

# Image Processing

## 실습 2.

2019. 03. 18

김 혁 진

gurwls9628@naver.com

# 실습 소개

- 과목 홈페이지

- 충남대학교 사이버 캠퍼스 ( <http://e-learn.cnu.ac.kr> )

- TA 연락처

- 김혁진
- 공대 5호관 627호 정보보호연구실
- [gurwls9628@naver.com](mailto:gurwls9628@naver.com)
  - [IP] 를 메일 제목에 붙여주세요
  - 과제 질문은 메일로만 해주세요

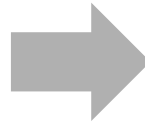
# 개요

- 1주차 과제 리뷰
  - Rgb2gray
- 실습
  - Quantization
  - Nearest-neighbor Interpolation
  - Histogram
- 구현
  - Bilinear Interpolation

# 1주차 과제 리뷰

- **rgb2gray 구현**

- RGB 영상의 행렬 값을 인자로 받아 Gray Scale의 영상의 행렬 값을 return하는 함수 구현



# 1주차 과제 리뷰

- **RGB image**

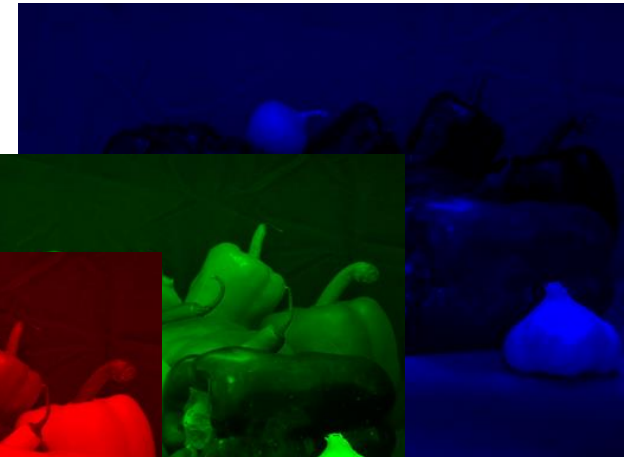
- Red, Green, Blue 의 3 색을 조합하여 이미지를 생성
- RGB 이미지의 행렬은 X(세로) x Y(가로) x 3(색) 으로 표현 가능
- 초록, 빨강, 파랑 순으로 빛을 민감하게 받아들임



X



Y



3

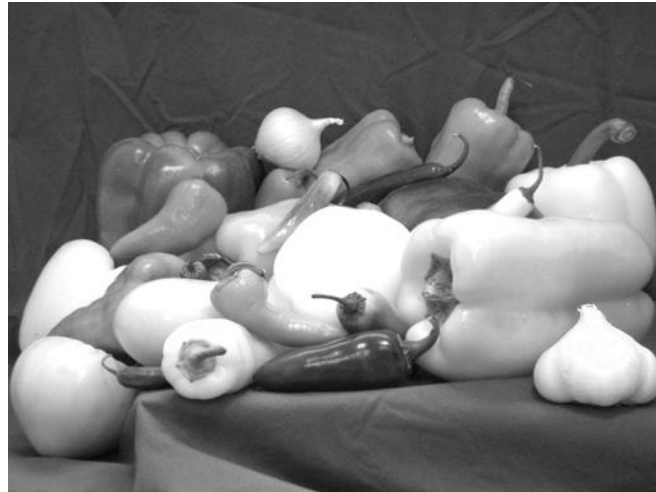
# 1주차 과제 리뷰

- **Gray Scale Image**

- 1가지 색상으로 표현되므로 3색을 구분할 필요 없음
- 0~255사이의 값으로 표현
- Gray Scale 이미지의 행렬은 X(세로) x Y(가로)로 표현 가능



단순 3분의 1



한가지 색상만 사용



3가지 색을 적절히 사용

# 1주차 과제 리뷰

- 이유?



한가지 색상만으로는  
모든 색을 표현하기 어려움



각 색상의 반응하는 정도가  
다름

# 1주차 과제 리뷰

- **rgb(:, :, 1)**
  - 1:2, 는 1-2의 값을 모두 가져오는 것이지만  
: 만 있을 경우에는 1-N번째까지 모두 가져온다.
  - :를 이용하지 않고 for문으로 반복해도 상관 없음
- **0.3, 0.6, 0.1은 정확한 수치는 크게 신경쓰지 않음**
  - Matlab은 0.2989, 0.5870, 0.1140의 수치를 R, G, B에 곱함
  - 결과값이 더 밝아지거나 어두워지지 않게 합이 1이어야 함

```
% RGB 영상의 행렬 값을 GrayScale으로 변경하는 함수
% rgb : RGB image, dimension (X, Y, 3)
% gray : Gray Image, dimension (X, Y)
function gray = my_rgb2gray(rgb)
gray = 0.3*rgb(:, :, 1)+0.6*rgb(:, :, 2)+0.1*rgb(:, :, 3);
% 채점 기준으로 정확한 수치는 요구하지 않지만 3 색상을 모두 사용해야 함
end
```

