CSC420 Assignment 1

Gabriel Luong
996268275

1

Algorithm 1 Convolution

```
1: procedure CONVOLUTION(Image(i × j), Filter(u × v))
2:     G ← new i × j image matrix
3:     for ImageY = 0 to j do
4:         for ImageX = 0 to i do
5:             sum ← 0
6:             for FilterY = -v to v do
7:                 for FilterX = -u to u do
8:                     sum ← sum + Filter(FilterX, FilterY) × Image(ImageX + FilterX, ImageY
   + FilterY)
9:                 end for
10:            end for
11:            G(ImageX, ImageY) = sum
12:        end for
13:    end for
14:    return G
15: end procedure
```

2

(a) For a given n x n image, I, and m x m filter, h, the process of performing a convolution requires m^2 operations per pixel in I which results in a time complexity of $O(n^2m^2)$. However, the complexity of convolution with the help of Fourier Transform can be reduced to $O(nlogn)$. If h can be separated to vertical and horizontal 1D filters, only 2 1D convolutions (m operations) will need to be computed for every pixel in I, and therefore it will only have a time complexity of $O(2m * n^2) = O(m * n^2)$.

(b) Applying first a Gauassian filter with $\sigma_1 = 3$, and then another Gaussian with $\sigma_2 = 4$ is equivalent to applying one Gaussian filter with $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2} = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$.

(c) Convolve **lena.png** with a Gaussian filter ($\sigma = 1$).

```
im = imread('lena.png');

% create the filter
sigma = 1;
filter = fspecial('gaussian', sigma * 3, sigma);

tic;
% convolve the image with the Gaussian filter
out = imfilter(im, filter, 'same', 'conv');
time_conv = toc;

imshow(out); % show the result of the convolution
fprintf('Time for sliding convolution: %0.4f\n', time_conv);
% Time for sliding convolution: 0.0020
```

(d) Since Gaussian filter F is separable, you can compute the horizontal and vertical filters, and perform a 1D convolution with the horizontal filter and image to each row followed by a 1D convolution with the vertical filter on each column. This is possible because of the associative property of convolution. You can verify the computed filters by finding the product of the horizontal and vertical filters is equal to the original Gaussian filter ($F = vh^T$). Separability allows for a faster

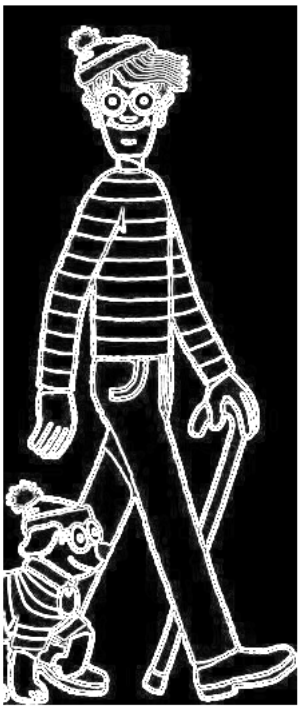
filtering process since only 2 1D convolutions will need to be computed for every pixel in the image. This is illustrated in the following code with a faster convolution time of 0.001 s.

