Technical University of Łódź
Institute of Electronics
Medical Electronics Division

Image Processing and Computer Graphics

Python Imaging Library 3

Author: Marek Kociński

March 2010

1 Purpose

To get acquainted with Image filtering using direct access to the pixels. The basic morphological operations will be applied to binary images. $3D$ image will be load into memory and selected slice will be displayed as $2D$ image.

Time

3×45 minutes

2 Tasks

1. Open Python interpreter window (Start→ Programy→ EPD32-6.0.2 → IDLE)
2. Open new Editor Window (File → New Window) and write your code into it.
3. Import needed modules, e.g. *Image*
4. Very often it is needed to get direct access to image data. Function *load()* returns a pixel access object that can be used to read and modify pixels. The access object behaves like a 2-dimensional array, so you can do:

```
pix = im.load()
print pix[x,y]
pix[x,y] = value
```

   Create inversion of the brighteness level of the *goldhill.bmp* image 1.

```
im = Image.open("goldhill.bmp")
im = im.convert(``L``)

sx = im.size[0]
sy = im.size[1]

pix = im.load()
for y in range(sy):
        for x in range(sx):
                pix[x,y] = 255 - pix[x,y]

im.save('neg.bmp')
```

5. Create an function that performs image convolution with 3×3 masks. (Hint: Designing and Implementing Linear Filters in the Spatial Domain).

(a) Gray-level image *(b) Inverted gray-level image*

Figure 1: Pixel operations

```
def convolution2d(img,k, size=3):
    """This function make image convolutin with cernel.

    Kernel as a list from 0 to 8."""
    fn="convolution2d"
    print "Function %s" %fn
    pix = img.load()

    if size == 3:
        print "***Kernel size = 3"
        sx = img.size[0]
        sy = img.size[1]

        nimg = Image.new (img.mode, img.size)
        npix = nimg.load()
        val=[]
        for y in range(1,sy-1):
            for x in range(1,sx-1):
                npix[x,y]= (k[0]*pix[x-1,y-1] + k[1]*pix[x,y-1]
                          + k[2]*pix[x+1,y-1] + k[3]*pix[x-1,y]
                          + k[4]*pix[x,y] + k[5]*pix[x+1,y]
                          + k[6]*pix[x-1,y+1] + k[7]*pix[x,y+1]
                          + k[8]*pix[x+1,y+1])
      return nimg
```

Load image *blood1.bmp* and perform convolution with Prewitt masks:

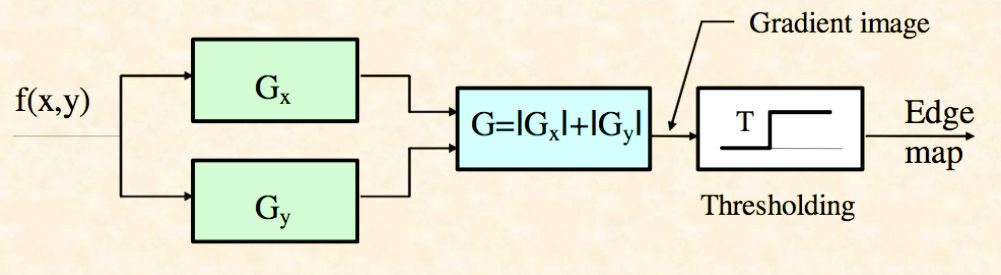


Figure 2: *Edge detection procedure (from* Image Processing *lectures by P.Strumiłło and M.Strzelecki)*

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

(a) Prewitt (horiz.) — k1 *(b) Prewitt (vertic.) — k3*

Pass convolution cernels as the function argument in the following convetnion:

$$k1 = [-1, -1, -1, 0, 0, 0, 1, 1, 1]$$

$$k3 = [-1, 0, 1, -1, 0, 1, -1, 0, 1]$$

Find edges of the image using procedure showed in the Fig. 2. Find appropriate threshold value (Fig. 3).