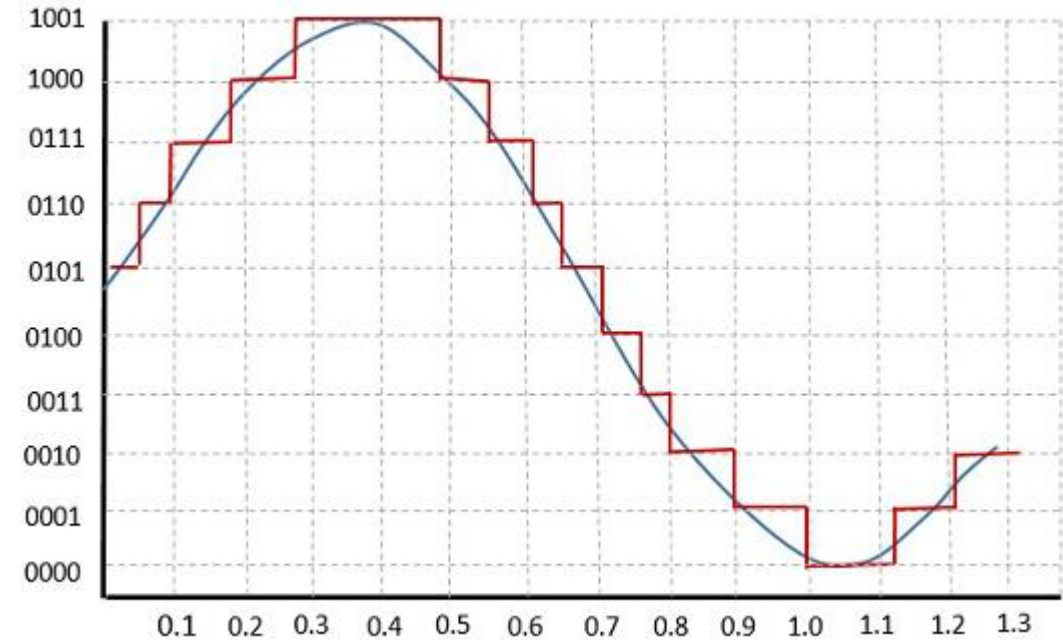




# Quantization

- 양자화

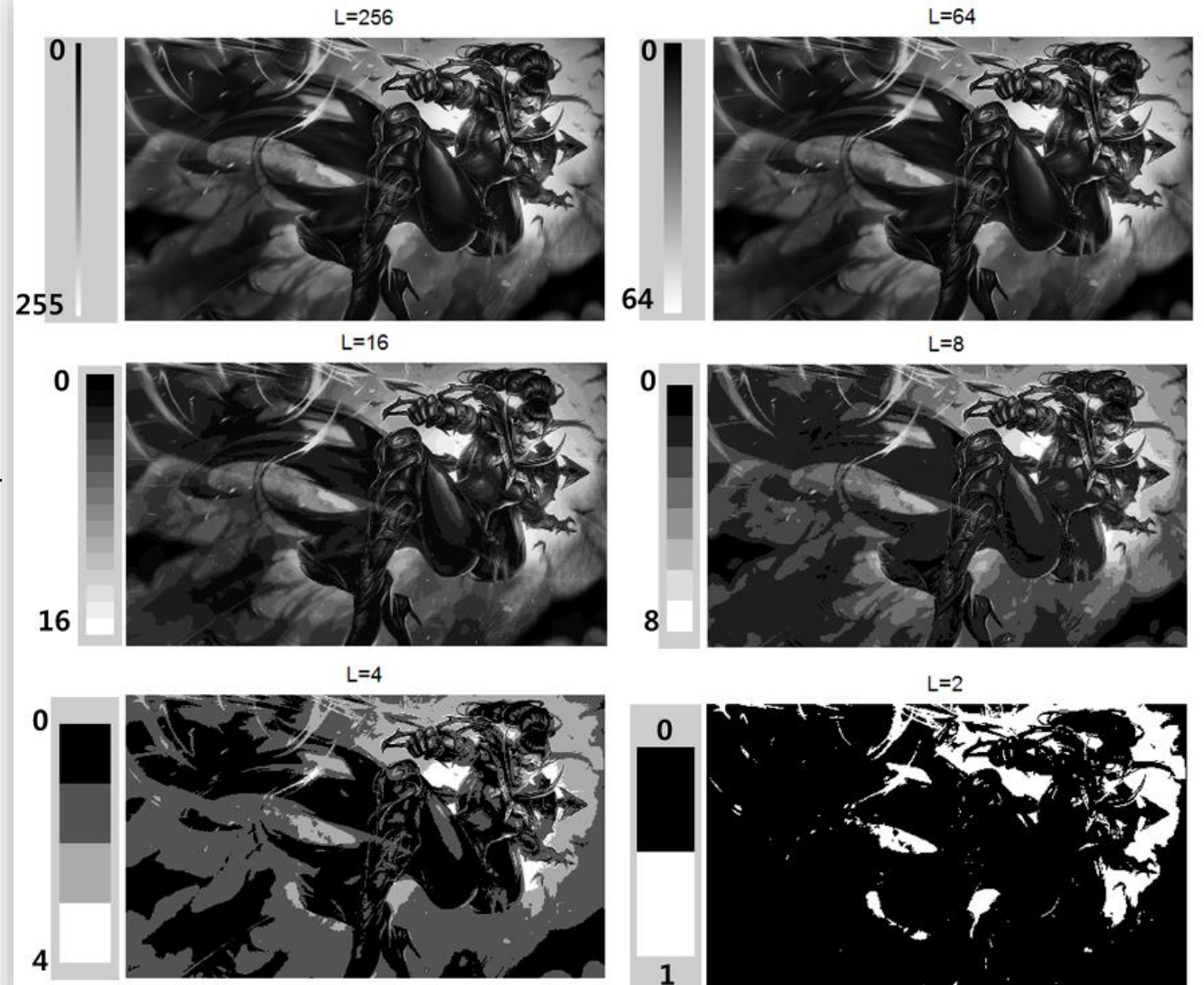
- Analogue로 표현된 값을 디지털로 표현함
- 세밀하게 표현할 수록 (양자화 Level이 높을수록) 기존 영상과 비슷하게 나오지만 더 많은 용량을 필요로 함



