ITE4052 Computer Vision

Programming Assignment2

2020095178 최윤선

0. Introduction

- File
  - main file: processing.m
  - function file: contrastStretching.m, gammaCorrection.m (3. Contrast Stretching 에서 사용)
- Original

1. Mean Filtering

- Implement Gaussian noise

  평균이 0, 표준편차가 0.1 인 Gaussian noisy image 생성

```
noisy_img = img + 0.1*randn(size(img));
```

➢ Mean Filtering & PSNR

Kernel size 가 3, 5, 7 일 때, 각각 mean filtered image 를 생성하고 PSNR 값을 계산한다.

```
% Mean Filtering
kernel_sizes = [3, 5, 7];
filtered_imgs = {1:numel(kernel_sizes)};
for i = 1:numel(kernel_sizes)
    filter = ones(kernel_sizes(i))/(kernel_sizes(i)^2);
    filtered_imgs{i} = convn(noisy_img, filter, 'same');
end
```

Mean filter $= \frac{1}{N^2}\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{N\times N}$ 이므로, 이를 이용하여 noisy image 와 filter 를 convolution 한다.

```
% PSNR
h = size(noisy_img, 1); w = size(noisy_img, 2);
my_psnr = {1:numel(kernel_sizes)};
noisy_img = rgb2gray(noisy_img);
for j = 1:numel(kernel_sizes)
    filtered_imgs{j} = rgb2gray(filtered_imgs{j});
    mse = sum(sum((noisy_img-filtered_imgs{j}).^2))/(w*h);
    my_psnr{j} = 10*log10((0-256)^2/mse);
    fprintf('Kernel Size: %d\t', kernel_sizes(j));
    fprintf('PSNR: %f\n', my_psnr{j});
end
```

$\text{PSNR} = 10\log_{10}\frac{(peak-to-peak\ value)^2}{\sigma_e^2} = 10\log_{10}\frac{(0-256)^2}{var(v-\hat{v})}$ 공식을 사용하여 kernel size 별로 noisy image 와 filter 를 적용한 image 사이의 PSNR 값을 구한다.

Mean filtered image 와 PSNR 결과값은 다음과 같다.

Original Image

Noisy Image

Kernel Size = 3

Kernel Size = 5

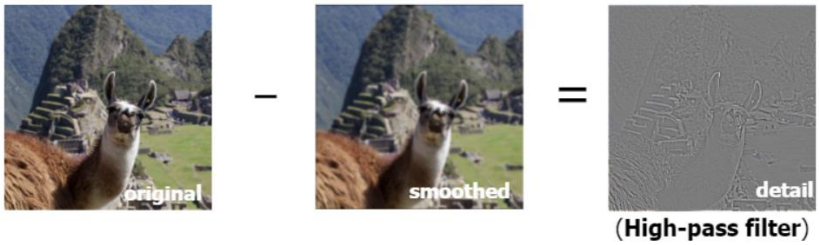
Kernel Size = 7

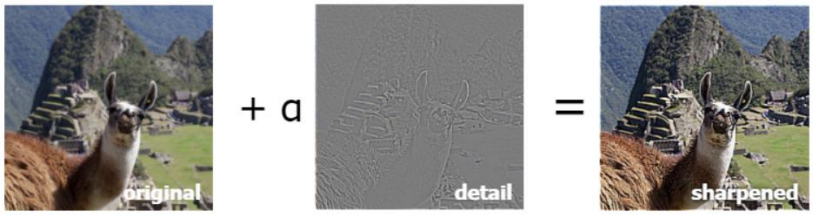
```
Kernel Size: 3  PSNR: 71.500804
Kernel Size: 5  PSNR: 70.694662
Kernel Size: 7  PSNR: 70.261150
```

PSNR 값이 클수록 손실 양이 적다는 뜻이다. 따라서 kernel size 를 3 일 때 PSNR 값이 가장 크므로, 3x3 filter 를 적용했을 때, noisy image 에 대한 filtering 이 적은 pixel 값 손실로 가장 잘 되었다는 것을 알 수 있다.

2. Unsharp Masking

- Sharpening





$$F + \alpha(F - \underbrace{F * H}_{\text{blurred image}}) = (1+\alpha)F - \alpha(F * H) = F * (\underbrace{[1+\alpha]e}_{\text{(Identity kernel)}} - \alpha H)$$

image

위의 공식을 사용하여 kernel size 별로 sharpened image 를 생성한다.

```
for i = 1:numel(kernel_sizes)
    % Smoothing Filter (Mean filtering)
    filter = ones(kernel_sizes(i))/(kernel_sizes(i)^2);
    smoothed_imgs{i} = convn(img, filter, 'same');
    % Detail (High-pass Filter)
    detail_imgs{i} = img - smoothed_imgs{i};
    % Sharpening
    sharpened_imgs{i} = img + 5*detail_imgs{i};
end
```

이때, smoothing 은 mean filtering 을 사용하였고, alpha 값은 임의로 5 로 지정하였다.

