Digital
Image
Process
Report #3

imresize
&
bwlabel

로봇기계공학과
21721160
최병희

Lab 2-2 MATLAB: imresize

- 문제 - imresize 커스텀 함수 구현

이번 Report의 목표는 Matlab 이미지 처리 함수 중 하나인 imresize의 nearest Neighbor(최근접화소) 방법과 Bilinearest interpolation(이중선형 보간법) 방법을 직접 함수로 구현해 보는 것입니다. nearest Neighbor 방법은 이미지를 resize할 때 새로 할당된 픽셀을 원래 이미지 픽셀의 가장 가까운 픽셀로 보간하는 방법입니다. 새로 할당된 이미지 size에서 필요한 픽셀의 좌표를 배율로 나누면 resize 하기 전 이미지의 어느 픽셀에 가까이 있는지 대강 예상할 수 있습니다. 하지만 픽셀 위치에 1.5번째 픽셀이란 말은 없으므로, 소수점이 나올 경우 이를 반올림하여 이전 이미지에서의 픽셀 위치를 결정합니다. Bilinearest interpolation은 주변 4개의 픽셀에 resize된 이미지로부터 나온 픽셀 위치에 대한 가중치를 곱하여 이미지를 resize하는 방법입니다. nearest Neighbor 방법과 마찬가지로 새로 할당된 이미지 size에서 배율을 나누어 이전 이미지의 어느 위치에 있는지 확인합니다. 그 위치를 포함하는 4개의 픽셀로부터 거리만큼 가중치를 곱하여 새로운 픽셀값을 할당합니다.

- Source code

- myResizeNN.m

```
function img = myResizeNN(image, rate)
if length(size(image)) >= 2
    [row column channel] = size(image);
else
    [row column]= size(image);
    channel = 1;
end % channel issue
resize = ceil([row column]*rate);
for i = 1:resize(1)
    x = i*1/rate;
    x(x<1) = 1;
    x(x>row) = row;
    for j = 1:resize(2)
        y = j*1/rate;
        y(y<1) = 1;
        y(y>column) = column;
        img(i,j,1:channel) = image(round(x), round(y), 1:channel);
    end
end
end
```

제가 수업의 알고리즘을 보며 직접 구현한 nearest Neighbor resize 함수입니다. 먼저 원본 이미지 채널을 저장합니다. Color 이미지의 경우 3채널, grayscale/binary 이미지의 경우 1채널을 사용하므로 이미지 size 배열의 길이에 따라 channel 변수의 값을 정해줍니다. 그 외 row, column은 원본 이미지의 세로, 가로 픽셀 수입니다. row, column에 배율

rate를 곱하여 이미지 resize 후 사이즈를 계산합니다.

그 다음으로 resize 후 픽셀에 들어갈 픽셀값을 구하기 위해 backward 방법을 통해 원본 이미지로부터 픽셀값을 구해 옵니다. nearest Neighbor 방법에서는 다시 원본 이미지로 돌아갔을 때 그 지점과 가장 가까운 픽셀의 픽셀값을 가져옵니다. 예를 들어 resize픽셀*1/배율= (1.3, 1.7) 이라면, 픽셀 좌표에 실수는 없으므로 가장 가까운 자연수 좌표인 (1, 2) 지점의 픽셀값을 가져옵니다. 따라서 이러한 경우 반올림한 좌표의 값을 가져오므로 round 함수를 사용합니다.

하지만 배율이 2보다 클 경우, 첫 번째 픽셀 index 1에 대한 원본 이미지에서의 좌표값이 0이 됩니다. 예를 들어 배율이 2.5일 경우 round(첫 번째 픽셀 index/rate) = round(1/2.5) = round(0.4) = 0인데, Matlab matrix는 index 1부터 시작하므로 오류가 발생합니다. 그러한 점을 막기 위해 x, y값이 1보다 작거나 원본 이미지의 최대 사이즈보다 큰 경우 다시 값을 1 또는 원본 이미지의 최대 사이즈로 되돌려서 resize 이미지에 할당해 줍니다.

