1.Algorithm & principal code fragment & your parameters:

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <cv.h>

#include <ctime>

using namespace std;

using namespace cv;

Mat picInitialSrc(Mat pic, Mat src, int row, int col){

int i,j;

for(i=0; i<col; i++){

for(j=0; j<row; j++){

pic.at<unsigned char>(i,j) = src.at<unsigned char>(i,j);

}

}

return pic;

}

void showSavePicture(Mat picture,string windowName,string saveName){

namedWindow( windowName, WINDOW\_AUTOSIZE );

imshow( windowName, picture );

imwrite( saveName , picture );

}

Mat Robert(Mat input){

int row = input.rows;

int col = input.cols;

Mat robert(row, col, CV\_8U);

int i,j,r1,r2;

double gradient = 0;

for(i=0; i<row-1; i++){

for(j=0; j<col-1; j++){

r1 = -(input.at<unsigned char>(i,j)) + input.at<unsigned char>(i+1,j+1);

r2 = -(input.at<unsigned char>(i,j+1))+ input.at<unsigned char>(i+1,j);

gradient = sqrt((r1\*r1)+(r2\*r2));

//printf("%lf\n",gradient);

if(gradient > 30)

robert.at<unsigned char>(i,j) = 0;

else

robert.at<unsigned char>(i,j) = 255;

}

}

return robert;

}

Mat Prewitt(Mat input){

int row = input.rows;

int col = input.cols;

Mat prewitt(row, col, CV\_8U);

int i, j, p1=0, p2=0;

double gradient;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

p1 = -(input.at<unsigned char>(i-1,j-1)) - (input.at<unsigned char>(i-1,j)) - (input.at<unsigned char>(i-1,j+1))

+ input.at<unsigned char>(i+1,j-1) + input.at<unsigned char>(i+1,j) + input.at<unsigned char>(i+1,j+1);

p2 = -(input.at<unsigned char>(i-1,j-1)) - (input.at<unsigned char>(i,j-1)) - (input.at<unsigned char>(i+1,j-1))

+ input.at<unsigned char>(i-1,j+1) + input.at<unsigned char>(i,j+1) + input.at<unsigned char>(i+1,j+1);

gradient = sqrt((p1\*p1)+(p2\*p2));

//printf("%lf\n",gradient);

if(gradient > 100)

prewitt.at<unsigned char>(i,j) = 0;

else

prewitt.at<unsigned char>(i,j) = 255;

}

}

return prewitt;

}

Mat Sobel(Mat input){

int row = input.rows;

int col = input.cols;

Mat sobel(row, col, CV\_8U);

int i, j, p1=0, p2=0;

double gradient;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

p1 = -(input.at<unsigned char>(i-1,j-1)) - 2 \* (input.at<unsigned char>(i-1,j)) - (input.at<unsigned char>(i-1,j+1))

+ input.at<unsigned char>(i+1,j-1) + 2 \* input.at<unsigned char>(i+1,j) + input.at<unsigned char>(i+1,j+1);

p2 = -(input.at<unsigned char>(i-1,j-1)) - 2 \* (input.at<unsigned char>(i,j-1)) - (input.at<unsigned char>(i+1,j-1))

+ input.at<unsigned char>(i-1,j+1) + 2 \* input.at<unsigned char>(i,j+1) + input.at<unsigned char>(i+1,j+1);

gradient = sqrt((p1\*p1)+(p2\*p2));

//printf("%lf\n",gradient);

if(gradient > 100)

sobel.at<unsigned char>(i,j) = 0;

else

sobel.at<unsigned char>(i,j) = 255;

}

}

return sobel;

}

Mat FaC(Mat input){

int row = input.rows;

int col = input.cols;

Mat fac(row, col, CV\_8U);

int i, j, p1=0, p2=0;

double gradient;

double sqrTwo = sqrt(2);

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

p1 = -(input.at<unsigned char>(i-1,j-1)) - sqrTwo \* (input.at<unsigned char>(i-1,j)) - (input.at<unsigned char>(i-1,j+1))

+ input.at<unsigned char>(i+1,j-1) + sqrTwo \* input.at<unsigned char>(i+1,j) + input.at<unsigned char>(i+1,j+1);

p2 = -(input.at<unsigned char>(i-1,j-1)) - sqrTwo \* (input.at<unsigned char>(i,j-1)) - (input.at<unsigned char>(i+1,j-1))

+ input.at<unsigned char>(i-1,j+1) + sqrTwo \* input.at<unsigned char>(i,j+1) + input.at<unsigned char>(i+1,j+1);

gradient = sqrt((p1\*p1)+(p2\*p2));

//printf("%lf\n",gradient);

if(gradient > 100)

fac.at<unsigned char>(i,j) = 0;

else

fac.at<unsigned char>(i,j) = 255;

}

}

return fac;

}

Mat Kirsch(Mat input){

int row = input.rows;

int col = input.cols;