# Quantization

- 양자화

- Analogue로 표현된 값을 디지털로 표현함
- 세밀하게 표현할 수록 (양자화 Level이 높을수록) 기존 영상과 비슷하게 나오지만 더 많은 용량을 필요로 함




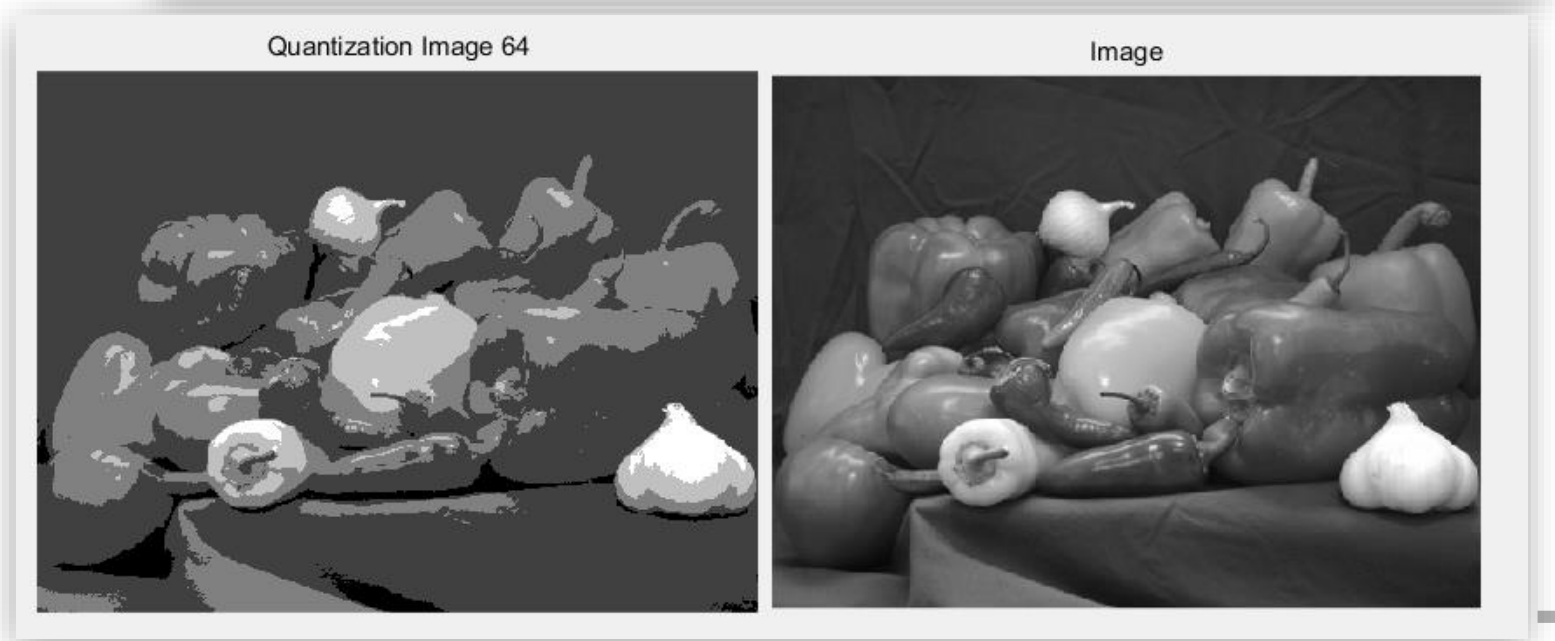
# Quantization

- 양자화의 레벨을 낮추면?

- 기존의 색상을 더 적은 수의 색으로 표현하여 더 적은 용량으로 표현
- 하지만 색상 구분이 어려워 짐

$$f = \text{floor}(\text{double}(x)/64)$$
$$q = \text{floor}(f * 64)$$

 gray	2019-03-14 오후...	PNG 파일	88KB
 quan	2019-03-14 오후...	PNG 파일	10KB



# Quantization

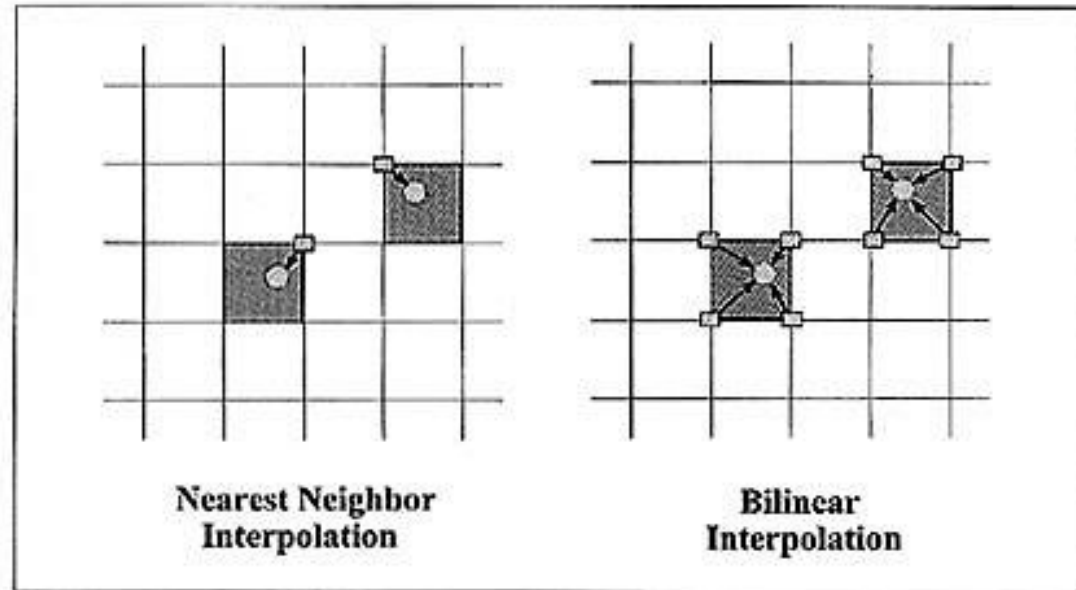
- 양자화의 레벨을 낮추는 것은?
  - 기존의 색상을 0~255로 표현했다면  
64로 색상의 값을 나누고 내림을 해  
레벨을 낮출 경우 0~3의 값으로 색을 표현
  - 0~63의 값이 모두 0으로 표현됨

```
img = imread('peppers.png');  
gray = rgb2gray(img);  
quan = floor(gray/64)*64;  
  
figure  
subplot(1, 1000, 1:499);  
imshow(quan);  
title('Quantization Image 64')  
  
subplot(1, 1000, 510:999);  
imshow(gray);  
title('Image')  
  
imwrite(gray, 'gray.png');  
imwrite(quan, 'quan.png');
```