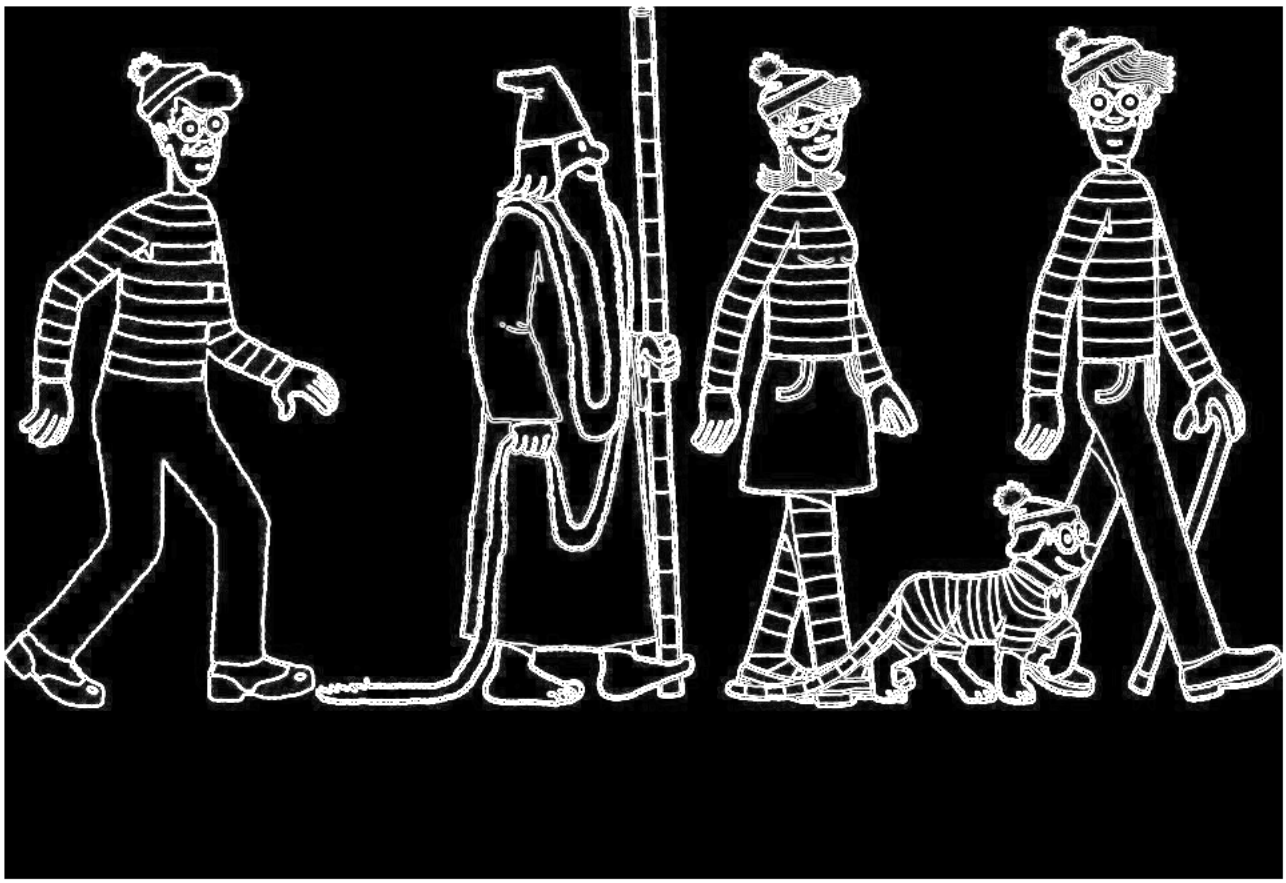
```
im = imread('lena.png');

% create the Gaussian filter
sigma = 1;
filter = fspecial('gaussian', sigma * 3, sigma);

% compute the vertical and horizontal filters of the Gaussian filter
[U,S,V] = svd(filter);
vertical_filter = sqrt(S(1,1)) * U(:,1);
horizontal_filter = sqrt(S(1,1)) * V(:,1)';

tic;
out = conv2(vertical_filter, horizontal_filter, im, 'same');
time_conv = toc;

imshow(out); % show the result of the convolution
fprintf('Time for convolution: %0.4f\n', time_conv);
% Time for convolution: 0.0010
```

3

(a) Compute the magnitude of gradients for **waldo.png** and **template.png**.

