單元十、灰階型態學影像處理

陳慶瀚

2004-12-05

灰階數學型態學的原理和方法的提出：

J. Serra, Image Analysis and Mathematical Morphology. London:Academic, 1982.

灰階數學型態學的影像處理應用：

S. R. Sternberg, “Grayscale morphology”, Comput. Vision, Graphics, Image Processing, vol. 35, pp.333-355, 1986.

灰階數學型態學的理論探討：

Henk J.A.M. Heijmans, “Theoretical aspects of gray-level morphology”,IEEE Trans. On Pattern Analysis and Machine Intelligence, vol.13, no.6, 1991.

1. 灰階數學型態學運算

基本運算

若有灰階值元素的結構單元 ***B*** 和影像 ***A***，

灰階擴張(*dilation*) $D_G(*)$定義爲:

$$D_G(\mathrm{A},\mathrm{B}) = \max_{[j,k]\in \mathrm{B}} \{a[m-j,n-k]+b[j,k]\}$$

灰階侵蝕(*erosion*) $E_G(*)$定義爲:

$$E_G(\mathrm{A},\mathrm{B}) = \min_{[j,k]\in \mathrm{B}} \{a[m+j,n+k]-b[j,k]\}$$

二階灰階型態學運算

斷開(Opening) –

$$O_G(\mathrm{A},\mathrm{B}) = D_G(E_G(\mathrm{A},\mathrm{B}),\mathrm{B})$$

閉合(Closing) –

$$C_G(\mathrm{A},\mathrm{B}) = -O_G(-\mathrm{A},-\mathrm{B})$$

結構單元 ***B*** 的設計在灰階型態學運算扮演重要的角色。一般我們使用對稱型態的 structuring elements，*b*[*j*,*k*] = *b*[-*j*,-*k*]。在簡化的情況下，也可以使用 ***B*** = *constant* = 0，如此將使得上述運算變成：

灰階擴張(*dilation*):

$$D_G(A,B) = \max_{[j,k]\in B}\{a[m-j,n-k]\} = \max_B(A)$$

灰階侵蝕(*erosion*):