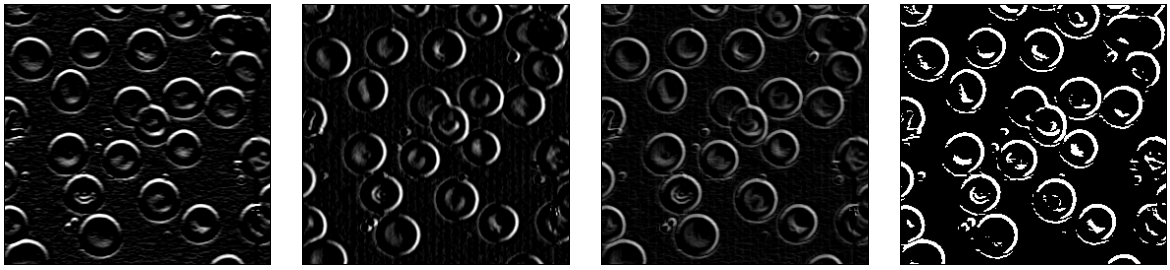
Add booth resulting images using pixel operations.

```
a1 = Image.new(im.mode, im.size)

pix_a1_v = a1_v.load()
pix_a1_h = a1_h.load()
pix_a1 = a1.load()

for y in range(sy):
    for x in range(sx):
        pix_a1[x,y] = (pix_a1_h[x,y] + pix_a1_v[x,y] )/2
```

6. Load images *bin1.bmp* and *bin2.bmp*. For both images apply function dilation *dilaet2D()* and erosion *erode2D()*. Use different structuring element (Fig. 4). Define each element as tupe or list, for example first element, which is *cross* shapled is defined as followes:

$$se1 = (0, 1, 0, 1, 1, 1, 0, 1, 0,)$$

Use at least 6 different structuring elements (Fig. 5).



(a) Prewitt (vert.) *(b) Prewitt (horiz.)* *(c) Sum of gradients* *(d) After thresholding*

Figure 3: Edge detection using Prewitt mask

```
def dilate2D(im,se,sh=1):
    """dilate2D function is to do dilation of img image (2D).
    se - tuple describes any FULL SIZE se.
    ------- se(n) - ON piksels

    ----Dilation for WHITE objects with BLACK background!!! ---"""

    fn="dilate2D(im,se)"
    if(sh): print "Function %s" %fn

    sX,sY = im.size
    imD = im.load()
    nim = Image.new (im.mode, im.size)
    nimD= nim.load()

    seX = 3 #size of se
    seY = 3
    hx = seX/2
    hy = seY/2

    for y in range(hy,sY-hy):
        for x in range(hx,sX-hx):
            if( imD[x,y] == 255 ):
                if(se[0]): nimD[x-1,y-1] =255
                if(se[1]): nimD[x,  y-1] =255
                if(se[2]): nimD[x+1,y-1] =255
                if(se[3]): nimD[x-1,y]   =255
                if(se[4]): nimD[x,  y]   =255
                if(se[5]): nimD[x+1,y]   =255
                if(se[6]): nimD[x-1,y+1] =255
                if(se[7]): nimD[x,  y+1] =255
                if(se[8]): nimD[x+1,y+1] =255
```