또한 Color/1채널 이미지 상관없이 채널만큼 이미지를 할당해 주도록 하였습니다.

- myResizeBil.m

```
function img = myResizeBil(image, rate)
if length(size(image)) >= 2
    [row column channel] = size(image);
else
    [row column]= size(image);
    channel = 1;
end
resize = ceil([row column]*rate);
image(row+1, 1:column, 1:channel) = zeros(1, column, channel);
image(1:row+1, column+1, 1:channel) = zeros(row+1, 1, channel);
for i = 1:resize(1)
    x = i*1/rate;
    x(x<1) = 1;
    x(x>row) = row;
    l = floor(x);
    a = x - l;
    for j = 1:resize(2)
        y = j*1/rate;
        y(y<1) = 1;
        y(y>column) = column;
        k = floor(y);
        b = y - k;
        img(i,j,1:channel)                                                    =
(1-a)*(1-b)*image(l,k,1:channel)+a*(1-b)*image(l+1,k,1:channel)+(1-
a)*b*image(l,k+1,1:channel)+a*b*image(l+1,k+1,1:channel);
```

```
    end
end
end
```

다음으로 제가 작성한 Bilinearest interpolation 함수입니다. 이미지의 채널 및 사이즈(row, column) 결정은 위 nearest Neighbor code에서의 방법과 같습니다. 하지만 resize 후 이미지에서의 픽셀과 원본 이미지에서의 픽셀이 일대일 대응하는 nearest Neighbor과는 다르게 Bilinearest interpolation 방법은 backward한 좌표 주변의 4픽셀로부터 값을 도출합니다. 따라서 처음 픽셀은 괜찮지만, 마지막 픽셀값을 가져올 때 원본 이미지 픽셀의 최대값+1 부분을 참조하는 문제가 발생합니다. 그 점을 해결하기 위해서 원본 이미지의 우측과 하단에 ㄱ자 형태로 0으로 pad해 줍니다. 만약 0행렬 pad가 없을 경우, 오류가 나지 않는다면 맨 밑줄에 대한 resize가 진행되지 않기 때문에 결과 이미지의 가로, 세로가 1픽셀씩 줄어들게 됩니다.

Bilinearest interpolation은 nearest Neighbor과는 다르게 계산을 시작하는 좌표의 지점이 backward된 좌표 기준으로 좌측 상단 좌표이기 때문에 반올림하지 않고 내림하게 됩니다. 이러한 점에서 발생하는 문제로, 배율이 1 이상일 때 첫 번째 좌표가 참조하는 값이 0번째 픽셀을 포함합니다. 따라서 Nearest Neighbor처럼 x, y값이 1보다 작거나 원본 이미지의 최대 사이즈보다 큰 경우 다시 값을 1 또는 원본 이미지의 최대 사이즈로 되돌려서 resize 이미지에 할당해 주었습니다.

