DOCUMENTATION DAY2

-----------------------------------------

1> To access pixel values for RGB image :

img.at<Vec3b>(i,j)[0]

img.at<Vec3b>(i,j)[1]

img.at<Vec3b>(i,j)[2]

2> Passing images by function calls :

void pic(Mat img){

//

}

pic(img) (call from main function)

By default the function call for images is call by reference,

so original image must be cloned before further operations.

3> To draw different shapes:

(a) Line : void line(Mat img,Point p1,Point p2,Scalar(),int thickness,linetype,shift)

(b) Rectangle :void rectangle(Mat img,Point p1,Point p2,Scalar(),int thickness,linetype,shift)

(c) Circle : void circle(Mat img,Point centre,int radius,Scalar(),thickness,linetype,shift)

4> Noise Reduction:

(a) Erosion : In this the anchor value is replaced with the minimum value of the

kernel pixels/matrix. So, for an image with black bakground and white

drawing, erosion will shred the edges and the lines will become thinner.

(b) Dilation : In this the anchor value is replaced with the maximum value of the

kernel pixels/matrix. So, for an image with black background and white

drawing, dilation will thicken the edges and lines.

(c) Greater the kernel size, greater will be the degree of erosion or dilation.

5> Filters:

(a) Mode : The anchor value is replaced with the mode of all pixel values in kernel.

(b) Mean : The anchor value is replaced with the mean of all pixel values in kernel.

(c) Median : The anchor value is replaced with the median of all pixel values in kernel.

(d) Gaussian : The gaussian function gives maximum weightage to the anchor and the

weightage decreases as the distance from the anchor increases. This

kernel is convolved with the image matrix and through this process,

value of the anchor is decided.

The Gaussian filter is the best among the four types.

The quality of filers is: Gaussian > Median > Mean > Mode.

6> Histograms : Images usually donot have uniform contrast. Histogram measures the

number of pixels per grayscale value. So, a higher peak at some value

indicates that more number of pixels are having that particular pixel

value. The histogram thus gives a graphical representation of the

uneven contrast in the image.

7> Equalizer : For images having uneven contrast the equalizer function spreads out

the peaks of the histograms,i.e, if too many pixels are concentrated

at one grayscale value, the function will distribute or spread out the

pixel values to values near to the peak on either side.

(a) Function : equalizeHist(src,dest);

(b) Header : #include "opencv2/improc/improc.hpp"

--------------------------------------

TASK:

--------------------------------------

CHESS BOARD PATTERN

--------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

int main()

{

int n, i, j, k, p, q;

Mat img(800, 800, CV\_8UC1, Scalar(255));

for (i = 0; i < 800; i++){

for (j = 0; j < 800; j++){

p = (i / 100);

q = (j / 100);

if ((p + q) % 2 == 0) img.at<uchar>(i, j) = 255;

else img.at<uchar>(i, j) = 0;

}

}

namedWindow("test1", WINDOW\_AUTOSIZE);

imshow("test1", img);

waitKey(0);

return 0;

}

--------------------------------

EROSION & DILATION

--------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

int main()

{

int n, i, j, k, p, q,min,max;

Mat img(400, 400, CV\_8UC1, Scalar(255));

Point p1, p2,p3;

p1.x = 100;

p1.y = 100;

p2.x = 300;

p2.y = 300;

p3.x = (p1.x + p2.x) / 2;

p3.y = (p1.y + p2.y) / 2;

rectangle(img, p1, p2, Scalar(0),5 , 8, 0);

circle(img,p3,50,Scalar(0),5,8,0);

Mat img1(img.rows,img.cols,CV\_8UC1,Scalar(255));

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

for (i = 1; i < img.rows-1; i++){

for (j = 1; j < img.cols - 1; j++){

min = 256;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

if (img.at<uchar>(i + p, j + q) < min) min = img.at<uchar>(i + p, j + q);

}

}

img1.at<uchar>(i, j) = min;

}

}

namedWindow("Erosion",WINDOW\_AUTOSIZE);

imshow("Erosion", img1);

Mat img2(img.rows, img.cols, CV\_8UC1, Scalar(255));

for (i = 1; i < img.rows; i++){

for (j = 1; j < img.cols; j++){

max = -1;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

if (img.at<uchar>(i + p, j + q)>max) max = img.at<uchar>(i + p, j + q);

}

}

img2.at<uchar>(i, j) = max;

}

}

namedWindow("Dilation", WINDOW\_AUTOSIZE);

imshow("Dilation", img2);

waitKey(0);

return 0;

}

----------------------------------------------------------------

EROSION & DILATION (with function calls and greater kernel size)

----------------------------------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void dilation(Mat img)

{

Mat img1 = img.clone();

int i, j, k, max, p, q;

for (i = 3; i < img.rows - 3; i++){

for (j = 0; j < img.cols-3; j++){

max = -1;

for (p = -3; p < 4; p++){

for (q = -3; q < 4; q++){

if (img.at<uchar>(i + p, j + q)>max) max = img.at<uchar>(i + p, j + q);

