영상처리 2주차 과제

학번: 201404376

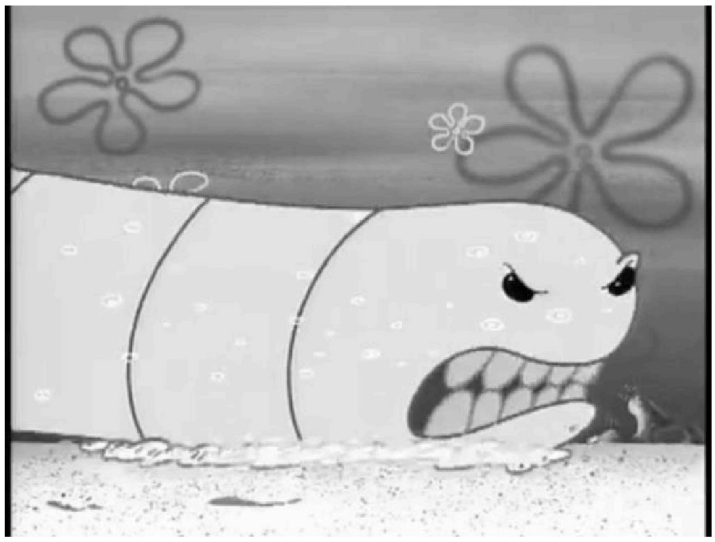
이름: 정성욱

1. 과제 내용

:: 구현한 과제에 대한 설명, 어떤 방식으로 접근해야 하는지

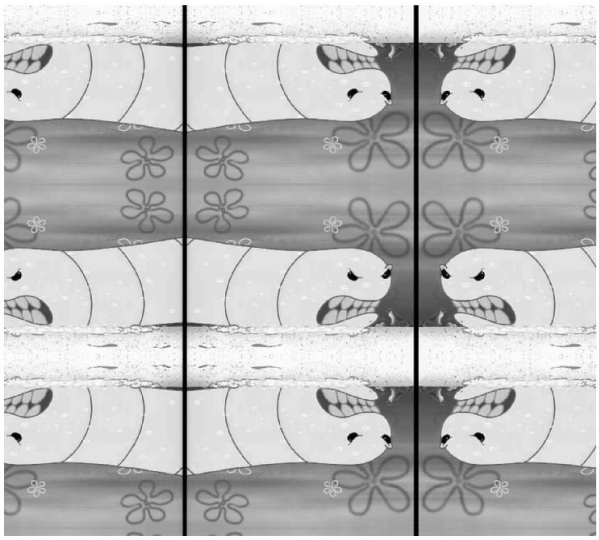
실행 화면:

원본:

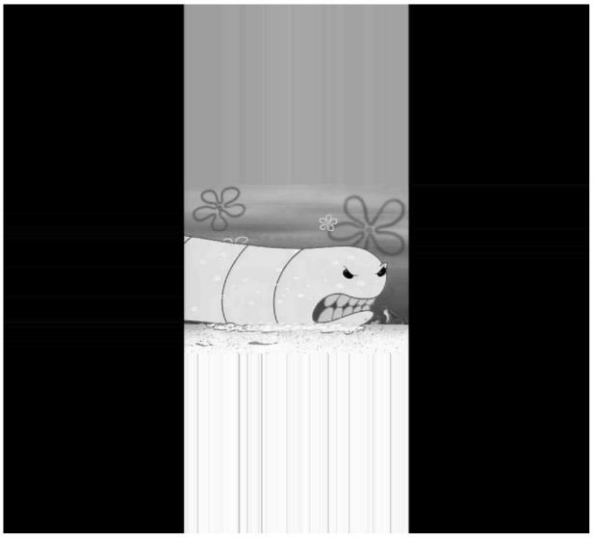


Padding

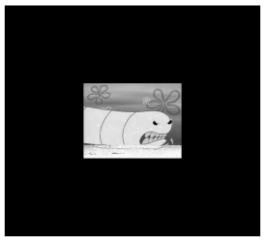
Mirror:



Repetition:

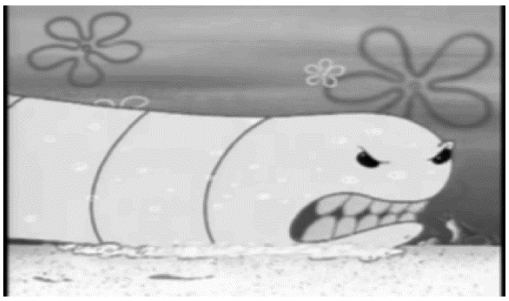


Zero padding:



Filter

Avr:

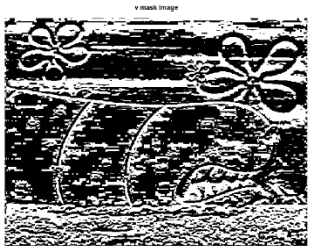


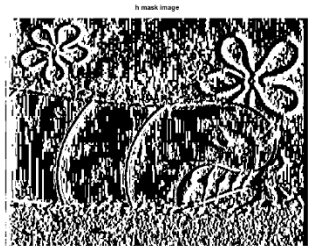
Weight:

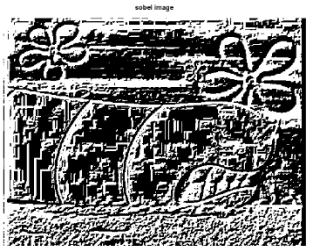
Laplacian:

median:

Sobel:







Unsharp:

구현한 방법에 대한 이유

:: 구현한 방법에 대한 설명 (왜 이렇게 구현했는지 자세히)

Padding:

Mirror:

```
strcmp(type, 'mirror') % mirror padding
 up_xcounter=0;
 up_ycounter=0;
 reverse=0;
 %up_mirror
 for lxp = pad:-1:1
   if up_xcounter==x
     reverse=1;
   elseif up_xcounter==1
     reverse=0;
   end

   if reverse
     up_xcounter=up_xcounter-1;
   else
     up_xcounter=up_xcounter+1;
   end
   for lyp =1:y
     pad_img(lxp,pad+lyp) = img(up_xcounter,lyp);
   end
 end
   %under_mirror
   reverse=0;
   under_xmirror = 0;
   under_xcounter=x;

   for lxp =x+pad+1:2*pad+x
     if under_xcounter==1
       reverse=1;
     elseif under_xcounter==x
       reverse=0;
     end

     if reverse
       under_xcounter=under_xcounter+1;
     else
       under_xcounter=under_xcounter-1;
     end

     for lyp =1:y
       pad_img(lxp,pad+lyp) = img(under_xcounter,lyp);
     end
   end
```

조금 길기 때문에 위 아래 먼저 설명하겠습니다. 우선 기본적으로 제로패딩이 되어있는 상태인데 먼저 위의 경우부터는 x루프는 패드에서부터 1까지 -1씩 감소하면서 가게하고 up_xcounter라는 변수를 두어 감소하는 반면 이미지를 읽어와서 반대가되도록 서로 반대 수를 가지도록 의도합니다. 그러다가 움직인 각 변수가 거리가 원본 이미지보다 크게되면, 읽는 입장에서는 이미지의 끝에 도달했다는 것인데 여기서 다시 반전이 있어야하므로, 끝부분일경우 reverse를 1로 주고 다시 끝부분부터 다시 반대로 -1씩 감소시키며 읽어들입니다. 그러다가 xcounter가 0일경우 revser를 0으로 주고 다시 1씩 증가시키면서 정방향으로 읽는 방법을 사용하였습니다.

마찬가지로 아래쪽도,x가 밑으로 커진다 뿐이지 전체적인 매커니즘은 같습니다. 다만 x의 루프가 반대방향으로 움직입니다.x는 위아래로 올라가고 y는 왼쪽에서 오른쪽으로 X가 움직일 때 마다

