Computational Approaches for Life Scientists

Faculty of Biology
Technion
Spring 2013

Digital Image Processing(2)

Based on lectures prepared by Benny Chor and Asaf Zaritsky,
Modified by Amir Rubinstein

Outline

Introduction to Digital Images

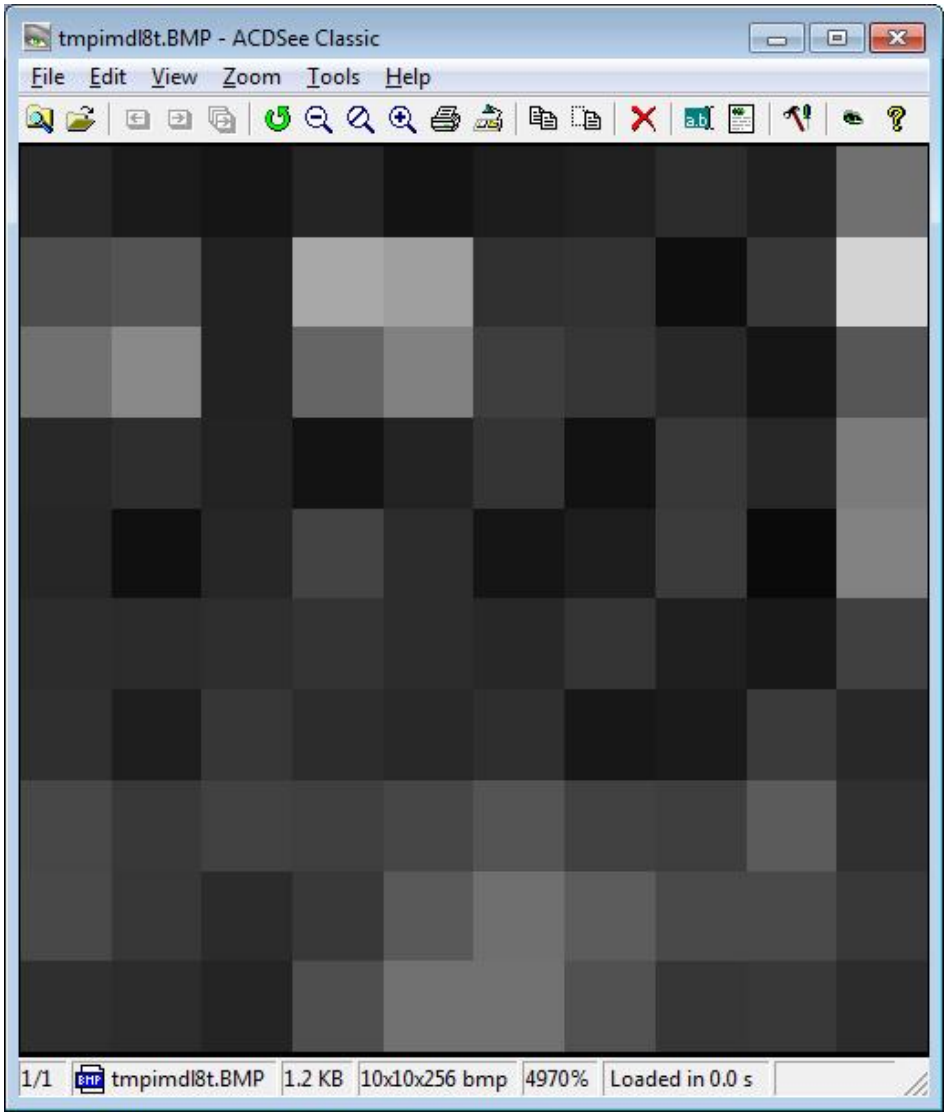
- **representation**
- **synthetic** images
- simple **operators** on images

- **Segmentation by thresholding**
  - Otsu segmentation

- Next time (today):
- Edge Detection
  - using erosion and dilation

- Noise Reduction (Denoising)
  - local means and local medians

Reminder – image representation



38, 26, 21, 36, 19, 28, 33, 44, 31, 112,

77, 83, 34, 168, 159, 48, 50, 14, 55, 211,

112, 137, 34, 101, 129, 62, 54, 40, 21, 86,

41, 46, 35, 19, 35, 52, 18, 57, 39, 123,

38, 16, 38, 67, 45, 21, 29, 59, 10, 130,

45, 43, 46, 51, 44, 39, 53, 31, 24, 64,

47, 30, 54, 45, 40, 46, 23, 26, 58, 40,

71, 57, 66, 63, 70, 84, 65, 62, 91, 49,

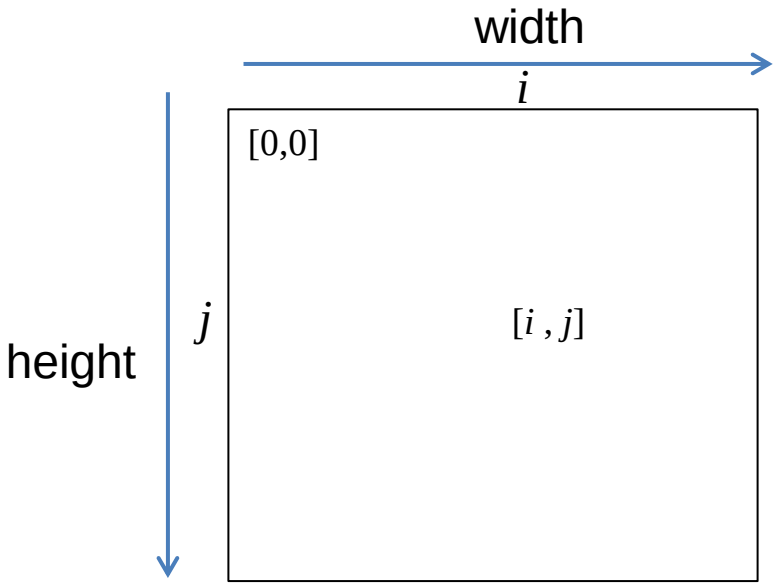
72, 55, 43, 57, 90, 111, 92, 73, 74, 56,