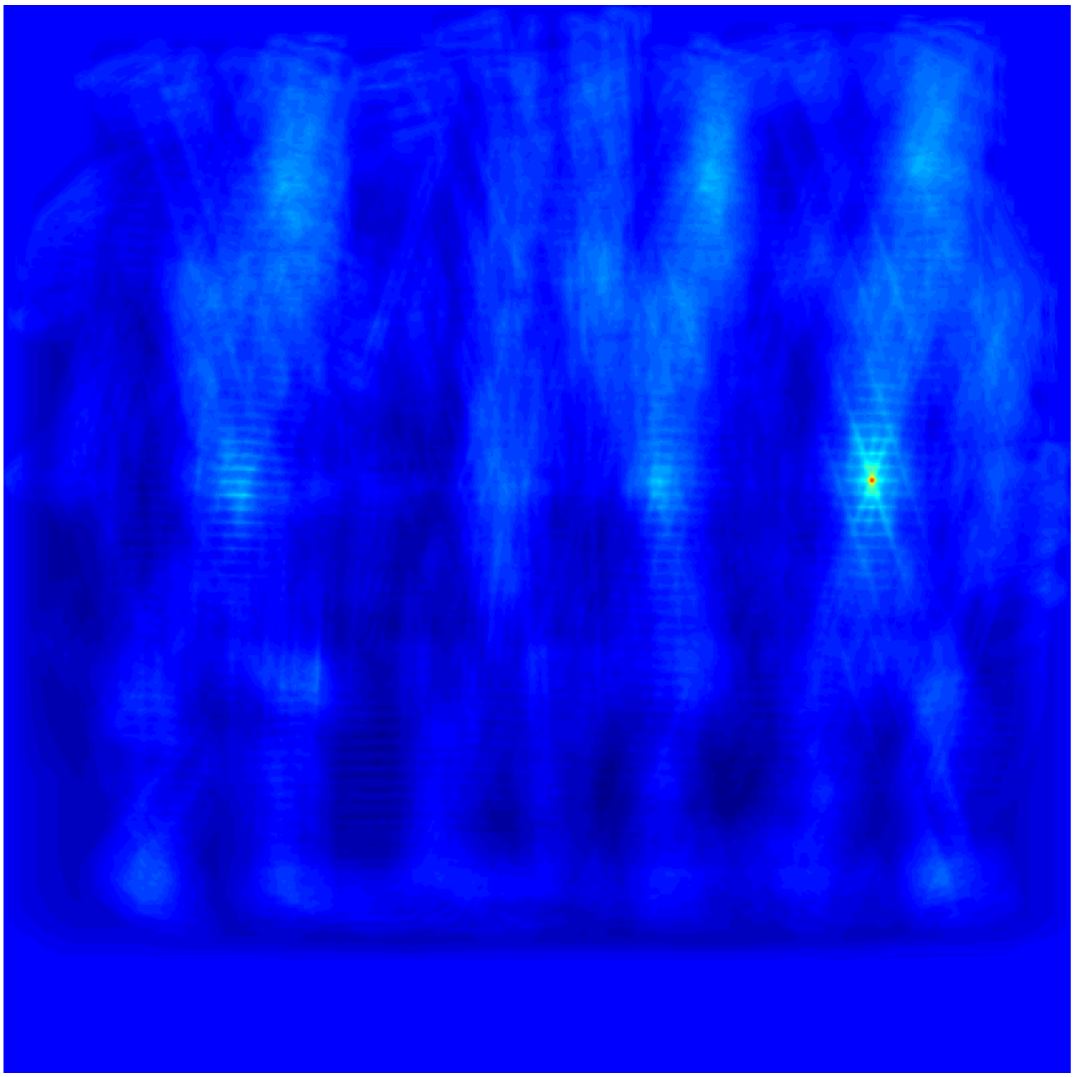
```
template = imread('template.png');
waldo = imread('waldo.png');

% compute the magnitude of gradients for the images
[template_grad_mag, template_grad_dir] = imgradient(rgb2gray(template));
[waldo_grad_mag, waldo_grad_dir] = imgradient(rgb2gray(waldo));

template_grad_mag = uint8(template_grad_mag);
waldo_grad_mag = uint8(waldo_grad_mag);

figure, imshow(template_grad_mag);
figure, imshow(waldo_grad_mag);
```

