Name: Trong-An Bui
Student ID: 107323903

Homework 2 and 3

This section addresses basic image manipulation and processing using the core scientific modules NumPy and SciPy. Some of the operations covered by this tutorial may be useful for other kinds of multidimensional array processing than image processing. In particular, the submodule `scipy.ndimage` provides functions operating on n-dimensional NumPy arrays.

1. Opening and writing to image files

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f) # uses the Image module (PIL)

import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```

2. Displaying images

```
f = misc.face(gray=True)  # retrieve a grayscale image
import matplotlib.pyplot as plt
```

```
plt.imshow(f, cmap=plt.cm.gray)
```

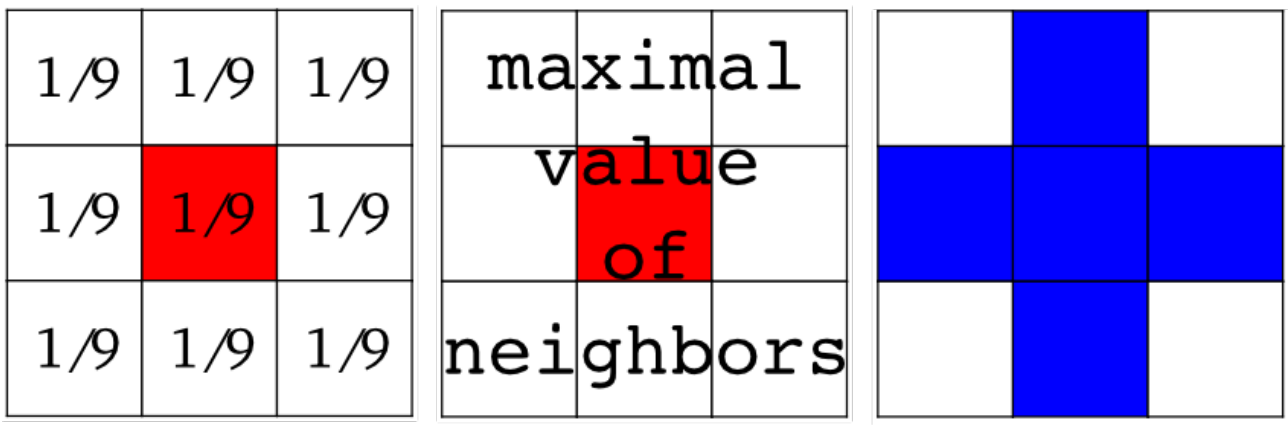
Increase contrast by setting min and max values

```
plt.imshow(f, cmap=plt.cm.gray, vmin=30, vmax=200)
# Remove axes and ticks

plt.axis('off')
```

Draw contour lines:

```
plt.contour(f, [50, 200])
```

3. Image filtering



Gaussian filter

```
from scipy import misc

face = misc.face(gray=True)

blurred_face = ndimage.gaussian_filter(face, sigma=3)

very_blurred = ndimage.gaussian_filter(face, sigma=5)
```

Uniform filter

```
local_mean = ndimage.uniform_filter(face, size=11)
```

4. Denoising

```
from scipy import misc

f = misc.face(gray=True)

f = f[230:290, 220:320]

noisy = f + 0.4 * f.std() * np.random.random(f.shape)
```

A **Gaussian filter** smoothes the noise out

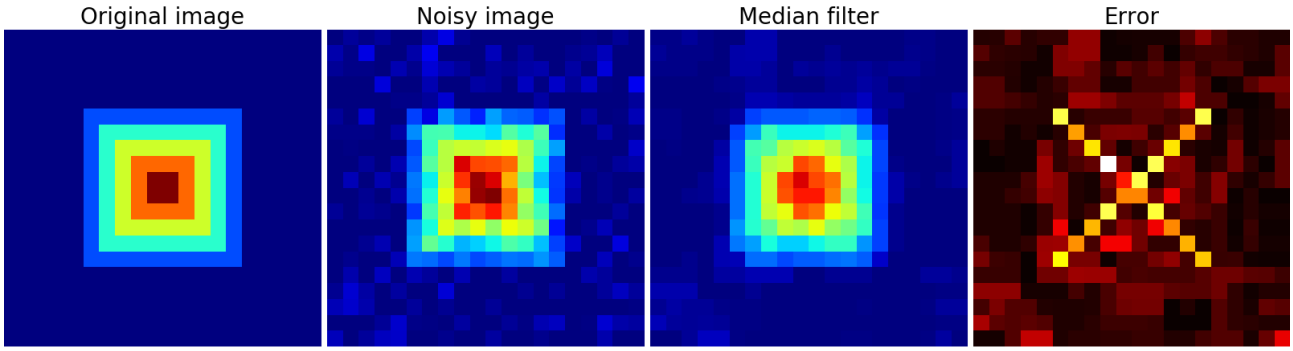
```
gauss_denoised = ndimage.gaussian_filter(noisy, 2)
```