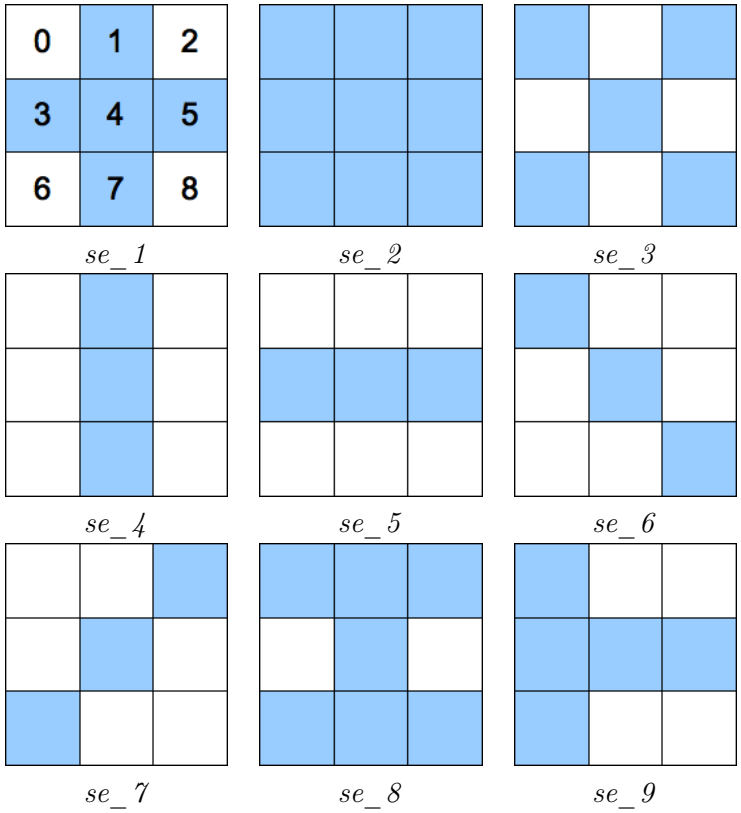


Figure 4: Various structuring elements for morphological operations

```
    if(sh): print "***dilation done..."
    return nim

def erode2D(im,se,sh=1):
    """erode2D function is to make erosion od img image (2D).
    se - tuple describes any FULL SIZE se.
    -------- se(n) - ON piksels

    EROSION for WHITE objects with BLACK background!!!"""

    fn="erode2D(im,se)"
    if(sh): print "Function %s" %fn

    sX,sY = im.size
    imD = im.load()
    nim = Image.new (im.mode, im.size)
    nimD= nim.load()
```

```
seX = 3 #size of se
seY = 3
hx = seX/2
hy = seY/2

for y in range(hy,sY-hy):
    for x in range(hx,sX-hx):
        x0=[]
        if( imD[x,y] == 255 ):
            if se[0]:x0.append(imD[x-1,y-1])
            if se[1]:x0.append(imD[x,  y-1])
            if se[2]:x0.append(imD[x+1,y-1])
            if se[3]:x0.append(imD[x-1,y])
            if se[4]:x0.append(imD[x,  y])
            if se[5]:x0.append(imD[x+1,y])
            if se[6]:x0.append(imD[x-1,y+1])
            if se[7]:x0.append(imD[x,  y+1])
            if se[8]:x0.append(imD[x+1,y+1])
            if all(x0) : nimD[x,y] = 255

if(sh): print "***erode done..."
return nim
```

7. Define new functions for morphological opening and for morphological closing of $2D$ images (Fig. 6):

   - *MorphologicalOpen2D(....)* which is combination of dilation and erosion of the image
   - *MorphologicalClose2D(...)* which is combination of erosion and dilation of the image
   - *Morphological gradient(...)* which is difference between dilation and erosion

