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;

}

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);

robert = Robert(src);

prewitt = Prewitt(src);

sobel = Sobel(src);

fac = FaC(src);

kirsch = Kirsch(src);

robinson = Robinson(src);

nab = NaB(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");

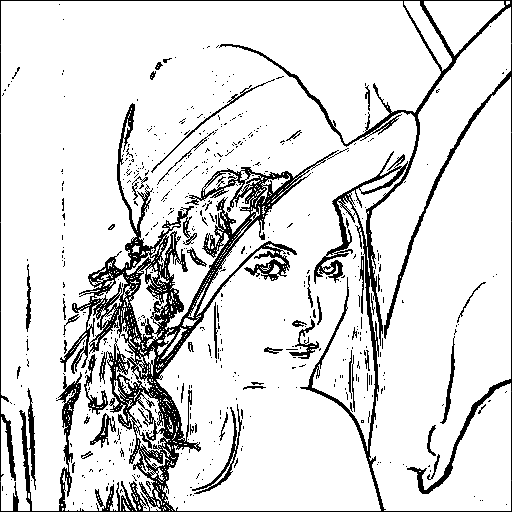
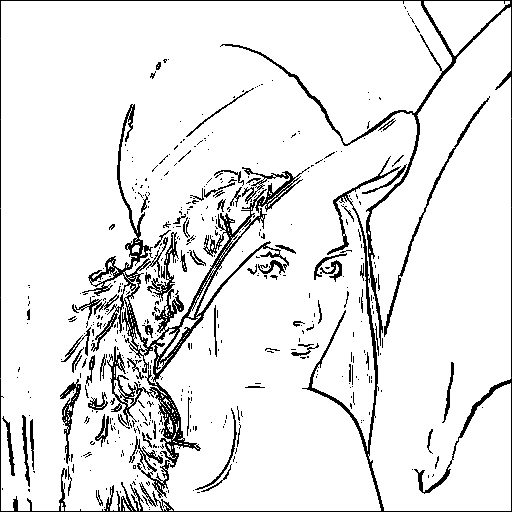
showSavePicture(nab,"Nevatia\_and\_Babu","Nevatia\_and\_Babu.bmp");

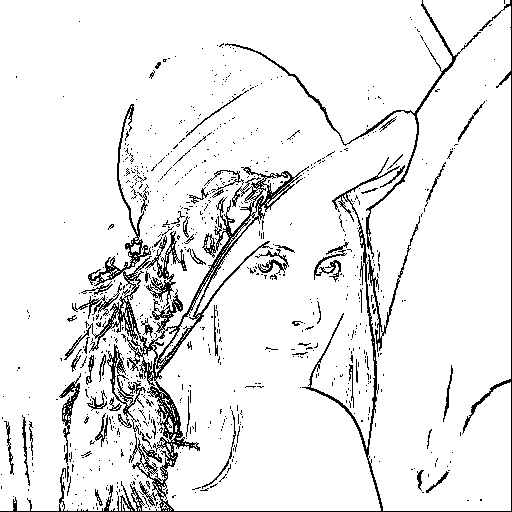
waitKey(0);

return 0;

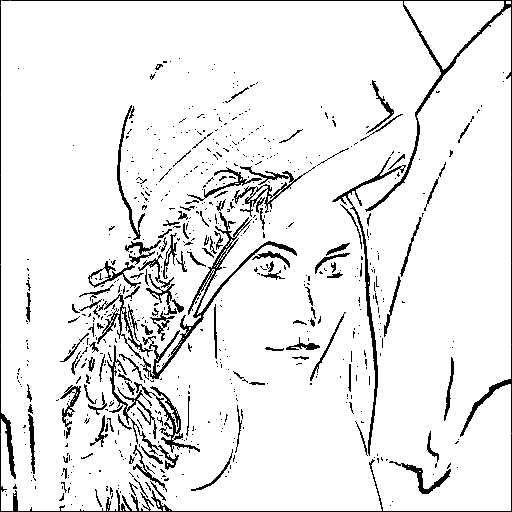
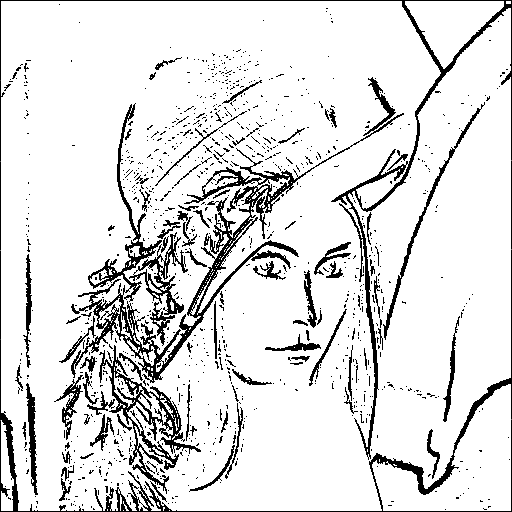
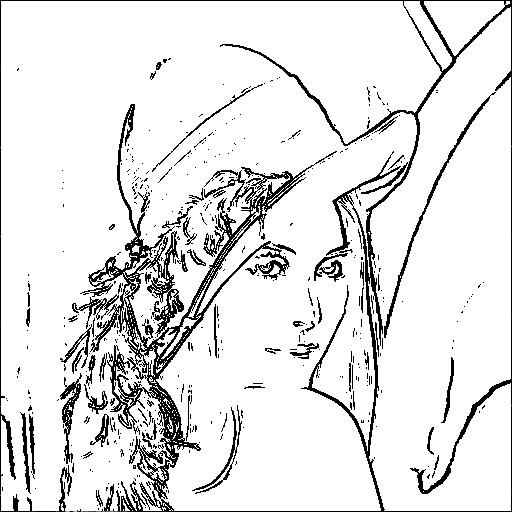
}

2.resulting images

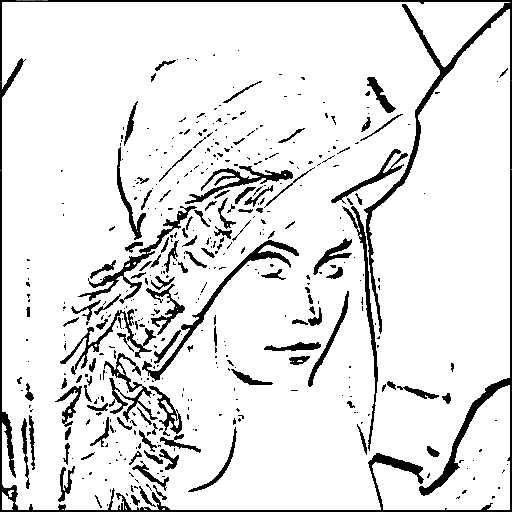




robert(30) prewitt(100) sobel(100)



Frei&Chen(100) kirsch(400) Robinson(200)



Nevatia\_and\_Babu(50000)