- DIP_3_1.m (메인 코드)

```
clc; clear all; close all;
img = imread('data/profile.jpg');
% img = imread('rice.png');

% Near Neighbor
imgNN081 = myResizeNN(img, 0.81);
imgNN143 = myResizeNN(img, 1.43);
figure(1);
subplot(1,3,1);
imshow(img);
title('Original','fontsize',16)
subplot(1,3,2);
imshow(imgNN081);
title('0.81','fontsize',16)
subplot(1,3,3);
imshow(imgNN143);
title('1.43','fontsize',16)

% Bilinear
imgBil081 = myResizeBil(img, 0.81);
imgBil143 = myResizeBil(img, 1.43);
figure(2)
```

```
subplot(1,3,1);
imshow(img);
title('Original','fontsize',16)
subplot(1,3,2);
imshow(imgBil081);
title('0.81','fontsize',16)
subplot(1,3,3);
imshow(imgBil143);
title('1.43','fontsize',16)

% NN Bil Compare
figure(3)
subplot(1,2,1);
imshow(imgNN143(2300:2700, 800:1500, 1:3));
title('NearNeighbor 1.43','fontsize',16)
subplot(1,2,2);
imshow(imgBil143(2300:2700, 800:1500, 1:3));
title('Bilinear 1.43','fontsize',16)

figure(4)
subplot(1,2,1);
imshow(imgNN143);
title('NearNeighbor 1.43','fontsize',16)
subplot(1,2,2);
imshow(imgBil143);
title('Bilinear 1.43','fontsize',16)

% Near Neighbor_builtin
imgNN081_builtin = imresize(img, 0.81, 'nearest');
imgNN143_builtin = imresize(img, 1.43, 'nearest');
figure(5);
subplot(2,2,1);
imshow(imgNN081);
title('0.81','fontsize',16)
subplot(2,2,2);
imshow(imgNN143);
title('1.43','fontsize',16)
subplot(2,2,3);
imshow(imgNN081_builtin);
title('0.81-builtin','fontsize',16)
subplot(2,2,4);
```

```
imshow(imgNN143_builtin);
title('1.43-builtin','fontsize',16)

% Bilinear_builtin
imgBil081_builtin = imresize(img, 0.81, 'bilinear');
imgBil143_builtin = imresize(img, 1.43, 'bilinear');
figure(6);
subplot(2,2,1);
imshow(imgBil081);
title('0.81','fontsize',16)
subplot(2,2,2);
imshow(imgBil143);
title('1.43','fontsize',16)
subplot(2,2,3);
imshow(imgBil081_builtin);
title('0.81-builtin','fontsize',16)
subplot(2,2,4);
imshow(imgBil143_builtin);
title('1.43-builtin','fontsize',16)
```

이미지 load, 함수 실행, subplot 작성 기능을 갖고 있는 메인 코드입니다.

- 결과 비교 및 배운 점

Lab 2-2의 실행 결과는 다음과 같습니다.