전반적인 픽셀을 채워주는 역할을 합니다.

```
%left_mirror
reverse=0;
under_ycounter = pad;
for lyp =pad:-1:1
  if under_ycounter==pad+2+y
    reverse=1;
  elseif under_ycounter==pad
    reverse=0;
  end

  if reverse
    under_ycounter=under_ycounter-1;
  else
    under_ycounter=under_ycounter+1;
  end
    for lxp =1:2*pad+x
    pad_img(lxp,lyp) = pad_img(lxp,under_ycounter);
  end
end

 %right_mirror
  reverse=0;
  under_ycounter = y+pad+2;
  for lyp =y+pad+2:2*pad+y
    if under_ycounter==pad
      reverse=1;
    elseif under_ycounter==y+pad+2
      reverse=0;
    end

    if reverse
      under_ycounter=under_ycounter+1;
    else
      under_ycounter=under_ycounter-1;
    end


      for lxp =1:2*pad+x
      pad_img(lxp,lyp) = pad_img(lxp,under_ycounter);
    end
  end
```

왼쪽과 오른쪽에 대한 설명입니다. 이역시 전체적인 매커니즘은 위아래 거울과 같지만

X가 픽셀 출력을 위한 루프를 돌고 y가 직접적으로 왼쪽 오른쪽으로 움직입니다. 왼쪽의 경우를 살펴보자면, under_y_counter라는 변수에 pad를 둡니다.(거울은 여기서부터 시작해야 되기 때문입니다.) Y의 경우 직접 왼쪽 오른쪽을 이동하면서 패딩을 채워야하는 역할입니다. Lyp lxp를 뒀는데 오른쪽에서는 lyp의 시작점은 pad+2+y(이미지 밖 바로 시작점)이고 왼쪽은 pad입니다. Lxp는 정확히 원래 이미지의 높이만큼의 픽셀을 읽어오는 것인데 마치 3d프린트가 물건깍을때 하나씩 움직여서 하나하나 스캔하는 기계팔 같은 역할을 하는 것입니다. 매 루프마다 왼쪽의 lyp는 -1 오른쪽은 lyp가 1씩 증가, 그리고 증가량이 원본이미지의 마지막 크기에 다달았다면 under_ycounter lxp와 함께 원본이미지를 스캔하는 만약 오른쪽으로 이동하고 있었다면 왼쪽으로 가게 방향을 틀어줍니다. 그러면 출력한 형상이 다시 나오게되는데 그것이 데칼코마니형태인 거

울 형태가 되는 것입니다.

Repetition :

```
elseif strcmp(type, 'repetition') % repetition padding
    up_xcounter=0;
    up_ycounter=0;
    for yupper = pad+1:y+pad+1
      for x_upper = pad:-1:1
        pad_img(x_upper,yupper) = pad_img(pad+1,yupper);
      end
      for x_upper = x+1+pad:2*pad+x
        pad_img(x_upper,yupper) = pad_img(x+pad,yupper);
      end
    end

    left_ycounter=pad+1;
    for yupper = pad:-1:1
      for x_upper = pad+1:pad+x
        pad_img(x_upper,yupper) = pad_img(x_upper,left_ycounter);
      end
    end
    right_ycounter=pad+y;
    for yupper = pad+y+1:pad*2+y
      for x_upper = pad+1:pad+x+1
        pad_img(x_upper,yupper) = pad_img(x_upper,right_ycounter);
      end
    end

    out_left_xcounter = pad+1;

    for x_upper = pad:-1:1
      for yupper = pad+1:-1:1
        pad_img(x_upper,yupper) = pad_img(out_left_xcounter,pad+1);
      end
    end

    out_left_xcounter = pad+1+x;
```

```
for x_upper = pad:-1:1
  for yupper = pad+1:-1:1
    pad_img(x_upper,yupper) = pad_img(out_left_xcounter,pad+1);
  end
end

out_left_xcounter = pad+1+x;

for x_under = pad+x+1:pad*2+x
  for yunder = pad+1:-1:1

    pad_img(x_under,yunder) = pad_img(out_left_xcounter,1+pad);
  end
end
for x_upper = pad:-1:1
  for yupper = pad+1:-1:1
    pad_img(x_upper,yupper) = pad_img(out_left_xcounter,pad+1);
  end
end

out_right_xcounter = pad+1+x;

for x_under = pad+x+1:pad*2+x
  for yunder = pad+1+y:pad*2+y

    pad_img(x_under,yunder) = pad_img(out_left_xcounter,1+pad);
  end
end

out_right_xcounter = pad+1;

for x_under = pad+1:-1:1
  for yunder = pad+1+y:pad*2+y

    pad_img(x_under,yunder) = pad_img(out_left_xcounter,1+pad+y);
  end
end
```

