EE211A Digital Image Processing
Winter 2018
Homework 2
(Deadline: 02/19/2018 6 pm)

YAO XIE
UID: 105036239

Problem 1 Perform dilation operation on the given 9*9 image with 3*3 structuring element. Check your results in computer (MATLAB, python ...) and compare it.

Binary Image:

0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0
0	0	1	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0	0

Structuring element:

1	0	1
0	0	0
0	0	1

Result:

Dilation is one of the two basic operators in the area of mathematical morphology, the other being erosion. It is typically applied to binary images, but there are versions that work on grayscale images. The basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller.

In problem 1, after applying dilation operation on the given 9*9 binary image with the 3*3 structuring element. We obtain a new binary image shown in Fig. 1. Comparing with the original binary image (Fig. 2) and the binary image we get by applying build-in function *imdilate* in Matlab

(Fig. 3) to check the correctness of the result. From three figures, we can know that all the images before/ after dilation have the same size 9*9. So dilation will not change the size of images. Dilation is realized by point-to-point operation. It has a reference center point each time. If the value of the point is 1, the surrounding 3*3 area will convolve with the structuring element. Comparing Fig. 1 and Fig. 2, it is obvious that the numbers of 1 are increased. In other words, dilation is to expand the non-zero area in the image. Comparing Fig. 2 and Fig. 3, we can know that the result is correct.

9x9 double

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	1	0
3	0	0	1	0	0	0	0	1	0
4	0	0	0	0	1	1	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	1	0
7	0	0	0	1	0	0	1	0	0
8	0	0	0	0	1	1	0	0	0
9	0	0	0	0	0	0	0	0	0

Fig. 1. Original binary image before dilation

9x9 double

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	1	0	1
2	0	1	0	1	0	0	1	0	1
3	0	0	0	1	1	1	1	0	1
4	0	0	0	1	0	0	0	0	1
5	0	1	0	1	0	1	1	0	1
6	0	0	1	0	1	1	0	1	0
7	0	0	0	1	1	1	1	0	1
8	0	0	0	0	1	0	0	1	0
9	0	0	0	0	0	1	1	0	0

Fig. 2. New binary image after dilation

9x9 double

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	1	0	1
2	0	1	0	1	0	0	1	0	1
3	0	0	0	1	1	1	1	0	1
4	0	0	0	1	0	0	0	0	1
5	0	1	0	1	0	1	1	0	1
6	0	0	1	0	1	1	0	1	0
7	0	0	0	1	1	1	1	0	1
8	0	0	0	0	1	0	0	1	0
9	-Inf	0	0	0	0	1	1	0	0

Fig. 3. Binary image obtained by build-in function *imdilation*

Code:

```
binary = [0 0 0 0 0 0 0 0 0; 0 0 1 0 0 0 0 1 0;0 0 1 0 0 0 0 1 0;
          0 0 0 0 1 1 0 0 0; 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 1 0;
          0 0 0 1 0 0 1 0 0; 0 0 0 0 1 1 0 0 0;0 0 0 0 0 0 0 0 0];
structuring = [1 0 1; 0 0 0; 0 0 1];
binary2 = zeros(9,9);
%Binary2 = imdilate(Binary,structuring);
for i = 1:9
    for j = 1:9
        if binary(i,j)==1
            binary2(i-1,j-1)=1;
            binary2(i-1,j+1)=1;
            binary2(i+1,j+1)=1;
        end
    end
end
binary3 = binary | binary2;
```

Problem 2

Perform a non-uniform lighting compensation on the given image(Text-Image) so that text is readable after thresholding. Show the results of thresholding, before and after non-uniform lighting is compensation applied. Do not use single line thresholding functions in MATLAB, python,... that perform this operation in single line.

Result:

In this problem, I processed several operations in order to perform a non-uniform compensation on the given image.

(1) Transform the rgb image into gray-value image.