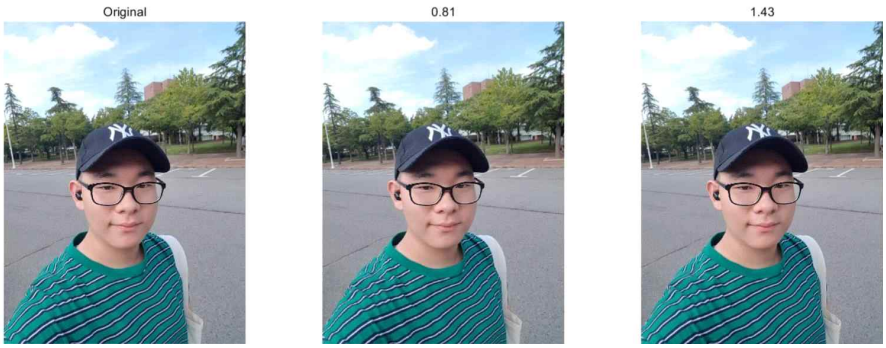

그림 2 myResizeNN의 결과

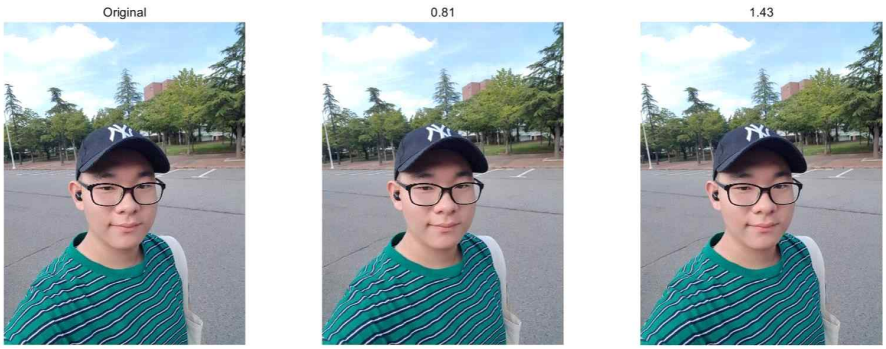

그림 3 myResizeBil의 결과

다음은 1.43 배율의 결과를 확대한 모습입니다. 확실히 Nearest Neighbor 방법보다 Bilinear 방법이 훨씬 부드럽게 이어지는 것을 확인할 수 있습니다.

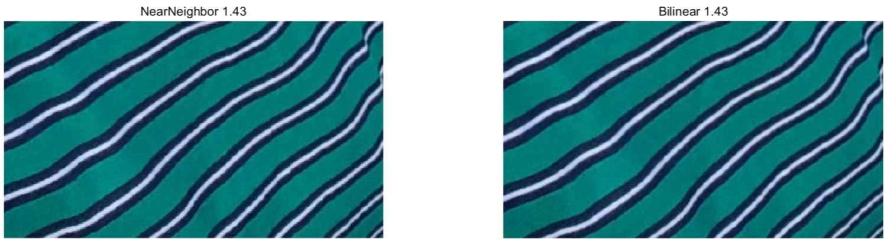

그림 4 Nearest Neighbor과 Bilinear의 결과 비교

다음은 매트랩에 내장되어 있는 함수와 직접 제작한 함수의 비교입니다.

그림 5 Nearest Neighbor 비교

그림 6 Bilinear 비교

외관상으로 큰 차이 없이 resize 되는 것을 볼 수 있습니다. 작업 공간의 size 결과 역시 내장함수의 결과와 동일합니다.

그림 7 작업 공간

Lab 2-5 Simple Image Labeling using yourCode

- 문제 - blob labeling 커스텀 함수 구현

  blob labeling은 이미지에서 물체가 몇 개 있는지를 분류하고 labeling 할 수 있는 알고리즘입니다. 4-adjacency, 8-adjacency 방법을 이용하여 근처 픽셀의 변화에 따라 물체의 edge로 추정되는 부분을 만날 때 labeling을 하고 그 물체를 분류합니다.

- Source code

- mybwlabel.m

```
function label = mybwlabel(image, adjacency)
label = double(padarray(image, [1 1], 0, 'both'));
if adjacency == 4
    filter = [0 1 0;
        1 0 0;
        0 0 0];
elseif adjacency == 8
    filter = [1 1 1;
        1 0 0;
        0 0 0];
end
[row column] = size(label);
l=1;
for i = 1:row-2
    for j = 1:column-2
        if label(i+1, j+1) == 1
            conv = label(i:i+2, j:j+2).*filter;
            if sum(conv, 'all') == 0
                label(i+1,j+1) = l;
                l=l+1;
            else
                conv_nonzero = conv(conv~=0);
                num = min(conv_nonzero);
                label(i+1,j+1) = num;
            end
        end
    end
end
label = label(2:row-1, 2:column-1);
end
```

제가 구현한 bwlabel의 함수는 다음과 같습니다. 첫 번째 문제로 타겟 픽셀의 주변 5개의 픽셀을 참조해야 하는데, 테두리의 픽셀은 주변 픽셀이 없는 부분이 있는 점을 해결해야 했습니다. 그래서 padding 함수를 이용해서 주변 테두리를 0 행렬로 바꾸어 줍니다. 이렇게 pad를 함으로서 주변 픽셀값을 검사할 수 있습니다. 또한 이 함수의 input이 binary 이미지라고 가정하여 double 형으로 형 변환을 해 줍니다. 이 변수에 다시 결과값을 할당하기 때문에 logical 함수는 사용할 수 없습니다. 그 다음, filter 형태로 주변 픽셀을 검사할 행렬 filter를 할당합니다. 받아오는 함수의 매개변수 값에 따라서 4-adjacency, 8-adjacency filter를 결정합니다.

