Biomedical Image Processing

Assignment 1 Part II

Guided Image Filtering

Dewi Kharismawati

14231619 / dek8v5

Theory

Image guided filtering is one method in edge-preserving filtering, which has a purpose to remove noise, smoothing, sharpening, texture decomposition, HDR compression, image abstraction, optical flow estimation, etc. In this assignment, we are focusing on the function of guided image for image smoothing. Image smoothing using guided filter has a similar result as bilateral filter, but it has a better edge behavior. Guided filtering gave the image output (q) by considering the content of guidance image (I) and filtering input image (p). Guide image can be either identical or different with the input image.

When the algorithm is implemented, the filtering output of each pixel I can be expressed as follow

$$q_i = \sum_j W_{ij}(I) p_j \quad (1)$$

where, W_{ij} is function of guidance image I and independent of p. With this equation, the filter is supposed to be linear.

In this assignment, we will design a function that can perform the guided image filtering with input, such as guidance image (I), filtering input image (p), radius (r), and regularization (ε). The main assumption from guided filter is a local linear model between the guidance I and output q. We also assume that q is linear transformation from I in a window wk centered at the pixel k, where

$$q_i = a_k I_i + b_k, \forall_i \in w_k \quad (2)$$

where, $ak\ bk$ are linear constant from the input picture itself, and Ii is the guidance image. Here, we use the square window of radius (r), and r = (w-1)/2. In my understanding, in this whole process of guided filtering, we are trying to get the a_k and b_k value from both I and q attributes, which include local mean, variance, and covariance. When, a_k and b_k are generated, we implement the mean of a and mean of b_k as the formula above to get the output (q). To get the a_k and b_k, we need constraints from the input p. n is unwanted component like noise and texture.

$$q_i = p_i + n_i \quad (3)$$

Also, we need to get a solution where minimize the difference between p and q, at the same time, maintaining the linear model. We get:

$$\mathrm{E(ak, bk)} = \sum_{i \in wk}((q_i - p_i)^2 + \epsilon a_k^2) \qquad (4)$$

Then, we substitute q from model (2) we get

$$\mathrm{E(ak, bk)} = \sum_{i \in wk}((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2) \qquad (5)$$

ε is regularization to regularize the a_k, so that we don't get too large a_k.

From the equation 5, we can solve both a_k and b_k.

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}, \qquad (6)$$

$$b_k = \bar{p}_k - a_k \mu_k. \qquad (7)$$

where, μ_k is the mean of I, σ_k^2 is the variance of I, $\bar{p}$k is the mean of p. Then we can solve the output of q by taking both mean from a and b:

$$q_i = \bar{a}_i I_i + \bar{b}_i, \qquad (8)$$

Implementation

Before implementing guided image filtering, the warm up is to create a local mean, variance, and covariance functions of given I, p, and window size (w).

In problem 3 implementation, I have main to provide inputs, and to call functions local_mean, variance, and covariance.

Here is the source code:

```
I = imread('cat.bmp'); %Guidance Image
p = imread('cat.bmp'); %inoput image

w = 3; %window size is given

%%%%%%%% No 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%call local mean function for I
local_mean_I = local_mean (I, w);
%call variance function for I
variance_I = variance (I, w);

%call local mean function for p
local_mean_p = local_mean (p, w);
%call variance for p
variance_p = variance (p, w);

%calculate the covariance of I and P by calling the covariance function
covariance_IP = covariance(I, p, w);
```

Local mean

In guided image, local mean is getting the mean of the sub matrix size based on the desire window size. Local mean function will be used to get the mean of I, p, a_k, b_k, also to get the variance and covariance.

