**Assignment 5**

**Part 1 soure code**

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <iostream>

#include <fstream>

#include <vector>

using namespace cv;

using namespace std;

//calculate the mean and modify the value of (i,j) to the mean num

void mean\_filter\_helper(IplImage\* img, int i, int j)

{

//calculate the sum

int sum = 0;

for (int row = i - 1; row < i + 2; row++)

{

uchar\* ptr = (uchar\*)img->imageData + row\*img->widthStep;

for (int col = j - 1; col < j + 2; col++)

{

sum += ptr[col];

}

}

//modify the value

uchar\* ptr = (uchar\*)img->imageData + i\*img->widthStep;

ptr[j] = sum / 9;

}

void mean\_filter(IplImage\* img)

{

//don't change the pixels in the boundary

for (int row = 1; row < img->height - 1; row++)

{

for (int col = 1; col < img->width - 1; col++)

{

mean\_filter\_helper(img, row, col);

}

}

}

//calculate the median and modify the value of (i,j) to the median num

void median\_filter\_helper(IplImage\* img, int i, int j)

{

//calculate the sum

int median = 0;

int count = 0;

int temp[9];

for (int row = i - 1; row < i + 2; row++)

{

uchar\* ptr = (uchar\*)img->imageData + row\*img->widthStep;

for (int col = j - 1; col < j + 2; col++)

{

temp[count] = ptr[col];

count++;

}

}

//sort the temp array

int max\_index = 9;

while (max\_index > 1)

{

//make the max number at the max\_index position

for (int temp\_index = 0; temp\_index < max\_index - 1; temp\_index++)

{

if (temp[temp\_index]>temp[temp\_index + 1])

{

//swap

int swap = temp[temp\_index + 1];

temp[temp\_index + 1] = temp[temp\_index];

temp[temp\_index] = swap;

}

}

//decrease the max\_index

max\_index--;

}

//modify the value

uchar\* ptr = (uchar\*)img->imageData + i\*img->widthStep;

ptr[j] = temp[4];

}

void median\_filter(IplImage\* img)

{

//don't change the pixels in the boundary

for (int row = 1; row < img->height - 1; row++)

{

for (int col = 1; col < img->width - 1; col++)

{

median\_filter\_helper(img, row, col);

}

}

}

void gaussian\_initialize(int (&mask)[5][5])

{

mask[0][0] = 1; mask[0][1] = 4; mask[0][2] = 7; mask[0][3] = 4; mask[0][4] = 1;

mask[1][0] = 4; mask[1][1] = 16; mask[1][2] = 26; mask[1][3] = 16; mask[1][4] = 4;

mask[2][0] = 7; mask[2][1] = 26; mask[2][2] = 41; mask[2][3] = 26; mask[2][4] = 7;

mask[3][0] = 4; mask[3][1] = 16; mask[3][2] = 26; mask[3][3] = 16; mask[3][4] = 4;

mask[4][0] = 1; mask[4][1] = 4; mask[4][2] = 7; mask[4][3] = 4; mask[4][4] = 1;

}

void gaussian\_filter\_helper(IplImage\* img, int i, int j)

{

const int sum\_weight = 273;

///use 5\*5 gaussin matrix within 3 sigma

int mask[5][5];

gaussian\_initialize(mask);

//calculate the sum

int sum = 0;

for (int row = i - 2; row < i + 3; row++)

{

uchar\* ptr = (uchar\*)img->imageData + row\*img->widthStep;

for (int col = j - 2; col < j + 3; col++)

{

//get the image\_data

int image\_data = ptr[col];

//get the mask\_data

int mask\_row = row - i + 2;// 4 - (row - i + 2);//since the matrix is symmetry

int mask\_col = col - j + 2;// 4 - (col - j + 2);

int mask\_data = mask[mask\_row][mask\_col];

sum += image\_data\*mask\_data;

}

}

//modify the value

uchar\* ptr = (uchar\*)img->imageData + i\*img->widthStep;

ptr[j] = sum / sum\_weight;

}

void gaussian\_filter(IplImage\* img)

{

//don't change the pixels in the boundary

for (int row = 2; row < img->height - 2; row++)

{

for (int col = 2; col < img->width - 2; col++)

{

gaussian\_filter\_helper(img, row, col);

}

}

}

int main()

{

//load the image

IplImage\* img = cvLoadImage("filters.png", 0);

//choose the type of filter

int mode = 0;

cout << "Please choose a type of filter(1-Mean 2-Median 3-Gaussian): ";

cin >> mode;

cout << endl;

//do different types of filter

if (mode == 1)

{

mean\_filter(img);

}

else if (mode == 2)

{

median\_filter(img);

}

else if (mode == 3)

{

gaussian\_filter(img);

}

else

{

cout << "Type invalid!\n";

system("pause");

exit(0);

}

//save result image

if (mode == 1)

{

cvSaveImage("mean\_filters.png", img);

}

else if (mode == 2)

{

cvSaveImage("median\_filter.png", img);

}

else

{

cvSaveImage("gaussian\_filter.png", img);

}

//release space

cvReleaseImage(&img);

system("pause");

return 0;

}

**Part 2 cite**

(PS:you can read comment of filter.cpp to get more info)

There are three pairs of functions:

1. mean\_filter/mean\_filter\_helper

This just use 3\*3 matrix with the same value 1 for each element.Just equal to calculate the average of 9 numbers to replace the origin data.It’s a smooth filter.

1. median\_filter/median\_filter\_helper

This just sort the 9 numbers around the middle point(I just use bubble sort).This cannot be replaced by a folding operation, so it’s not a smooth filter.

1. gaussian\_filter/gaussian\_filter\_helper

I just use 5\*5 matrix as a mask within 3 sigma.I make a folding operation between the image data and the mask.

As a conclusion, the larger mask I use, the more ambigious image i get.

In my view, gaussian filter is just similar with mean filter.The only difference is that each weight of the matrix is various in gaussian filter.So the gaussian can keep more details of the origin image compared with mean\_filter.

While the median filter is more suitable for salt-and-pepper noise.

**Part 3 output**

(1)mean\_filter\_image



The implementation of this filter is easy enough and can make the image smooth. However, it just decreases the influence of noises without remove it effectively. The noises are still exist which just become smaller.

1. median\_filter\_image



This method isn’t a liner method, however maybe median is more objective than mean, so this filter seems protect the verge of sharpen areas. Compared with the former filter, we can find that all of the noises are almost removed and the picture seems more clear than that handled by mean filter.

1. gaussian\_filter\_image



Gaussian filter is just like a development version of mean filter which uses a weighted matrix as a mask. It is also a liner filter and can decreases the effect of noises without removing it. So the results of those two filters are similar.