그 후 현재 label 값으로 사용할 변수 l을 1로 초기화합니다. 맨 처음 시작이므로, 첫 번째

물체를 발견하면 그 물체를 1로 labeling한 후 1을 증가시켜 다음 label로 사용합니다. labeling 과정은 다음과 같습니다. 먼저 반복문을 통해 이미지의 모든 픽셀에 filter로 컨볼루션 연산을 진행합니다. foreground 즉 배경과 다른 점이 존재하는 영역을 물체로 인식하고 labeling을 진행합니다. 맨 첫 번째로 감지된 물체에 label을 부여하고 1을 증가시킵니다. 컨볼루션 연산 결과의 합이 0인 경우 물체가 탐지되었다고 간주합니다.

이 물체가 이전에 탐지된 물체와 같은 물체인지를 확인하는 알고리즘은 else문 내부의 내용과 같습니다. 컨볼루션 연산 결과에서 0을 모두 제외하고 남은 label값 중 가장 낮은 label을 저장합니다.

위와 같은 과정을 반복하여 label된 이미지를 얻을 수 있습니다. 하지만 반복문을 끝내고 나서도 아직 pad가 남아 있으므로, pad를 제외한 부분만을 추출하여 return 해 줍니다.

- DIP_3_2.m(메인 코드)

```
clc; clear all; close all;
img = imread('rice.png');
rice_BW = imbinarize(img);

% source data
BW = logical([
1 1 1 0 0 0 0 0;
1 1 1 0 1 1 0 0;
1 1 1 0 1 1 0 0;
1 1 1 0 0 0 1 0;
1 1 1 0 0 0 1 0;
1 1 1 0 0 0 1 0;
1 1 1 0 0 1 1 0;
1 1 1 0 0 0 0 0 ]);
L = mybwlabel(BW, 8);
rice_L = mybwlabel(rice_BW, 8);
L_builtin = bwlabel(BW, 8);
rice_L_builtin = bwlabel(rice_BW, 8);

figure(1);
subplot(1,3,1); imagesc(BW); colormap jet; colorbar;
% imshow(BW);
title('Original','fontsize',16);
subplot(1,3,2); imagesc(L); colormap jet; colorbar;
% imshow(L);
title('Labeled','fontsize',16);
subplot(1,3,3); imagesc(L_builtin); colormap jet; colorbar;
% imshow(L_builtin);
title('Labeled-builtin','fontsize',16);
```

```
figure(2);
subplot(1,3,1); imagesc(rice_BW); colormap jet; colorbar;
% imshow(rice_BW);
title('Original','fontsize',16);
subplot(1,3,2); imagesc(rice_L); colormap jet; colorbar;
% imshow(rice_L);
title('Labeled','fontsize',16);
subplot(1,3,3); imagesc(rice_L_builtin); colormap jet; colorbar;
% imshow(rice_L_builtin);
title('Labeled-builtin','fontsize',16);
```

이미지 로드와 함수 실행, subplot 작성을 담당하는 메인 코드입니다.

- 결과 비교 및 배운 점

Lab 2-3의 결과는 다음과 같습니다.

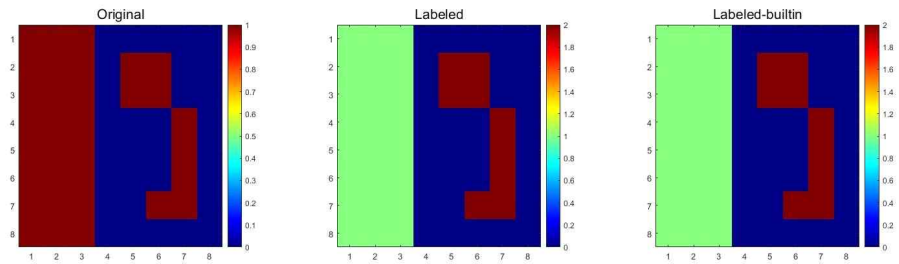

그림 8 Lab 2-3의 결과

Matlab 내장 함수와 같은 결과가 나오는 것을 확인할 수 있습니다. 하지만 Lab 2-4의 결과는 조금 달랐습니다. Lab 2-4의 결과는 다음과 같습니다.

