Extended Introduction to Computer Science
CS1001.py

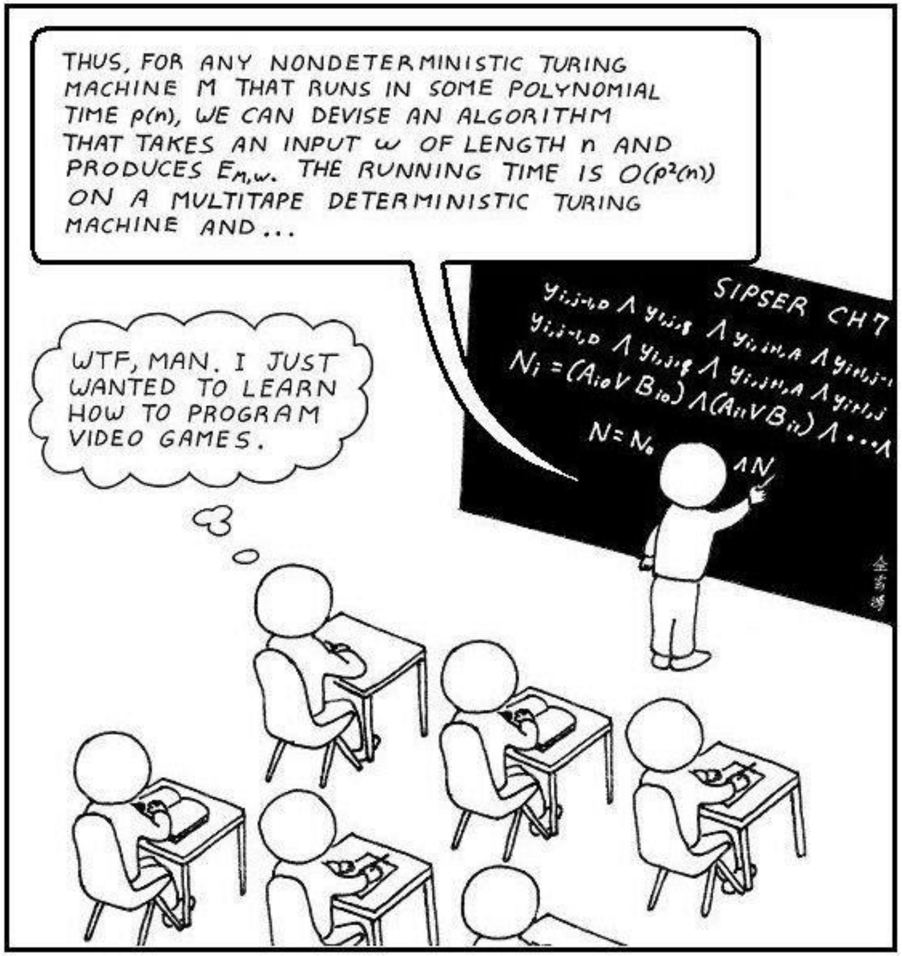
Module J Introduction to Digital Image Representation and Processing

Instructors: Elhanan Borenstein, Michal Kleinbort
Teaching Assistants: Noam Parzanchevsky,
Asaf Cassel, Shaked Dovrat, Omri Porat

School of Computer Science
Tel-Aviv University
Srping Semester 2020-21
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

And Now For Something Completely Different*,**



Source: https://www.pinterest.ca/pin/202380576975644458/

* שקף עם כותרת זו יישמש להדגשת המעבר בין חלקים שונים בקורס. מי שהלכו קצת לאיבוד, זו הזדמנות לקפוץ חזרה על הרכבת.

** אנו מזמינים אתכם לשלוח לנו הצעות לתמונות שיופיעו על שקפים אלו

Lecture 24-25: Plan

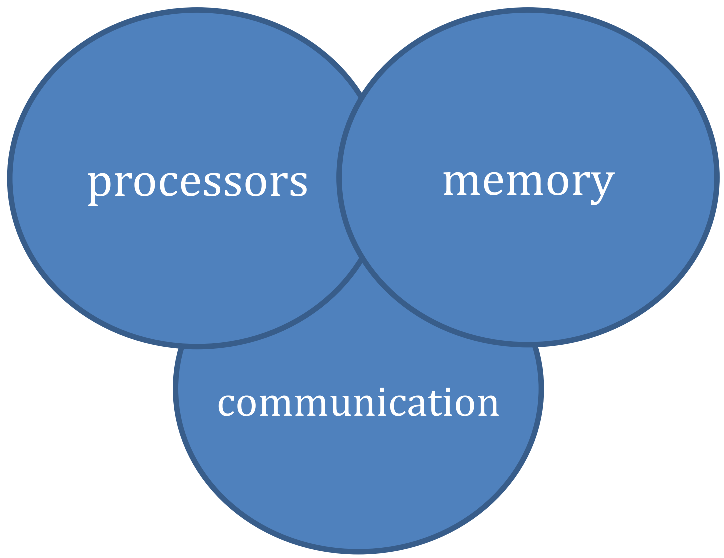
- Introduction to Digital Image Representation:
    - Greyscale and color images
    - Bit depth, resolution
    - Generating synthetic images
    - Manipulating images

- Basics of Digital Image Processing
    - Noise reduction:
        - Noise models: Gaussian, Salt & Pepper
        - De-noising with local means, local medians
    - Additional examples, time permitting

Brief "Historical" Technological Context

- early 1980's
- today

transistors	speed	RAM	Hard Disk
29 K	4.77 MHz	640 KB	5 MB
15 G	3.7 GHz	8 GB	500 GB



e-mail, simple text (128 ascii chars)
tons of data, inc. images (next slide)

A Brief Historical Context, Few Decades Later

- With the proliferation of
  (1) larger and faster memory,
  (2) strong, inexpensive processors,
  (3) faster internet,

  it became possible to efficiently
  (1) store,
  (2) process, and
  (3) transmit large digital images.