Mat kirsch(row, col, CV\_8U);

int i, j, k, p1=0, p2=0;

double gradient;

int arrayk0[8] = {-3,-3,5,-3,5,-3,-3,5};

int arrayk1[8] = {-3,5,5,-3,5,-3,-3,-3};

int arrayk2[8] = {5,5,5,-3,-3,-3,-3,-3};

int arrayk3[8] = {5,5,-3,5,-3,-3,-3,-3};

int arrayk4[8] = {5,-3,-3,5,-3,5,-3,-3};

int arrayk5[8] = {-3,-3,-3,5,-3,5,5,-3};

int arrayk6[8] = {-3,-3,-3,-3,-3,5,5,5};

int arrayk7[8] = {-3,-3,-3,-3,5,-3,5,5};

int arrayMap[8];

int tempCal = 0;

int answerNow = 0;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

arrayMap[0] = input.at<unsigned char>(i-1,j-1);

arrayMap[1] = input.at<unsigned char>(i-1,j);

arrayMap[2] = input.at<unsigned char>(i-1,j+1);

arrayMap[3] = input.at<unsigned char>(i,j-1);

arrayMap[4] = input.at<unsigned char>(i,j+1);

arrayMap[5] = input.at<unsigned char>(i+1,j-1);

arrayMap[6] = input.at<unsigned char>(i+1,j);

arrayMap[7] = input.at<unsigned char>(i+1,j+1);

for(k=0; k<8; k++){

tempCal += arrayk0[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk1[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk2[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk3[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk4[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk5[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk6[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk7[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

gradient = answerNow;

//printf("%lf\n",gradient);

if(gradient > 400)

kirsch.at<unsigned char>(i,j) = 0;

else

kirsch.at<unsigned char>(i,j) = 255;

answerNow = 0;

tempCal = 0;

}

}

return kirsch;

}

Mat Robinson(Mat input){

int row = input.rows;

int col = input.cols;

Mat robinson(row, col, CV\_8U);

int i, j, k, p1=0, p2=0;

double gradient;

int arrayk0[8] = {-1,0,1,-2,2,-1,0,1};

int arrayk1[8] = {0,1,2,-1,1,-2,-1,0};

int arrayk2[8] = {1,2,1,0,0,-1,-2,-1};

int arrayk3[8] = {2,1,0,1,-1,0,-1,-2};

int arrayk4[8] = {1,0,-1,2,-2,1,0,-1};

int arrayk5[8] = {0,-1,-2,1,-1,2,1,0};

int arrayk6[8] = {-1,-2,-1,0,0,1,2,1};

int arrayk7[8] = {-2,-1,0,-1,1,0,1,2};

int arrayMap[8];

int tempCal = 0;

int answerNow = 0;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

arrayMap[0] = input.at<unsigned char>(i-1,j-1);

arrayMap[1] = input.at<unsigned char>(i-1,j);

arrayMap[2] = input.at<unsigned char>(i-1,j+1);

arrayMap[3] = input.at<unsigned char>(i,j-1);

arrayMap[4] = input.at<unsigned char>(i,j+1);

arrayMap[5] = input.at<unsigned char>(i+1,j-1);

arrayMap[6] = input.at<unsigned char>(i+1,j);

arrayMap[7] = input.at<unsigned char>(i+1,j+1);

for(k=0; k<8; k++){

tempCal += arrayk0[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk1[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk2[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk3[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk4[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk5[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk6[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<8; k++){

tempCal += arrayk7[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

gradient = answerNow;

//printf("%lf\n",gradient);

if(gradient > 200)

robinson.at<unsigned char>(i,j) = 0;

else

robinson.at<unsigned char>(i,j) = 255;

answerNow = 0;

tempCal = 0;

}

}

return robinson;

}

Mat NaB(Mat input){

int row = input.rows;

int col = input.cols;

Mat nab(row, col, CV\_8U);

int i, j, k, p1=0, p2=0;

double gradient;

int arrayk0[25] = {100,100,100,100,100,100,100,100,100,100,0,0,0,0,0,-100,-100,-100,-100,-100,-100,-100,-100,-100,-100};

int arrayk1[25] = {100,100,100,100,100,100,100,100,78,-32,100,92,0,-92,-100,32,-78,-100,-100,-100,-100,-100,-100,-100,-100};

int arrayk2[25] = {100,100,100,32,-100,100,100,92,-78,-100,100,100,0,-100,-100,100,78,-92,-100,-100,100,-32,-100,-100,-100};

int arrayk3[25] = {-100,-100,0,100,100,-100,-100,0,100,100,-100,-100,0,100,100,-100,-100,0,100,100,-100,-100,0,100,100};

int arrayk4[25] = {-100,32,100,100,100,-100,-78,92,100,100,-100,-100,0,100,100,-100,-100,-92,78,100,-100,-100,-100,-32,100};

int arrayk5[25] = {100,100,100,100,100,-32,78,100,100,100,-100,-92,0,92,100,-100,-100,-100,-78,32,-100,-100,-100,-100,-100};

int arrayMap[25];

int tempCal = 0;

int answerNow = 0;

for(i=2; i<row-2; i++){

for(j=2; j<col-2; j++){

arrayMap[0] = input.at<unsigned char>(i-2,j-2);

arrayMap[1] = input.at<unsigned char>(i-2,j-1);

arrayMap[2] = input.at<unsigned char>(i-2,j);

arrayMap[3] = input.at<unsigned char>(i-2,j+1);

arrayMap[4] = input.at<unsigned char>(i-2,j+2);

arrayMap[5] = input.at<unsigned char>(i-1,j-2);

arrayMap[6] = input.at<unsigned char>(i-1,j-1);