47, 45, 36, 78, 114, 113, 81, 54, 57, 44

Reminder - images in Python

- The Python Imaging Library – PIL
  - Read / write images
  - Display image
  - Basic image processing

- download: http://www.lfd.uci.edu/~gohlke/pythonlibs/#pil
  - take the version that fits your machine and python version



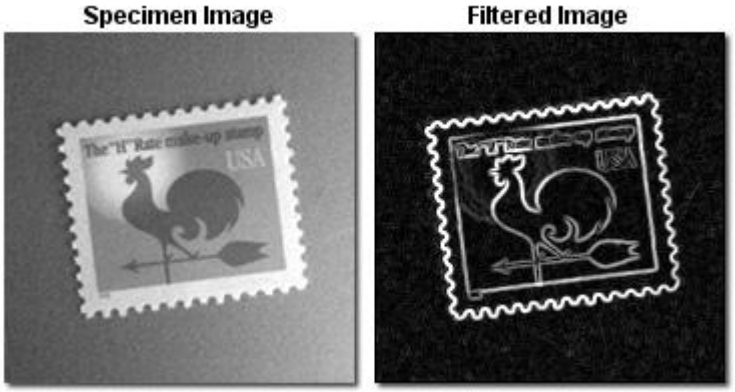
Reminder – handling images in Python

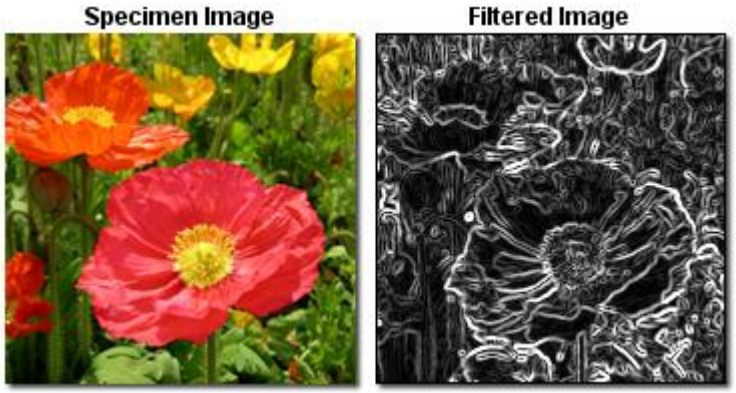
```
from PIL import Image

# load image
im = Image.open("./guess.jpg")

# turn a color image to B&W for simplicity
im = im.convert('L')

# get image data
#loads image into matrix
#changes to matrix WILL affect image
im_mat = im.load()
print(im_mat[0,0])
im_mat[0,0] = 255

print(im.size, im.format, im.mode)
```

```
# display
im.show()

# crop
w, h = im.size
region = im.crop((200,410,210,420))
region.show()

# rotate
rot = im.rotate(45)
rot.show()

#save
im.save("./tmp.bmp", "bmp")
```

Motivation – Edge Detection

- Edge - sharp change in intensity between close pixels
- Usually captures much of the meaningful information in the image



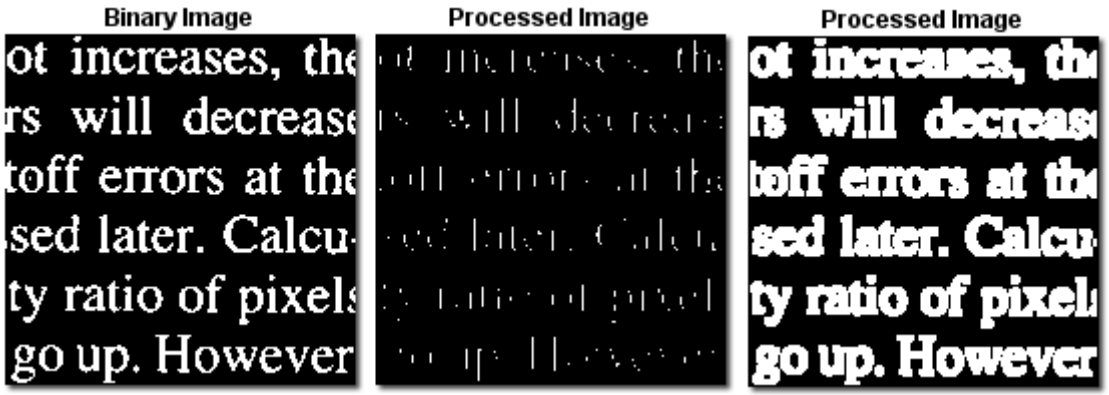


images extracted using Sobel filter from:

http://micro.magnet.fsu.edu/primer/java/digitalimaging/russ/sobelfilter/index.html

Erosion and Dilation

- Erosion - the removal of pixels from the periphery of a (white) feature.
- Dilation - the adding of pixels to that periphery.



Erosion Dilation

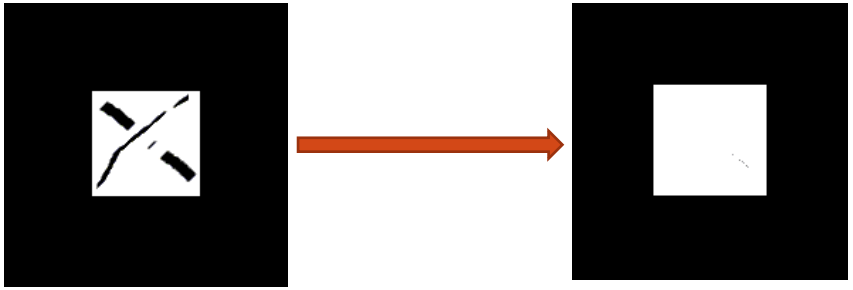
- If we assume bright foreground and dark background:
  - Erosion shrinks foreground areas, and holes grow.
  - Dilation enlarges foreground areas, and holes shrink.

