Non Local Mathematical Morphology

Introduction

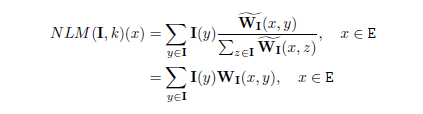
Mathematical morphology is an approach to image analysis that characterises an image by transformations with simple geometrical interpretation . The original image, denoted by I, is studied by its interaction with small subsets, named structuring elements (SEs), obtained by convolution in the max-plus algebra . It has been applied successfully to a large number of fields including biomedical microscopy, material science, remote sensing, and medical imaging. The classical approach is characterised by two main properties : (1) SE is \_xed, i.e., does not depend on the spatial position at which it is centred; (2) the basic morphological operations are invariant under translation. This idea has been extended to grey scale images, using a complete lattice formulation [23]. This paper deals with a case of adaptive mathematical morphology. Adaptive mathematical morphology refers to morphological filtering techniques that adjust the SE to the local context of the image. The approach in this paper is based on the adaptive morphology framework, but where the local structuring

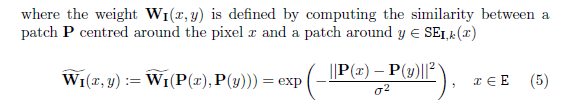
element is \estimated" taking into consideration the whole image. We thus refer it as a "nonlocal" approach, following the terminology initiated in [1]. The term of \nonlocal morphology" has been already considered in previous works. On the one hand, Salembier in [22] proposed a straightforward generalisation of nonlocal means filter to morphological filters. The nonlocal PDE is solved using numerical methods which includes nonlocal distances as weights. It is obvious that such PDE-based approach.

Non-flat Nonlocal Morphology

In order to fully understand how and why nonlocal morphology works, we will begin with a detailed description of nonlocal means and the theory which support the approach. Nonlocal processing refers to the general methodology of designing energies using nonlocal comparison of patches extracted in the image. Starting

from the initial paper by Baudes et al., nonlocal energies have proved to be efficient for many imaging problems, including denoising , semi-supervised classification and segmentation. Recently, nonlocal schemes for image processing have received a lot of attention. Rather than considering only the vector associated to one pixel to compute pixel similarities, patches around these pixels are considered. These patches capture the dependencies of neighbouring pixels and thus can distinguish textural patterns. Nonlocal means filters have been proposed in mainly for denoising applications. The filtering idea consists in computing a weighted average of the input image in a neighbourhood of size k:





Here, ||Pj||is the Euclidean norm of the patch P of size lxl as a vector in Rlxl and sigma is a smoothing parameter. Thus, pixels with similar neighbourhoods are given larger weights compared to pixels whose neighbourhoods look different. The algorithm makes explicit use of the fact that repetitive patterns appear in most of the natural images.

Code:

#include<bits/stdc++.h>

using namespace std;

#include "image\_io.h"

#include "adaptive\_functions.h"

int main()

{

string file\_src="test.pgm";

image img=read\_image(file\_src);

int s\_row,s\_col,s\_x,s\_y;

int patch\_size,sigma;

cout<<"Enter the number of row in the structuring element: ";

cin>>s\_row;

cout<<"Enter the number of column in the structuring element: ";

cin>>s\_col;

cout<<"Enter the x coordinate of origin: ";

cin>>s\_x;

cout<<"Enter the y coordinate of origin: ";

cin>>s\_y;

cout<<"Enter the size of patch(patch will always be square so enter the dimesion of one side): ";

cin>>patch\_size;

cout<<"Enter the value of smoothing factor: ";

cin>>sigma;

//img.print\_image();

image e\_img1=adaptive\_erosion(img,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img1=adaptive\_dilation(img,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img1.write\_image("eroded\_lenna1.pgm");

d\_img1.write\_image("dilated\_lenna1.pgm");

///////////////////////////////////////////////////////////////////////////////////////////////////////////

/\* image e\_img2=adaptive\_erosion(e\_img1,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img2=adaptive\_dilation(d\_img1,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img2.write\_image("eroded\_lenna2.pgm");

d\_img2.write\_image("dilated\_lenna2.pgm");

////////////////////////////////////////////////////////////////////////////////////////////////////////////

image e\_img3=adaptive\_erosion(e\_img2,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img3=adaptive\_dilation(d\_img2,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img3.write\_image("eroded\_lenna3.pgm");

d\_img3.write\_image("dilated\_lenna3.pgm");

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image e\_img4=adaptive\_erosion(e\_img3,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img4=adaptive\_dilation(d\_img3,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img4.write\_image("eroded\_lenna4.pgm");

d\_img4.write\_image("dilated\_lenna4.pgm");

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image e\_img5=adaptive\_erosion(e\_img4,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img5=adaptive\_dilation(d\_img4,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img5.write\_image("eroded\_lenna5.pgm");

d\_img5.write\_image("dilated\_lenna5.pgm");

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image e\_img6=adaptive\_erosion(e\_img5,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image d\_img6=adaptive\_dilation(d\_img5,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

e\_img6.write\_image("eroded\_lenna6.pgm");

d\_img6.write\_image("dilated\_lenna6.pgm");\*/

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image o\_img1=adaptive\_opening(img,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image c\_img1=adaptive\_closing(img,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

o\_img1.write\_image("opening\_lenna1.pgm");

c\_img1.write\_image("closing\_lenna1.pgm");

c\_img1=d\_img1-e\_img1;

c\_img1.write\_image("outline.pgm");