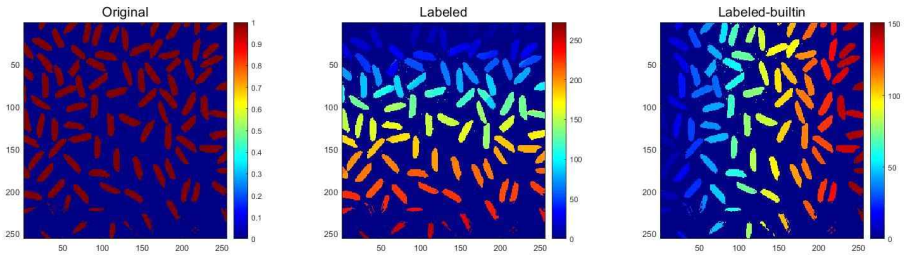

그림 9 Lab 2-4의 결과

Matlab 내장 함수에 rice.png 파일을 실행시킨 결과입니다. 제가 구현한 함수는 물체를 가로로 labeling하는 반면, Matlab 내장 함수는 물체를 세로로 labeling하는 것을 볼 수 있었습니다. 그래서 builtin 함수와 filter 진행방향이 다른 점이 문제라고 생각하여 filter의 방향과 연산 방향을 90도 돌려서 연산한 결과는 다음과 같았습니다.

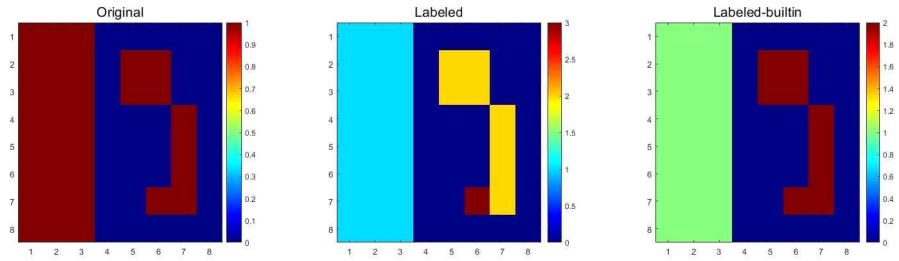

그림 10 filter를 반시계방향으로 90도 돌려서 연산한 Lab 2-3 결과

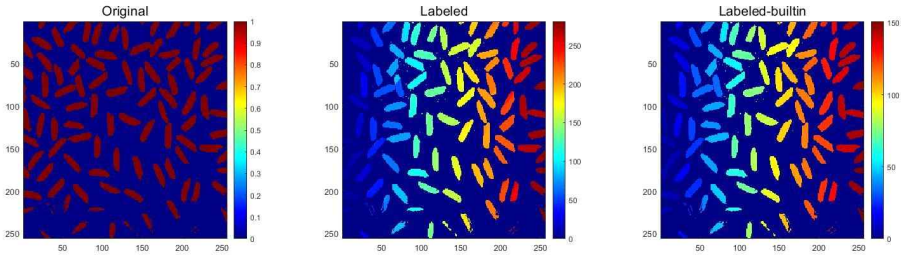

그림 11 filter를 반시계방향으로 90도 돌려서 연산한 Lab 2-4 결과

rice.png에 대한 결과는 내장함수와 비슷하게 나오지만 Lab 2-3에서 사용한 예시는 제대로 나오지 않음을 볼 수 있었습니다.