A median filter preserves better the edges:

```
med_denoised = ndimage.median_filter(noisy, 3)
```

Median filter: better result for straight boundaries (**low curvature**):

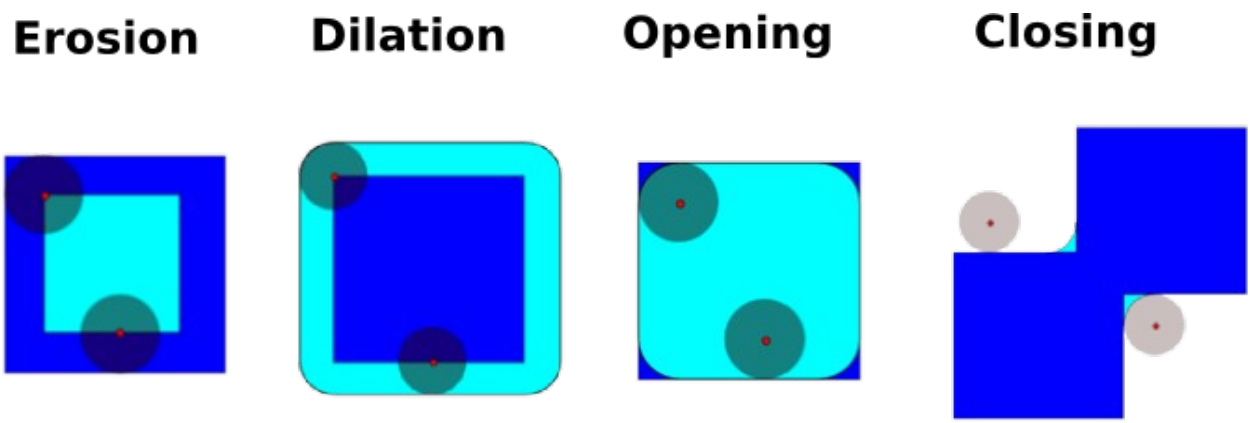
```
im = np.zeros((20, 20))

im[5:-5, 5:-5] = 1

im = ndimage.distance_transform_bf(im)

im_noise = im + 0.2 * np.random.randn(*im.shape)

im_med = ndimage.median_filter(im_noise, 3)
```



5. Mathematical morphology

Erosion = minimum filter. Replace the value of a pixel by the minimal value covered by the structuring element.:

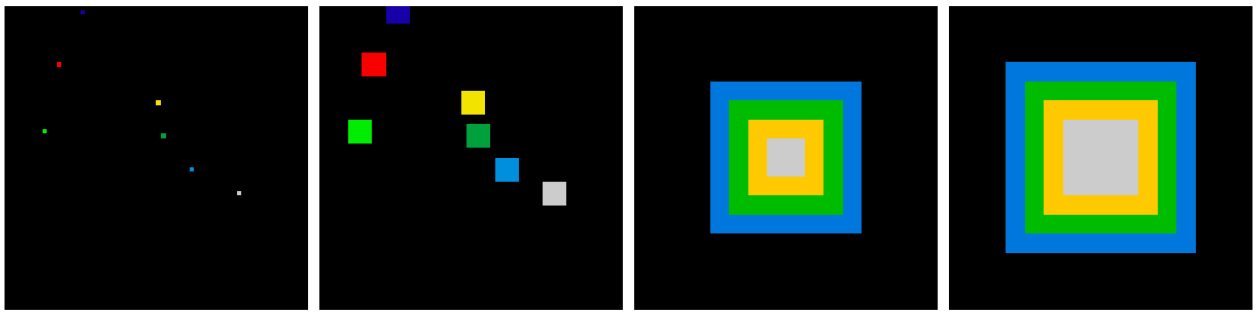
```
a = np.zeros((7,7), dtype=np.int)

a[1:6, 2:5] = 1

ndimage.binary_erosion(a).astype(a.dtype)

#Erosion removes objects smaller than the structure

ndimage.binary_erosion(a,
structure=np.ones((5,5))).astype(a.dtype)
```