$$E_G(A,B) = \min_{[j,k]\in B}\{a[m-j,n-k]\} = \min_B(A)$$

灰階斷開*(Opening)* –

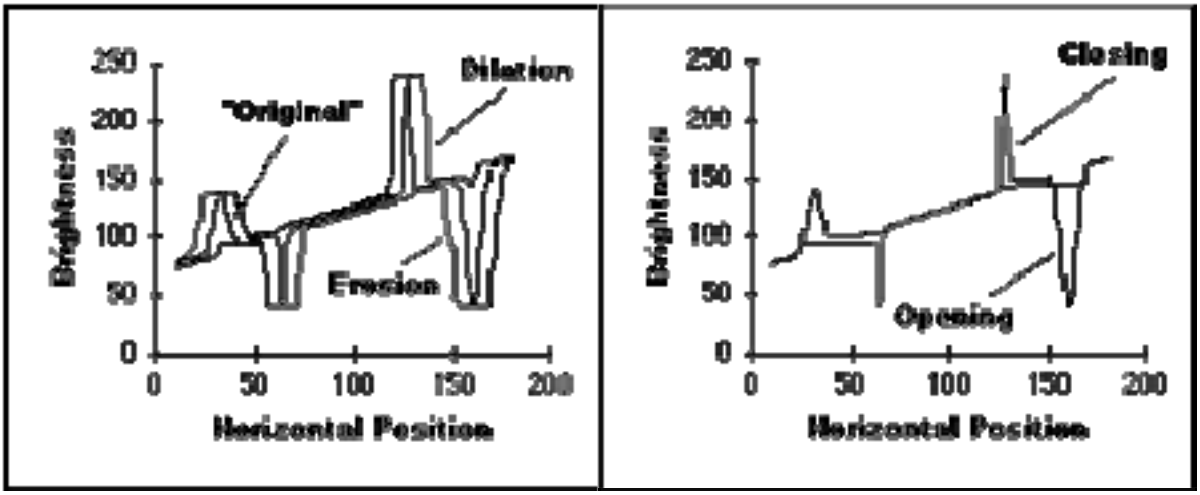
$$O_G(A,B) = \max_B(\min_B(A))$$

灰階閉合*(Closing)* –

$$C_G(A,B) = \min_B(\max_B(A))$$

在此條件下，灰階型態學運算就成爲 *maximum filter* 和 *minimum filter*。

下圖爲灰階型態學運算的 1-D 範例。



高階灰階型態學運算

型態學平滑化(Morphological smoothing)

$$\begin{aligned} MorphSmooth(A,B) &= C_G(O_G(A,B),B) \\ &= \min(\max(\max(\min(A)))) \end{aligned}$$

型態學梯度(Morphological gradient)

典型的梯度濾波器產生一個包含強度(magnitude)和方向(orientation)的向量，型態學梯度僅估測梯度強度(*gradient magnitude)*:

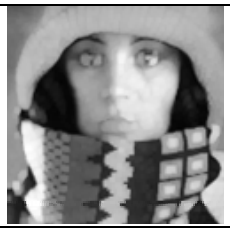
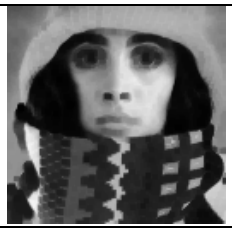
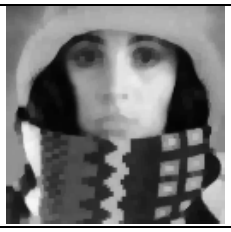
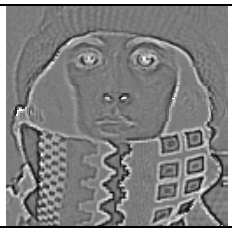
$$\begin{aligned} Gradient(\mathtt{A},\mathtt{B}) &= \frac{1}{2}\left(D_G(\mathtt{A},\mathtt{B}) - E_G(\mathtt{A},\mathtt{B})\right) \\ &= \frac{1}{2}\left(\max(\mathtt{A}) - \min(\mathtt{A})\right) \end{aligned}$$

型態學 Laplacian

$$\begin{aligned} Laplacian(\mathtt{A},\mathtt{B}) &= \frac{1}{2}\left(\left(D_G(\mathtt{A},\mathtt{B}) - \mathtt{A}\right) - \left(\mathtt{A} - E_G(\mathtt{A},\mathtt{B})\right)\right) \\ &= \frac{1}{2}\left(D_G(\mathtt{A},\mathtt{B}) + E_G(\mathtt{A},\mathtt{B}) - 2\mathtt{A}\right) \\ &= \frac{1}{2}\left(\max(\mathtt{A}) + \min(\mathtt{A}) - 2\mathtt{A}\right) \end{aligned}$$

灰階型態學影像處理範例

 (原始影像)

				
Dilation	Erosion	Smoothing	Gradient	Laplacian

習題 1、設計以下函式並測試femme136x136.raw影像的各函式運算結果：

```
void grayDilation(uc2D &im1, uc2D &im2)
void grayErosion(uc2D &im1, uc2D &im2)
void grayClosing(uc2D &im1, uc2D &im2)
void grayOpenning(uc2D &im1, uc2D &im2)
void graySmoothing(uc2D &im1, uc2D &im2)
```

```
void grayGradient(uc2D &im1, uc2D &im2)
void grayLaplacian(uc2D &im1, uc2D &im2)
```

其中結構單元的宣告可改成如下形式：

```
int seX[NumPtSet]={0,-1,0,1,0},
    seY[NumPtSet]={-1,0,0,0,1},
    seZ[NumPtSet]={0,0,0,0,0}; //gray level of SE, 0 by default
```