Algorithm:

1. Get the argument statement, Img (input image) and window_size (desired window size). Window size usually is odd number (3,5,7,9,…)
2. Create the temporary matrix based on the desired window. In here, the window_matrix is matrix with dimension window_size $\times$ window size, and the content is 1/window_size^2. When we have window_size = 3, the result of window_matrix will be
```
0.111111111111111       0.111111111111111       0.111111111111111
0.111111111111111       0.111111111111111       0.111111111111111
0.111111111111111       0.111111111111111       0.111111111111111
```
3. Then, we get the local mean with function con2, which 2D convolving between Img matrix and window_matrix, and I choose the shape to be ‘same’. So that, I can get the same dimension as Img matrix. Therefore, we do not lose any information. Convolution here means computing mean in neighborhood based on window size and centered at each pixel.

local_mean function implementation source code:

```
function [ loc_mean ] = local_mean( Img, window_size )
%UNTITLED2 Summary of this function goes here
%   Img = input image matrix
%   window_size = window size for local mean

Img = double(Img); %convert image into double

%creating the window_matrix for the convolution 2D
%this window matrix is w*w matrix, which contains 1/(w^2)
window_matrix = 1/window_size^2 .* (ones(window_size));

%2D convolving the Image with the window matrix
%shape same means will have the sane aize as the Img size
img_mean = conv2(Img, window_matrix, 'same');

%return values
loc_mean = img_mean;

end
```

Variance

Variance is the difference between the square of the input local mean and the square of the local mean itself. If we have image I, and we want to take the variance, it can be done by the following:

$$\sigma^2 = E(I^2) - (E(I))^2$$

Algorithm:

1. Get the argument statement, Img (input image) and window_size (desired window size). Window size usually is odd number (3,5,7,9,…)
2. Compute the local mean of Img^2
3. Compute the square of Image local mean.
4. Finally, get the variance by subtracting the result of step 2 and step 3.

Variance function implementation source code:

```
function [ variance_Img ] = variance( Img, window_size);
%local_variance getting the variance of the image.
%variance is for indentify the change of intensity in the image
%   Img = input image
%   window_size = window size desire for the

Img_square = Img.^2;

local_mean_img_square = local_mean(Img_square, window_size);

loc_mean_img = local_mean(Img, window_size);

%loc_variance is derived from (local_mean X^2) - (local_mean X)^2

variance_Img = local_mean_img_square - (loc_mean_img.^2);

end
```

Squaring before taking the local mean and after compute the local mean resulting in very different result. This difference is what we call as the variance. In this image processing case, greater the actual variation in the values of intensity (for instance), the greater variance will be.

Covariance

The idea of covariance is similar to variance. But, in here, we have two different variables, let say that another image is p). In this case, we can get the covariance by the following equation.

$$\sigma^2 = E(Ip) - (E(I)\times E(p))$$

Algorithm:

1. Get the argument statement, Img (input image) and window_size (desired window size). Window size usually is odd number (3,5,7,9,…)
2. Compute the local mean of (Img $\times$ p)
3. Compute the local mean of Img
4. Compute the local mean of p
5. Finally, get the variance by subtracting the result of step 2 and (step 3 $\times$ step 4).

Covariance function implementation source code:

```
function [ cov ] = covariance( Img, p, window_size )
%covariance getting the covariance from Image and guided image
%   Img: the original image
%   p: guided image
%   window_size: desired window size

local_mean_img_p = local_mean(Img.*p, window_size);

local_mean_img = local_mean(Img, window_size);

local_mean_p = local_mean(p, window_size);


%covariance is derived from (local_mean XY)-(local_mean X * Local_mean Y)
cov = local_mean_img_p - (local_mean_img .* local_mean_p);
end
```

the function of covariance also exactly the same as the variance. However, in covariance we are measuring how much change occur in between two images. The change can be anything, but, right now, we are focusing on the intensity of the images. The covariance will be large if the change is large, otherwise it will be small.

Image Guided Filtering

As we have discussed in theory, we need to find a_k and b_k to do the filtering, where a_k and b_k are linear constant from the input picture (p) itself. The image guided filter is implemented in myimguidedfilter.m function, where it needs several arguments, which are guidance Image (I), input image (p), radius (r), and regularization (epsilon). From equation (6), we can solve ak bk

$$a_k = \frac{\frac{1}{|\omega|}\sum_{i\in\omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon},$$

$$b_k = \bar{p}_k - a_k \mu_k.$$

ak, can be solved if we have mean, variance, covariance, and regularization. Therefore, we will find each of the value by following the algorithm as follow:

The algorithm to get the output q as describe in the part 3.1 algorithm 1 is as follow:

1. The input guidance image (I) and input image (p) are still in intensity range of 1-255. We need to convert it into 0-1 range, because guided image only works in range 0-1.
2. Get the window size. From the main function, we want to pass radius r. However, in our personal local mean, we use window_size as a function argument. Therefore, we need window_size, which can be retrieved from w = 2r+1.
3. Compute the mean of both images input I and p. We use our personal local_mean that has been written for problem number 3.
4. Get the correlation of guidance Image I and correlation of guidance image and input image together. Correlation I is derived from the local mean of I^2 (because it is only one variable). And correlation Ip is derived from the local mean of (I.×p). Both correlation will be used in the calculation of variance and covariance in the next step
5. Retrieve the variance of I and covariance of I and p. As we have discussed the covariance is to measure the change in intensity from the input image. We can get the variance and covariance by subtracting the local mean of the variables from correlation of the variables.
6. We have all the required value to compute ak and bk. To get the ak, we divide the covariance of Ip over variance I plus regularization. To get bk, we subtract local mean of input image by a multiply local mean input image I.
7. Then, we get both ak and bk local means.
8. Getting the output q from multiply local mean a and I and add the result with local mean b.

9. Get the output q, and plot the image in the figure.

The source code for myimguidedfilter is as follow:

```
function [ q ] = myimguidedfilter( I, p, r, epsilon )
%myimguidedfilter is a filtering method to smooth images
%   I = guidance image with range intensity 0-255
%   p = input image with range intensity 0-255
%   r = radius desired for the window size to get the local mean
%   epsilon = regularization to keep the ak controlled (not too big)

%converting the intensity of I and p from 0-255 to 0-1
I = mat2gray(I);
p = mat2gray(p);

%getting the window size, in this function we are going to use our own
%local mean function, which it needs window_size (based on number 3
%problem), to get the local mean.
w = 2*r + 1;

%step 1
%computing the mean and covariance of both images input
mean_I = local_mean(I, w);

mean_p = local_mean(p, w);
corr_I = local_mean(I.*I, w);
corr_Ip = local_mean(I.*p, w);

%step 2
%getting the variance of guidance image and covariance of guidance and
%input images using mean and correlation we compute in step 1
var_I = corr_I - mean_I.*mean_I;
cov_Ip = corr_Ip - mean_I.*mean_p;


%step 3
%get the value of a from variance divided by the sum of variance guidance img and epsilon
%getting the value of b by substracting the multiplication of a and guidance
%image local mean from input image local mean
a = cov_Ip ./ (var_I + epsilon);
b = mean_p - a.*mean_I;

%step 4
%retrieve the local mean of a and b
mean_a = local_mean(a, w);
mean_b = local_mean(b, w);

%step 5
%compute the output by multiply mean_a and I and add the result with mean_b
q = (mean_a.*I) + mean_b;

end
```

main for calling myimguidedfilter, based on the desired radius and epsilon from the user. In this implementation, we tried 9 trial that are similar to the paper. Then, we can see the result and compare it to the paper.

```
%call the myguidedfilter function to get the output image
%function syntax q = myimguidedfilter(I, p, r, epsilon);

figure,
subplot(1,2,1)
imshow(p);
title('original image');
subplot(1,2,2)
imhist(p)
title('histogram of original image')
```

```
figure,
subplot(3,3,1)
r = 2;
epsilon = 0.1^2;
tic
q1 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q1);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,2)
epsilon = 0.2^2;
tic
q2 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q2);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,3)
epsilon = 0.4^2;
tic
q3 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q3);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);



subplot(3,3,4)
r = 4;
epsilon = 0.1^2;
tic
q4 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q4);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,5)
epsilon = 0.2^2;
tic
q5 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q5);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,6)
epsilon = 0.4^2;
tic
q6 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q6);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);


subplot(3,3,7)
r=8;
epsilon = 0.1^2;
tic
q7 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q7);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,8)
epsilon = 0.2^2;
tic
q8 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q8);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);

subplot(3,3,9)
epsilon=0.4^2;
tic
q9 = myimguidedfilter(I, p, r, epsilon);
toc
imshow(q9);
title(['r=' num2str(r) ' and epsilon=' num2str(sqrt(epsilon)) '^2']);
```

```
figure,
subplot(3,3,1);
imhist(q1);
title('r=2 and epsilon=0.1^2');
subplot(3,3,2);
imhist(q2);
title('r=2 and epsilon=0.2^2');
subplot(3,3,3);
imhist(q3);
title('r=2 and epsilon=0.4^2');
subplot(3,3,4);
imhist(q4);
title('r=4 and epsilon=0.1^2');
subplot(3,3,5);
imhist(q5);
title('r=4 and epsilon=0.2^2');
subplot(3,3,6);
imhist(q6);
title('r=4 and epsilon=0.4^2');
subplot(3,3,7);
imhist(q7);
title('r=8 and epsilon=0.1^2');
subplot(3,3,8);
imhist(q8);
title('r=8 and epsilon=0.2^2');
subplot(3,3,9);
imhist(q9);
title('r=8 and epsilon=0.4^2');
```