arrayMap[7] = input.at<unsigned char>(i-1,j);

arrayMap[8] = input.at<unsigned char>(i-1,j+1);

arrayMap[9] = input.at<unsigned char>(i-1,j+2);

arrayMap[10] = input.at<unsigned char>(i,j-2);

arrayMap[11] = input.at<unsigned char>(i,j-1);

arrayMap[12] = input.at<unsigned char>(i,j);

arrayMap[13] = input.at<unsigned char>(i,j+1);

arrayMap[14] = input.at<unsigned char>(i,j+2);

arrayMap[15] = input.at<unsigned char>(i+1,j-2);

arrayMap[16] = input.at<unsigned char>(i+1,j-1);

arrayMap[17] = input.at<unsigned char>(i+1,j);

arrayMap[18] = input.at<unsigned char>(i+1,j+1);

arrayMap[19] = input.at<unsigned char>(i+1,j+2);

arrayMap[20] = input.at<unsigned char>(i+2,j-2);

arrayMap[21] = input.at<unsigned char>(i+2,j-1);

arrayMap[22] = input.at<unsigned char>(i+2,j);

arrayMap[23] = input.at<unsigned char>(i+2,j+1);

arrayMap[24] = input.at<unsigned char>(i+2,j+2);

for(k=0; k<25; k++){

tempCal += arrayk0[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<25; k++){

tempCal += arrayk1[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<25; k++){

tempCal += arrayk2[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<25; k++){

tempCal += arrayk3[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<25; k++){

tempCal += arrayk4[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

for(k=0; k<25; k++){

tempCal += arrayk5[k] \* arrayMap[k];

}

if(tempCal > answerNow) answerNow = tempCal;

gradient = answerNow;

//printf("%lf\n",gradient);

if(gradient > 50000)

nab.at<unsigned char>(i,j) = 0;

else

nab.at<unsigned char>(i,j) = 255;

answerNow = 0;

tempCal = 0;

}

}

return nab;

}

Mat Laplacian1(Mat input){

int row = input.rows;

int col = input.cols;

Mat lap(row, col, CV\_8U);

int i, j, p1=0, p2=0;

int gradient;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

gradient = input.at<unsigned char>(i,j)\*4\*(-1) + input.at<unsigned char>(i-1,j) + input.at<unsigned char>(i,j-1) + input.at<unsigned char>(i+1,j) + input.at<unsigned char>(i,j+1);

if(gradient > 15)

lap.at<unsigned char>(i,j) = 0;

else

lap.at<unsigned char>(i,j) = 255;

}

}

return lap;

}

Mat Laplacian2(Mat input){

int row = input.rows;

int col = input.cols;

Mat lap(row, col, CV\_8U);

int i, j, p1=0, p2=0;

double gradient;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

gradient = input.at<unsigned char>(i,j)\*8\*(-1) + input.at<unsigned char>(i-1,j) + input.at<unsigned char>(i,j-1) + input.at<unsigned char>(i+1,j) + input.at<unsigned char>(i,j+1) + input.at<unsigned char>(i-1,j-1) + input.at<unsigned char>(i+1,j-1) + input.at<unsigned char>(i-1,j+1) + input.at<unsigned char>(i+1,j+1);

gradient = gradient / 3;

if(gradient > 15)

lap.at<unsigned char>(i,j) = 0;

else

lap.at<unsigned char>(i,j) = 255;

}

}

return lap;

}

Mat minVarLaplacian(Mat input){

int row = input.rows;

int col = input.cols;

Mat mvl(row, col, CV\_8U);

int i, j, p1=0, p2=0;

double gradient;

for(i=1; i<row-1; i++){

for(j=1; j<col-1; j++){

gradient = input.at<unsigned char>(i,j)\*4\*(-1) - input.at<unsigned char>(i-1,j) - input.at<unsigned char>(i,j-1) - input.at<unsigned char>(i+1,j) - input.at<unsigned char>(i,j+1) + 2\*input.at<unsigned char>(i-1,j-1) + 2\*input.at<unsigned char>(i+1,j-1) + 2\*input.at<unsigned char>(i-1,j+1) + 2\*input.at<unsigned char>(i+1,j+1);

gradient = gradient / 3;

if(gradient > 20)

mvl.at<unsigned char>(i,j) = 0;

else

mvl.at<unsigned char>(i,j) = 255;

}

}

return mvl;

}

Mat LoG(Mat input){

int row = input.rows;

int col = input.cols;

Mat lg(row, col, CV\_8U);

int i, j, k, p1=0, p2=0;

double gradient;

int mask[121] = {0,0,0,-1,-1,-2,-1,-1,0,0,0,

0,0,-2,-4,-8,-9,-8,-4,-2,0,0,

0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0,

-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1,

-1,-8,-22,-14,52,103,52,-14,-22,-8,-1,

-2,-9,-23,-1,103,178,103,-1,-23,-9,-2,

-1,-8,-22,-14,52,103,52,-14,-22,-8,-1,

-1,-4,-15,-24,-14,-1,-14,-24,-15,-4,-1,

0,-2,-7,-15,-22,-23,-22,-15,-7,-2,0,

0,0,-2,-4,-8,-9,-8,-4,-2,0,0,

0,0,0,-1,-1,-2,-1,-1,0,0,0};

int arrayMap[121];

int tempCal = 0;