Dilation: maximum filter:

```
a = np.zeros((5, 5))

a[2, 2] = 1

ndimage.binary_dilation(a).astype(a.dtype)

np.random.seed(2)

im = np.zeros((64, 64))
```

```
x, y = (63*np.random.random((2, 8))).astype(np.int)

im[x, y] = np.arange(8)


bigger_points = ndimage.grey_dilation(im, size=(5, 5),
structure=np.ones((5, 5)))


square = np.zeros((16, 16))

square[4:-4, 4:-4] = 1

dist = ndimage.distance_transform_bf(square)

dilate_dist = ndimage.grey_dilation(dist, size=(3, 3), \

        structure=np.ones((3, 3)))
```



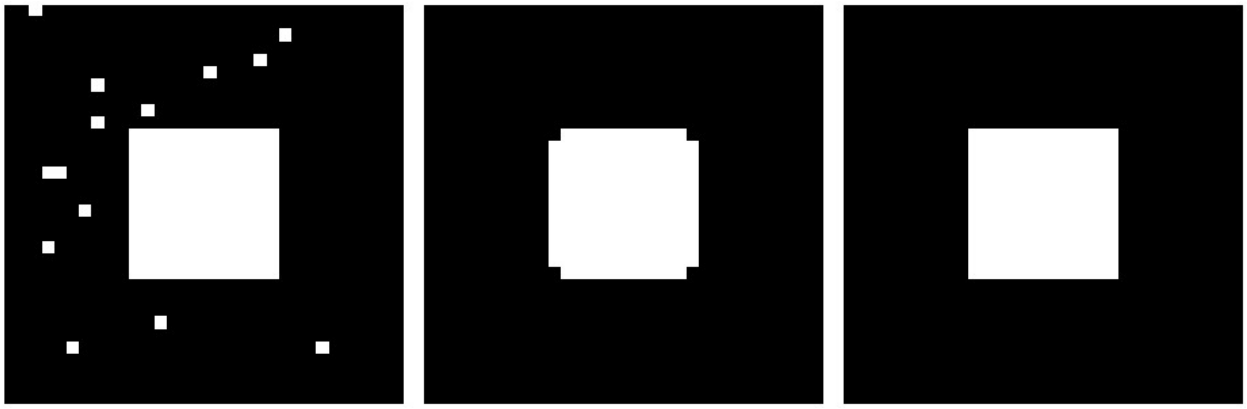
Opening: erosion + dilation:

```
a = np.zeros((5,5), dtype=np.int)

a[1:4, 1:4] = 1; a[4, 4] = 1

# Opening removes small objects

ndimage.binary_opening(a, structure=np.ones((3,3))).astype(np.int)

# Opening can also smooth corners

ndimage.binary_opening(a).astype(np.int)
```

Application: remove noise:

```
square = np.zeros((32, 32))

square[10:-10, 10:-10] = 1

np.random.seed(2)

x, y = (32*np.random.random((2, 20))).astype(np.int)
```

```
square[x, y] = 1


open_square = ndimage.binary_opening(square)


eroded_square = ndimage.binary_erosion(square)

reconstruction = ndimage.binary_propagation(eroded_square,
mask=square)
```