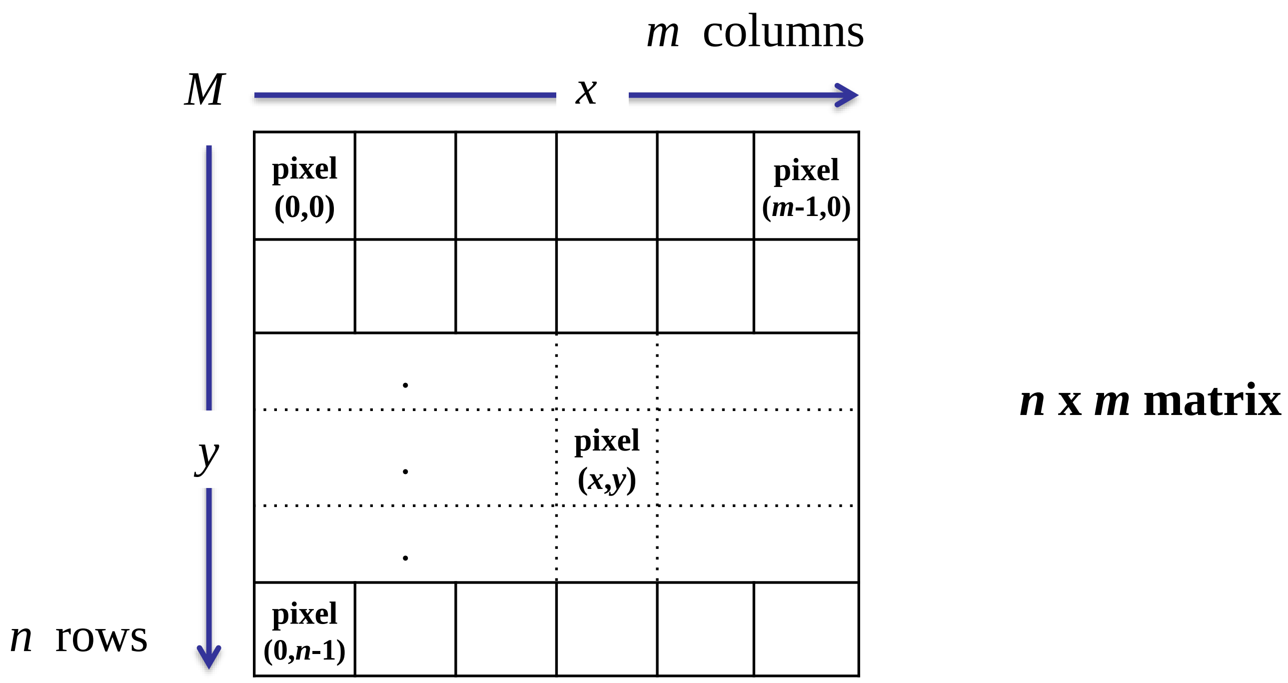
- Facebook stores about 350 million photos DAILY (4000/sec, reported 2019).
  >250 billion photos where uploaded in total.
- The total number of photos+videos shared on Instagram is 40 billion (2010-2019).

- This dramatic technological progress is reflected by the following saying, often attributed (apparently incorrectly) to Bill Gates, in 1981: "640KB ought to be enough for anybody".

Digital Image Representation

- A digital image is commonly represented as a numeric 2D matrix.

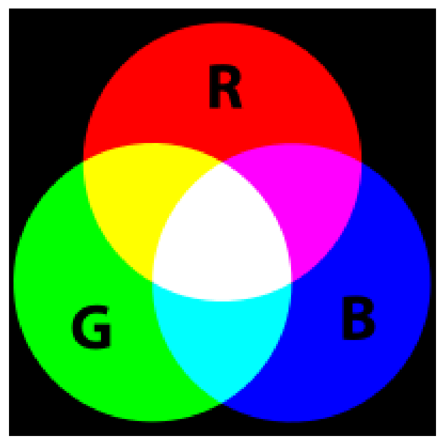
- Each element $M[x, y]$ is called a pixel (picture element).
  Pixel values convey information about the light intensity / color at that location of the image.

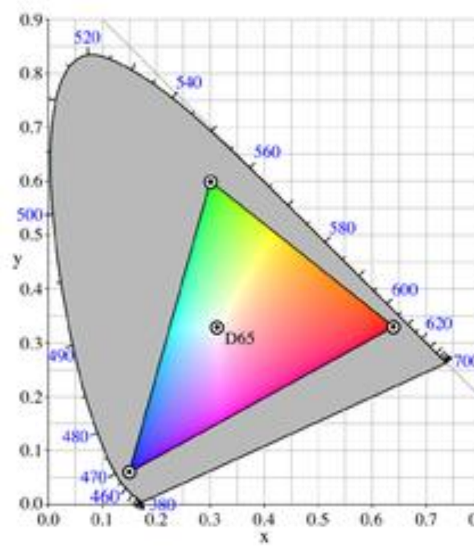


RGB vs. Greyscale Images

- For standard RGB* color images, $M[x, y]$ is a triplet of values, representing the red, green, and blue components of the light intensity at that pixel.