# Nearest-Neighbor Interpolation

- **최단입점 보간법**

- 이미지의 크기를 변경하거나 회전할 때  
가장 가까운 지점의 색을 이용해 색을 추론
- 각 픽셀의 위치는 정수 값
- 만약 참조하려는 픽셀의 위치가  
정수가 아닌 실수일 경우  
어떻게 값을 가져오는지에 대해



# Nearest-Neighbor Interpolation

- **최단입점 보간법**

- 이미지의 가장 가까운 점의 값을 이용해 값을 추론

37	60
40	51



37	?	?	?	60	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
40	?	?	?	51	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?

# Nearest-Neighbor Interpolation

- 최단입점 보간법

- 이미지의 가장 가까운 점의 값을 이용해 값을 추론

37	60
40	51



37	?	?	?	60	?	?	?
?	37	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
40	?	?	?	51	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?

# Nearest-Neighbor Interpolation

- 최단입점 보간법

1	2	12	47	58	78	95	61
21	15	67	16	78	45	15	11
10	22	31	34	44	45	66	78
99	87	10	0	20	30	1	11
15	13	16	17	18	19	17	21
22	23	22	20	1	54	6	12
10	7	5	23	51	37	7	89
1	0	10	90	97	99	1	50



?	?
?	?



# Nearest-Neighbor Interpolation

## • 최단입점 보간법

1	2	12	47	58	78	95	61
21	? 15	67	16 ?	78	45	13 ?	11
10	22	31	34	44	45	66	78
99	87	10	0	20	30	1	11
15	? 13	16	17 ?	18	19	17 ?	21
22	23	22	20	1	54	6	12
10	? 7	5	23 ?	51	37	7 ?	89
1	0	10	90	97	99	1	50

정확하게 한 픽셀과 대치되지 않으면?



?	?	?
?	?	?
?	?	?

# Nearest-Neighbor Interpolation

## • 최단입점 보간법

89	2	12	47	58	78	95	61
21	15	67	16	78	45	15	11
10	22	31	34	44	45	66	78
99	87	10	0	20	30	0	11
15	13	16	17	18	19	17	21
22	23	22	20	1	54	6	12
10	7	5	23	51	37	7	89
4	0	10	90	97	99	1	50

가장 가까운 픽셀의 값을 사용

?	?	?
?	?	?
?	?	?

$$(2, 2) \times 8/3 = (16/3, 16/3)$$

# Nearest-Neighbor Interpolation

## • 최단입점 보간법

- padding으로 이미지 처리를 용이하게 함
- 반올림을 통해서 픽셀 위치를 정수 값으로 만들어 값을 받음

```
function pad_img = padding(img)
[x, y] = size(img);
pad_img = zeros(x+2, y+2);
pad_img(2:x+1, 2:y+1) = img;

% 꼭지점 부분 패딩
pad_img(1,1) = img(1, 1);
pad_img(1,y+2) = img(1, y);
pad_img(x+2,1) = img(x, 1);
pad_img(x+2,y+2) = img(x, y);

% 모서리 부분 패딩
pad_img(2:x+1, 1) = img(1:x, 1);
pad_img(2:x+1, y+2) = img(1:x, y);
pad_img(1, 2:y+1) = img(1, 1:y);
pad_img(x+2, 2:y+1) = img(x, 1:y);
pad_img = uint8(pad_img);
end
```

```
function re_img = my_imresize(img, row, col)
re_img = zeros(row, col);
[x, y] = size(img);

img = padding(img);
r_p = x/row;
c_p = y/col;

for i = 1:row
    for j = 1:col
        % 반올림해서 가까운 픽셀의 값을 가져옴
        x = round(i*r_p)+1;
        y = round(j*c_p)+1;

        re_img(i, j) = img(x, y);
    end
end
re_img = uint8(re_img);
end
```

# Nearest-Neighbor Interpolation

- 최단입점 보간법

- 단순히 값을 가져오기 때문에 계단 현상이 심하게 일어남

```
>> re = my_imresize(img, 50, 50);  
>> imshow(re)  
>> re = my_imresize(re, 500, 500);  
>> imshow(re)
```