int answerNow = 0;

for(i=5; i<row-5; i++){

for(j=5; j<col-5; j++){

arrayMap[0] = input.at<unsigned char>(i-5,j-5);

arrayMap[1] = input.at<unsigned char>(i-5,j-4);

arrayMap[2] = input.at<unsigned char>(i-5,j-3);

arrayMap[3] = input.at<unsigned char>(i-5,j-2);

arrayMap[4] = input.at<unsigned char>(i-5,j-1);

arrayMap[5] = input.at<unsigned char>(i-5,j);

arrayMap[6] = input.at<unsigned char>(i-5,j+1);

arrayMap[7] = input.at<unsigned char>(i-5,j+2);

arrayMap[8] = input.at<unsigned char>(i-5,j+3);

arrayMap[9] = input.at<unsigned char>(i-5,j+4);

arrayMap[10] = input.at<unsigned char>(i-5,j+5);

arrayMap[11] = input.at<unsigned char>(i-4,j-5);

arrayMap[12] = input.at<unsigned char>(i-4,j-4);

arrayMap[13] = input.at<unsigned char>(i-4,j-3);

arrayMap[14] = input.at<unsigned char>(i-4,j-2);

arrayMap[15] = input.at<unsigned char>(i-4,j-1);

arrayMap[16] = input.at<unsigned char>(i-4,j);

arrayMap[17] = input.at<unsigned char>(i-4,j+1);

arrayMap[18] = input.at<unsigned char>(i-4,j+2);

arrayMap[19] = input.at<unsigned char>(i-4,j+3);

arrayMap[20] = input.at<unsigned char>(i-4,j+4);

arrayMap[21] = input.at<unsigned char>(i-4,j+5);

arrayMap[22] = input.at<unsigned char>(i-3,j-5);

arrayMap[23] = input.at<unsigned char>(i-3,j-4);

arrayMap[24] = input.at<unsigned char>(i-3,j-3);

arrayMap[25] = input.at<unsigned char>(i-3,j-2);

arrayMap[26] = input.at<unsigned char>(i-3,j-1);

arrayMap[27] = input.at<unsigned char>(i-3,j);

arrayMap[28] = input.at<unsigned char>(i-3,j+1);

arrayMap[29] = input.at<unsigned char>(i-3,j+2);

arrayMap[30] = input.at<unsigned char>(i-3,j+3);

arrayMap[31] = input.at<unsigned char>(i-3,j+4);

arrayMap[32] = input.at<unsigned char>(i-3,j+5);

arrayMap[33] = input.at<unsigned char>(i-2,j-5);

arrayMap[34] = input.at<unsigned char>(i-2,j-4);

arrayMap[35] = input.at<unsigned char>(i-2,j-3);

arrayMap[36] = input.at<unsigned char>(i-2,j-2);

arrayMap[37] = input.at<unsigned char>(i-2,j-1);

arrayMap[38] = input.at<unsigned char>(i-2,j);

arrayMap[39] = input.at<unsigned char>(i-2,j+1);

arrayMap[40] = input.at<unsigned char>(i-2,j+2);

arrayMap[41] = input.at<unsigned char>(i-2,j+3);

arrayMap[42] = input.at<unsigned char>(i-2,j+4);

arrayMap[43] = input.at<unsigned char>(i-2,j+5);

arrayMap[44] = input.at<unsigned char>(i-1,j-5);

arrayMap[45] = input.at<unsigned char>(i-1,j-4);

arrayMap[46] = input.at<unsigned char>(i-1,j-3);

arrayMap[47] = input.at<unsigned char>(i-1,j-2);

arrayMap[48] = input.at<unsigned char>(i-1,j-1);

arrayMap[49] = input.at<unsigned char>(i-1,j);

arrayMap[50] = input.at<unsigned char>(i-1,j+1);

arrayMap[51] = input.at<unsigned char>(i-1,j+2);

arrayMap[52] = input.at<unsigned char>(i-1,j+3);

arrayMap[53] = input.at<unsigned char>(i-1,j+4);

arrayMap[54] = input.at<unsigned char>(i-1,j+5);

arrayMap[55] = input.at<unsigned char>(i,j-5);

arrayMap[56] = input.at<unsigned char>(i,j-4);

arrayMap[57] = input.at<unsigned char>(i,j-3);

arrayMap[58] = input.at<unsigned char>(i,j-2);

arrayMap[59] = input.at<unsigned char>(i,j-1);

arrayMap[60] = input.at<unsigned char>(i,j);

arrayMap[61] = input.at<unsigned char>(i,j+1);

arrayMap[62] = input.at<unsigned char>(i,j+2);

arrayMap[63] = input.at<unsigned char>(i,j+3);

arrayMap[64] = input.at<unsigned char>(i,j+4);

arrayMap[65] = input.at<unsigned char>(i,j+5);

arrayMap[66] = input.at<unsigned char>(i+1,j-5);

arrayMap[67] = input.at<unsigned char>(i+1,j-4);

arrayMap[68] = input.at<unsigned char>(i+1,j-3);

arrayMap[69] = input.at<unsigned char>(i+1,j-2);

arrayMap[70] = input.at<unsigned char>(i+1,j-1);

arrayMap[71] = input.at<unsigned char>(i+1,j);

arrayMap[72] = input.at<unsigned char>(i+1,j+1);