}

}

img1.at<uchar>(i, j) = max;

}

}

namedWindow("Dilation", WINDOW\_AUTOSIZE);

imshow("Dilation", img1);

waitKey(5000);

return;

}

void erosion(Mat img)

{

Mat img1 = img.clone();

int i, j, p, q, min;

for (i = 3; i < img.rows-3; i++){

for (j = 3; j < img.cols-3; j++){

min = 256;

for (p = -3; p < 4; p++){

for (q = -3; q < 4; q++){

if (img.at<uchar>(i + p, j + q) < min) min = img.at<uchar>(i + p, j + q);

}

}

img1.at<uchar>(i, j) = min;

}

}

namedWindow("Erosion", WINDOW\_AUTOSIZE);

imshow("Erosion", img1);

waitKey(5000);

return;

}

int main()

{

int n, i, j, k, min, max;

Mat img(400, 400, CV\_8UC1, Scalar(0));

Point p1, p2,p3;

p1.x = 100;

p1.y = 100;

p2.x = 300;

p2.y = 300;

p3.x = (p1.x + p2.x) / 2;

p3.y = (p1.y + p2.y) / 2;

rectangle(img, p1, p2, Scalar(255), 5, 8, 0);

circle(img, p3, 50, Scalar(255), 5, 8, 0);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

waitKey(5000);

erosion(img);

dilation(img);

waitKey(0);

return 0;

}

------------------------------------------------

FILTERS- MODE,MEAN,MEDIAN :

------------------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void bubblesort(int A[],int l,int r)

{

int n, i, j, t;

for (i = r; i >= l; i--){

for (j = l; j < r; j++){

if (A[j] > A[j + 1]){

t = A[j];

A[j] = A[j + 1];

A[j + 1] = t;

}

}

}

return;

}

void medianfilter(Mat img)

{

Mat img1 = img.clone();

int n, i, j, k, A[9], p, q;

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

n = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

A[n++] = img.at<uchar>(i + p, j + q);

}

}

bubblesort(A, 0, 8);

img1.at<uchar>(i, j) = A[4];

}

}

namedWindow("Median", WINDOW\_AUTOSIZE);

imshow("Median", img1);

waitKey(5000);

return;

}

void meanfilter(Mat img)

{

Mat img1 = img.clone();

int sum, n, i, j, p, q;

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

sum = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

sum += img.at<uchar>(i + p, j + q);

}

}

img1.at<uchar>(i, j) = (sum / 9);

}

}

namedWindow("Mean", WINDOW\_AUTOSIZE);

imshow("Mean", img1);

waitKey(5000);

return;

}

void modefilter(Mat img)

{

Mat img1 = img.clone();

int n, i, j, k, p, q, A[256],ind,max,maxind;

for (i = 1; i < img.rows; i++){

for (j = 1; j < img.cols; j++){

for (n = 0; n < 256; n++){

A[n] = 0;

}

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

ind = img.at<uchar>(i + p, j + q);

A[ind]++;

}

}

max = 0;

maxind = 0;

for (n = 0; n < 256; n++){

if (A[n] > max){

max = A[n];

maxind = n;

}

}

img1.at<uchar>(i, j) = maxind;

}

}

namedWindow("Mode", WINDOW\_AUTOSIZE);

imshow("Mode", img1);

waitKey(5000);

return;

}

int main()

{

int n, i, j, k;

Mat img;

img = imread("noisyimages.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

waitKey(5000);

modefilter(img);

meanfilter(img);

medianfilter(img);

waitKey(0);

return 0;

}

-------------------------------------------

MEDIAN FILTER (colour image)

-------------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void bubblesort(int A[], int l, int r)

{

int i, j, t;

for (i = r; i >= l; i--){

for (j = l; j < i; j++){

if (A[j] > A[j + 1]){

t = A[j];

A[j] = A[j + 1];

A[j + 1] = t;

}

}

}

return;

}

void medianfilter(Mat img)

{

Mat img1 = img.clone();

int i, j, k, n, p, q, A[9];

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

n = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

A[n++] = img.at<Vec3b>(i + p, j + q)[0];

}

}

bubblesort(A, 0, 8);

img1.at<Vec3b>(i, j)[0] = A[4];

}

}

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

n = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

A[n++] = img.at<Vec3b>(i + p, j + q)[1];

}

}

bubblesort(A, 0, 8);

img1.at<Vec3b>(i, j)[1] = A[4];

}

}

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

n = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

A[n++] = img.at<Vec3b>(i + p, j + q)[2];

}

}

bubblesort(A, 0, 8);

img1.at<Vec3b>(i, j)[2] = A[4];

}

}

namedWindow("Median", WINDOW\_AUTOSIZE);

imshow("Median", img1);

waitKey(1000);

return;

}

int main()

