IP Coursework Two

Ting QIAO

Dec 2016

Note: In this coursework, the statical data as each parameter changing is recorded. The code is in statistic_result.m file. The data is stored at Statics_data.txt. However, I don't have enough time to analyse them.

Coursework report

Basic Section

1. **Define in your own words what an Integral Image is, and describe in detail an algorithm to calculate it.**

The Integral Image method is aiming to calculate the result of a patch of image in a relative 'tricky' way. The result not necessary the sum of a patch. It could be the product of the the pixel in the patch as well. The normal solution only focus on the patch itself. Intuitively, the calculation of the specific patch will not benefit the further calculation and will not do any form of pre-computing for the falling patches. On the contrast, Integral Image method considering the image as a whole. Integral Image simplify the patch calculation by pre-computing the result for each pixel in the image.

The Integral algorithm is starting from the top-left corner of the whole image, iteratively loop through every pixels in the image. For each pixel, the result of Integral Image algorithm is the sum(Could be product as well, depends on the purpose) of the results from the pixel at left top area(including the pixel itself and the row and column the pixel is locating at) of the image. After calculating the result of each pixel in the image, the sum(product) result for the specific patch is the result of the pixel at right-bottom of the patch(in the patch) subtract the result at the pixel which is located at the top-right of the patch(one pixel out the patch), subtract the result at the pixel which is located at the left-bottom of the patch(one pixel out the patch), and then add the result from the pixel located at the left-top of the patch(diagonal one pixel out the patch). For the product result, the operation will be multiplication and division.

2. **Explain how the Integral Image Method can be used to accelerate template matching.**

The basic idea of template matching algorithm is for each pixel(reference pixel) in the image, find a set of in searching area pixels that similar to the reference pixel(the surrounding area of the pixel are similar). The similarity is measured by the Sum of Squared Differences. After find the similar pixels, the pixels was weighted sum up to the new estimating pixel. The relative position of selected patch is represented by the offset. Since all of the pixels in the image need to be smoothed, we need to calculate the SSD for each pixels with each possible offsets.

The Integral Image method can be used to calculate the SSD for each pixels in the image with a specific offset. It will consist the third dimension of the image. Then by weighted sum the third dimension pixel values. The denoised pixel can be estimated.

Pseudocode:

```
Function BuildIntegralImageArray(Image)
IntegralImagesWithDifferentOffsets = ImageArray(Number of possible offsets);
For each possible offset in Searching Window
    %Part One, Calculate the Integral Image
    ImageShift = ShiftImage(Image, Offset_x,Offset_y);
    ImageDifference = Image - ImageShift;
    SquaredDifference = Elementwise Multiplication(ImageDifference,ImageDifference);
    %Calculate the Integral Image
    SumImage = CalculateIntegral(SquaredDifference);
    IntegralImagesWithDifferentOffsets(Offset) = SumImage;
End For
Return IntegralImagesWithDifferentOffsets;
End Function

Function CalculateIntegral(Image)
    SumImage = Zeros(Same Dimension with Image);
    %Calculate the Integral Image
    For each Row in Image
        For each Pixel in the Row
            SumImage.ValueAt(Row,Pixel) = SumImage.ValueAt(Row-1,Pixel-1) +
            SumOfPartRow(Row,Start,End) + SumOfPartColumn(Pixel,Start,End)
        End For
    End For
    Return SumImage;
End Function

%Evaluate the sum at specific X,Y
Function EvaluationAt(IntegralImage,X,Y,HalfPatchSize)
    SumAt(X,Y) = IntegralImage.ValueAt(X+HalfPatchSize,Y+HalfPatchSize)-
    IntegralImage.ValueAt(X+HalfPatchSize,Y-HalfPatchSize-1) -
    IntegralImage.ValueAt(X -HalfPatchSize-1,Y+HalfPatchSize)+
    IntegralImage.ValueAt(X -HalfPatchSize-1,Y-HalfPatchSize-1);
End Function
```

Parameters needed for calculating the Integral for several offsets is the searching window size. The size of searching window indicating how many possible offsets should be considered. It also decides how many Integral Images will be generated. The offsets was used to build the DifferenceImage whose value is the value for Integral Image.