- For grey-level images, $M[x, y]$ is a non-negative number, representing the light intensity at that pixel.





(images from Wikipedia)

* RGB is one common representation of colors. CMY is another one

Some Fun with Color Representation (for self exploration)

From Computer Science Field Guide:

- https://csfieldguide.org.nz/en/chapters/data-representation/images-and-colours/

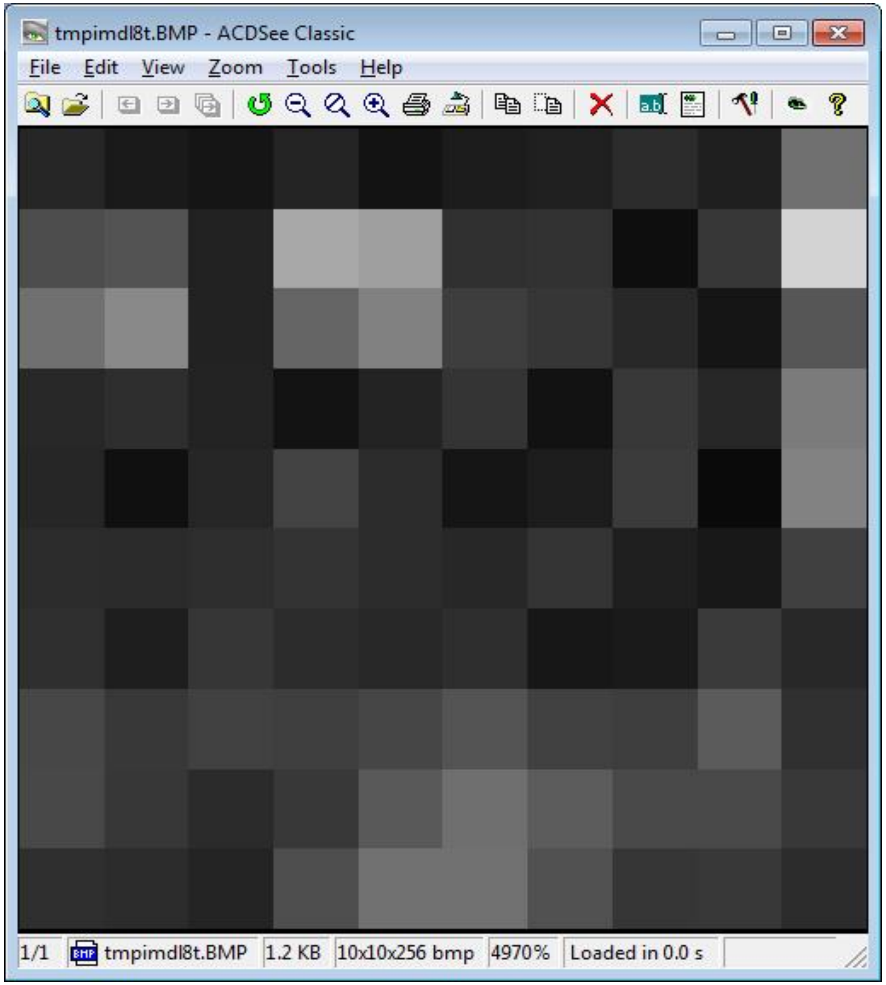
Grey Level format

- For the sake of simplicity, the remainder of this class will deal with greyscale images only.
  - However, what we will do is applicable to color images as well with minor modifications.

- Real numbers expressing visual signal have to be discretized in order to enable their representation on bounded precision digital devices. A good quality greyscale photograph (that is, good by human visual inspection) has 256 grey-level values per pixel.
  - This requires 8 bits per pixel
  - The value 0 represents black, while 255 represents white.
  - For each pixel, the closer its value is to 0, the blacker it is. So 128 is considered a mid-way grey.

Grey Level Images - Example

- 256 grey level image: 0 = black, 255 = white



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

Image Bit Depth

- Bit depth = number of bits per pixel:

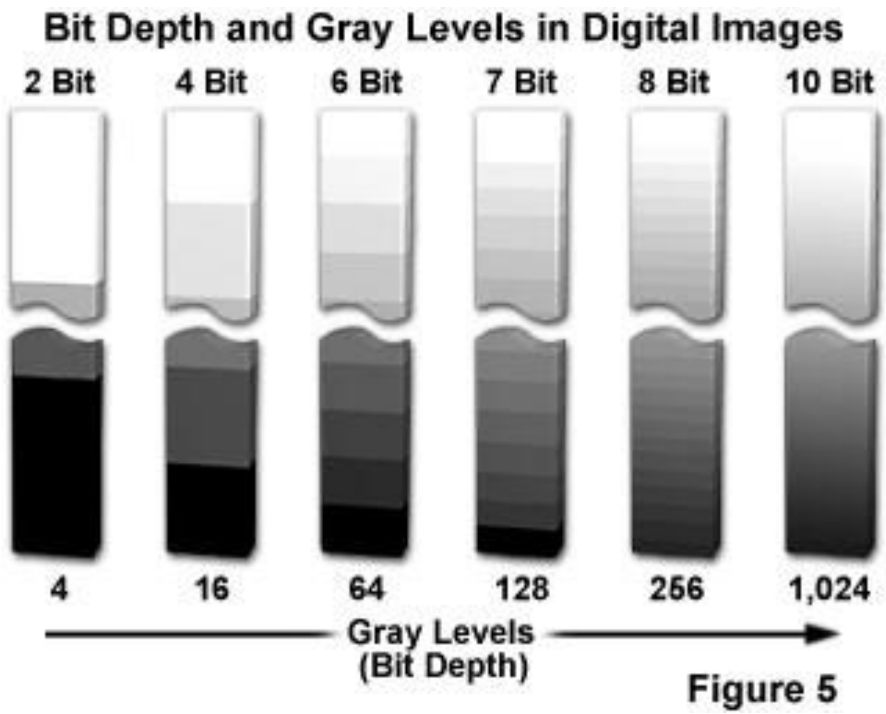


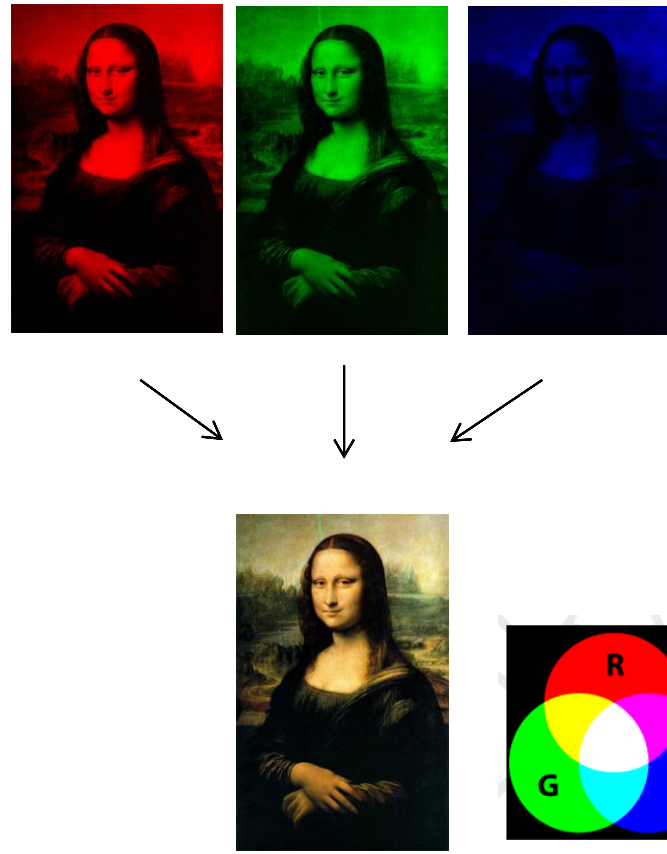
Image from:
http://micro.magnet.fsu.edu/

- A human observer is able to discriminate between at most a few hundreds shades of gray in optimal conditions (some estimations are lower, depending also on the background, distance from the image etc.).
- We remark that in some applications, such as medical imaging, 4096 grey levels (12 bits) are used. Higher bit depth images are sometimes aimed for an automated analysis by a computer.

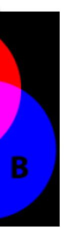
BW / Grayscale / RGB - Summary

B&W
(1 bpp)

256 gray level image
(8 bpp)



"true color" image
(8+8+8 = 24 bpp)



Images from: http://www.csse.uwa.edu.au/~wongt/matlab.html

12

Python Imaging Library – PIL/PILLOW

- We will demonstrate topics with "real" images using the external package PILLOW

- Installation instruction: open command prompt and type:

```
python -m pip install --upgrade pip
```

and then

```
python -m pip install --upgrade Pillow
```

(if this doesn't work replace `python` with `python3`)

- Upon successful installation, the following line should not raise an error:

```
>>> from PIL import Image
```

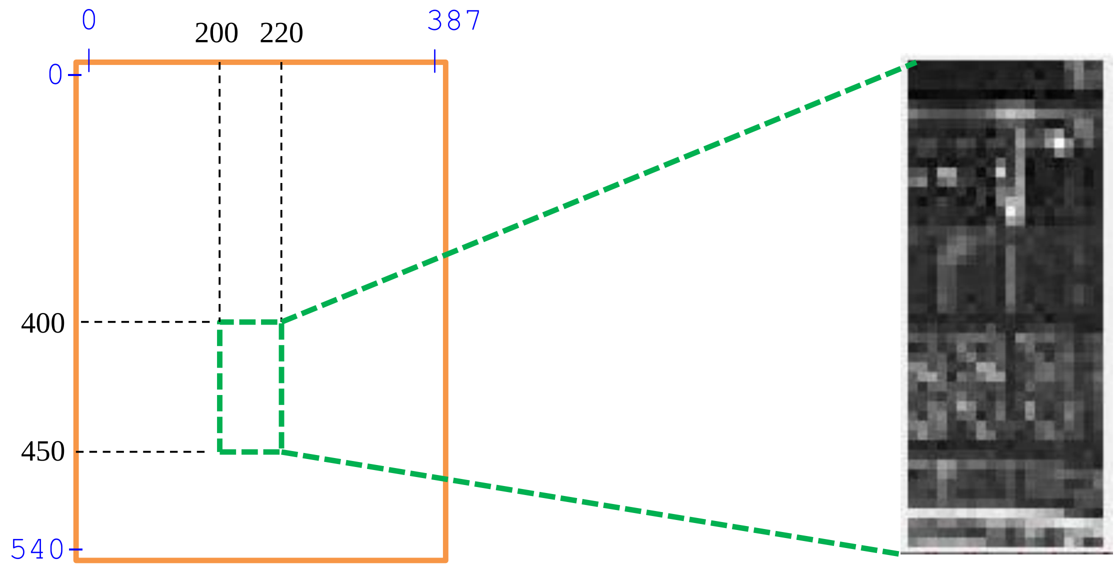
Basic Handling of Images using PIL

```
>>> from PIL import Image

# Open image
>>> img = Image.open("./guess.bmp")

>>> img.size
(388, 541) #width, height


>>> img.show() # display

# convert to 256 gray levels
# so the code we write later will work
>>> img = img.convert('L')

# get grey levels distribution (0-255)
>>> img.histogram()

# Save as a new file
>>> img.save("./new_image", "bmp")
```