Original Image

Kernel Size = 3

Kernel Size = 5

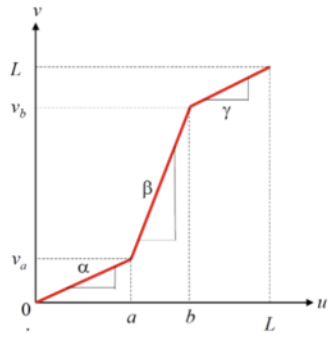
Kernel Size = 7

3. Contrast Stretching

Contrast Stretching 과 Gamma Correction 은 함수만 모아 둔 파일을 따로 만들어 실행하였다.

➢ Contrast Stretching

$$v=\begin{cases}\alpha u, & 0\le u<a\\ \beta(u-a)+v_a & a\le u<b\\ \gamma(u-b)+v_b & b\le u<L\end{cases}$$



위의 공식을 참고하였고, 임의의 parameter 를 설정하여 Contrast Stretching 함수를 작성하였다.

```
function v = contrastStretching(u)
a = 0.4; b = 0.6;
alpha = 0.5;
beta = 0.5;
gamma = 1.5;
v_a = alpha*a;
v_b = beta*(b-a)+v_a;
v = zeros(size(u));
for i = 1:numel(u)
    if u(i) >= 0 && u(i) < a
        v(i) = alpha*u(i);
    elseif u(i) >= a && u(i) < b
        v(i) = beta*(u(i)-a)+v_a;
    elseif u(i) >= b && u(i) < 1
        v(i) = gamma*(u(i)-b)+v_b;
    end
end
end
```

<Parameters>

- $a = 0.4$
- $b = 0.6$
- $\alpha = 0.5$
- $\beta = 0.5$
- $\gamma = 1.5$

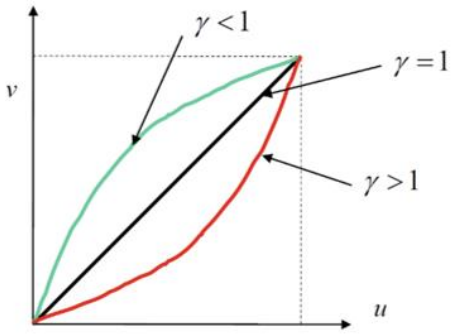
<실행 코드>

```
cs_img = contrastStretching(img);
```

결과 이미지는 다음과 같다.

➢ Gamma Correction

$$v = cu^{\gamma}$$



위의 공식을 참고하였고, 임의의 parameter 를 설정하여 Gamma Correction 함수를 작성하였다.

```
function v = gammaCorrection(u, gamma)
c = 1;
if gamma < 1 || gamma > 1
    v = c*(u.^gamma);
end
end
```

<Parameters>

· $c = 1$

<실행 코드>

```
gc_img = gammaCorrection(img, 0.3);
```

$\gamma = 0.3$일 때, 결과 이미지는 다음과 같다.

4. Histogram Equalization

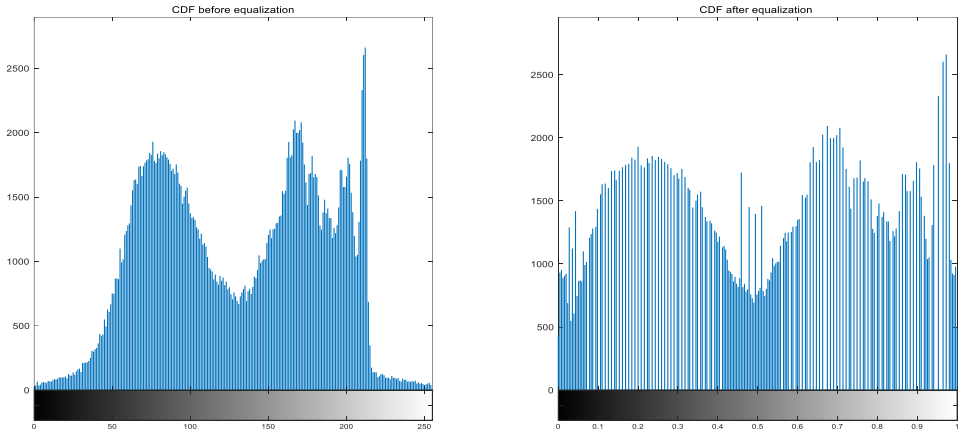
Gray scale image 로 histogram equalization 을 진행하였다