2. *Top-hat transform*

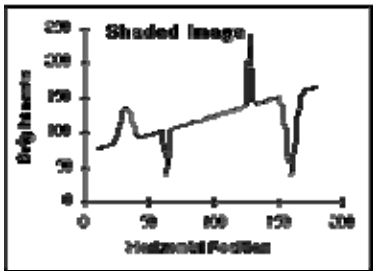
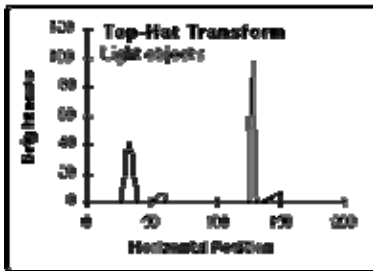
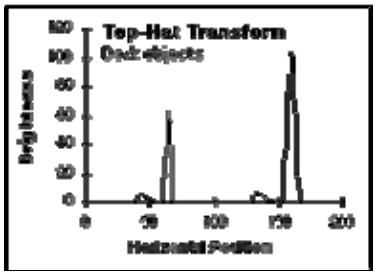
針對以灰階值呈現的物件，Meyer 提出 *top-hat transform* 的型態學影像處理：

亮的物件位於暗的背景 –

$$TopHat(\mathtt{A},\mathtt{B}) = \mathtt{A} - (\mathtt{A} \circ \mathtt{B}) = \mathtt{A} - \max_{\mathtt{B}}\left(\min_{\mathtt{B}}(\mathtt{A})\right)$$

暗的物件位於亮的背景 –

$$TopHat(\mathtt{A},\mathtt{B}) = (\mathtt{A} \bullet \mathtt{B}) - \mathtt{A} = \min_{\mathtt{B}}\left(\max_{\mathtt{B}}(\mathtt{A})\right) - \mathtt{A}$$

		
原始訊號	亮的物件轉換	暗的物件轉換

適應性二值化(Thresholding)

由灰階型態學的觀點，一個可隨區域變化的二值化臨界值可定義如下：

$$\theta[m,n] = \frac{1}{2}\left(\max(\mathtt{A}) + \min(\mathtt{A})\right)$$

適應性對比增強(contrast enhancement)

根據結構單元 B 的區域範圍，可以定義區域對比值:

$$LocalContrast(\mathtt{A},\mathtt{B}) = \max(\mathtt{A}) - \min(\mathtt{A})$$

可隨區域變化的適應性對比增強則定義如下:

$$c[m,n] = scale \cdot \frac{A - \min(A)}{\max(A) - \min(A)}$$

習題 2、設計以下函式並測試finger300x300.raw影像的各函式運算結果：

```
void morphologicalThresholding(uc2D &im1, uc2D &im2)
void morphologicalContrastEnhance(uc2D &im1, uc2D &im2)
```

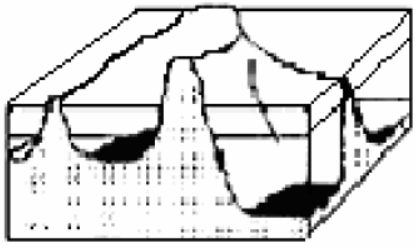
3. 分水嶺演算法

分水嶺演算法或分水嶺轉換是近年來形態學影像處理最受矚目的方法，它主要應用在影像切割(segmentation)，Meyer首先於1990年提出Morphological segmentation的原始構想和演算法，由於計算複雜度極高， Vincent和Soille隨後於1991年提出快速的演算法版本。

F. Meyer, S. Beucher, “Morphological segmentation”, Journal of Visual Communication and Image Representation, vol. 1, no.1, pp.21-46. (1990)

Luc Vincent and Pierre Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, VOL. 13, NO. 6, pp. 583-598, June 1991.