"./" = current folder
"../" = parent folder

Guessing Game

```
>>> img.size
(388, 541) #width, height

# crop(min_x, min_y, max_x, max_y)
>>> region = img.crop((200,400,220,450))
>>> region.show()
```



The Matrix Representing an Image

- In order to change the image pixels we load the matrix representing it
- Note: changes in the matrix WILL affect the image

```
>>> im = Image.open("./some_image.jpg").convert('L')
>>> mat = im.load()

>>> mat[0,0] #upper left corner
31

>>> mat[0,0] = 255
>>> mat[0,0]
255

>>> for x in range(20):
        for y in range(20):
            mat[x,y] = 255

>>> im.show()
```

Generating Synthetic Images

```
def create_img(w, h, op):
    ''' create a w X h image
        assign pixel x,y with op(x,y) '''
    img = Image.new(mode='L', size=(w,h), color=255)
    mat = img.load()

    for x in range(w):
        for y in range(h):
            mat[x,y] = op(x,y)

    return img
```

256 grayscale format

Initial color

Some Examples (Executions in Class)

```
# Define constants to ease code readability
WHITE = 255
BLACK = 0

rnd_img = create_img(256, 256, lambda x,y: random.randint(0,255))

ver_lines = create_img(100, 300, lambda x,y: BLACK if x%10==0 else WHITE)

n=512
diagonal = create_img(n, n, lambda x,y: BLACK if x-y==0 else WHITE)

framed_diagonal = create_img(n, n, lambda x,y:
                                   BLACK if x==0 or y==0 or \
                                            x==n-1 or y==n-1 or \
                                            x==y or x+y==n-1 \
                                         else WHITE)

what = create_img(n, n, lambda x,y,c=1: (c*(x-y))%256)

circles = create_img(n, n, lambda x,y,c=1: (c*(x**2 + y**2)) % 256)

product = create_img(n, n, lambda x,y,c=1: (c*x*y) % 256)
```

Padlet for your own creative images

- We urge you to play with the code, be as creative as you can, or simply use trial and “error”.

https://padlet.com/amirr6/euhqfxrey5f4emmu

Manipulating Images

20

Manipulating Images

```
def process_img(img, op):
    ''' process image img (PIL.Image object) '''

    w,h = img.size
    mat = img.load()
    new_img = img.copy()
    new_mat = new_img.load()

    for x in range(w):
        for y in range(h):
            new_mat[x,y] = op(mat, x, y)

    return new_img
```

Some Examples (Executions in Class)

```
# Define constants to ease code readability
WHITE = 255
BLACK = 0

img = Image.open("./some_image.jpg").convert('L')



white_square = \
 process_img(img, lambda mat, x, y: WHITE if x<100 and y<100 else mat[x,y])


color_shifted = process_img(img, lambda mat, x, y, k=30: (mat[x,y]+k)%256 )


negative = process_img(img, lambda mat, x, y: 256-mat[x,y])


w,h = img.size
upside_down = process_img(img, lambda mat, x, y: mat[x,h-y-1])
```

Tiling Multiple Images

- A useful utility function that tiles several images together, horizontally, assuming all images are of the same size:

```
def tile(*images):
    ''' Join several images horizontally for easy display.
        Assume all images are of the same size
        The * before the parameter means a variable number of parameters '''

    w,h = images[0].size
    n = len(images) #number of images

    new = Image.new('L',(w*n+n,h), 255) #+n for some space between images

    for i in range(len(images)):
        new.paste(images[i], (w*i+i,0)) #+i for some space between images

    return new
```

Do you understand this?

Noise Reduction

Signals

- A signal is any physical quantity, measurable through time or over space.
  - Examples: radio, telephone, radar, sound, light,…
- Signal processing: applying mathematical techniques for the extraction, transformation and interpretation of signals.
  - Signal processing may take two major flavors:
    1) digital (discrete)
    2) analog (continuous)
  - Naturally, in this course we explore the first option