Images from:
http://micro.magnet.fsu.edu/primer/java/digitalimaging/russ/erosiondilation/index.html

Erosion and Dilation - examples

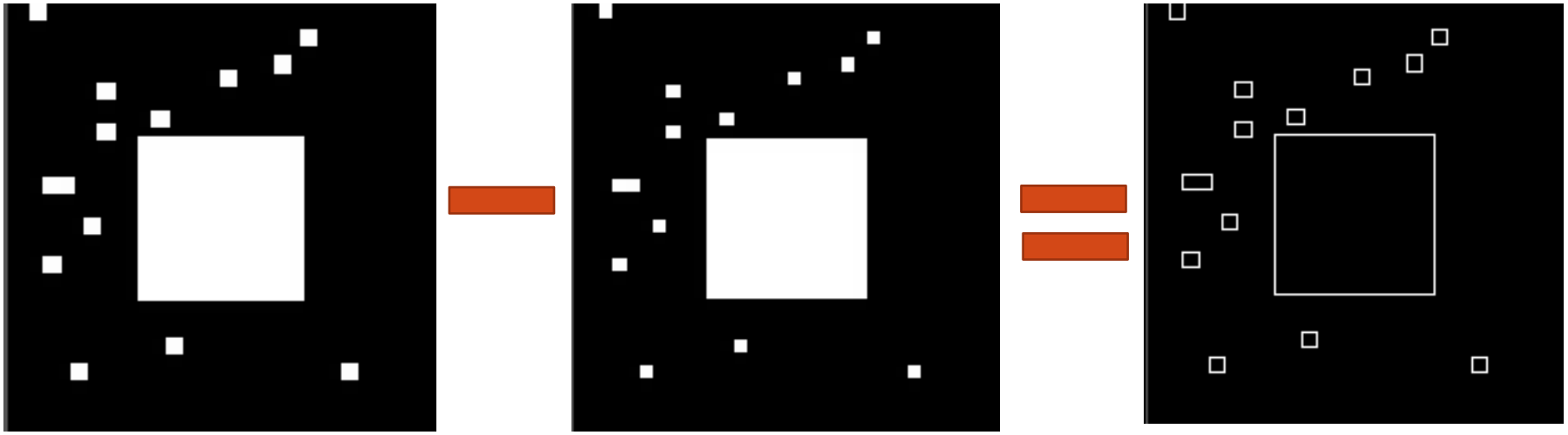
Erosion

Dilation



Dilation for Edge Detection

```
im1 = Image.open('square.bmp')
im2 = dilation(im, 1, 1)
edges = diff(im2, im)
```



- Next: - diff
  - erosion/dilation using morphological operators

Diff - Code

```
# create diff between two images
def diff(im1, im2):
    assert im1.size == im2.size

    out = Image.new('L', im1.size, 'white')
    out_pix = out.load()
    mat1 = im1.load()
    mat2 = im2.load()
    width, height = im1.size

    for y in range(height):
      for x in range(width):
            out_pix[x, y] = mat1[x, y] - mat2[x, y])

    return out
```

Neighborhood

- Neighborhood/Environment of the pixel [x,y]:

  the set of all pixels whose coordinates are **close** to [x,y].

- A neighborhood commonly considered is the (2k + 1)-by-(2k + 1) square matrix of coordinates centered at [x,y]

- k is a small integer - typically 1 or 2.

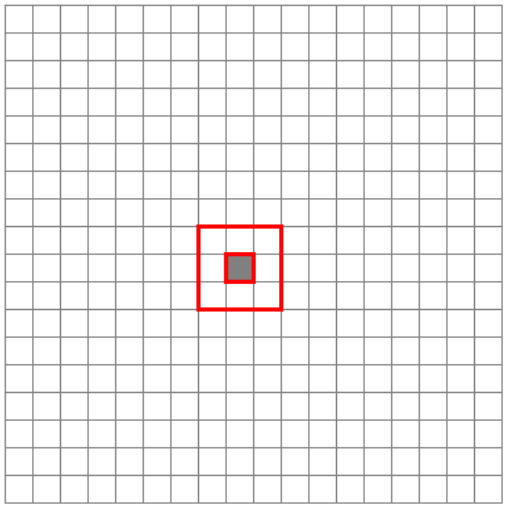
$$N_{3\times 3}(x,y) = \begin{bmatrix} x-1, y+1 & x, y+1 & x+1, y+1 \\ x-1, y & x, y & x+1, y \\ x-1, y-1 & x, y-1 & x+1, y-1 \end{bmatrix}$$

- We may, however, prefer a non-square neighborhood in some cases.
  - rectangular
  - circular / elliptical
  - non-symmetric…
- We may also want to assign different weights to different neighbors

Neighbors - Code

Create a pixel's environment:

```
def neighbours(mat, width, height, x, y, nx=1, ny=1):
    nlst = [ ]
    for yy in range(max(y-ny, 0), min(y+ny+1, height)):
        for xx in range(max(x-nx, 0), min(x+nx+1, width)):
            nlst.append(mat[xx, yy])
    return nlst
```



- What's the neighborhood in the image boundaries?

Morphological Operators - Code

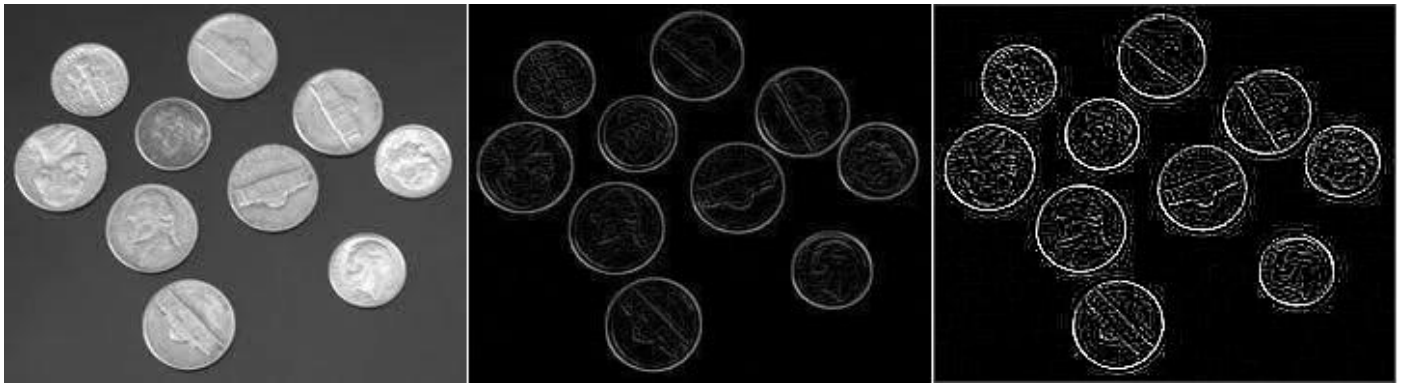
Framework:

```
def morphological(im, operator, nx = 1, ny = 1):
    w, h = im.size
    out_im = Image.new('L',(width, height), 'white')
    in_pix = im.load()
    out_pix = out_im.load()

    for y in range(h):
        for x in range(w):
            nlst = neighbours(in_pix, w, h, x, y, nx, ny)
            out_pix[x, y] = operator(nlst)

    return out_im
```