Result

Original image and the histogram

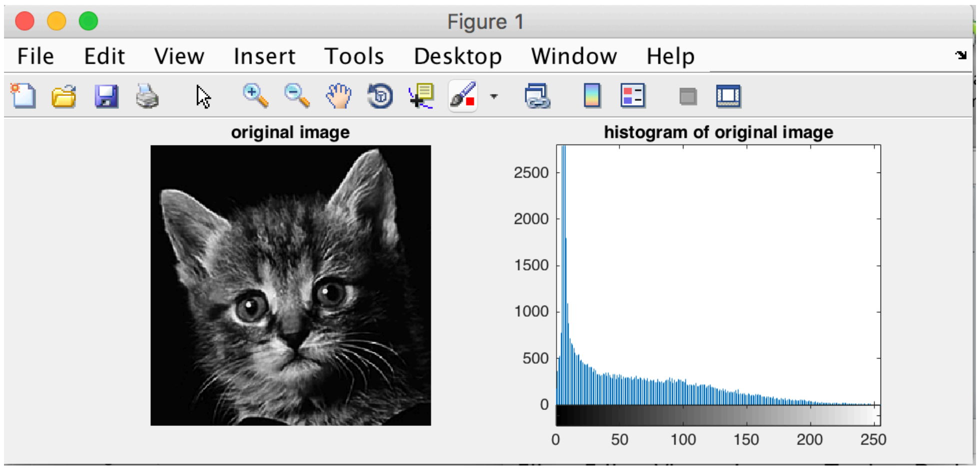


Figure 1
File Edit View Insert Tools Desktop Window Help
original image
histogram of original image
2500
2000
1500
1000
500
0
0
50
100
150
200
250

The result of running the myimguidedfilter function with combinations of 3 values of r and epsilon. We get the result figure bellow