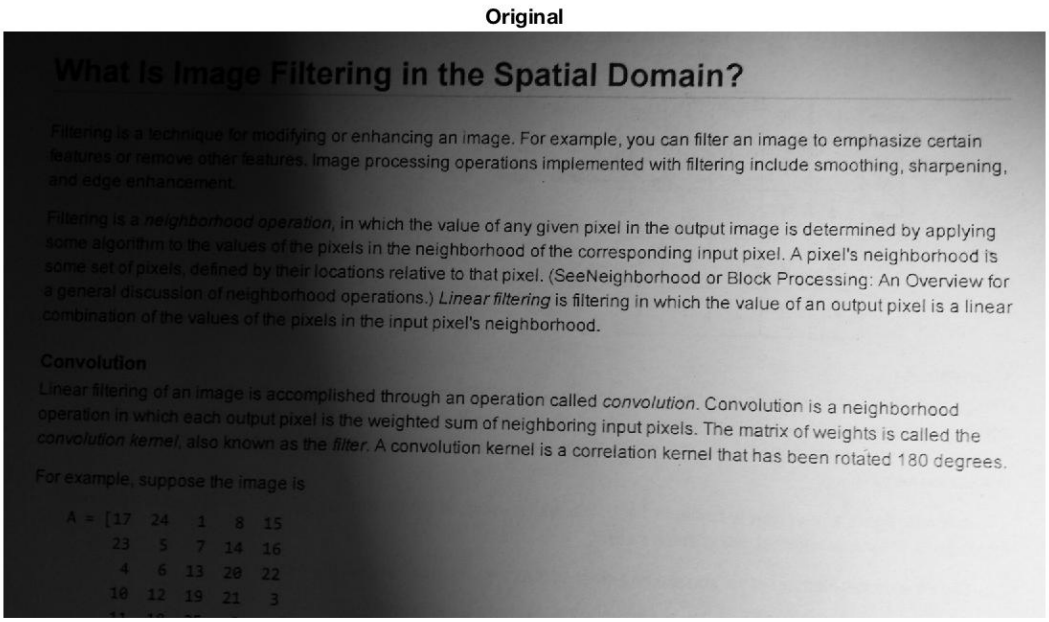

Fig. 4. Original image

(2) Perform dilation on the original image. The structuring element in the dilation is se = strel('disk',5). After trying different values, I think the result is best when it is 5. Dilation is to eliminate the text in the original image, in other words, extracting the non-uniform lighting background from the original image.