arrayMap[73] = input.at<unsigned char>(i+1,j+2);

arrayMap[74] = input.at<unsigned char>(i+1,j+3);

arrayMap[75] = input.at<unsigned char>(i+1,j+4);

arrayMap[76] = input.at<unsigned char>(i+1,j+5);

arrayMap[77] = input.at<unsigned char>(i+2,j-5);

arrayMap[78] = input.at<unsigned char>(i+2,j-4);

arrayMap[79] = input.at<unsigned char>(i+2,j-3);

arrayMap[80] = input.at<unsigned char>(i+2,j-2);

arrayMap[81] = input.at<unsigned char>(i+2,j-1);

arrayMap[82] = input.at<unsigned char>(i+2,j);

arrayMap[83] = input.at<unsigned char>(i+2,j+1);

arrayMap[84] = input.at<unsigned char>(i+2,j+2);

arrayMap[85] = input.at<unsigned char>(i+2,j+3);

arrayMap[86] = input.at<unsigned char>(i+2,j+4);

arrayMap[87] = input.at<unsigned char>(i+2,j+5);

arrayMap[88] = input.at<unsigned char>(i+3,j-5);

arrayMap[89] = input.at<unsigned char>(i+3,j-4);

arrayMap[90] = input.at<unsigned char>(i+3,j-3);

arrayMap[91] = input.at<unsigned char>(i+3,j-2);

arrayMap[92] = input.at<unsigned char>(i+3,j-1);

arrayMap[93] = input.at<unsigned char>(i+3,j);

arrayMap[94] = input.at<unsigned char>(i+3,j+1);

arrayMap[95] = input.at<unsigned char>(i+3,j+2);

arrayMap[96] = input.at<unsigned char>(i+3,j+3);

arrayMap[97] = input.at<unsigned char>(i+3,j+4);

arrayMap[98] = input.at<unsigned char>(i+3,j+5);

arrayMap[99] = input.at<unsigned char>(i+4,j-5);

arrayMap[100] = input.at<unsigned char>(i+4,j-4);

arrayMap[101] = input.at<unsigned char>(i+4,j-3);

arrayMap[102] = input.at<unsigned char>(i+4,j-2);

arrayMap[103] = input.at<unsigned char>(i+4,j-1);

arrayMap[104] = input.at<unsigned char>(i+4,j);

arrayMap[105] = input.at<unsigned char>(i+4,j+1);

arrayMap[106] = input.at<unsigned char>(i+4,j+2);

arrayMap[107] = input.at<unsigned char>(i+4,j+3);

arrayMap[108] = input.at<unsigned char>(i+4,j+4);

arrayMap[109] = input.at<unsigned char>(i+4,j+5);

arrayMap[110] = input.at<unsigned char>(i+5,j-5);

arrayMap[111] = input.at<unsigned char>(i+5,j-4);

arrayMap[112] = input.at<unsigned char>(i+5,j-3);

arrayMap[113] = input.at<unsigned char>(i+5,j-2);

arrayMap[114] = input.at<unsigned char>(i+5,j-1);

arrayMap[115] = input.at<unsigned char>(i+5,j);

arrayMap[116] = input.at<unsigned char>(i+5,j+1);

arrayMap[117] = input.at<unsigned char>(i+5,j+2);

arrayMap[118] = input.at<unsigned char>(i+5,j+3);

arrayMap[119] = input.at<unsigned char>(i+5,j+4);

arrayMap[120] = input.at<unsigned char>(i+5,j+5);

for(k=0; k<121; k++){

tempCal += mask[k] \* arrayMap[k];

}

if(tempCal > 3000)

lg.at<unsigned char>(i,j) = 0;

else

lg.at<unsigned char>(i,j) = 255;

answerNow = 0;

tempCal = 0;

}

}

return lg;

}

Mat DoG(Mat input){

int row = input.rows;

int col = input.cols;

Mat dg(row, col, CV\_8U);

int i, j, k, p1=0, p2=0;

double gradient;

int mask[121] = {-1,-3,-4,-6,-7,-8,-7,-6,-4,-3,-1,

-3,-5,-8,-11,-13,-13,-13,-11,-8,-5,-3,

-4,-8,-12,-16,-17,-17,-17,-16,-12,-8,-4,

-6,-11,-16,-16,0,15,0,-16,-16,-11,-6,

-7,-13,-17,0,85,160,85,0,-17,-13,-7,

-8,-13,-17,15,160,283,160,15,-17,-13,-8,

-7,-13,-17,0,85,160,85,0,-17,-13,-7,

-6,-11,-16,-16,0,15,0,-16,-16,-11,-6,

-4,-8,-12,-16,-17,-17,-17,-16,-12,-8,-4,

-3,-5,-8,-11,-13,-13,-13,-11,-8,-5,-3,

-1,-3,-4,-6,-7,-8,-7,-6,-4,-3,-1, };

int arrayMap[121];

int tempCal = 0;

int answerNow = 0;

for(i=5; i<row-5; i++){

for(j=5; j<col-5; j++){

arrayMap[0] = input.at<unsigned char>(i-5,j-5);

arrayMap[1] = input.at<unsigned char>(i-5,j-4);

arrayMap[2] = input.at<unsigned char>(i-5,j-3);