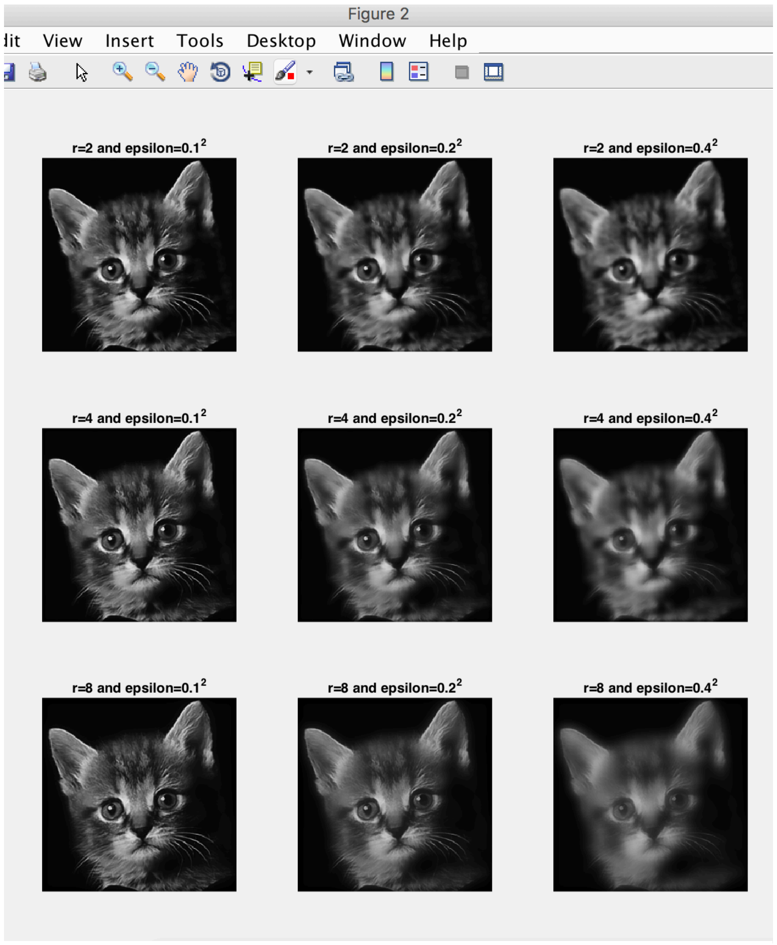


Figure 2 result from myimguidedfilter

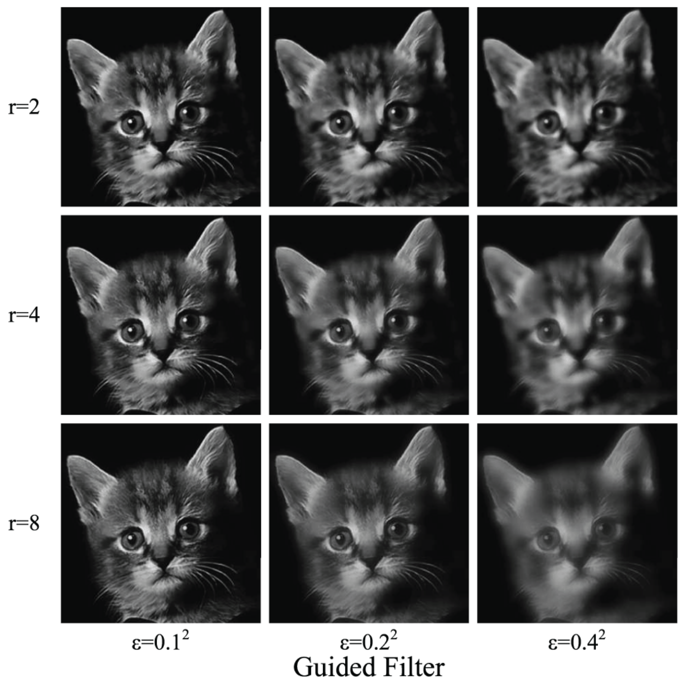


Figure 1 result from paper

Histogram from corresponding myimguidedfilter result

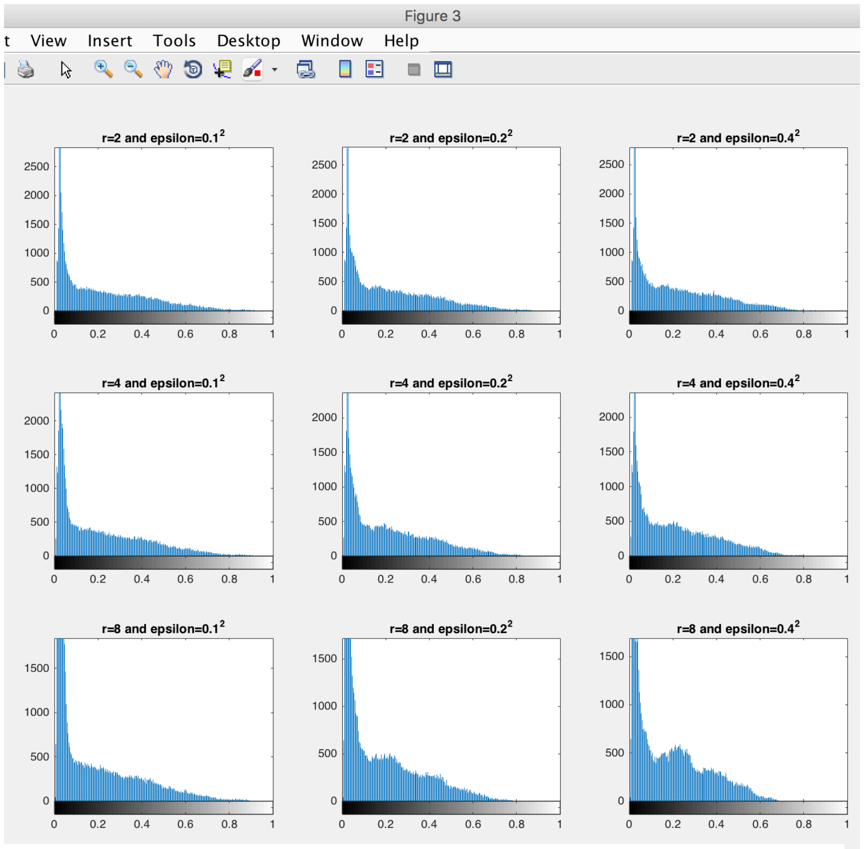
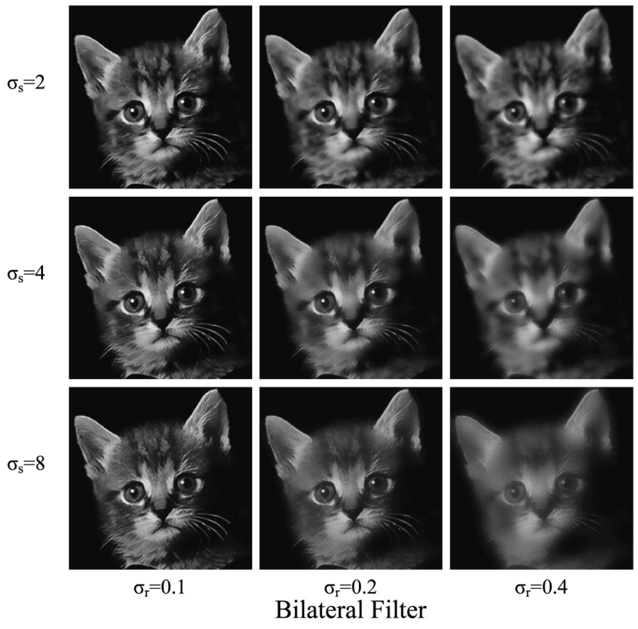


Figure 3 corresponding histogram from myimguidedfilter



As we compare both result, they provide the same exact result for the same r and epsilon. This means that our implementation is successful. With the simple guided filtering algorithm, we can get the exact same result as in bilateral filter.

In this experiment, we want to see how guided filter can smoothen input image p with guide image I, where I and p is identical. Smoothing in guided filtering is edge-preserving smoothing, where we want to remove noise or texture and get a more accurate intensity, but at the same time, we want to keep the sharp edges in the image. From the experiment that has been done, we can observe the influenced of window radius and regularization in smoothing. We use a combination between 3 different values of r and ε. We can see the effect of window radius in the vertical directions, and the effect of epsilon in the horizontal directions. There are three cases that I want to point out:

1. In the first row, we can see the effect of increasing ε in the small radius. We can compare the original image, first image, and that the last image (in the first row). The third image

is smoothed even though it is not that much, and there is no significant change in the histogram. Therefore, the bigger the ε, it is smoothing more. It also can be seen in the 2nd and 3rd row.

2. Then, let see the influenced of window radius increasing in image smoothing by observing the first column of the result by comparing the first and third image (from that column), and compared it to the original image. In a glance, there is no change at all in the image. The cat is still sharp in every part. But, I notice the cat's forehead is less textured. I can see almost M shape in the third image in the column. Also, we can see from the histogram of the first column, there is a big difference in the histogram distribution. From here, we can see that even though we have increasing radius, but if we have a small ε, it will not be smoothed that much.
3. The bigger both r and ε, more smoothed image we get. However, when we put too big r and ε, it will be too smooth and we could lose information from the image. We can see from the last image, it is smoothed too much, but we can still see clear edges in the cat's ears. The transition from background to the ears are still clearly displayed. Therefore, we still need to balance up between r and ε, so that we can enough reduce the noise but not removing too much information.