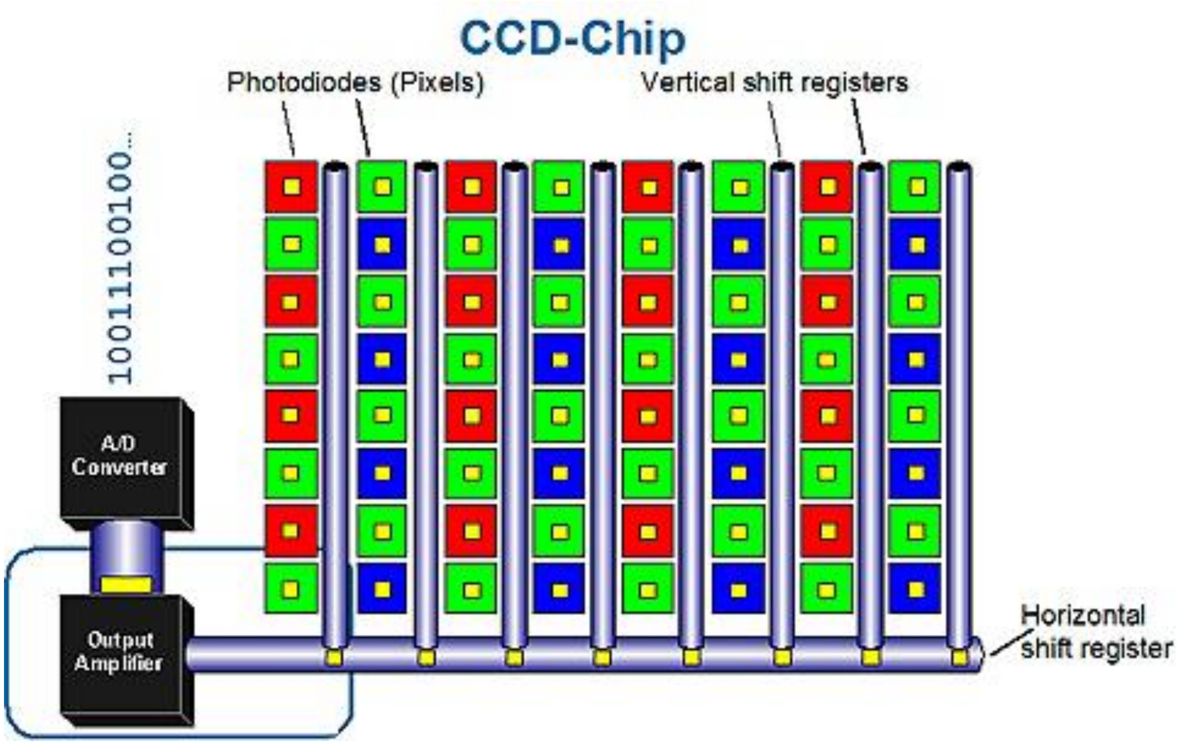
Digital Image Processing

- Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame.
- The output of image processing may be either an image or a set of characteristics or parameters related to the image.
- Most image processing techniques involve treating the image as a two-dimensional signal and applying standard signal processing techniques to it .

(text and figure taken from Wikipedia).

CCD (for reference only)

- CCD (charge coupled device): transforming light (photons) to electrical voltage
- Each captor of the CCD is roughly a square area, in which the number of incoming photons is being counted for a fixed time period.



(image and text taken from http://www.axis.com/edu/axis/)

Noise

- Digital cameras (as well as traditional film cameras, microscopes, etc.) are susceptible to noise formation.

- Noise sources include flecks of dust inside the camera, faulty sensors or recording elements, the deviation of electrons from their original path (a phenomenon called *electron hiss*), etc.

- A basic noise model:

  At any pixel (x,y) the observed value $S(x,y)$ equals the sum of the "true" value $I(x,y)$ plus some noise $N(x,y)$.

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

- The goal of noise reduction, or denoising algorithms, is to produce a new image, which should be as close as possible to the "true" image I.
  - Note that the values $I(x,y)$ are not known to us! All we have is $S(x, y)$.

Noise and Denoising Models

- Two very basic noise types we will see:
  1. Gaussian noise
  2. Salt and Pepper noise

- Two basic denoising approaches, based on local operators:
  - Local means (operator = average)
  - Local medians (operator = median)

Other, non-local methods, consider farther parts of the image.

Assumptions on the Images

- We assume the image is piecewise smooth:

  Most of the image's area consists of regions where light intensity varies smoothly: if $M[x_1, y_1]$ and $M[x_2, y_2]$ are neighbors, then they attain close enough values.

Gaussian Noise Model

- The noise ingredient $N(x, y)$ at each pixel is a random variable.
- It is usually assumed that $N(x, y)$ is distributed normally and independently of the noise at other pixels.
- So each pixel in the image is changed from its original value by some (usually small) amount. Small deviations from the original value are more likely than large ones.

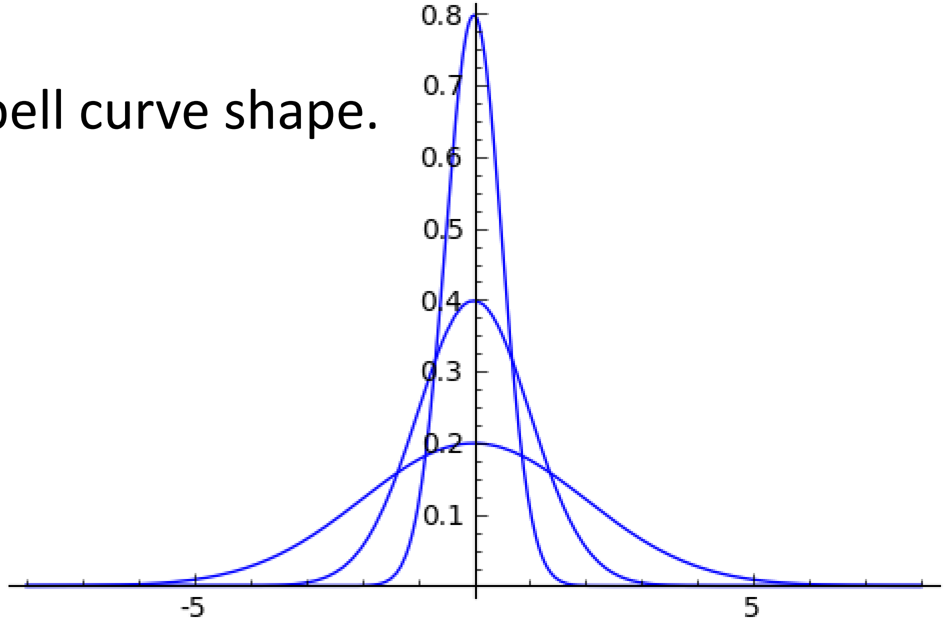
Gaussians (for reference only)

- The probability density function

$$G_\sigma(x) = \frac{e^{-x^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

is called the Gaussian, or normal, distribution. It has mean 0 and standard deviation (SD) σ. This is a continuous function, which is the limit of the Binomial distribution, as the number of events tends to infinity.