arrayMap[3] = input.at<unsigned char>(i-5,j-2);

arrayMap[4] = input.at<unsigned char>(i-5,j-1);

arrayMap[5] = input.at<unsigned char>(i-5,j);

arrayMap[6] = input.at<unsigned char>(i-5,j+1);

arrayMap[7] = input.at<unsigned char>(i-5,j+2);

arrayMap[8] = input.at<unsigned char>(i-5,j+3);

arrayMap[9] = input.at<unsigned char>(i-5,j+4);

arrayMap[10] = input.at<unsigned char>(i-5,j+5);

arrayMap[11] = input.at<unsigned char>(i-4,j-5);

arrayMap[12] = input.at<unsigned char>(i-4,j-4);

arrayMap[13] = input.at<unsigned char>(i-4,j-3);

arrayMap[14] = input.at<unsigned char>(i-4,j-2);

arrayMap[15] = input.at<unsigned char>(i-4,j-1);

arrayMap[16] = input.at<unsigned char>(i-4,j);

arrayMap[17] = input.at<unsigned char>(i-4,j+1);

arrayMap[18] = input.at<unsigned char>(i-4,j+2);

arrayMap[19] = input.at<unsigned char>(i-4,j+3);

arrayMap[20] = input.at<unsigned char>(i-4,j+4);

arrayMap[21] = input.at<unsigned char>(i-4,j+5);

arrayMap[22] = input.at<unsigned char>(i-3,j-5);

arrayMap[23] = input.at<unsigned char>(i-3,j-4);

arrayMap[24] = input.at<unsigned char>(i-3,j-3);

arrayMap[25] = input.at<unsigned char>(i-3,j-2);

arrayMap[26] = input.at<unsigned char>(i-3,j-1);

arrayMap[27] = input.at<unsigned char>(i-3,j);

arrayMap[28] = input.at<unsigned char>(i-3,j+1);

arrayMap[29] = input.at<unsigned char>(i-3,j+2);

arrayMap[30] = input.at<unsigned char>(i-3,j+3);

arrayMap[31] = input.at<unsigned char>(i-3,j+4);

arrayMap[32] = input.at<unsigned char>(i-3,j+5);

arrayMap[33] = input.at<unsigned char>(i-2,j-5);

arrayMap[34] = input.at<unsigned char>(i-2,j-4);

arrayMap[35] = input.at<unsigned char>(i-2,j-3);

arrayMap[36] = input.at<unsigned char>(i-2,j-2);

arrayMap[37] = input.at<unsigned char>(i-2,j-1);

arrayMap[38] = input.at<unsigned char>(i-2,j);

arrayMap[39] = input.at<unsigned char>(i-2,j+1);

arrayMap[40] = input.at<unsigned char>(i-2,j+2);

arrayMap[41] = input.at<unsigned char>(i-2,j+3);

arrayMap[42] = input.at<unsigned char>(i-2,j+4);

arrayMap[43] = input.at<unsigned char>(i-2,j+5);

arrayMap[44] = input.at<unsigned char>(i-1,j-5);

arrayMap[45] = input.at<unsigned char>(i-1,j-4);

arrayMap[46] = input.at<unsigned char>(i-1,j-3);

arrayMap[47] = input.at<unsigned char>(i-1,j-2);

arrayMap[48] = input.at<unsigned char>(i-1,j-1);

arrayMap[49] = input.at<unsigned char>(i-1,j);

arrayMap[50] = input.at<unsigned char>(i-1,j+1);

arrayMap[51] = input.at<unsigned char>(i-1,j+2);

arrayMap[52] = input.at<unsigned char>(i-1,j+3);

arrayMap[53] = input.at<unsigned char>(i-1,j+4);

arrayMap[54] = input.at<unsigned char>(i-1,j+5);

arrayMap[55] = input.at<unsigned char>(i,j-5);

arrayMap[56] = input.at<unsigned char>(i,j-4);

arrayMap[57] = input.at<unsigned char>(i,j-3);

arrayMap[58] = input.at<unsigned char>(i,j-2);

arrayMap[59] = input.at<unsigned char>(i,j-1);

arrayMap[60] = input.at<unsigned char>(i,j);

arrayMap[61] = input.at<unsigned char>(i,j+1);

arrayMap[62] = input.at<unsigned char>(i,j+2);

arrayMap[63] = input.at<unsigned char>(i,j+3);

arrayMap[64] = input.at<unsigned char>(i,j+4);

arrayMap[65] = input.at<unsigned char>(i,j+5);

arrayMap[66] = input.at<unsigned char>(i+1,j-5);

arrayMap[67] = input.at<unsigned char>(i+1,j-4);

arrayMap[68] = input.at<unsigned char>(i+1,j-3);

arrayMap[69] = input.at<unsigned char>(i+1,j-2);

arrayMap[70] = input.at<unsigned char>(i+1,j-1);

arrayMap[71] = input.at<unsigned char>(i+1,j);

arrayMap[72] = input.at<unsigned char>(i+1,j+1);

arrayMap[73] = input.at<unsigned char>(i+1,j+2);

arrayMap[74] = input.at<unsigned char>(i+1,j+3);

arrayMap[75] = input.at<unsigned char>(i+1,j+4);

arrayMap[76] = input.at<unsigned char>(i+1,j+5);

arrayMap[77] = input.at<unsigned char>(i+2,j-5);