Erosion and Dilation - Code

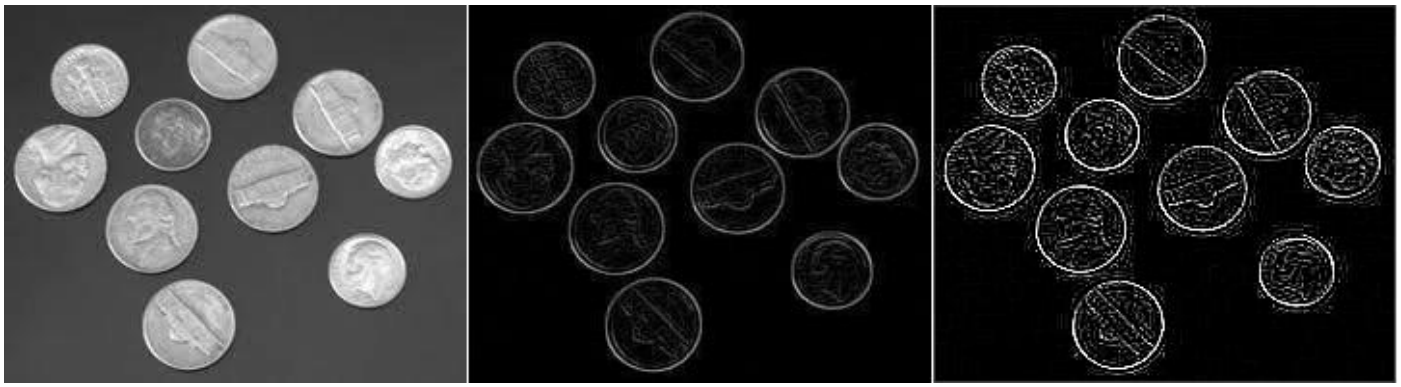
```
def erosion(im, nx = 1, ny = 1):
    return morphological(im, min, nx, ny)

def dilation(im, nx = 1, ny = 1):
    return morphological(im, max, nx, ny)
```

Runs

	original	Diff	PIL's filter
coins.jpg			

```
from PIL import ImageFilter

im = Image.open("./coins.jpg")
new = im.filter(ImageFilter.FIND_EDGES)
new.show()
```

Join images for easy display



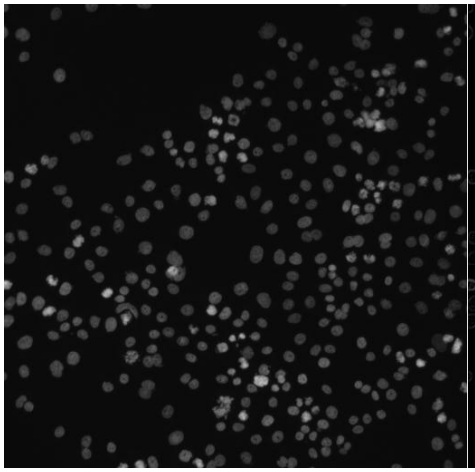
```
def join(*images):
    w,h = images[0].size
    n = len(images) #number of images
    new = Image.new('L',(w*n+n,h), 'white')

    for i in range(len(images)):
        new.paste(images[i], (w*i+i,0))
    return new
```

```
from PIL import ImageFilter

im = Image.open("./coins.jpg")
im = im.convert('L')
out_im = diff(dilation(im,1,1),im)
out_im2 = im.filter(ImageFilter.FIND_EDGES)
join(im,out_im, out_im2).show()
```

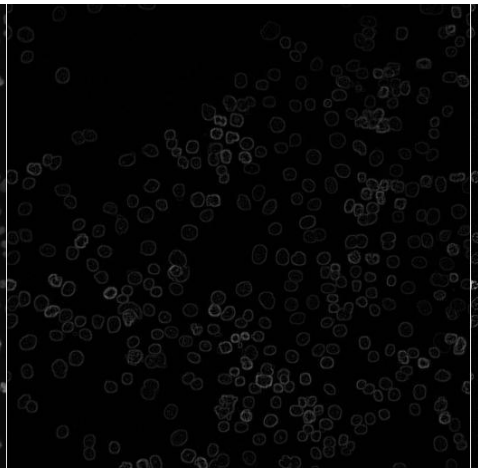
Runs (2)

T29 human cancer

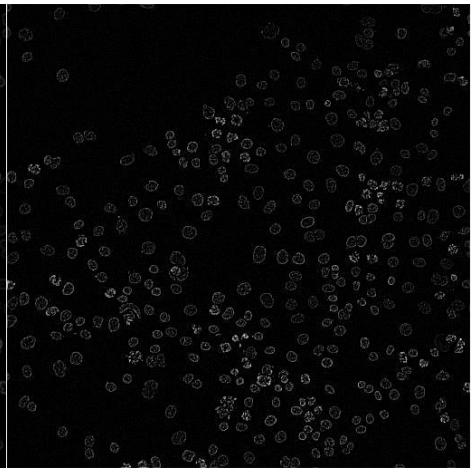
original



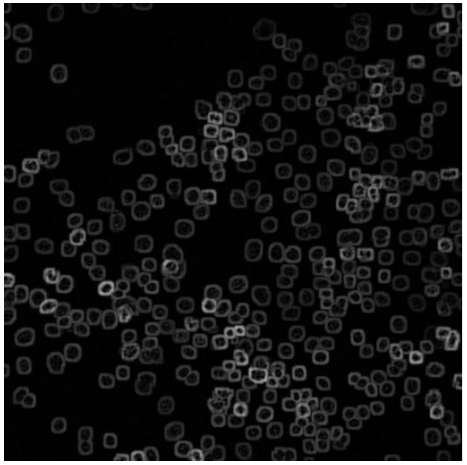
Diff (3X3)