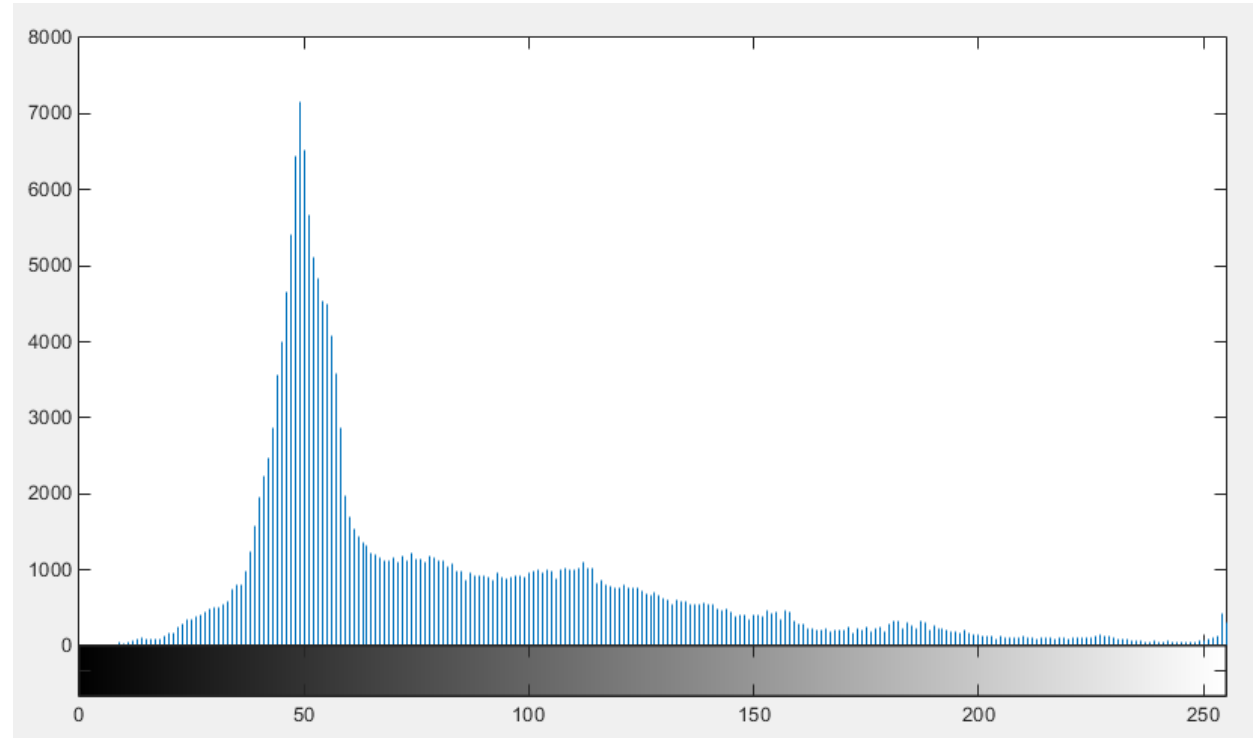


# Histogram

- 히스토그램

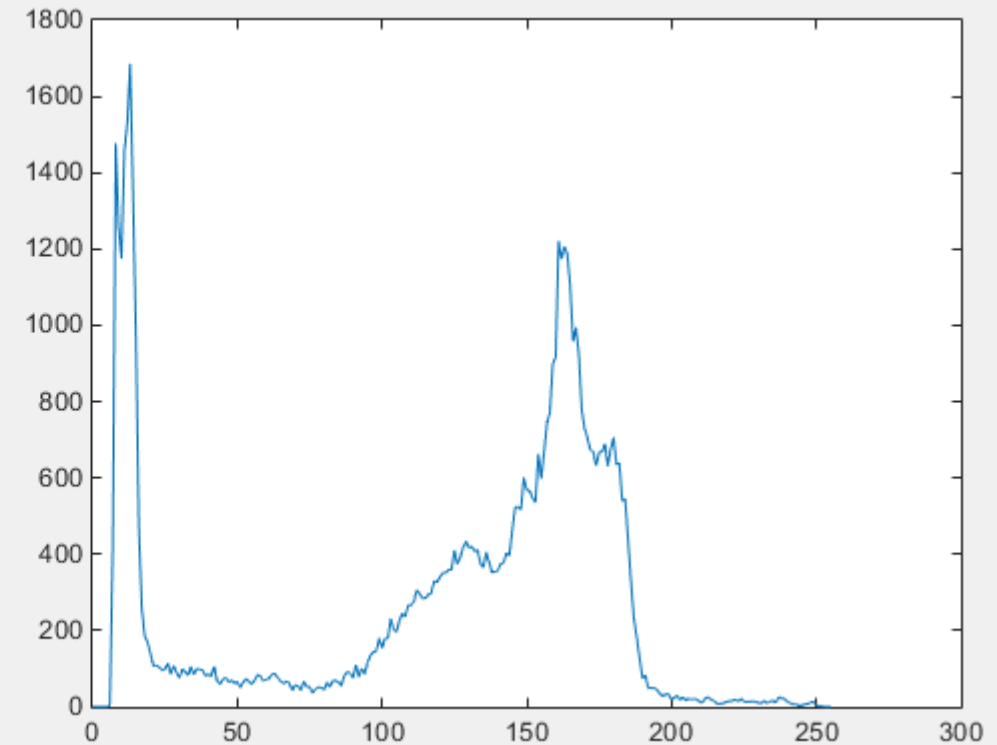
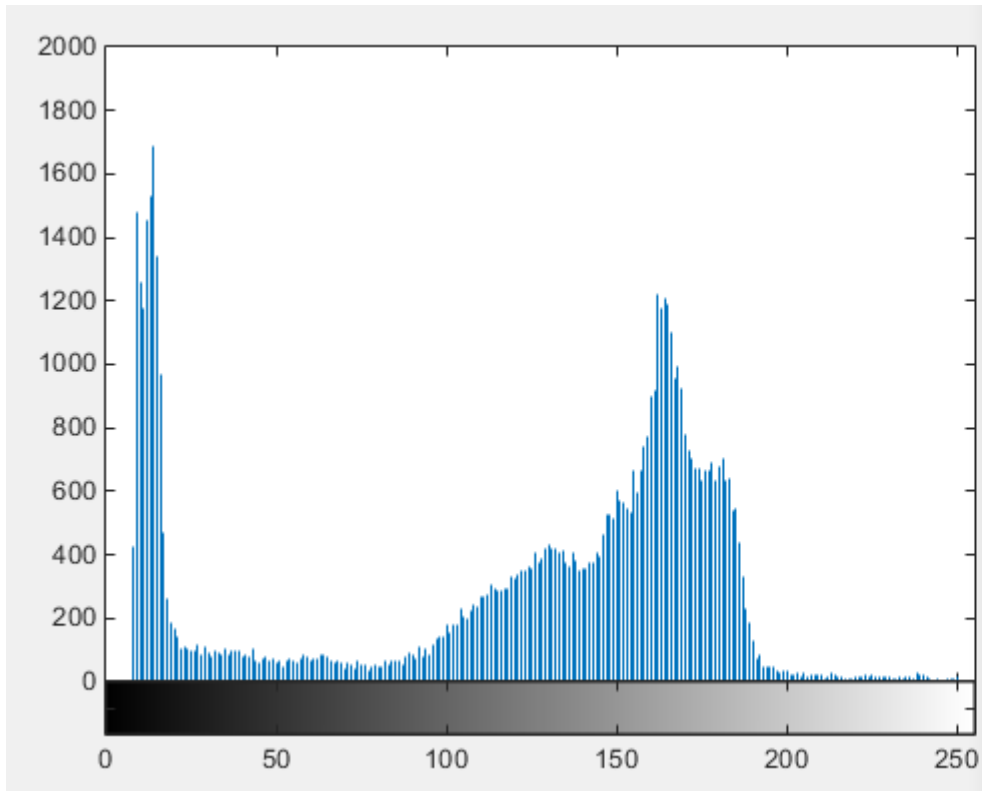
- PDF (Probability Distribution Function)
- 이미지에 어느 값이 얼마나 있는지  
알 수 있음

```
>> imhist(gray)  
>> ylim([0 8000])
```