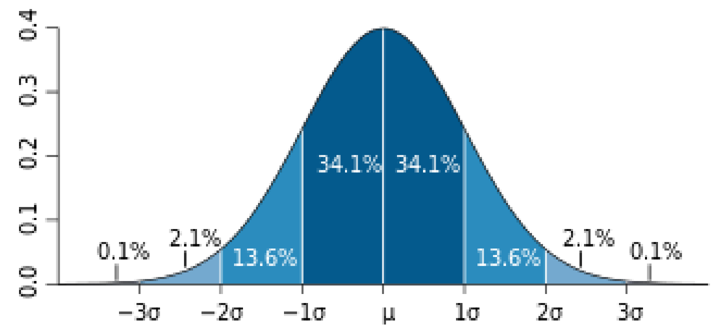
The Gaussian has the well known bell curve shape.

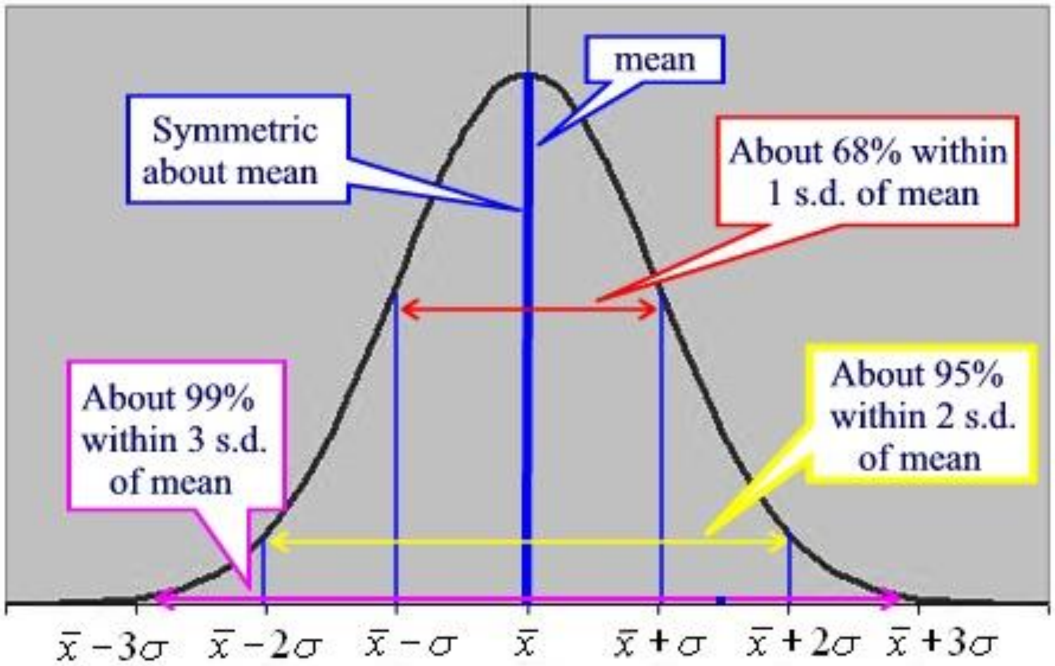


Three Gaussians, with σ = 0.5, 1, 2 (σ = 0.5 is the narrowest).

More on Gaussians

68% of the distribution lies within one standard deviation of the mean. **95%** of the distribution lies within two standard deviations of the mean. **99.7**% of the distribution lies within three standard deviations of the mean. These percentages are known as the "empirical rule".





http://www.regentsprep.org/regents/math/algtrig/ats2/normallesson.htm

Gaussian Noise: Python Code

- The function random.gauss(mu, sigma) returns a floating point number, distributed according to a Gaussian distribution with expected value (mean) μ and standard deviation σ.
- We will use μ = 0, and a default value σ = 10. When added to pixel values, we will round the noise and make sure the outcome falls within 0…255.

14 out of 20 (70%) between -10 and 10.
19 out of 20 (95%) between -20 and 20

```
>>> import random
>>> random.gauss(0,10)
0.36121514047571907
>>> random.gauss(0,10)
21.643048694527852
>>> lst = [round(random.gauss(0,10)) for i in range(20)]
>>> lst
[-8, 22, 12, 4, -1, 2, 11, 6, -16, -1, 4, -9, -3, 1, -5, -3, 5, 18, 19, 1]
>>> sorted(lst)
[-16, -9, -8, -5, -3, -3, -1, -1, 1, 1, 2, 4, 4, 5, 6, 11, 12, 18, 19, 22]
```

Adding Gaussian Noise: Python Code

```
def add_gaussian_noise(img, sigma=10):
    ''' Generates Gaussian noise with mean 0 and SD sigma.
        Adds indep. noise to pixel, keeping values in 0..255
    '''n

    def g_noise_op(mat, x, y):
        g_noise = round(random.gauss(0,sigma))
        return min(max(mat[x,y] + g_noise, 0), 255)

    return process_img(img, g_noise_op)
```