Here is a brief description of the watershed algorithm on a gradient image. The gradient image is of the same size of the original one with its pixels being gradients represented by integer grey-level values. A gradient image can be considered as a topographic surface as follow:



A *minimum* refers to a connected component in which the pixels are with the same grey level value and the value of this component is strictly lower than the values of its neighboring pixels.

Imagine that we pierce each minimum of the topographic surface and that we plunge this surface into a lake at a constant vertical speed. The water entering through the holes floods the surface. During the flooding, two or more floods coming from different minima may merge. In order to avoid this, dams are built up at potential merging points of the surface. At the end, only dams emerge. These dams are *watersheds*.

The original serial watershed transformation by ordered queue starts by detecting and labelling the

initial seeds (minima). Ordered region growing is then performed starting from these minima. Non-labeled pixels are connected to different components in an increasing order of gray-levels. Inside the unlabelled flat areas, called plateaus of non-minima, components progress synchronously such that they incorporate equal extents within the plateau. The recursive label propagation (flooding) is performed using an ordered queue, which is an array of H FIFO queues, with one queue for each of the H gray-levels in the image. For the case of grayscale image in raw format, one requires 256 FIFOs of size equal to the image size, since gray-scale information is represented using one byte per pixel.

演算法

(1) 資料結構設計

For implementing the ordered queue algorithm, the size of all the FIFOs required should be equal to the image size. For 256 gray-levels, this amounts to utilizing 256 such FIFOs. However, a pixel having an intensity, say *h*, and having a downward brim to its plateau, is flooded earlier than a pixel having an intensity greater than *h*. This implies that unless all the pixels having intensity *h* have been flooded, flooding of pixels of intensity *h+1* will not take place. Therefore, we conclude that if we take the histogram first and then perform the flooding step, we will need only one FIFO of size equal to the image size, and not 256 FIFOs as required in the original algorithm. However, this implementation requires proper addressing of the pixels belonging to each gray-level. For software implementation, a simple `new` operator will do the job of dynamically allocating portions of the FIFO to respective frequencies corresponding to the different gray-levels. For the hardware implementation, we require a small amount of additional memory to keep track of the FIFO's starting and ending locations, in order to locate pixels of particular gray-level.

This algorithm assumes the availability of functions to which if a memory location is passed, it gives the number and addressesNG(p) (x[k] in C code) of neighboring pixels. This assumption is made for the sake of simplicity of explanation and is not made in the architectural implementation of the algorithm. Figure 1 shows the overall initializations required for the algorithm. N is the number of gray-levels.

```
#define N 256   //256 gray level
#define S 65536 // S denotes the number of pixels in the 256x256 image
#define T 65535 // T is equal to S-1
declarations: input[S],label[S],sort[S],
output[S],fifoq[S],outputlabel[S],*fifo[256];
```

(2)**Sorting**

Let us assume that we already have sorted the image pixels and have the histogram in the array hist[i], where I denotes the particular gray-level. Knowing the frequency distribution, we can allocate the memory for the respective gray-levels dynamically. The code segment shows the dynamic allocation of memory:

```
for(i=0;i<=255;i++)
{
    fifo[i] = new int(hist[i]);
}
```

(3)Locating minima

This requires two scannings of the whole image. In our case, we do it row wise. In the first scan, we detect the pixels which are not regional minima, i.e., the pixels whose neighborhoods have at least one pixel of gray-scale value less than itself. The second scan checks to find out the single minima pixels, minima plateau and non-minima plateau. In the same scan, output image of labels (output label) is also given unique labels for minima pixels to reduce the number of memory fetches later on. The algorithm to detect and label the minima is given:

```
- First Scan the entire image
   For every pixel p' ∈N_G(p)
   such that input[p]>input[p']
   label p is (not a minimum);
   break;
- Second Scan the entire image
   if p is not (not a minimum) then
   {
    For every pixel p' ∈N_G(p)
    {
        if input[p] = input[p'] then
        {
            if p' is (not a minimum) then
            {
                Nonmimima plateau detected;
                label p is (not a minimum)
            }
        }
        else
        {
            plateau detected;
            label p is (not a minimum)
        }
    }
    if both nonminima plateau and minima plateau remains undetected then
    {
       label p is minima;
       output[i] = outlabel;
       outlabel++
   }
   if nonminima plateau detected, then
        label all the pixels of this plateau as (not a minimum)
   if plateau detected then
```

```
   find whether this plateau is a minima plateau or a nonminima plateau;
   put all the pixels in fifoq;
   {
        if minima plateau, then
        label all the pixels of this plateau to minima;
        {
            for every pixel q of this plateau
            output[q] = outlabel; outlabel++;
        }
    }
 }
```