arrayMap[78] = input.at<unsigned char>(i+2,j-4);

arrayMap[79] = input.at<unsigned char>(i+2,j-3);

arrayMap[80] = input.at<unsigned char>(i+2,j-2);

arrayMap[81] = input.at<unsigned char>(i+2,j-1);

arrayMap[82] = input.at<unsigned char>(i+2,j);

arrayMap[83] = input.at<unsigned char>(i+2,j+1);

arrayMap[84] = input.at<unsigned char>(i+2,j+2);

arrayMap[85] = input.at<unsigned char>(i+2,j+3);

arrayMap[86] = input.at<unsigned char>(i+2,j+4);

arrayMap[87] = input.at<unsigned char>(i+2,j+5);

arrayMap[88] = input.at<unsigned char>(i+3,j-5);

arrayMap[89] = input.at<unsigned char>(i+3,j-4);

arrayMap[90] = input.at<unsigned char>(i+3,j-3);

arrayMap[91] = input.at<unsigned char>(i+3,j-2);

arrayMap[92] = input.at<unsigned char>(i+3,j-1);

arrayMap[93] = input.at<unsigned char>(i+3,j);

arrayMap[94] = input.at<unsigned char>(i+3,j+1);

arrayMap[95] = input.at<unsigned char>(i+3,j+2);

arrayMap[96] = input.at<unsigned char>(i+3,j+3);

arrayMap[97] = input.at<unsigned char>(i+3,j+4);

arrayMap[98] = input.at<unsigned char>(i+3,j+5);

arrayMap[99] = input.at<unsigned char>(i+4,j-5);

arrayMap[100] = input.at<unsigned char>(i+4,j-4);

arrayMap[101] = input.at<unsigned char>(i+4,j-3);

arrayMap[102] = input.at<unsigned char>(i+4,j-2);

arrayMap[103] = input.at<unsigned char>(i+4,j-1);

arrayMap[104] = input.at<unsigned char>(i+4,j);

arrayMap[105] = input.at<unsigned char>(i+4,j+1);

arrayMap[106] = input.at<unsigned char>(i+4,j+2);

arrayMap[107] = input.at<unsigned char>(i+4,j+3);

arrayMap[108] = input.at<unsigned char>(i+4,j+4);

arrayMap[109] = input.at<unsigned char>(i+4,j+5);

arrayMap[110] = input.at<unsigned char>(i+5,j-5);

arrayMap[111] = input.at<unsigned char>(i+5,j-4);

arrayMap[112] = input.at<unsigned char>(i+5,j-3);

arrayMap[113] = input.at<unsigned char>(i+5,j-2);

arrayMap[114] = input.at<unsigned char>(i+5,j-1);

arrayMap[115] = input.at<unsigned char>(i+5,j);

arrayMap[116] = input.at<unsigned char>(i+5,j+1);

arrayMap[117] = input.at<unsigned char>(i+5,j+2);

arrayMap[118] = input.at<unsigned char>(i+5,j+3);

arrayMap[119] = input.at<unsigned char>(i+5,j+4);

arrayMap[120] = input.at<unsigned char>(i+5,j+5);

for(k=0; k<121; k++){

tempCal += mask[k] \* arrayMap[k];

}

if(tempCal < 1)

dg.at<unsigned char>(i,j) = 0;

else

dg.at<unsigned char>(i,j) = 255;

answerNow = 0;

tempCal = 0;

}

}

return dg;

}

int main(int argc,char\*\* argv ){

Mat src;

src = imread("lena.bmp",CV\_LOAD\_IMAGE\_GRAYSCALE);

int row = src.rows;

int col = src.cols;

/\*Mat robert(row, col, CV\_8U);

Mat prewitt(row, col, CV\_8U);

Mat sobel(row, col, CV\_8U);

Mat fac(row, col, CV\_8U);

Mat kirsch(row, col, CV\_8U);

Mat robinson(row, col, CV\_8U);

Mat nab(row, col, CV\_8U);\*/

Mat laplacian1(row, col, CV\_8U);

Mat laplacian2(row, col, CV\_8U);

Mat mvlaplacian(row, col, CV\_8U);

Mat lg(row, col, CV\_8U);

Mat dg(row, col, CV\_8U);

/\*robert = Robert(src);

prewitt = Prewitt(src);

sobel = Sobel(src);

fac = FaC(src);

kirsch = Kirsch(src);

robinson = Robinson(src);

nab = NaB(src);\*/

laplacian1 = Laplacian1(src);

laplacian2 = Laplacian2(src);

mvlaplacian = minVarLaplacian(src);

lg = LoG(src);

dg = DoG(src);

/\*showSavePicture(src,"src","src.bmp");

showSavePicture(robert,"robert","robert.bmp");

showSavePicture(prewitt,"prewitt","prewitt.bmp");

showSavePicture(sobel,"sobel","sobel.bmp");

showSavePicture(fac,"Frei\_&\_Chen","Frei\_&\_Chen.bmp");

showSavePicture(kirsch,"kirsch","kirsch.bmp");

showSavePicture(robinson,"Robinson","Robinson.bmp");

showSavePicture(nab,"Nevatia\_and\_Babu","Nevatia\_and\_Babu.bmp");\*/

showSavePicture(laplacian1,"Laplacian1","Laplacian1.bmp");

showSavePicture(laplacian2,"Laplacian2","Laplacian2.bmp");

showSavePicture(mvlaplacian,"minVarLaplacian","minVarLaplacian.bmp");

showSavePicture(lg,"Laplacian of Gaussian","Laplacian of Gaussian.bmp");