Repetition에 관한 설명입니다. 끝부분의 값을 그대로 유지한채 패딩의 크기만큼 늘려주는 것이 목적입니다. 이미지와 직접 붙어있는 왼쪽,오른쪽,남,북부터 보자면 , 기준점의 위치는 (pad+1,pad+1), (pad+x,pad+1), (pad+1,pad+y), (pad+x,pad+y)로 나눌 수 있습니다. 왼쪽은 pad+1+x,pad+1(위에서 아래로) 가는것을 목표로 스캔을 합니다. 오른쪽도 (pad+x,pad+y)역시 오른쪽의 위에서 아래로 스캔, 남북은 왼쪽에서 오른쪽으로 북의 경우 (pad+1,pad+y), 남의 경우 (pad+x,pad+y)로 가는 경우를 봅니다. 그렇게 스캔한 값을 가지고 pad_size만큼 채워주면 동서남북에 대한 패딩처리는 됩니다. 동북, 서북, 동남, 서남 이 비게 됩니다. 이것에 대해 방금 패딩한 동서남북의 패딩을 확장을 해야합니다. 동북의 경우 왼쪽으로 패딩한 끝부분의 값을 다시 카피하여 x_upper, x가 위로 움직이면서 y가 값을 출력하는 식으로 처리합니다. 동남 역시 왼쪽아래의 값을 복사하여 y는 프린터의 잉크 출력기 처럼 루프돌고 처음으로 루프돌고 처음으로를 반복하고, x는 밑으로 쭉 내려가면서 왼쪽 아래의 마지막 픽셀값을 복사하여 끝까지 밀어넣습니다.

서남 서북 역시 같은 방법으로합니다만, 이때 차이는 x_upper시작점이 x+pad+1에서해서

x+2*pad로 끝난 다는 것과 y의 활동 범위가 y+pad+1에서 y+pad*2까지로 변경된다는 점입니다.

Zero padding:

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

가장 간단한 패딩이 아닐까합니다 단순히 0으로된 행렬을 원래 x의값 +2*pad 크기와 원래 y의 값 +2*pad 크기만큼만들어주고 x= 1+pad,y=1+pad 자리부터 원본 이미지를 가운데에 넣어서 이미지가 한가운데부터 시작하게 하는 패딩입니다. 전체적인 색상은 픽셀값이 0이라 검은색이고 가운데 이미지만 원래의 이미지 색상입니다.

Filter

자주 사용되는 전역변수

Pad_size = filter_Size/2

x,y는 [x, y] = size(img); 여기서 img는 함수를 실행할 때 넣어주는 이미지.

Pad_img : 원본 이미지에 pad_size만큼 mirror 패딩을 취한 이미지임.

```
if strcmp(type, 'avr')
    for i = 1:x
        for j = 1:y
            % fil_size-1까지로 해야 filter size만큼의 mask가 잡힘
            filter_img(i,j) = mean(mean(pad_img(i:i+filter_size-1, j:j+filter_size-1)));
        end
    end
elseif strcmp(type, 'weight')
    % 다른 필터의 마스크를 만들 때에 참고
    mask = [1:pad_size+1 pad_size:-1:1]' * [1:pad_size+1 pad_size:-1:1];
    % 필터의 합이 1이 되게 하기 위해 sum을 미리 구함
    s = sum(sum(mask));
    for i = 1:x
       for j = 1:y
            filter_img(i,j) = sum(sum(double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*mask))/s;
       end
    end
elseif strcmp(type, 'laplacian') % Laplacian Filter
     %mask = [1:pad_size+1 pad_size:-1:1]' * [1:pad_size+1 pad_size:-1:1];
     laple_masks= -1*ones(filter_size,filter_size);
     center_x = pad_size+1;
     center_y = pad_size+1;
     coeff =0;
     base = filter_size*filter_size;
     cx = 0;
     cy = 0;
     laple_masks(center_x,center_y) = base-1;

     for i = 1:x
       for j = 1:y
            filter_img(i,j) = sum(sum((double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*laple_masks)));
       end
    end
```

avr:

필터 이미지를 원본 이미지 크기 만큼 만듭니다. 그리고 원본 이미지에서 패딩한 이미지에 대해

필터 사이즈만큼 평균의 합을 구합니다 그러면 각 열에 대한 평균값으로 1차원 값으로 줄어듭니다. 거기서 다시 1차원 배열의 평균을 구해서 i,j값에 넣어줍니다. 평균값으로 나눈 것이기 때문에 전체적으로 픽셀값이 뭉그러집니다.(비율은 유지됨)

weight:

mask를 둡니다. 생성 소스는 [1:pad_size+1 pad_size:-1:1]' * [1:pad_size+1 pad_size:-1:1]; 인데

pad_Size가 3일 경우

1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	16	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1

이런식으로 pad+1을 중심으로 가장 큰 가중치를 두고 양, 옆, 위, 아래로 조금씩 값을 줄여 나가는 분포를 주는 것을 만듭니다. 값 연산을 하기 위한 변수 s에 행렬 합을 구하기 위해 sum연산을 두 번한 값을 저장합니다. S를 패드이미지에 점 곱한 결과값의 합에 대해 나눕니다. 그러면 자동적으로 가운데 가중치가 높으므로 비뚤한 픽셀값이 조금은 뭉그러집니다.

laplacian:

laple_masks로 (filter,filter)크기의 1로 채운 필터를 하나 만들어주고 -1을 곱해 전부다 -1의 값을 갖는 필터로 만들어 줍니다. 또한 가운데 값을 filter*filter-1값으로 base라는 변수에 저장합니다.

그리고 가운데 인덱스 값을 정해줘야 하므로, center_x,center_y를 두고 패드사이즈+1을 각각 넣어줬습니다. 처음 만들어뒀던 laple mask의 가운데 base값을 대입하여 전체합이 0이 되도록 맞춰주고 패드이미지에 점곱을 한뒤에 그 행렬의합(sum두번)을 구해서 filter이미지의 I,j에 넣어줍니다.

Median:

```
elseif strcmp(type, 'median') % Median Filter
    x_count_stack = 1;
    real_x = pad_size+1;
    real_y = pad_size+1;
    for pad_img_x= 1:x+pad_size
      for pad_img_y = 1:y +pad_size
        index = 1;
        list_value = [];
        add_val = 1;
        for fx = pad_img_x:pad_img_x+filter_size
            if fx>x+pad_size
             add_val =0;
             break;
            end
            for fy =pad_img_y:pad_img_y+filter_size
              if fy>y+pad_size
               add_val =0;
               break;
              end
              list_value(index) = pad_img(fx,fy);
              list_value;
              pad_img(fx,fy);
              index=index+1;
              index;
            end
         end
         if add_val ==1
         m_value = median(list_value);
         filter_img(real_x,real_y) =m_value;
         end

      real_y=real_y+1;
    end
      real_y=pad_size+1;
      %x_count_stack=x_count_stack+1
      real_x=real_x+1;
    end
    % Fill here
```

여기엔 패드이미지 전체를 도는 루프, 그 루프 내에서 필터사이즈만큼 도는 루프 x,y축해서 총 4개의 루프가 사용되었습니다. real_x,y는 실제 filter median값을 넣어줄 곳을 가르키는 인덱스 입니다. 패드이미지를 돌면서 필터사이즈 만큼 이미지의 끝부분부터 마지막까지 스캔을 한뒤에 매 스캔 루프마다 모인 값 중에 중간 값을 필터이미지의 real,x,y에 넣고 다시 돌리는 식으로 처리하였습니다. 필터 이미지의 루프 굴리는 것은 real_y 기준으로 반복하면서 1줄 1줄씩 채우게 구성했기 때문에 매번 초기화 되는 것은 real_y 입니다.