- Code

```
cdf = cumsum(imhist(gray_img));
cdf = cdf/numel(gray_img);
hist_img = reshape(cdf(gray_img(:)+1), size(gray_img));
```

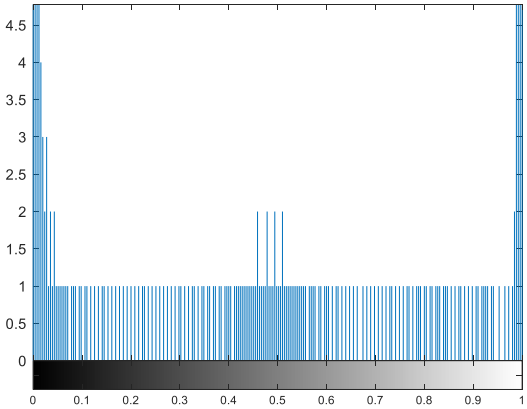
이미지의 CDF 를 cumsum()을 사용하여 구한 후, 이미지의 크기로 나누어 CDF 값을 0 과 1 사이로 정규화 해준다. 구한 CDF 로 다시 이미지를 생성한다. Histogram equalization 을 하기 전과 후의 히스토그램과 이미지는 다음과 같다.

- Histogram



- Image

- CDF after Histogram Equalization



5. Image Upsampling

➢ Down-Sample (×0.25)

imresize() 함수를 사용하여 원래 이미지를 1/4 크기로 축소하였다.

```
down_img = imresize(img, 0.25);
```

➢ Up-Sample (×4)

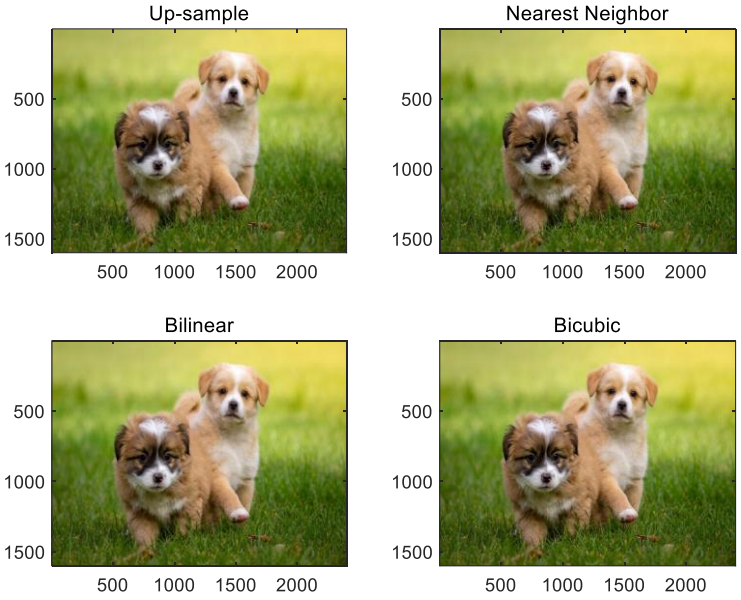
기본 upsampling 은 Scaling 행렬을 사용하였고, nearest neighbor, bilinear, bicubic interpolation 은 imresize() 함수를 활용하였다.

```
% Up-sample of original image
scaling = projtform2d([4,0,0; ...
                       0,4,0; ...
                       0,0,1]);
up_img = imwarp(img, scaling);

% Nearest Neighbor
nn_img = imresize(img, 4, 'nearest');

% Bilinear
bil_img = imresize(img, 4, 'bilinear');

% Bicubig
bic_img = imresize(img, 4, 'bicubic');
```



➢ PSNR

Gray Scale 로 변환 후 original image 와의 PSNR 값을 비교하였다.

```
[h, w] = size(up_img);

nn_mse = sum(sum((up_img-nn_img).^2))/(w*h);
nn_psnr = 10*log10((0-256)^2/nn_mse);

bil_mse = sum(sum((up_img-bil_img).^2))/(w*h);
bil_psnr = 10*log10((0-256)^2/bil_mse);

bic_mse = sum(sum((up_img-bic_img).^2))/(w*h);
bic_psnr = 10*log10((0-256)^2/bic_mse);
```

```
PSNR (Nearest Neighbor): 76.661316
PSNR (Bilinear): 77.254843
PSNR (Bicubic): 77.187715
```

Bilinear interpolation 을 적용하여 up-sampling 을 하였을 때, PSNR 값이 가장 크므로, 가장 적은 pixel 의 손실로 up-sampling 된 것을 확인할 수 있다.