Talking about the edge preserving, guided filter uses two cases to keep the sharp edge. If the variance (σ^2) is smaller than ε (no edge identified), are smoothed (this is called the flat patch: case 2). Whereas, if the variance (σ^2) is larger than ε (edge may be identified) (this is called high variance: case 1), it is not smoothed. This is pretty much how the guided image smoothing, yet maintained the edge of an input image.

From the experiment implementation, I put tic toc to get the run time of myimguidedfilter function. It interesting to see the result of the time elapsed from each function call. These experiments are run in a macbook pro 2017, with core i7, and RAM 16GB. Following are the running time for the corresponding case:

```
Elapsed time is 0.124302 seconds.
Elapsed time is 0.004578 seconds.
Elapsed time is 0.003868 seconds.
Elapsed time is 0.006716 seconds.
Elapsed time is 0.006642 seconds.
Elapsed time is 0.005429 seconds.
Elapsed time is 0.013409 seconds.
Elapsed time is 0.013645 seconds.
Elapsed time is 0.013998 seconds.
```

The first result seems out from the other, this is because the first call is initiating the execution. I tried to see the effect if I call the function before the first case, it turns out that the time is similar to the 2nd function called. From the time result, there is no particular pattern in running time. However, case with the larger window radius has a longer execution time. This is because I use local_mean function that I wrote in problem 3 to get the mean in guided filter case. If I use moving sum as algorithm 2 described, or using integral image to get the mean, I believe the running time will be a lot smaller.

To implemented the guided filtering, we need to be careful on using the intensity range. It only works with 0-1 range of image, but not 0-255. This is happened because with the double calculation, a and b values are ranged between 0-1, and also their means. Therefore, when we use range 0-255 in last calculation of retrieving q, guidance I (with intensity range 0-255) will multiplied by double less than one, it will not give the significant change to guidance I. Therefore, there is no smoothing can really be seen. Here is the result of experiment using 0-255 intensity range and in double type. There is no smoothing displayed in the result.