3. Details in basic SSD implementation.

The template matching for the pixel at Image(Row,Column).

Pseudocode:

```
%Concatenate Image row-wise
Image_1_2 = Concatenate(2,Image,Image);
Image_1_3 = Concatenate(2,Image_1_2,Image);
%Concatenate Image column-wise
Image_2_3 = Concatenate(1,Image_1_3,Image_1_3);
Image_3_3 = Concatenate(1,Image_2_3,Image_1_3);
Radius = SearchingWindowSize / 2;
HalfPatchSize = PatchSize / 2;

For Offset_Row = -Radius:Radius
    For Offset_Column = -Radius: Radius
        Shift_Row = Row + Offset_Row;
        Shift_Column = Column + Offset_Row;

        DiffSum = 0;
        For PatchRow = -HalfPatchSize: HalfPatchSize
            For PatchColumn = -HalfPatchSize: HalfPatchSize
                Ori=Image.ValueAt(Row+PatchRow,Column+PatchColumn);
                Off=Image.ValueAt(Shift_Row+PatchRow, Shift_Column
                +PatchColumn);
                Diff = Ori - Off;
                DiffSum = DiffSum + Diff^2;
            End For
        End For
    End For
End For
```

The problem I want to talk about here is avoiding the edge problem. For the referencing pixel near the edge, their searching window could exceed the edge. In order to mitigate the influence of the edge problem, I concatenated the image as a 3X3 new matrix. Each cell of the matrix is the original image. If we use the normal template matching method, the speed of processing will not be affected dramatically. However, if we use the Integral Image method, the speed will be slowed roughly 9 times. The reason is the size of Integrated Image is increased.

4. Details in SSD implementation with Integral Image

The pseudo code of generating the Integral Image with different offset have been shown at the Question 2. Given original pixel at Image(Row,Column).

```
%Concatenate Image row-wise
Image_1_2 = Concatenate(2,Image,Image);
Image_1_3 = Concatenate(2,Image_1_2,Image);
%Concatenate Image column-wise
Image_2_3 = Concatenate(1,Image_1_3,Image_1_3);
Image_3_3 = Concatenate(1,Image_2_3,Image_1_3);
Radius = SearchingWindowSize / 2;
HalfPatchSize = PatchSize / 2;

DistancesWithOffset = Zeros(3,Number_Of_Offsets)
%For Each Possible offset
For Offset_Row = -Radius:Radius
    For Offset_Column = -Radius: Radius
        DistancesWithOffset(1,Counter) = Offset_Row;
        DistancesWithOffset(2,Counter) = Offset_Column;
        ImageShift = ShiftImage(Image, Offset_x,Offset_y);
        ImageDifference = Image-ImageShift;
        SquaredDifference=Elementwise.
        Multiplication(ImageDifference,ImageDifference);
        SumImage = CalculateIntegral(SquaredDifference);
        Distance = EvaluationAt(SumImage,Row,Column,HalfPatchSize);
        DistancesWithOffset(3,Counter) =Distance;
        End For
    End For
End For
Return DistancesWithOffset
End Function
```

- In this function, I used the 'imtranslate' function to shift the image. The problem might affect the result is the parameter of the 'imtranslate' function. The translate's coordinate frame is using the left top of the image as origin. The x-axis is from left to right and the y-axis is from top to bottom. So if the offset is Row=-5, Column = -5, then the parameters in the 'imtranslate' function should be [5,5].

- Comparing the result between naive template matching and Integral Image template matching can partially check the correctness of two algorithm. However, the accurate correctness needs to be proved by logic.

5. When the Naive method is faster than the Integral Image method?

Naive method and Integral Image method which is better is depends on the patch size and the number of points we want to calculate the Sum of Squared Difference. It may also depends on the padding area which was used to avoid the edge problem.

The Naive method only calculate the squared difference in the patch and sum them up. For each point we want to calculate the SSD, assume the patch size is N, it will take 3N^2 time.

The Integral Image method iteratively calculate the 'Integral' of the sum square. For a continuous image with size Row and Col, it have to take Row*Col for pre-computing. But the patch sum can be calculated in linear time.

Assume we have K different points needs the SSD, for the naive method, the cost will be 3*K*N^2. For the Integral Image method, the cost will be Row*Col + K. We want the cost of Naive method is less than Integral Image method.