PIL's filter



Diff (7X7)



Noise and Denoising

Noise and Denoising

The observed value at pixel x, y, $S(x, y)$, equals the sum of the true value $I(x, y)$ plus noise $N(x, y)$.

$$S(x, y) = I(x, y) + N(x, y) \ .$$

The goal of denoising algorithms is, given the observed image $S(x, y)$ $(0 \leq x < k,\ 0 \leq y < \ell)$ to produce a new image, $\hat{I}(x, y)$, which should be close to the original image $I(x, y)$ $(0 \leq x < k,\ 0 \leq y < \ell)$.

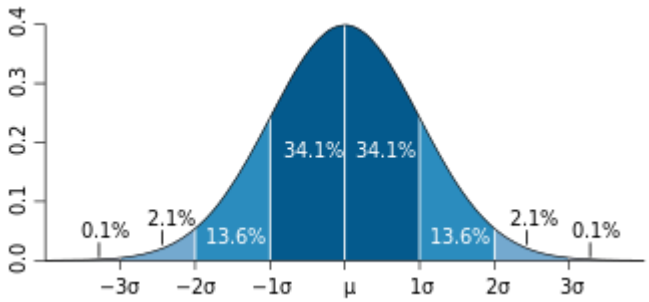
Obviously such goal is not well defined, and thus cannot be solved, if there are no constraints on the image and on the noise.

Image model: We assume the image is piecewise smooth: Most of the image's area consists of large, smooth regions where light intensity varies continuously – if x_1, y_1 and x_2, y_2 are neighbors, then $I(x_1, y_1)$ and $I(x_2, y_2)$ attain close enough values.

Gaussian Noise Model

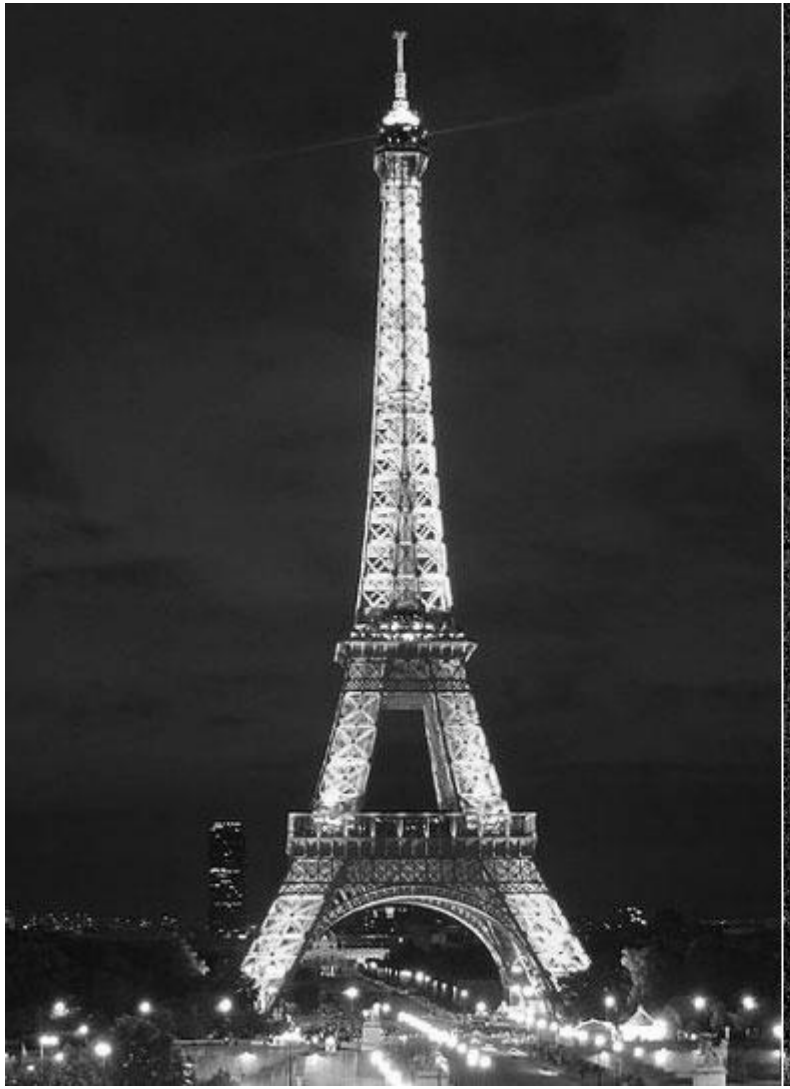
Gaussian noise model: The noise at pixel x, y, $N(x, y)$, is a random variable. It is usually assumed that $N(x, y)$ is "white noise", distributed independently of the noise at other pixels.

```
def add_gauss(im, sigma=20):
    ''' Generates Gaussian noise with mean 0 and SD sigma.
        Adds indep. noise to pixel, keeping values in 0..255'''
    out_im = im.copy()
    in_pix = im.load()
    out_pix = out_im.load()

    w,h = im.size
    for y in range(h):
        for x in range(w):
            noise = round(random.gauss(0,sigma))
            out_pix[x, y] = min(max(in_pix[x, y] + noise, 0), 255)

    return out_im
```