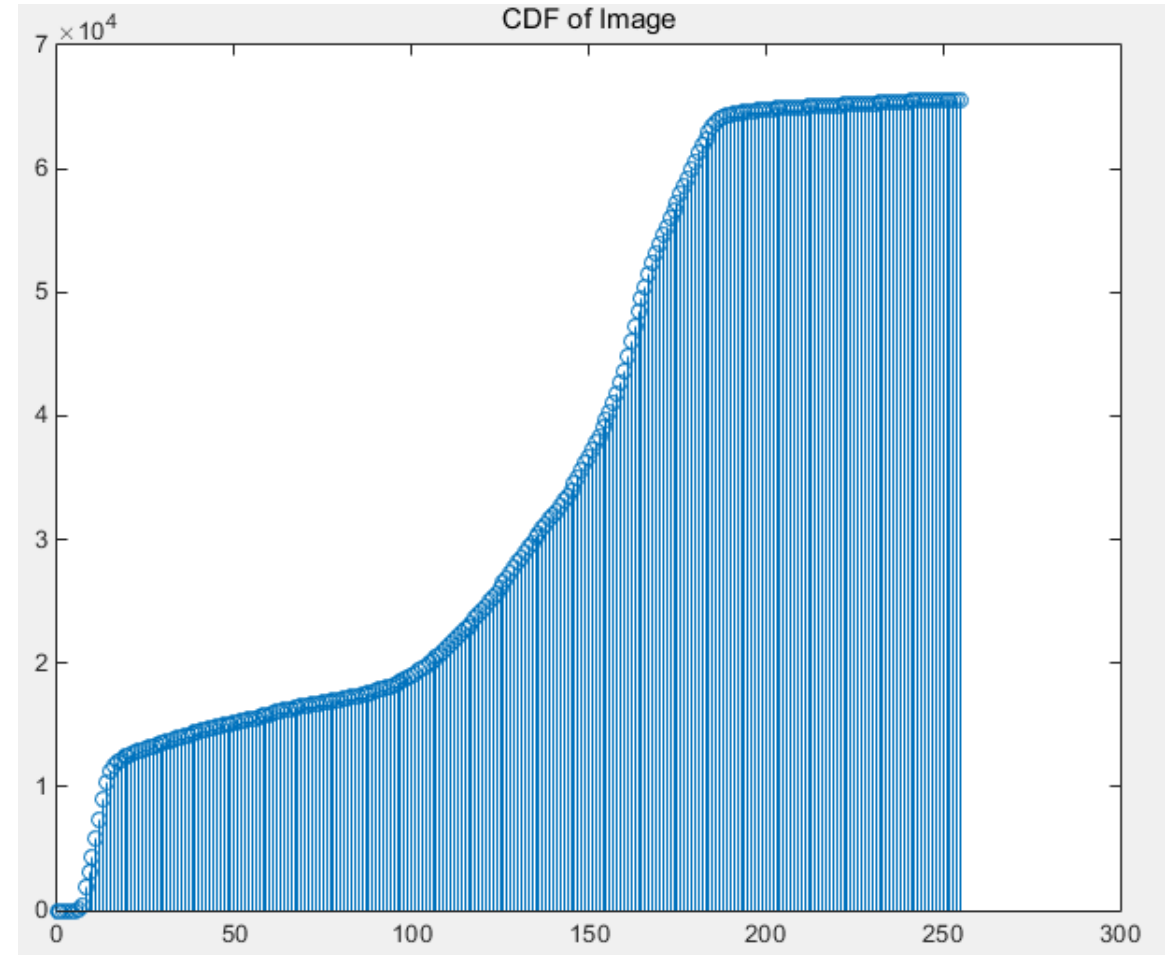
# Histogram

- 히스토그램
  - PDF (Probability Distribution Function)