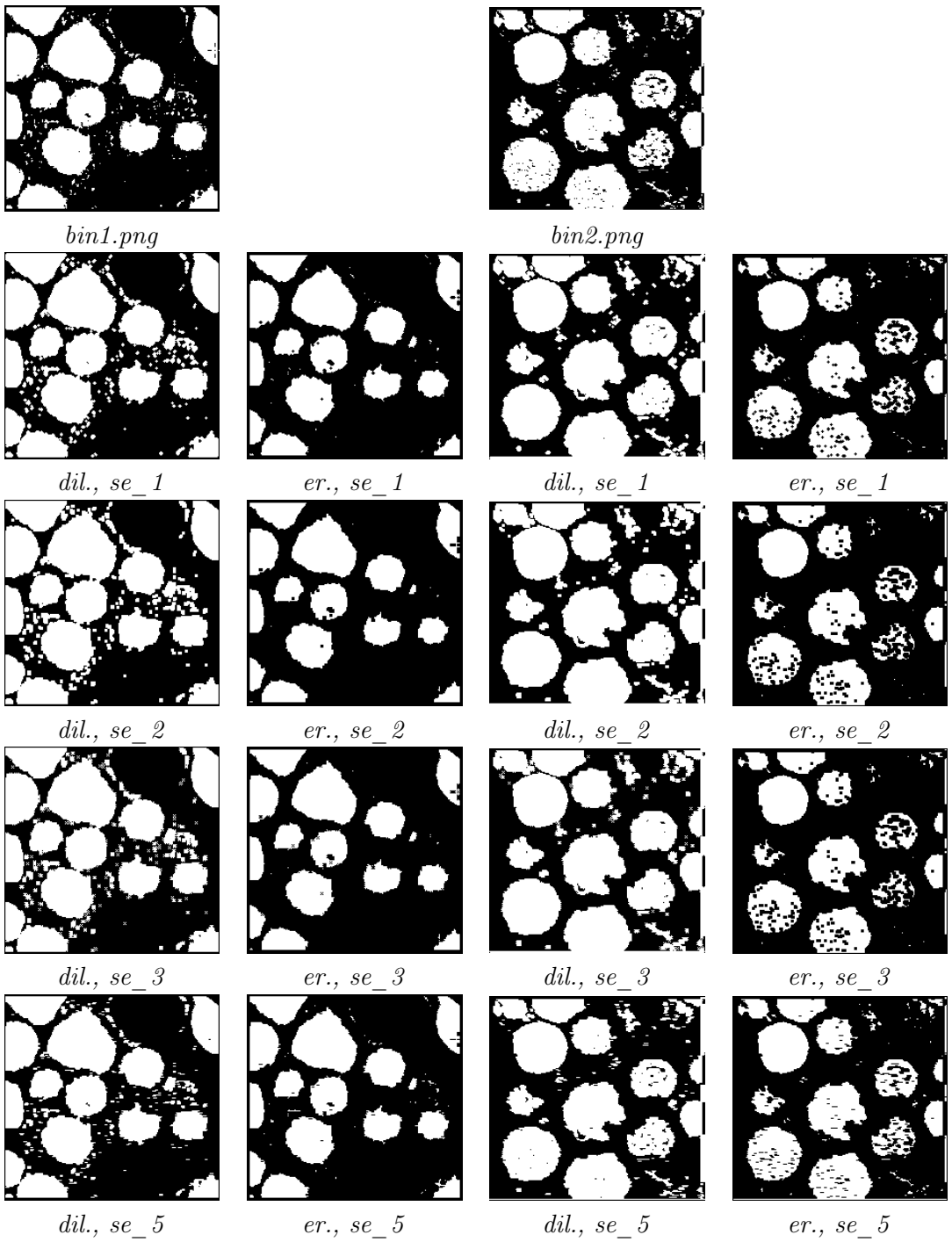
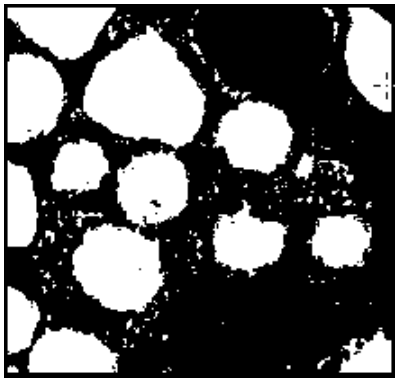
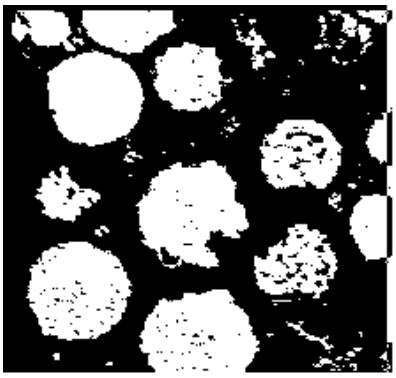