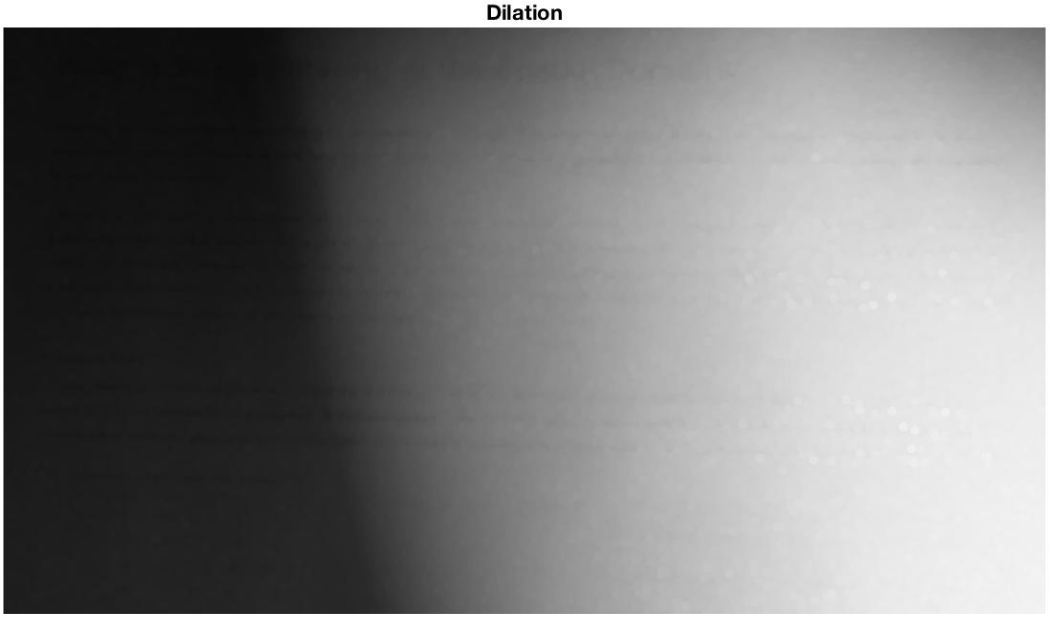

Fig. 5. Image after dilation

(3) Perform a rank filter after the dilation image. In this case, I chose the 5th brightest pixel in a 15*15 range filter. And the result is regarded are the background which has a same non-uniform light distribution as the original image. Rank filters operating on images assign the kth value of the gray levels from the window consisting of M pixels arranged according to their value to the center point of the window. The rank filter in this case improves the dilation image we get in the former step.

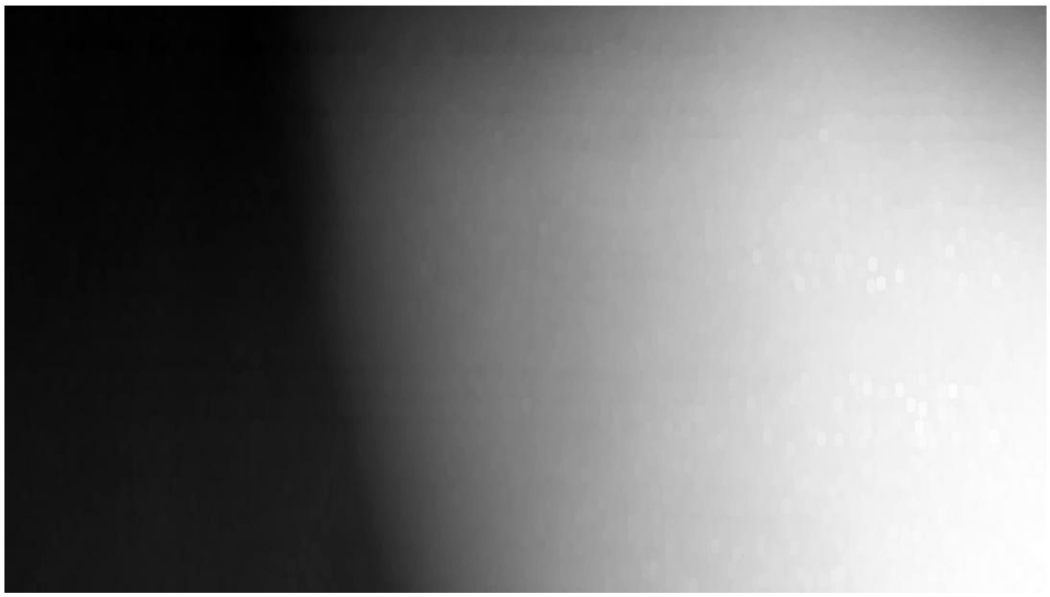

Fig. 6. Background image after rank filtering

(4) Subtracting the background image from the original image and then normalize the image. In this step, after subtracting the background, the non-uniform lighting compensation is performed. In some degree, the non-uniformity is improved and the differences between pixel values of background and texts are enhanced. However, we can not say that the result is satisfying because on the left side of the image, it is still dark and obscure. Thus thresholding is still needed.