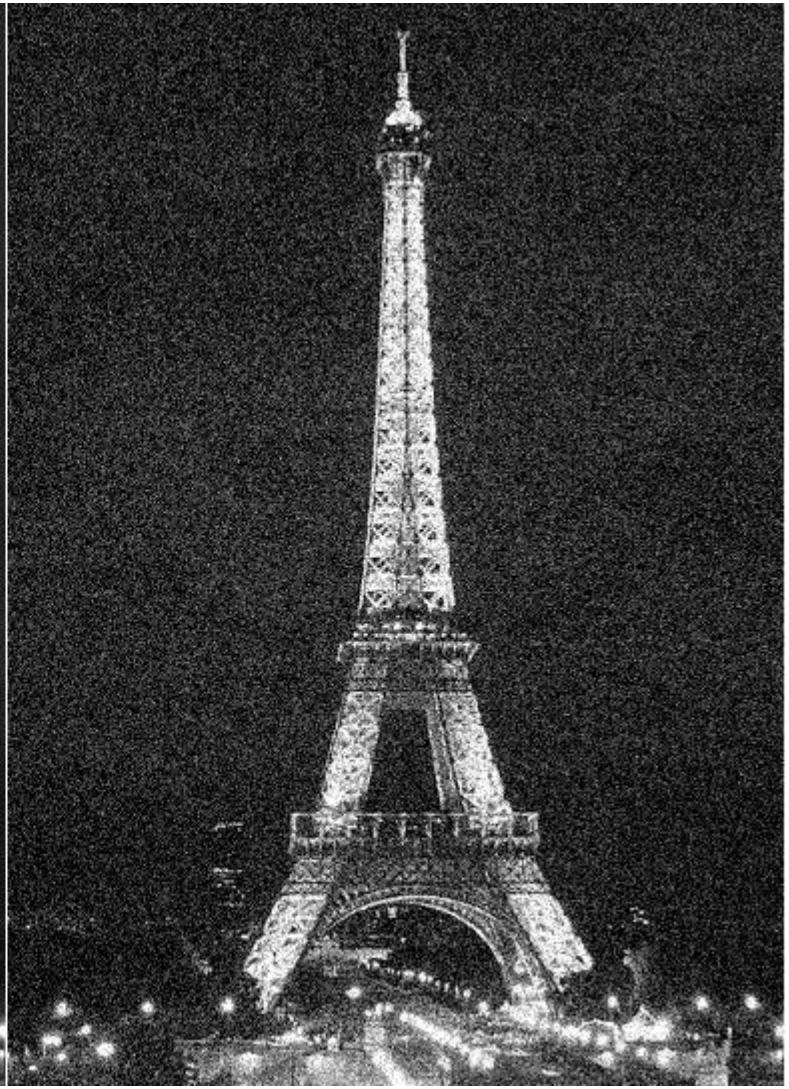


The function random.gauss(mu, sigma) returns a floating point number of Gaussian distribution with expected value μ and standard deviation σ

Gaussian Noise example



original



add_gauss(im, 30)

Salt and Pepper Noise Model

A different type of noise is the so called salt and pepper noise – extreme grey levels (white and black), or bursts, appearing at random and independently in a small number of pixels.

```
def add_SP(im, p=0.01):
    out_im = im.copy()
    in_pix = im.load()
    out_pix = out_im.load()

    w,h = im.size
    for y in range(h):
        for x in range(w):
            rand = random.random()
            if rand < p:
                if rand < p/2:
                    out_pix[x, y] = 0
                else:
                    out_pix[x, y] = 255
    return out_im
```

50:50 chance for black or white

Salt and Pepper Noise example

original | add_gauss(im, 30) | add_SP(im, 0.05)

Denoising algorithms

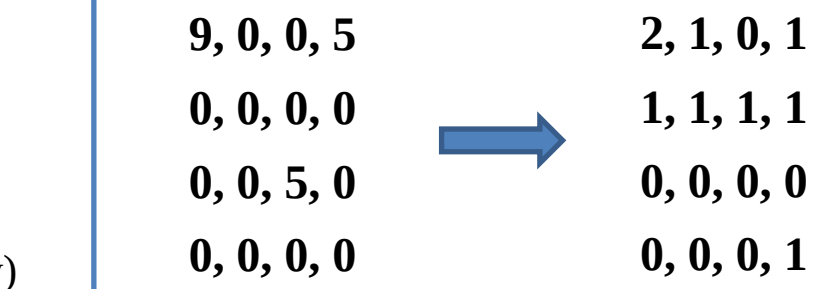
- We will discuss two approaches to denoising:
  1. Local **Means**
  2. Local **Medians**

- Of course, these are only the tip of the iceberg.

Local Means

- Replace pixel at [x,y] by the average (mean) of its neighborhood.

```
def mean(lst):
    return sum(lst)//len(lst)

def denoise_mean(im, nx = 1, ny = 1):
    return morphological(im, mean, nx, ny)
```



- if image is piecewise smooth → small change → preserves original signal
- average SD reduces to $\sigma/(2k+1)$ → noise is reduced

→ good for Gaussian noise

Weighted Local Means

Uniform averaging based on the whole neighborhood, as discussed before, can be expressed as the matrix dot product

$$\begin{pmatrix} S[x-1,y+1] & S[x,y+1] & S[x+1,y+1] \\ S[x-1,y] & S[x,y] & S[x+1,y] \\ S[x-1,y-1] & S[x,y-1] & S[x+1,y-1] \end{pmatrix} \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

A common variant puts more weight close to the central pixel. For example, in our case of the 3-by-3 neighborhood, replacing the $1/9$ matrix by