{

Mat img = imread("noisycolor.jpg", CV\_LOAD\_IMAGE\_COLOR);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

medianfilter(img);

waitKey(0);

return 0;

}

----------------------------------------------

GAUSSIAN FILTER (3x3 kernel for coloured image)

----------------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void gaussianfilter(Mat img)

{

Mat img1 = img.clone();

int i, j, k, A[3][3],sum,p,q;

A[0][0] = 1;

A[0][1] = 2;

A[0][2] = 1;

A[1][0] = 2;

A[1][1] = 4;

A[1][2] = 2;

A[2][0] = 1;

A[2][1] = 2;

A[2][2] = 1;

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols-1; j++){

sum = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

sum += A[p+1][q+1] \* (img.at<Vec3b>(i + p, j + q)[0]);

}

}

sum = sum / 16;

img1.at<Vec3b>(i, j)[0] = sum;

}

}

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

sum = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

sum += A[p + 1][q + 1] \* (img.at<Vec3b>(i + p, j + q)[1]);

}

}

sum = sum / 16;

img1.at<Vec3b>(i, j)[1] = sum;

}

}

for (i = 1; i < img.rows - 1; i++){

for (j = 1; j < img.cols - 1; j++){

sum = 0;

for (p = -1; p < 2; p++){

for (q = -1; q < 2; q++){

sum += A[p + 1][q + 1] \* (img.at<Vec3b>(i + p, j + q)[2]);

}

}

sum = sum / 16;

img1.at<Vec3b>(i, j)[2] = sum;

}

}

namedWindow("Gaussian", WINDOW\_AUTOSIZE);

imshow("Gaussian", img1);

waitKey(1000);

return;

}

int main()

{

Mat img = imread("noisycolor.jpg", CV\_LOAD\_IMAGE\_COLOR);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

gaussianfilter(img);

waitKey(0);

return 0;

}

-----------------------------------------------

GAUSSIAN FILTER (5x5 kernel for coloured image)

-----------------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void gaussianfilter(Mat img)

{

Mat img1 = img.clone();

int i, j, k, sum, p, q;

int A[5][5] = { 1, 4, 7, 4, 1, 4, 16, 26, 16, 4, 7, 26, 41, 26, 7, 4, 16, 26, 16, 4, 1, 4, 7, 4, 1 };

for (i = 2; i < img.rows - 2; i++){

for (j = 2; j < img.cols-2; j++){

sum = 0;

for (p = -2; p < 3; p++){

for (q = -2; q < 3; q++){

sum += A[p+2][q+2] \* (img.at<Vec3b>(i + p, j + q)[0]);

}

}

sum = sum / 273;

img1.at<Vec3b>(i, j)[0] = sum;

}

}

for (i = 2; i < img.rows - 2; i++){

for (j = 2; j < img.cols - 2; j++){

sum = 0;

for (p = -2; p < 3; p++){

for (q = -2; q < 3; q++){

sum += A[p + 2][q + 2] \* (img.at<Vec3b>(i + p, j + q)[1]);

}

}

sum = sum / 273;

img1.at<Vec3b>(i, j)[1] = sum;

}

}

for (i = 2; i < img.rows - 2; i++){

for (j = 2; j < img.cols - 2; j++){

sum = 0;

for (p = -2; p < 3; p++){

for (q = -2; q < 3; q++){

sum += A[p + 2][q + 2] \* (img.at<Vec3b>(i + p, j + q)[2]);

}

}

sum = sum / 273;

img1.at<Vec3b>(i, j)[2] = sum;

}

}

namedWindow("Gaussian", WINDOW\_AUTOSIZE);

imshow("Gaussian", img1);

waitKey(1000);

return;

}

int main()

{

Mat img = imread("noisycolor.jpg", CV\_LOAD\_IMAGE\_COLOR);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

gaussianfilter(img);

waitKey(0);

return 0;

}

------------------------------------

HISTOGRAM & EQUALIZER

------------------------------------

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

using namespace cv;

using namespace std;

void histogram(Mat img)

{

Mat img1(500, 256, CV\_8UC1, Scalar(255));

int A[256], i, j, n, k;

for (k = 0; k < 256; k++){

A[k] = 0;

}

for (i = 0; i < img.rows; i++){

for (j = 0; j < img.cols; j++){

A[img.at<uchar>(i, j)]++;

}

}

for (i = 0; i < 255; i++){

for (j = 0; j < (500) - A[i]; j++){

img1.at<uchar>(j, i) = 255;

}

for (j = (500 - A[i]); j < 500; j++){

img1.at<uchar>(j, i) = 0;

}

}

namedWindow("Histogram", WINDOW\_AUTOSIZE);

imshow("Histogram", img1);

waitKey(5000);

return;

}

int main()

{

Mat img = imread("noisyimages.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

namedWindow("Image", WINDOW\_AUTOSIZE);

imshow("Image", img);

histogram(img);

Mat img2;

equalizeHist(img, img2);

histogram(img2);

waitKey(0);

return 0;

}