Figure 5: Results of morphological operations of image filtering with various structuring elements



(a) Oryginal image 1



(b) Oryginal image 2

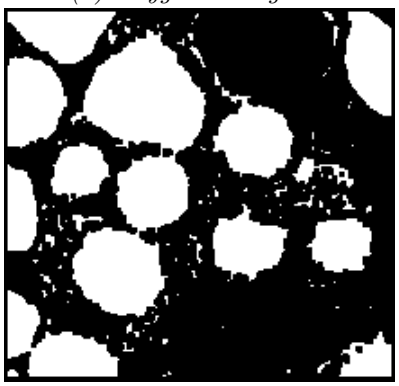


image 1 — opening

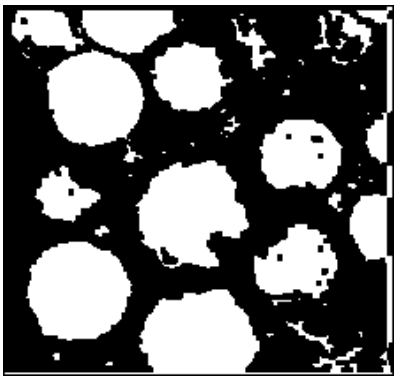


image 2 — openinig

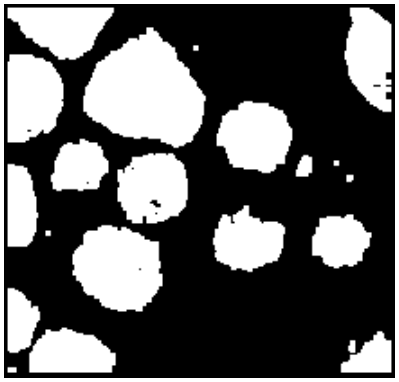


image1 — closing

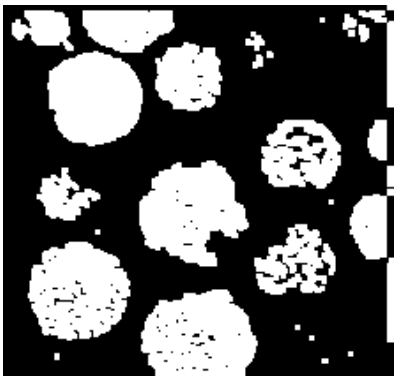


image 2 — closing

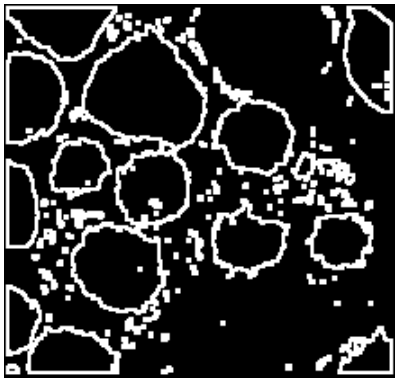


image1 — gradient

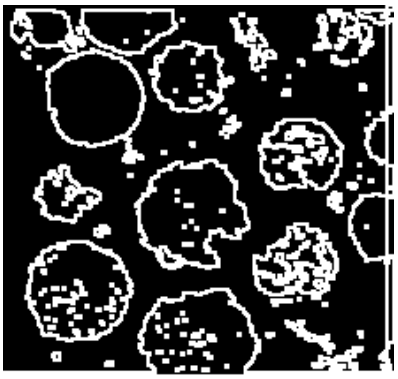


image 2 — gradient

Figure 6: Morphological operations