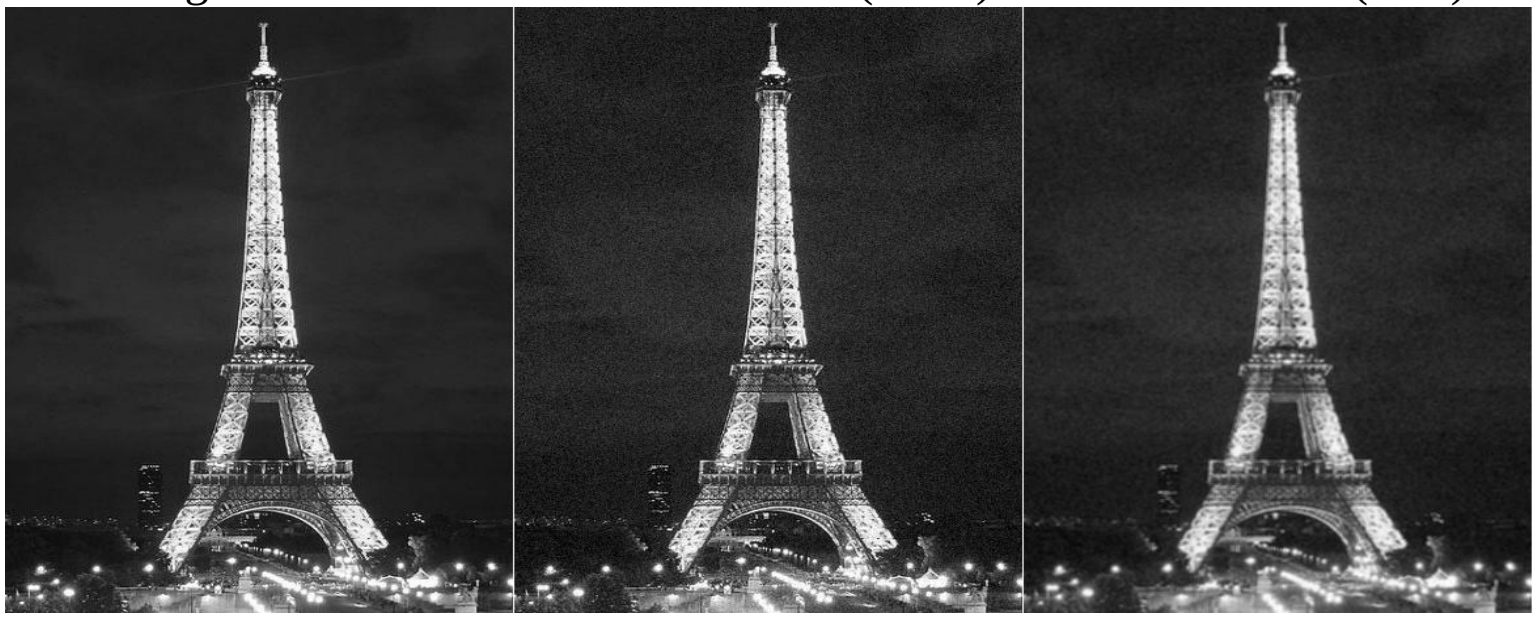
$$\begin{pmatrix} 1/12 & 1/12 & 1/12 \\ 1/12 & 1/3 & 1/12 \\ 1/12 & 1/12 & 1/12 \end{pmatrix}.$$

We point out that while this maintains more of the original signal, the noise reduction here is smaller.

Local Means - limitations

- Does not perform well on areas not piecewise smooth (edges !)

original | Gaussian noise (σ=10) | local means(3X3)



- sensitive to extreme outliers:

noise

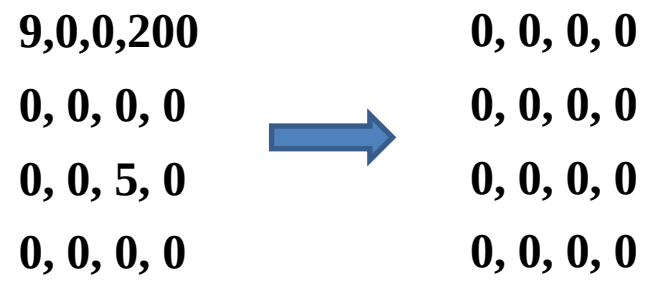
9,0,0,200	**2, 1, 33, 50**
0, 0, 0, 0	**1, 1, 22, 34**
0, 0, 5, 0	**0, 0, 0, 0**
0, 0, 0, 0	**0, 0, 0, 1**

➔ bad for Salt and pepper noise

Local Medians

- Replace pixel at [x,y] by the median of its neighborhood.

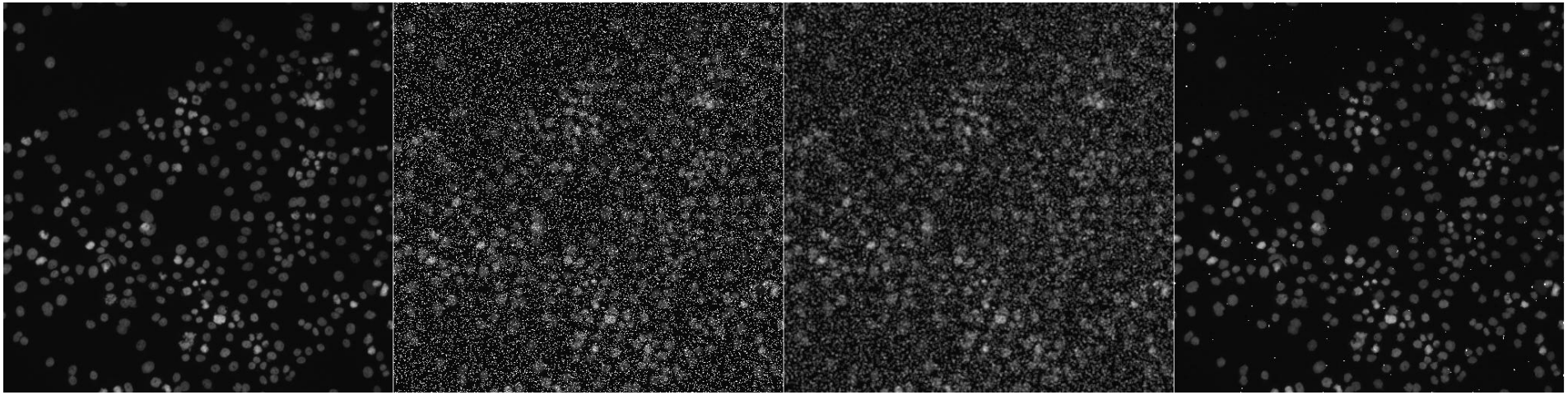
```
def median(lst):
    lst.sort()
    return lst[len(lst)//2]

def denoise_median(im, nx = 1, ny = 1):
    return morphological(im, median, nx, ny)
```



- advantages:
  - preserves edges
  - not sensitive to extreme outliers ➔ good for S&P noise
- disadvantages:
  - eliminate small, fine features
  - slower than local means (median calculation takes more time than average)