(4) Initialization

In this step, we need to scan the array name *label*, which indicates whether a particular pixel is a minima or a nonminima. Since we use a single FIFO with locations equal to the image size, we need two arrays, mincount[256] and compcount[256] to take care of the starting and ending locations of a particular FIFO. Hence, in the initialization step we scan the label array and get the frequency distribution of minima's. Following code shows the code for initialization of FIFO memory with minima addresses.

```
for(i=0;i<=T;i++)
{
    if(label[i] == 2)
    {
        *(fifo[(input[i])] + mincount[(input[i])]) = i;
        mincount[input[i]] = mincount[input[i]] + 1;
    }
}
```

(5)**Flooding**

In this phase of the algorithm, we start from the lowest gray-level minima, check to see if output label is not enabled (outputlabel = 1 indicates that the particular pixel has already been visited and labeled), get the neighboring pixels and give them the same label as one given to that particular pixel. Further, all the neighboring pixels are put in a queue. We then check to see if the outputlabels are enabled for the neighboring pixels and if they are not labeled, their outputlabels are enabled after putting them all in the queues of their respective FIFOs. This process is repeated until the entire image is covered. The flooding algorithm is shown:

```
for(i=0;i<=255;i++)//1
{
    if (mincount[i] == 0);
```

```
    else //2
    {
       for(j=0;j<hist[i];j++)//3
       {
          if((outputlabel[*(fifo[i] + compcount[i])]) == 0)
              foutputlabel[*(fifo[i] + compcount[i])] = 1;
       }
       tempcall = *(fifo[i] + compcount[i]);
       call = neighbor(x,tempcall);
       compcount[i] = compcount[i] + 1;
       for(k=0;k<call;k++)//4
       {
          if((outputlabel[x[k]]==0) && (label[x[k]] != 2)) //5
          {
             output[x[k]] = output[*(fifo[i] + compcount[i] -1)];
             outputlabel[x[k]] = 1;
             if ((input[x[k]]==i) && (label[x[k]] == 2));
             else
             {
                 *(fifo[input[x[k]]] + mincount[input[x[k]]]) = x[k];
                  mincount[input[x[k]]] = mincount[input[x[k]]] + 1;
             }
           } //5
         } //4
       } //3
    } //2
} //1
```

下圖展示分水嶺演算法的影像切割結果：

10	8	6	9	6	8	8	8	8	8
4	7	6	9	9	6	8	8	8	8
4	4	8	8	8	8	6	8	8	8
4	4	8	2	2	2	8	6	8	8
4	4	8	2	2	2	8	6	8	8
4	4	8	8	8	8	8	6	8	8
9	9	9	9	6	6	6	10	10	10
9	9	9	3	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10
9	9	9	9	9	9	9	9	9	9

(a)

0	0	0	0	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1
0	0	2	2	2	2	2	2	1	1
0	2	2	2	2	2	2	2	2	1
3	3	2	2	2	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3

(b)

(a) Input image, and after processing detected minimas (shown in

bold) (b) Output image of labels after flooding the input image

習題 3、設計以下函式並測試finger300x300.raw影像的各函式運算結果：

```
void morphologicalThresholding(uc2D &im1, uc2D &im2)
void morphologicalContrastEnhance(uc2D &im1, uc2D &im2)
```