# Histogram

- 히스토그램
  - CDF (Cumulative Distribution Function)
  - PDF의 적분
  - Discrete한 값이기때문에 적분을 구하기 쉬움



# Histogram

- 히스토그램

- 모든 픽셀의 값을 모두 셈
- CDF는 PDF 값을 받아 이전 값을 다음 CDF값에서 더해 받음

```
function my_hist(img)
pdf = zeros(1, 256);
[x, y] = size(img);
% PDF
for i = 1:x
    for j = 1:y
        pdf(img(i,j)) = pdf(img(i,j)) + 1;
    end
end
% CDF
cdf = pdf;
for i = 2:256
    cdf(i) = cdf(i)+cdf(i-1);
end
```

% Plotting

figure

subplot(1, 1000, 1:470);

stem((1:256)-1, pdf)

%plot((1:256)-1, pdf)

title('PDF of Image');

subplot(1, 1000, 530: 1000);

stem((1:256)-1, cdf)

%plot((1:256)-1, cdf)

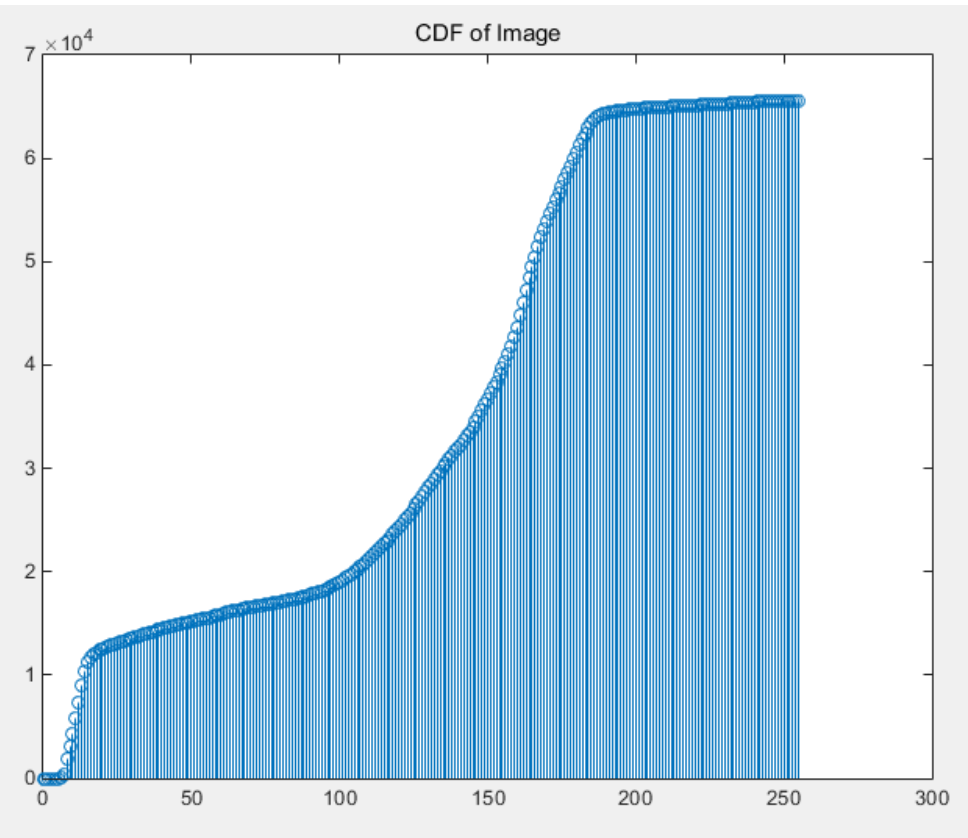
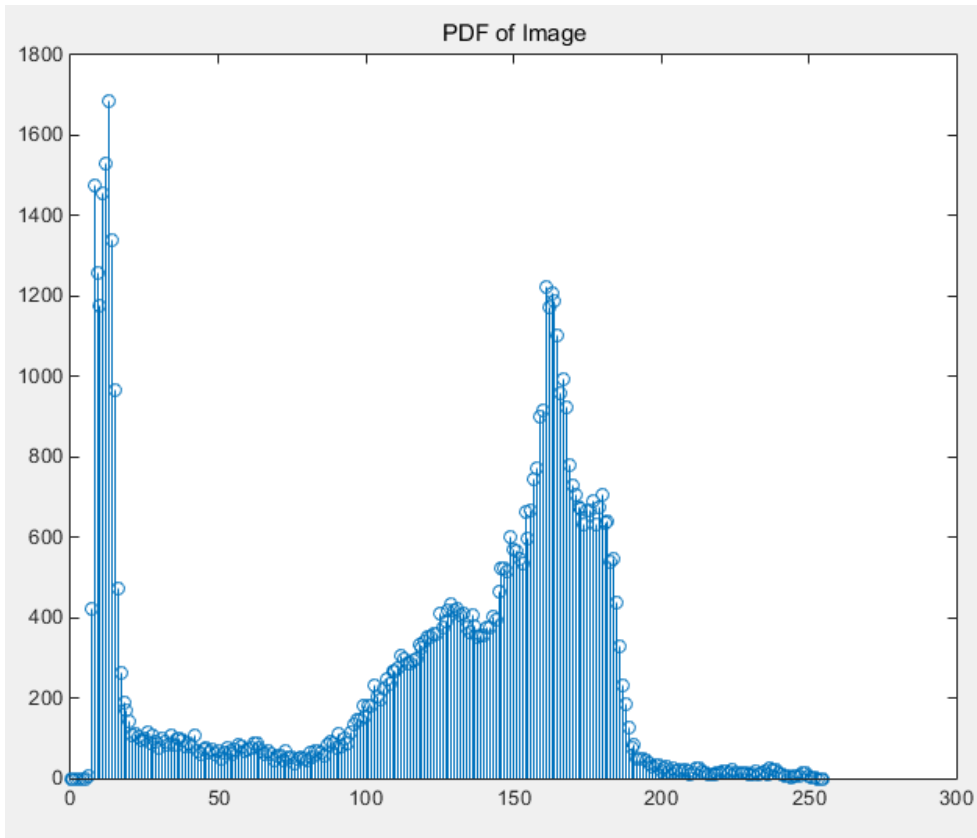
title('CDF of Image');