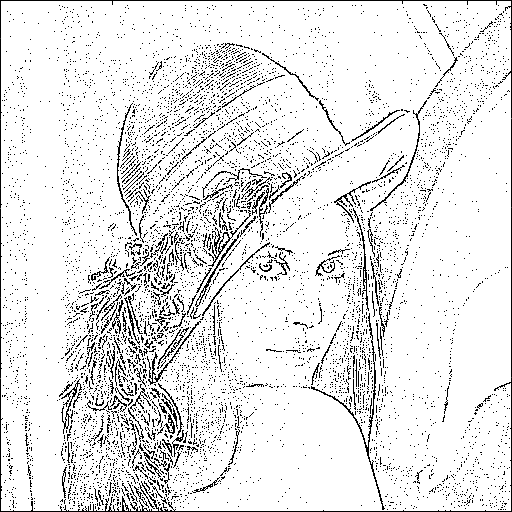
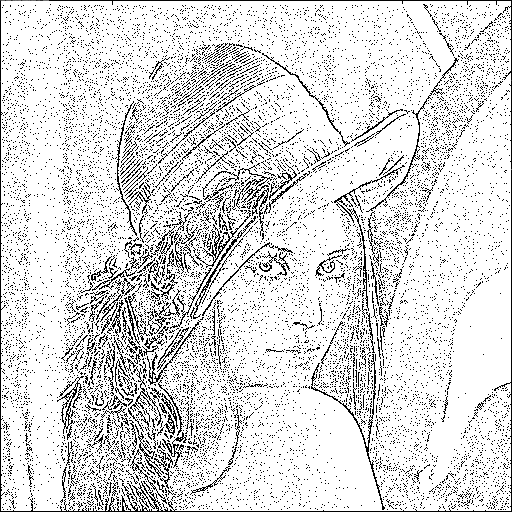
showSavePicture(dg,"Difference of Gaussian","Difference of Gaussian.bmp");

waitKey(0);

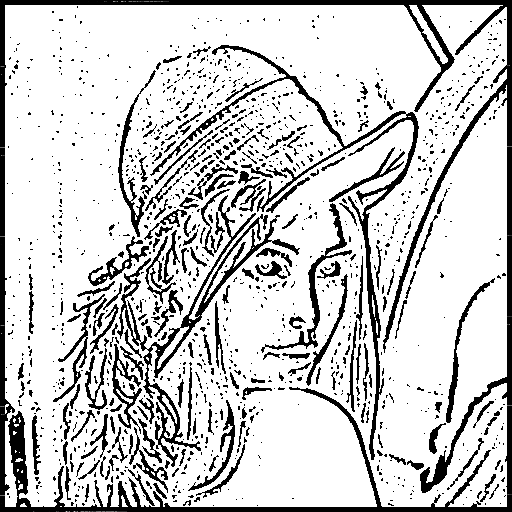
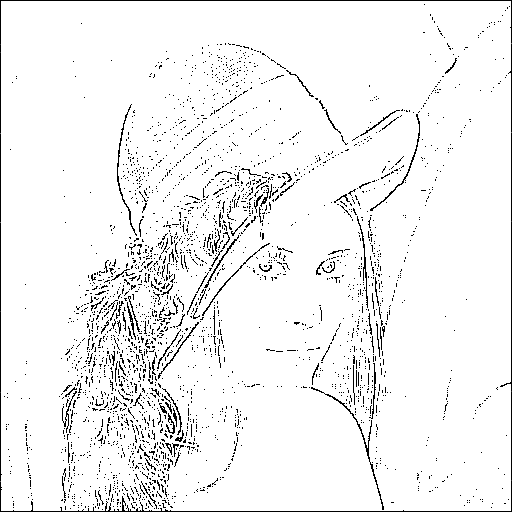
return 0;

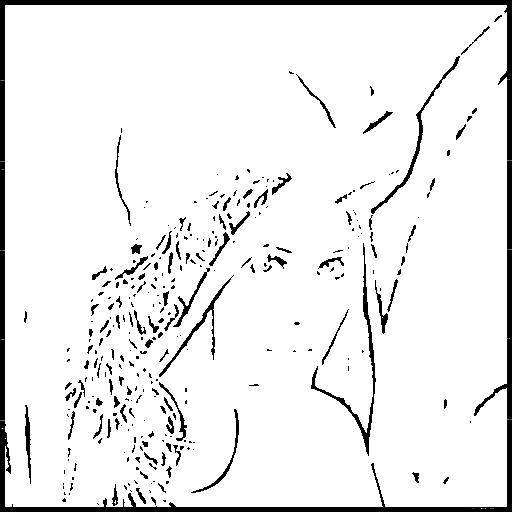
}

2.resulting images



Laplacian1 Laplacian2



minVarLaplacian Laplacian of Gaussian

Difference of Gaussian

Laplace Mask Type1: 15

Laplace Mask Type2: 15

Minimum variance Laplacian: 20

Laplace of Gaussian: 3000

Difference of Gaussian: 1