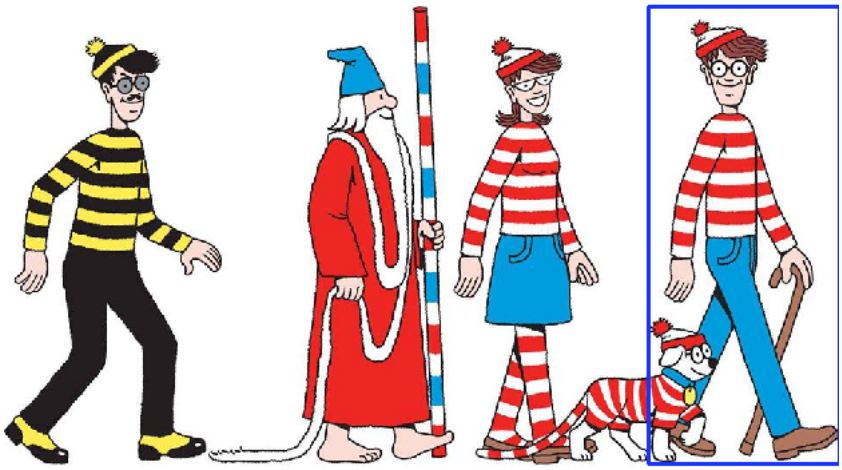




(b) Localize the template in the image **waldo.png** based on the magnitude of gradients.

```
template = imread('template.png');
waldo = imread('waldo.png');

% compute the magnitude of gradients for the images
[template_grad_mag, template_grad_dir] = imgradient(rgb2gray(template));
[waldo_grad_mag, waldo_grad_dir] = imgradient(rgb2gray(waldo));

template_grad_mag = uint8(template_grad_mag);
waldo_grad_mag = uint8(waldo_grad_mag);

% Following code reused from lecture 2:
% normalized cross-correlation
out = normxcorr2(template_grad_mag, waldo_grad_mag);

% plot the cross-correlation results
figure('position', [100,100,size(out,2),size(out,1)]);
subplot('position',[0,0,1,1]);
imagesc(out)
axis off;
axis equal;

% find the peak in response
[y,x] = find(out == max(out(:)));
y = y(1) - size(template_grad_mag, 1) + 1;
x = x(1) - size(template_grad_mag, 2) + 1;

% plot the detection's bounding box
figure('position', [300,100,size(im,2),size(im,1)]);
subplot('position',[0,0,1,1]);
imshow(waldo)
axis off;
axis equal;
rectangle('position', [x,y,size(template_grad_mag,2),size(template_grad_mag,1)], 'edgecolor',
    [0.1,0.2,1], 'linewidth', 3.5);
```





(c) Compute the image pyramid with 3 levels for **waldo.png**.

```
function out = impyramid(im)
% create the Gaussian filter
sigma = 1;
filter = fspecial('gaussian', sigma * 3, sigma);

% blur the image with the Gaussian filter
blur_im = imfilter(im, filter, 'same', 'conv');

% separate the channels of an RGB image
im_red = blur_im(:,:,1);
im_green = blur_im(:,:,2);
im_blue = blur_im(:,:,3);

% downsample the channels by a factor of 2
downsample_im_red = im_red(1:2:size(im_red,1), 1:2:size(im_red, 2));
downsample_im_green = im_green(1:2:size(im_green,1), 1:2:size(im_green, 2));
downsample_im_blue = im_blue(1:2:size(im_blue,1), 1:2:size(im_blue, 2));

% put the channels back together
out = cat(3, downsample_im_red, downsample_im_green, downsample_im_blue);
end

% compute the image pyramid with 3 levels for waldo.png
im = imread('waldo.png');
im1 = impyramid(im);
im2 = impyramid(im1);
im3 = impyramid(im2);
```

```
28
29 imshow(im);
30 figure, imshow(im1);
31 figure, imshow(im2);
32 figure, imshow(im3);
```

4

To reduce the amount of fine, detailed edges that are detected with the Canny edge detector, one can (1) raise the threshold of the low threshold, and (2) use a Gaussian filter with a larger σ.

5

(a) Compute the magnitude of gradients for an image.

```
im = imread('temple.jpg');

% compute the magnitude of gradients of the image
[imMag, imDir] = imgradient(rgb2gray(im));

imMag = uint8(imMag);

figure, imshow(im);
figure, imshow(imMag);
```