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/\* image o\_img2=adaptive\_opening(o\_img1,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image c\_img2=adaptive\_closing(c\_img1,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

o\_img2.write\_image("opening\_lenna2.pgm");

c\_img2.write\_image("closing\_lenna2.pgm");

image o\_img3=adaptive\_opening(o\_img2,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image c\_img3=adaptive\_closing(c\_img2,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

o\_img3.write\_image("opening\_lenna3.pgm");

c\_img3.write\_image("closing\_lenna3.pgm");

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image o\_img4=adaptive\_opening(o\_img3,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image c\_img4=adaptive\_closing(c\_img3,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

o\_img4.write\_image("opening\_lenna4.pgm");

c\_img4.write\_image("closing\_lenna4.pgm");

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

image o\_img5=adaptive\_opening(o\_img4,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

image c\_img5=adaptive\_closing(c\_img4,s\_row,s\_col,s\_x,s\_y,patch\_size,sigma);

o\_img5.write\_image("opening\_lenna5.pgm");

c\_img5.write\_image("closing\_lenna5.pgm"); }

Code For Producing Image:

#include<bits/stdc++.h>

using namespace std;

#include "image\_io.h"

int main()

{

image img;

img.set\_height(30);

img.set\_width(30);

img.set\_range\_max(255);

vector<vector<int> > bitmap(30,vector<int> (30,0));

for(int i=0;i<30;i++)

{

for(int j=0;j<30;j++)

{

bitmap[i][j]=i\*j;

bitmap[i][j]%=255;

}

}

img.set\_bitmap(bitmap);

img.write\_image("test.pgm");

}

Code for Image IO (Header File):

#define vvi vector<vector<int> >

#define vi vector<int>

class image

{

private:

int width,height,range\_max;

//height is the number of rows

//width is the number of cols

public:

vvi bitmap;

void set\_width(int width)

{

this->width=width;

}

void set\_height(int height)

{

this->height=height;

}

void set\_range\_max(int range\_max)

{

this->range\_max=range\_max;

}

void set\_bitmap(vvi bitmap)

{

this->bitmap=bitmap;

}

int get\_width()

{

return this->width;

}

int get\_height()

{

return this->height;

}

int get\_range\_max()

{

return this->range\_max;

}

vvi get\_bitmap()

{

return this->bitmap;

}

image()

{

}

image(int width,int height,int range\_max)

{

this->width=width;

this->height=height;

this->range\_max=range\_max;

bitmap.resize(height,vector<int>(width));

}

void print\_image()

{

cout<<width<<" "<<height<<" "<<range\_max<<endl;

for(int i=0; i<height; i++)

{

for(int j=0; j<width; j++)

{

cout<<bitmap[i][j]<<endl;

}

}

}

void print\_image(int param)

{

cout<<"width="<<width<<" height="<<height<<" range="<<range\_max<<endl;

// cout<<bitmap.size()<<endl;

// cout<<bitmap[0].size();

}

void write\_image(string src\_file\_name)

{

int dimx, dimy, range;

dimx=this->width;

dimy=this->height;

range=this->range\_max;

int i, j;

FILE \*fp = fopen(src\_file\_name.c\_str(), "wb"); /\* b - binary mode \*/

(void) fprintf(fp, "P5\n%d %d\n%d\n", dimx, dimy,range);

for (j = 0; j < dimy; ++j)

{

for (i = 0; i < dimx; ++i)

{

int temp=this->bitmap[j][i];

temp+=(range+1);

static unsigned char color[1];

// color[0]+=(range+1);

color[0] = temp%(range+1);

(void) fwrite(color, 1, 1, fp);

}

}

(void) fclose(fp);

}

image duplicate\_image(image img)

{

image dup\_image;

dup\_image.set\_width(img.get\_width());

dup\_image.set\_height(img.get\_height());

dup\_image.set\_range\_max(img.get\_range\_max());

dup\_image.set\_bitmap(img.get\_bitmap());

return dup\_image;

}

image operator-(image& img)

{

image sub\_img;

sub\_img=sub\_img.duplicate\_image(img);

int height=img.get\_height();

int width=img.get\_width();

//cout<<height<<" "<<width<<endl;

for(int i=0;i<height;i++)

{

for(int j=0;j<width;j++)

{

sub\_img.bitmap[i][j]=this->bitmap[i][j]-img.bitmap[i][j];

}

}

return sub\_img;

}

image operator+(image& img)

{

image add\_img;

add\_img=add\_img.duplicate\_image(img);

int height=img.get\_height();

int width=img.get\_width();

//cout<<height<<" "<<width<<endl;

for(int i=0;i<height;i++)

{

for(int j=0;j<width;j++)

{

add\_img.bitmap[i][j]=this->bitmap[i][j]+img.bitmap[i][j];

}

}

return add\_img;

}

};

image read\_image(string src\_file\_name)

{

int buff\_len=256;

FILE \*pf=fopen(src\_file\_name.c\_str(), "rb");

char buf[buff\_len], \*t;

unsigned int w, h, d;

int r;

t = fgets(buf, buff\_len, pf);

do

{

t = fgets(buf, buff\_len, pf);

}

while ( strncmp(buf, "#", 1) == 0 );

r = sscanf(buf, "%u %u", &w, &h);

r = fscanf(pf, "%u", &d);

fseek(pf, 1, SEEK\_CUR);

image img(w,h,d);

for(int i=0; i<h; i++)

{

for(int j=0; j<w; j++)

{

unsigned char arr[1];

fread(arr, sizeof(unsigned char), 1, pf);

img.bitmap[i][j]=(unsigned int)arr[0];

}

}

return img;

}