Sobel:

```
elseif strcmp(type, 'sobel') % Sobel Filter
    basic_filter_size = 3;
    long_sx = 0;
    % Fill it
    v_img = zeros(x, y);
    h_img = zeros(x, y);

      long_sx = zeros(filter_size,filter_size);
      center_x = pad_size+1;
      center_y = pad_size+1;
      for lsix = 1:pad_size
        for lsiy =1:filter_size
          if lsiy == center_y
            long_sx(lsix,lsiy) = -2;
          else
            long_sx(lsix,lsiy) = -1;
          end
        end
    %  end
          under_sx = -1*flip(long_sx(1:center_x-1,1:filter_size));
          long_sx(center_x+1:filter_size,1:filter_size) = under_sx;
          long_sy = long_sx';

        for i = 1:x
       for j = 1:y
           v_img(i,j) = sum(sum(double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*long_sx));
           h_img(i,j) = sum(sum(double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*long_sy));
       end
      end

  end
    % Fill here
    filter_img = h_img+v_img;
    % Sobel에서 만들어지는 이미지 비교
    figure
    subplot(1, 1500, 1:490), imshow(v_img), title('v mask image');
    subplot(1, 1500, 501:990), imshow(h_img), title('h mask image');
    subplot(1, 1500, 1001:1490), imshow(filter_img), title('sobel image');
```

Low와 high의 크기를 정하고 시작합니다.

소밸의 x를 정의해주고 y를 x의 역행렬을 넣어주면 되기 때문에 x먼저 만드는 것에 집중합니다.

가운데 (가로든 세로든 한줄만)center_x,y의 열이 전부 0이어야하고 그 0을 중심으로 대칭이 되면서 서로 반대속성을 지녀야합니다 우선 윗줄부터 1씩 채우다가 가운데 center값이 나오면 2로 채웁니다. 그러면 만약에 필터값이 5라하면

11211

11211

00000

00000 (이거 밑에 줄입니다)00000 << 원ㄹㅐ 5x5 행렬

이런 형태가 만들어집니다. 그 후 center(pad_size+1)을 기준으로 아래쪽 두 줄을 만들어야하는데, 그냥 위쪽에 만들어진

11211

11211 을 싹 긁어와서 -1을 붙인 뒤에 pad_size+1부터 filter_size까지의 슬라이싱을 해준 후 붙여 넣어주면

11211

11211

00000

-1-1-2-1-1

-1-1-2-1-1

이되면서 소밸이 만들어지고 이것의 역행렬을 하게되면

110-1-1

110-1-1

220-2-2

110-1-1

110-1-1

이 되고 이것이 각각 v_img ,h_img가 되고 filter은 이것을 패딩 이미지에 씌운 뒤 두 개의 행렬곱 값을 덧셈해주면 filter_img인 sobel이 되는 것입니다.

Unsharp:

```
elseif strcmp(type, 'unsharp')
    k = 0.5;
    l_filter = zeros(filter_size,filter_size);
    l_filter(pad_size+1,pad_size+1) = 1;
    l_filter =l_filter*(1/(1-k));
    h_filter = -1*ones(filter_size,filter_size)/(filter_size*filter_size);
    h_filter = h_filter*(k/(1-k));

    for i = 1:x
        for j = 1:y
            filter_img(i,j) = sum(sum(double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*l_filter+double(pad_img(i:i+filter_size-1,j:j+filter_size-1)).*h_filter)),
        end
    end

    % k는 조정해도 괜찮음 (0 <= k <= 1)

    % Fill it
end
```

이것의 경우 가운데 점에 가중치를 둔 저주파 l_filter와 고주파인 h_filter로 나누어야합니다.

l_filter의 경우 필터의 크기만큼 만들어주고 가운데 pad_size+1,pad_size+1의 부분에 1을 넣어줍니다. 반대로 h_filter는 처음의 weight처름 모두 1로 맞추고 필터의 크기^2 만큼 나누어주고 -1을 곱해줍니다. 그 후 정의된 상수 k를 곱해줍니다.

L_filter와 h_filter에 (1-k)를 나누어주고 pad_img에 필더를 점곱 한 뒤 두 행렬을 더합니다. 그 후 행과 열에 대한 값의 합을 구한후(2차원인데-> 1차원 1차원에서 값으로 바뀜) 필터 이미지의 하나의 인덱스에 값을 넣어주면서, 필터를 끝까지 채웁니다.

느낀 점

:: 구현하면서 느낀 점, 어려웠던 점, 혹은 설명이 필요하다고 느낀 부분

필터부분이 정확히 어떻게 이루어지는지 궁금합니다. 인터넷으로 찾아보면서 공부하긴했지만, 실질적인 연산을 다루는 부분은 잘 안나와 있더군요..