Assume the N, Row, Col are already been decided, the smaller K is , the cost of naive method is relatively smaller.

Assume the K, Row,Col are already been decided, the smaller N can reduce the cost of naive method. The threshold is XXXXXXXXXXXXXX

6. Performance and limitation for both methods.

Both algorithms take a long time to process one image. Compared with convolution algorithms, both of the methods needs two much computation for each pixel. Searching the similar pixels is expensive.

Besides, Both algorithm needs a lot of parameters, find the proper parameters could be a difficult optimisation problem.

7. Deal with the borders.

There are several ways to deal with the borders. The purpose of extending the border is avoid the offset and patch exceed the image. In this coursework, the basic section's border problem was handled by duplicate and concatenating the images to a 3X3 matrix. Each cell is the original image. We only focus on the central image and the rest 8 images is concatenated for the exceeding patch. The reason for selecting this method is it is easily to be implemented. The disadvantage of this method is it will increase the image size dramatically(9 times). It will lead to very slow speed for the Integral Image method.

In the advanced section, the border was extended by using the 'padarray' function. The extended border was created by mirroring the pixels near the border. The extended border was created symmetrically about the original border. For all four edges, the border are extended.

The border can be extended by using different methods, fulfil the extended border by zero is another way. However, in this case, fill the extended border with the information from the image itself is better.

Advanced Section

1. Pseudocode

```
Function Non_Local_Means_Filter(Image, Sigma, h, patch_size,window_size)
    ExtendImage(Image, [patch_size/2],'symmetric','both');
    IntegralImages = ImageArray;
    For each possible offset
        Translated = TranslateImage(Image);
        Difference = Translated - Image;
        DiffSquare = Difference ^ 2;
        IntegralImage = CalculateIntegral(Image)
        IntegralImages(offset) = IntegralImage;
    End For

    Result = Zeros(Same dimension with Original Image)
    For each Pixel(X,Y) in the Image
        WeightedSum = 0;
        SumOfWeight = 0;
        For each IntegralImage in IntegralImages
            Distance = EvaluationAt(IntegralImage,X,Y,HalfPatchSize)
            Weight = ComputeWeight(Distance);
            WeightedSum = WeightedSum + Weight * Image.ValueAt(X,Y);
            SumOfWeight = SumOfWeight + Weight;
        End For
        Result(X,Y) = WeightedSum / SumOfWeight;
    End For
End Function
```

A. Extended size

The size of extending edge is depends on the patch size and searching window's size. It is half of patch size add half of searching window's size. The reason is that the extreme circumstance of the patch exceed the original image is that the pixel at four corners and with searching window centre to it.

B. Extended pixels' position

The pixel's location in the extended image is different from the original image. Both the Row and Column value in the extended needs to be add the width and height of the extended area. It will make sure that we can access the right corresponding pixel.

C. Compute weighting

The weight is calculated by using the formula. The parameters are the distance(d), decay parameter(h), standard deviation(sigma), and the patch size.

$$w(p,q) = e^{-\frac{\max(d^2 - 2\sigma^2, 0)}{h^2}}$$

D. Result

The result of estimated pixel is not the weighted sum of estimated pixel from all patches. We have to divided it with the sum of weights to normalise it.

2. Different images will have different responses to the same parameter sets. Here, for each parameter we only focus on one image.

• Window Size:

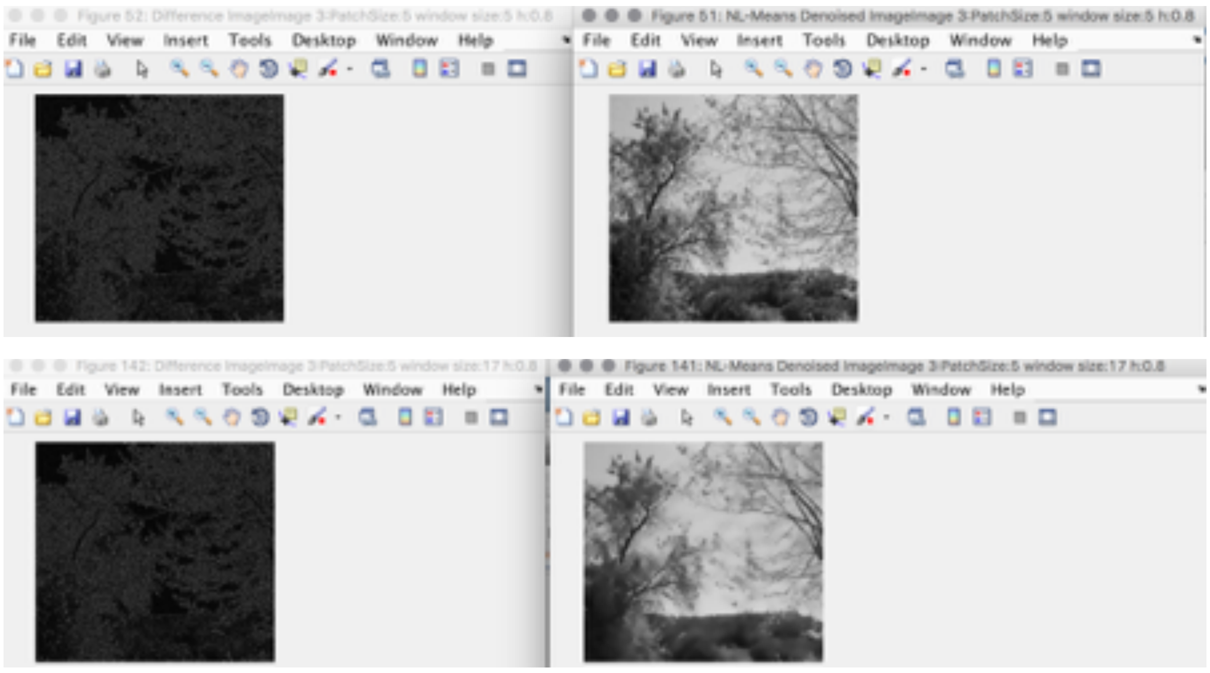
Figure 52: Difference Imageimage 3 PatchSize:5 window size:5 h:0.8
Figure 51: NL-Means Denoised Imageimage 3 PatchSize:5 window size:5 h:0.8
Figure 142: Difference Imageimage 3 PatchSize:5 window size:17 h:0.8
Figure 141: NL-Means Denoised Imageimage 3 PatchSize:5 window size:17 h:0.8
File Edit View Insert Tools Desktop Window Help

The image demonstrated that the different window size can have significant influence on image. The too large window size will leads to image been blurred(or you can say more smooth).

• patch size

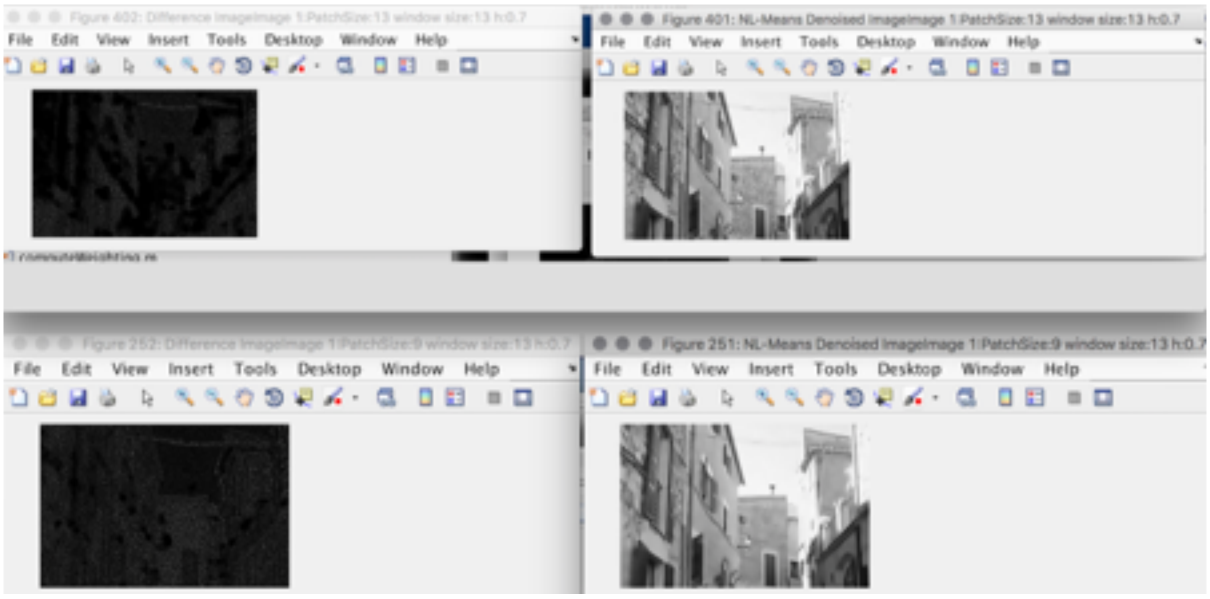
Figure 402: Difference Imagelmage 1:PatchSize:13 window size:13 h:0.7
Figure 401: NL-Means Denoised Imagelmage 1:PatchSize:13 window size:13 h:0.7
Figure 252: Difference Imagelmage 1:PatchSize:9 window size:13 h:0.7
Figure 251: NL-Means Denoised Imagelmage 1:PatchSize:9 window size:13 h:0.7
File Edit View Insert Tools Desktop Window Help