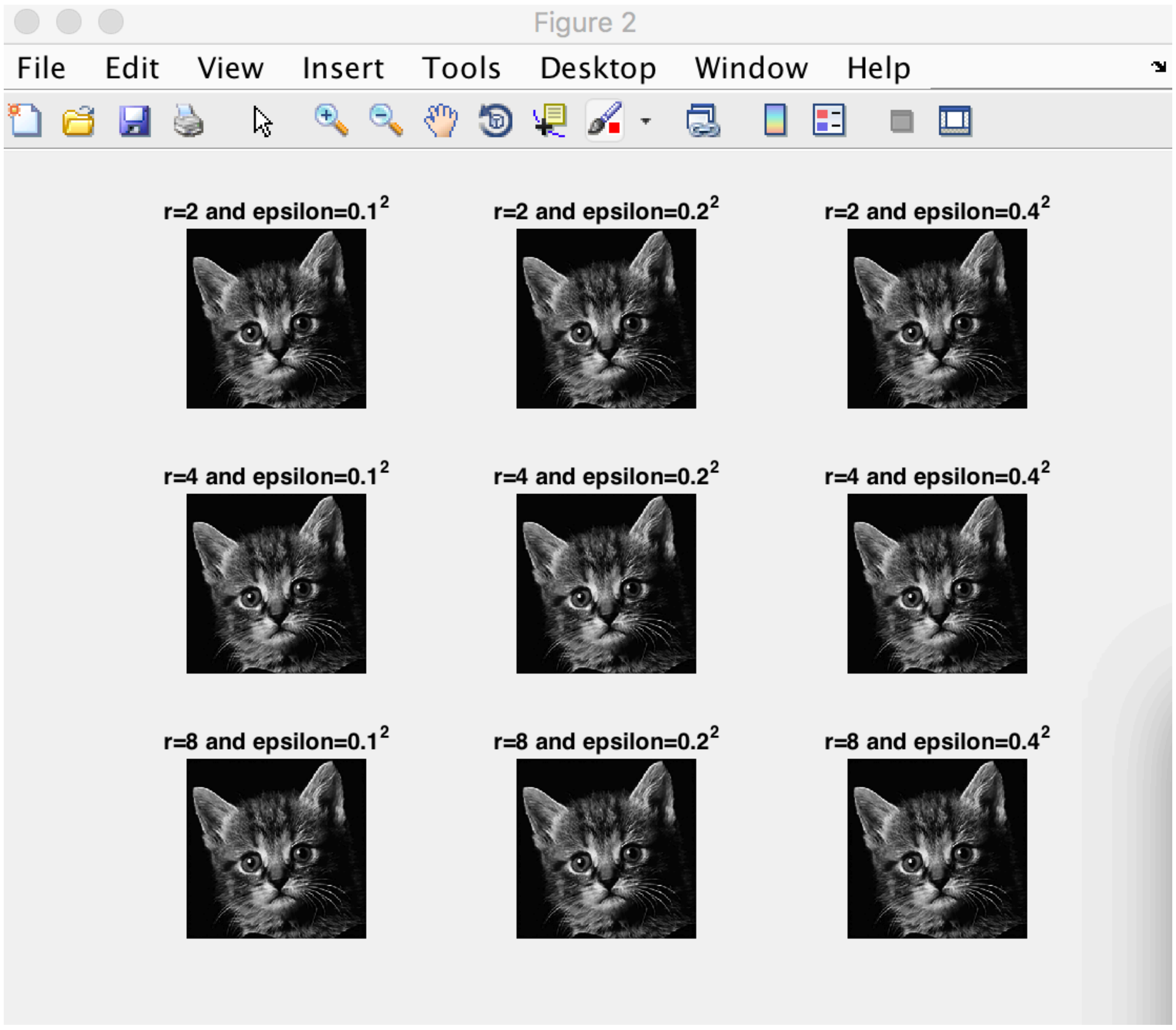


Figure 4 guided filter with intensity range 0-255

Advantage

As we have already discussed in the result, there are a lot of advantages from using guided filtering. As well as, it able to solve other filtering methods problem. The advantages are listed as follow:

1. It is an edge-preserving filter. Guided filter able to smooth images, removing edges and texture, yet it can maintain sharp edges. Therefore, we will not lose information that brought by the edges.

2. Guided filtering is more generic in function rather than only for smoothing, because it is proven to be effective as well for other functions, such as HDR compression, flash/no-flash denoising, and haze removal.
3. It is non-iterative, which means smoothing image and still retaining image edge details.
4. Guided filter can preserve gradient in the input image. This means, we can keep the gradient information in case we need to use gradient.
5. And the most important advantage, guided filter runs in linear time complexity O(N) which the complexity is proportional the amount of pixel in the image. This linear complexity is non-approximate algorithm, which is not influenced by the window radius and the intensity range. The main problem of complexity in guided filter is in the computation of mean in I and p, also in a and b with window radius of r. But, this can be normalized with other algorithms, such as integral image and moving sum to get O(N). This is not implemented in this assignment.

Therefore, guided image is included as one of the most fast and accurate filtering method.

Disadvantage

There is one major disadvantage from guided filter, this limitation is similar to other explicit filters. It may exhibit halos, unwanted and unavoidable smoothing of edges, near some edges. There are some cases, such as when we want to smooth the strong texture, but the weaker edges are also become smoothed in effect of the strong texture smoothing.

Conclusion

In conclusion, guided image filtering is one methods of filtering that able to filter images with the help of guidance image. Guided image filtering is basically an edge preserving filter. Guidance image can be identical or different with the input image p. With the algorithm of guided filter, we can get the exact result as bilateral filter. However, guided filter is more effective and efficient compare to bilateral filter because guided filter has a linear complexity of O(N), which the complexity is proportional the amount of pixel in the image. This linear complexity is non-approximate algorithm, which is not influenced by the window radius and the intensity range. This is happened because guided filter does not use any iterations at all. In addition, it also solved the bilateral problem gradient distortion. Guided filter is preserving the gradient. Therefore, guided image is faster, yet more accurate than bilateral filter. However, there is one drawback from guided filter, which may exhibit halos near some edge. These halos are unavoidable, and happened when filters are forced to smooth some edges that we do not want to be smoothed.