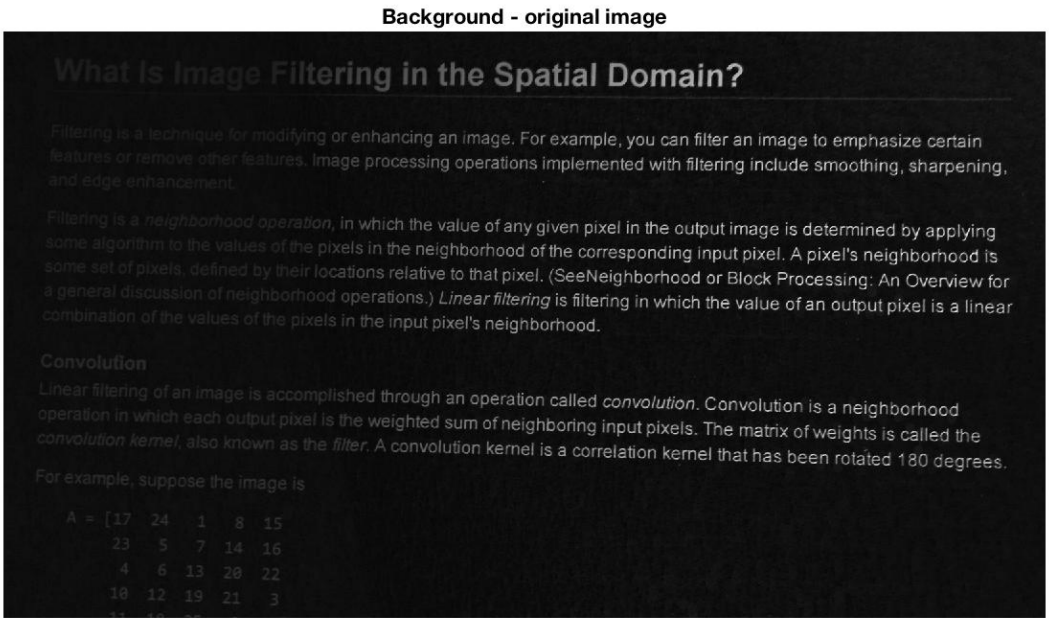

Fig. 7. Background subtracting from the original image

(5) Perform thresholding on the image we obtained in step 4. To achieve a better result, I perform a half-and-half thresholding, which means the threshold value are different in the right and left part of the image. I tried several values of thresholding and found that the image is best when the right thresholding is 0.03 and the left is 0.08.

What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a *neighborhood operation*, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. (SeeNeighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) *Linear filtering* is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

Convolution

Linear filtering of an image is accomplished through an operation called *convolution*. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the *convolution kernel*, also known as the *filter*. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

```
A = [17  24   1   8  15
     23   5   7  14  16
      4   6  13  20  22
     10  12  19  21   3
```

Fig. 8. Final image we get after thresholding

In conclusion, we perform the non-uniform lighting compensation by following the 5 steps above. The final image is satisfying and the texts can be seen clearly.

Code:

```
png_name = 'text.png';
data = imread(png_name);
data=rgb2gray(data);
[x,y]=find(data ~= 255);
data = data(x(1):x(length(x)),y(1):y(length(y)));
%data = im2double(data);
[m,n] = size(data);
figure(1)
imshow(data);
title('Original');
%first: dilation
se = strel('disk',5);
data2 = imdilate(data,se);
figure(2)
imshow(data2);
title('Dilation');
%second: rank filter, filp the dilated image right and left, up and down to
%process boundaries
data_1 = [data2(:,n:-1:1) data2(:,:) data2(:,n:-1:1)];
newdata = [data_1(m:-1:1,:); data_1(:,:); data_1(m:-1:1,:)];
background = zeros(m,n);
for i=1:m
    for j=1:n
        data3 = newdata(m+i-7:m+i+7,n+j-7:n+j+7);
        rankdata = sort((data3),'descend');
        background(i,j) = rankdata(5);
    end
end
figure(3)
imshow(background,[]);
%third: background-image
datadouble = im2double(data);
background2 = background./255;
data4 = (background2 - datadouble);
figure(4)
imshow(data4,[]);
title('Background - original image');
%fourth: globle thresholding
data5 = zeros(size(data4));
for i=1:m
    for j=1:400
         if data4(i,j) >= 0.03
             data5(i,j) = 0;
         else data5(i,j)=1;
         end
    end
end
for i= 1:m
```

```
    for j=401:n
        if data4(i,j) >= 0.08
             data5(i,j) = 0;
         else data5(i,j)=1;
         end
    end
end
figure(5)
imshow(data5,[]);
```