The result demonstrates that if the size of patch is larger, the detail of the image will be kept. However, the smaller patch size will make the filtered image more smooth. One interesting thing is that, some of the texture in the image will be distorted.

• h

The parameter h will affect the weight calculation for each pixel. The following images are the image with h = 0.9 and h = 0.1;

h = 0.9:

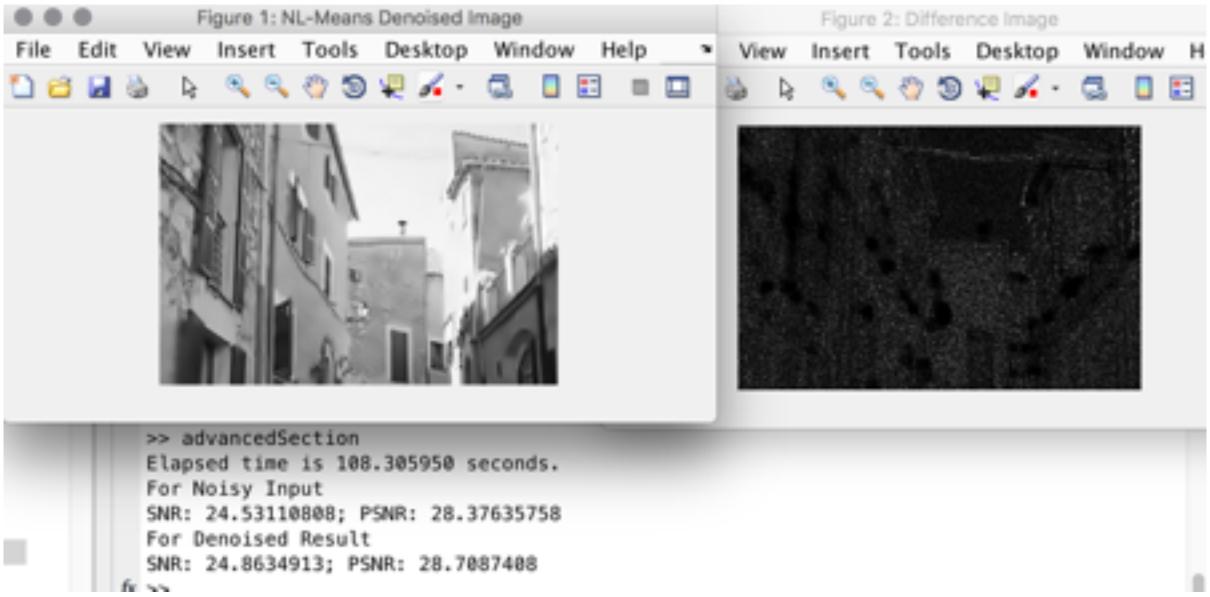
Figure 1: NL-Means Denoised Image
Figure 2: Difference Image
File Edit View Insert Tools Desktop Window Help

```
>> advancedSection
Elapsed time is 108.305950 seconds.
For Noisy Input
SNR: 24.53110808; PSNR: 28.37635758
For Denoised Result
SNR: 24.8634913; PSNR: 28.7087408
fx >>
```

h=0.1:

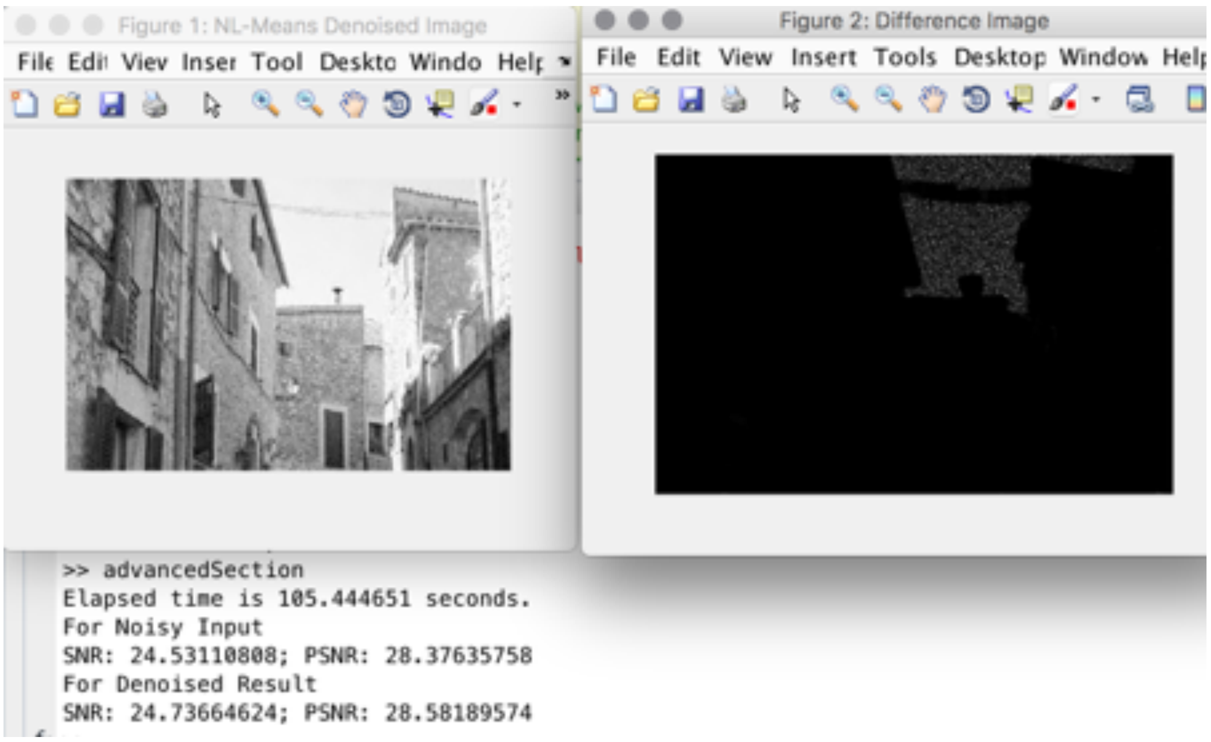
Figure 1: NL-Means Denoised Image
File Edit View Insert Tools Desktop Window Help
Figure 2: Difference Image
File Edit View Insert Tools Desktop Window Help

```
>> advancedSection
Elapsed time is 105.444651 seconds.
For Noisy Input
SNR: 24.53110808; PSNR: 28.37635758
For Denoised Result
SNR: 24.73664624; PSNR: 28.58189574
```

Different h parameter will controls the filter sensitive to which section of grayscale. For example, the larger ‘h’ is, the filter will be more sensitive to the brighter part of the image. The reason could be that the larger ‘h’ will leads to larger average weight. So the brighter section was emphasised.

- Sigma

The sigma is given in this coursework. The effect of sigma is very similar to the parameter ’h’. It will decide the distribution of the weights.

3. The detail range of parameters are depends on the image’s texture. The result here are only for the three images in the coursework.

- Window size

To smooth the image, increase the window size. The edge will be blurred as well.

- Patch size

More sensitive to small noise, decrease the patch size.

- h

More sensitive to brighter pixel, increase the h.