Local Medians examples

original | 20% S&P | local means | local medians



A glance to Non-Local approaches

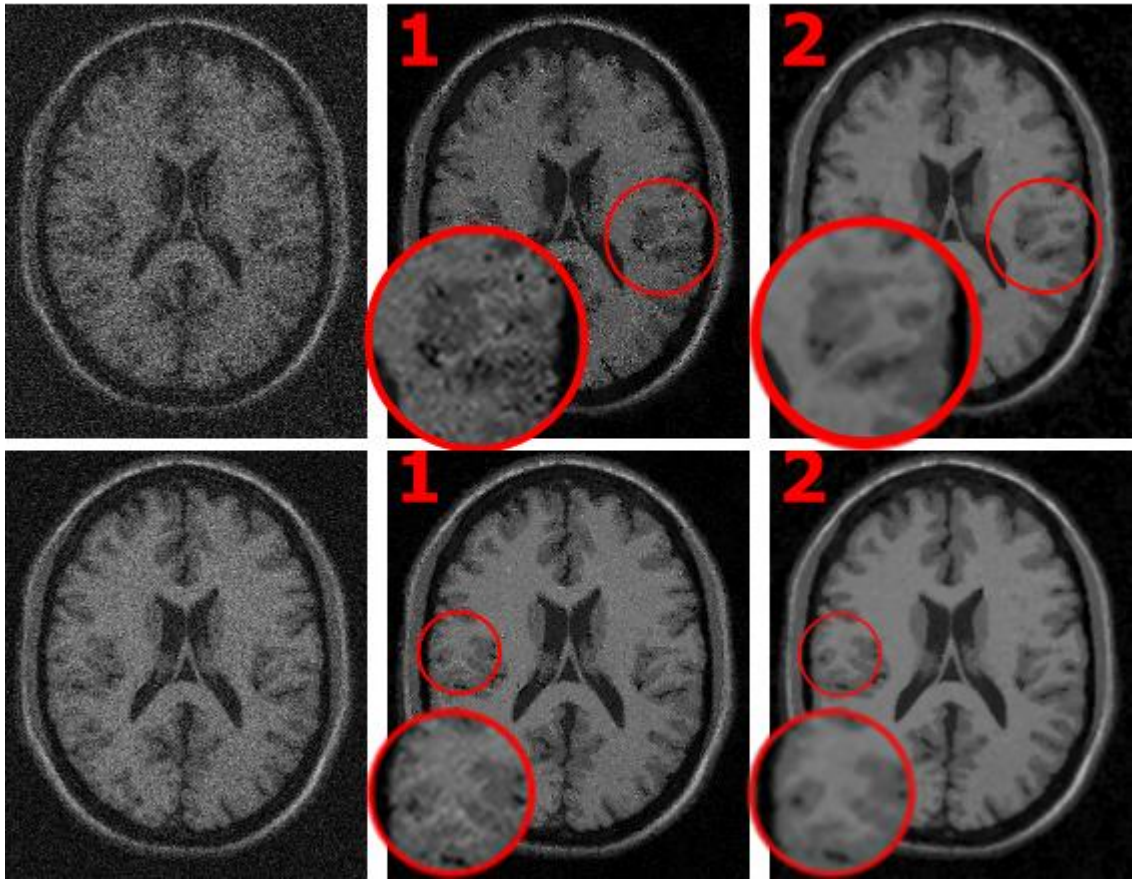
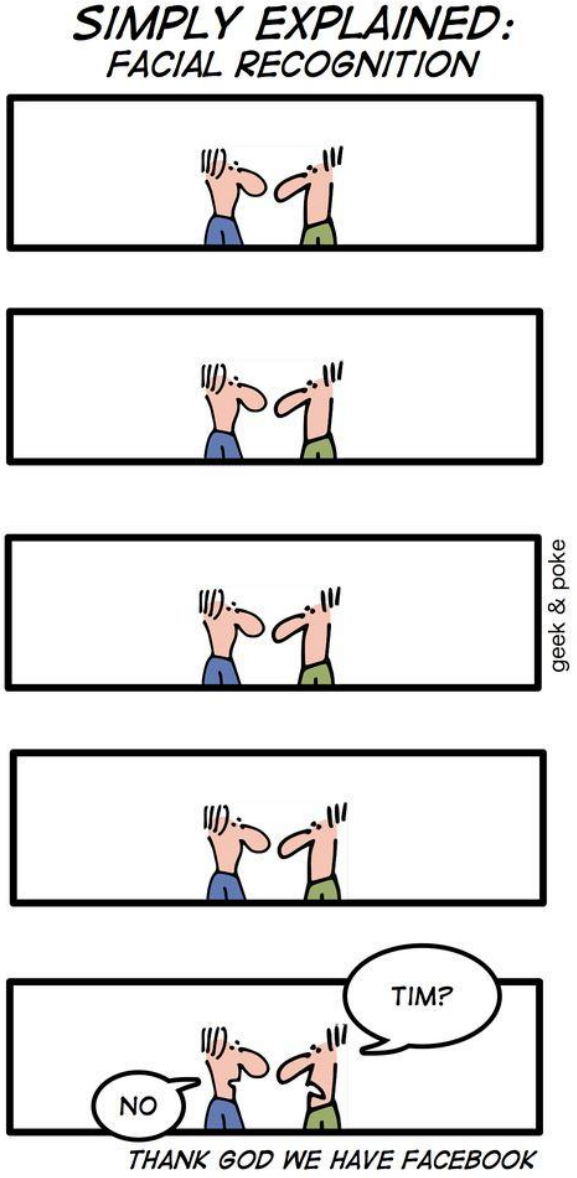


image from:

http://www.nitrc.org/project/list_screenshots.php?group_id=518&screenshot_id=400


SIMPLY EXPLAINED:
FACIAL RECOGNITION
geek & poke
TIM?
NO
THANK GOD WE HAVE FACEBOOK

Exercises

1. Here's an image with a spot of size 4X4 somewhere.
   Which method would you use to denoise it, with what neighborhood size (the parameter k)?

   Check your answer by experiments:

   - First download the image (eifel.jpg).
   - Create a 4X4 white spot on a dark part of it, as shown:
   - try denoising with local means/medians,
     and with several neighborhood sizes

2. Iterated denosing:
   Instead of using larger k's, try using k=1 several times to clean the spot from Q1.

   Draw your conclusions.