-end



# Histogram

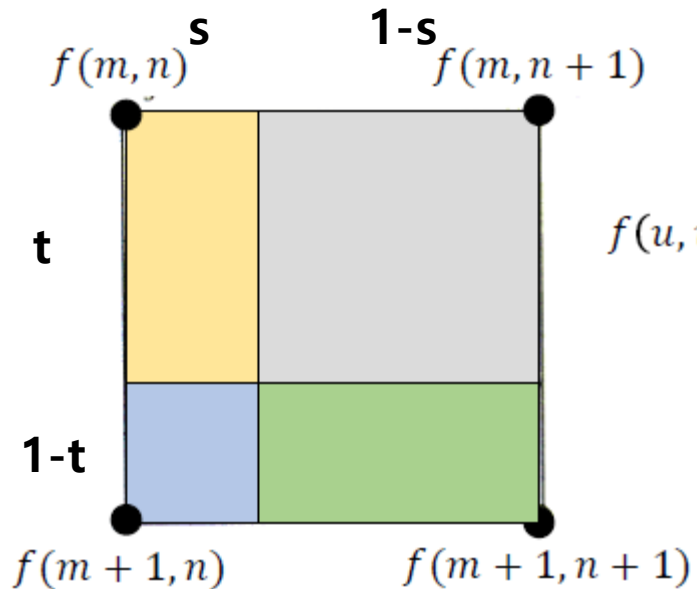
- 히스토그램



# Bilinear Interpolation

- 선형 보간법

- 계단 현상이 일어나는 것을 줄이기 위해  
이미지를 확대하거나 줄일 때 처리를 다르게 함



$$\begin{aligned} f(u, v) = & (1 - s)(1 - t) \cdot f(m, n) \\ & + s(1 - t) \cdot f(m, n + 1) \\ & + (1 - s)t \cdot f(m + 1, n) \\ & + st \cdot f(m + 1, n + 1) \end{aligned}$$

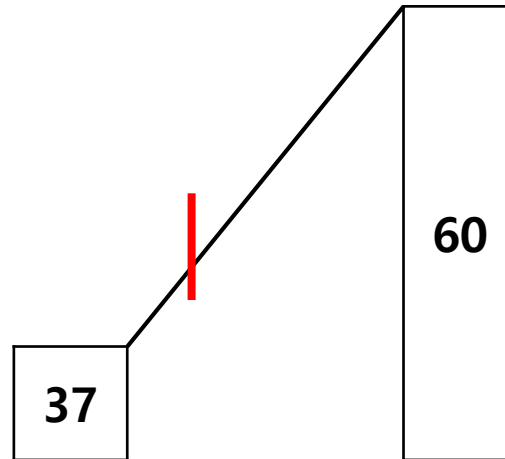


# Bilinear Interpolation

- 기본적인 가정

- 값이 변할 때에 선형적인 변화가 있을 것
- 그 값을 예상해서 값을 채워 넣음
- 확대된이미지의 위치가 어디에 해당하는지 예상해 값을 추측

37	60
40	51



# Bilinear Interpolation

- 구현해서 .m파일 제출
  - Bilinear 방식으로 이미지 resize하는 함수를 작성해야 함
  - 이러닝에 올린 .m파일을 토대로 작성

```
function re_img = my_bilinear(img, row, col)
% TODO: Fill it
|
end
```

# 과제

- 보고서
  - 이러닝에 올라온 보고서 양식을 보고 작성
  - **파일 이름:**
    - [IP]20xxxxxxx\_이름\_2주차\_과제.pdf

# 과제

- **제출 기한**

- 3월 24일 23시 59분까지

- **추가 제출 기한**

- 3월 25일 0시 10분까지 (최대 점수 9점, 과제 총점 계산 후 -1점)
- 3월 25일 0시 20분까지 (최대 점수 8점, 과제 총점 계산 후 -2점)
- 3월 25일 0시 30분까지 (최대 점수 7점, 과제 총점 계산 후 -3점)
- 3월 25일 0시 40분까지 (최대 점수 6점, 과제 총점 계산 후 -4점)
- 3월 25일 0시 50분까지 (최대 점수 5점, 과제 총점 계산 후 -5점)
- 3월 25일 1시 00분까지 (최대 점수 4점, 과제 총점 계산 후 -6점)
- 3월 31일 23시 59분까지 (최대 점수 3점, 과제 총점 계산 후 -7점)

## 과제 요약

### 1. imresize 내장 함수를 사용하지 않고 이미지 resize 함수를 구현

- 이러닝에 올라온 my\_bilinear.m 파일의 함수를 채운 후 제출

#### • 제출 파일

- my\_bilinear.m 파일
- .pdf 보고서 파일
- 위의 파일을 압축해서 [IP]20xxxxxx\_이름\_2주차\_과제.zip로 제출