpolation을 실시하니 보다 자연스러운 결과물을 얻을 수 있었다. 경계선 부분에서 잘못된 픽셀 값 고려로 지그재그 패턴의 부정확한 표현이 이루어졌지만, 픽셀 값 간의 차이를 고려한 interpolation의 경우 그 표현력이 더욱 정확함을 확인할 수 있었다.

PSNR			
Input image	Lena	Babbon	Pepper
Bilinear interpolation	35.55	26.97	33.78
개선된 interpolation	35.62	27.04	33.78

PSNR 의 경우 매우 근소한 차이가 있었다. Pepper 를 제외한 나머지 이미지에서는 약간의 개선이 있었다. Pixel value 차이가 크지 않은 부분에서는 dramatic 한 차이를 보여주지 못한 까닭으로 생각된다.