6. Feature extraction

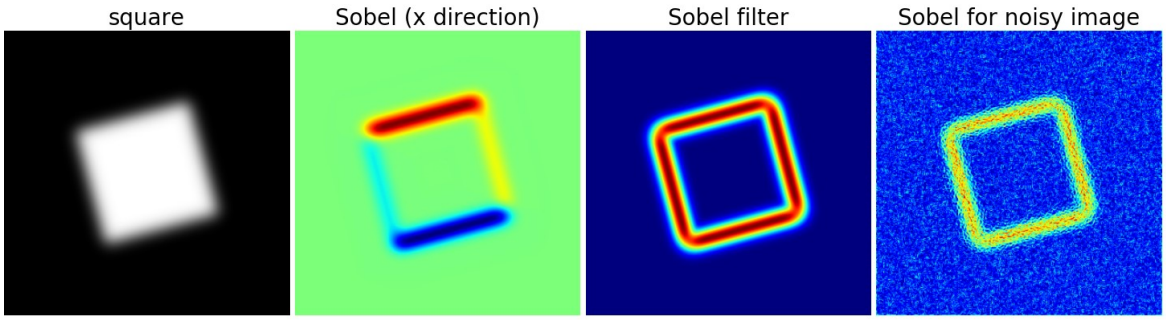
6.1. Edge detection

Synthetic data:

```
im = np.zeros((256, 256))

im[64:-64, 64:-64] = 1

im = ndimage.rotate(im, 15, mode='constant')

im = ndimage.gaussian_filter(im, 8)
```

Use a **gradient operator** (**Sobel**) to find high intensity variations:

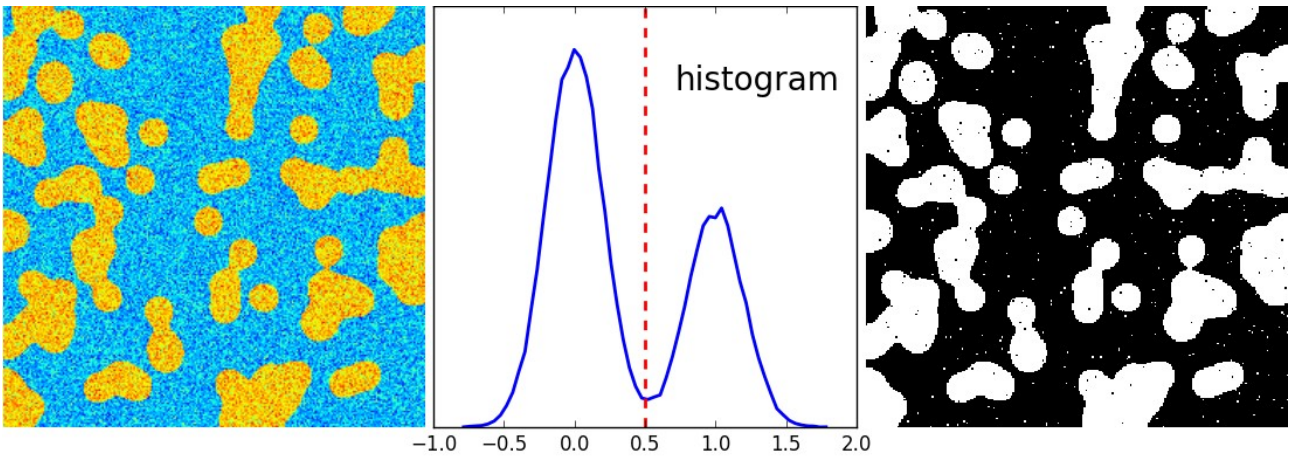
```
sx = ndimage.sobel(im, axis=0, mode='constant')

sy = ndimage.sobel(im, axis=1, mode='constant')

sob = np.hypot(sx, sy)
```



6.2. Segmentation

Histogram-based segmentation (no spatial information)

```
n = 10

l = 256

im = np.zeros((l, l))

np.random.seed(1)

points = l*np.random.random((2, n**2))

im[(points[0]).astype(np.int), (points[1]).astype(np.int)] = 1

im = ndimage.gaussian_filter(im, sigma=l/(4.*n))

mask = (im > im.mean()).astype(np.float)

mask += 0.1 * im

img = mask + 0.2*np.random.randn(*mask.shape)

hist, bin_edges = np.histogram(img, bins=60)

bin_centers = 0.5*(bin_edges[:-1] + bin_edges[1:])

binary_img = img > 0.5
```



Use mathematical morphology to clean up the result:

```
# Remove small white regions

open_img = ndimage.binary_opening(binary_img)

# Remove small black hole

close_img = ndimage.binary_closing(open_img)
```