Adding Gaussian Noise: Example

```
>>> img = Image.open("…").convert("L")
>>> img_gaussian_noise = add_gaussian_noise(img)
>>> tile(img, img_gaussian_noise).show()
```

Original image

Gaussian noise (σ=10)

Salt and Pepper Noise Model

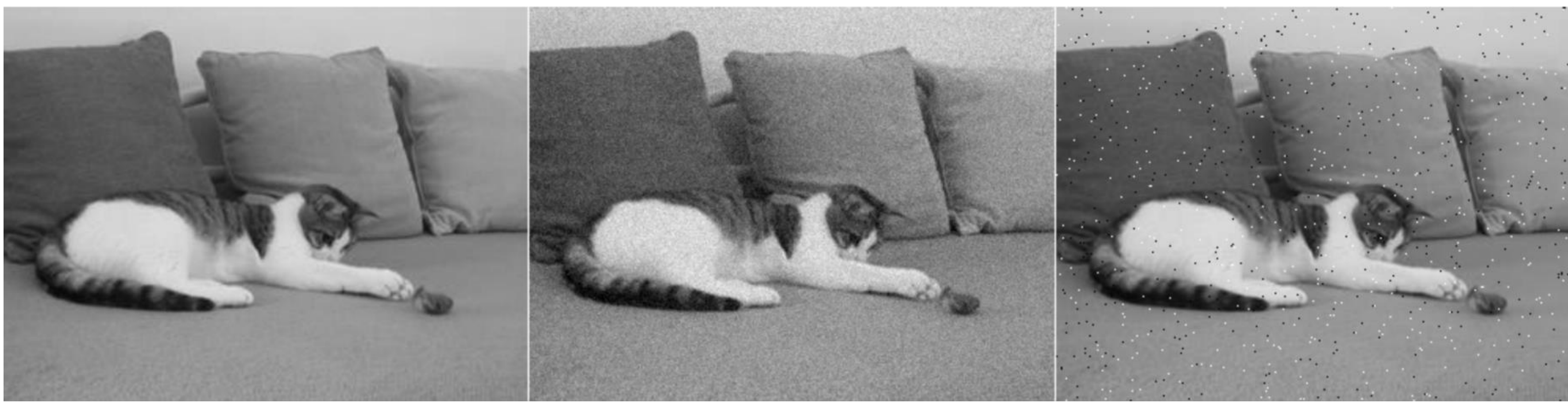
A different type of noise is the so called salt and pepper noise: extreme grey levels (white and black), or bursts, appearing at random and independently in a small number of pixels.

Adding Salt & Pepper Noise: Python Code

```
def add_SP_noise(img, p=0.01):
    ''' Add salt and pepper noise: Each pixel is "hit" independently
        with probability = p.
        If hit, it has 50:50 chance of becoming white or black '''

    def sp_noise_op(mat, x, y):
        sp_noise = BLACK if random.random()<0.5 else WHITE # 50:50
        r = random.random()
        if r<p: #noise occurs with prob. p
            return sp_noise
        else:
            return mat[x,y]

    return process_img(img, sp_noise_op)
```

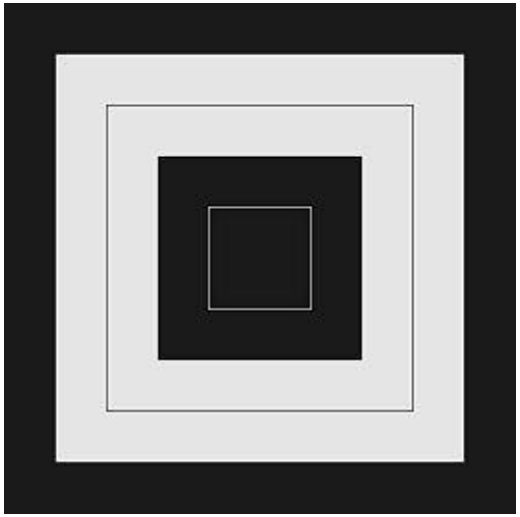
Adding Noise: Example

```
>>> img = Image.open("…").convert("L")
>>> img_gaussian_noise = add_gaussian_noise(img)
>>> img_sp_noise = add_SP_noise(img)
>>> tile(img, img_gaussian_noise, img_sp_noise).show()
```

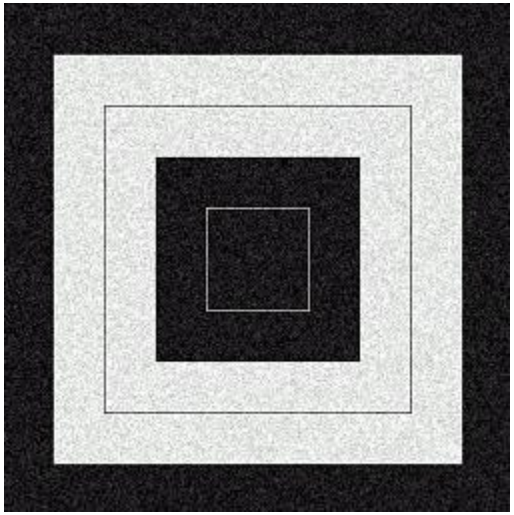


Original image　　Gaussian noise (σ=10)　　Salt & pepper noise (p=0.01)

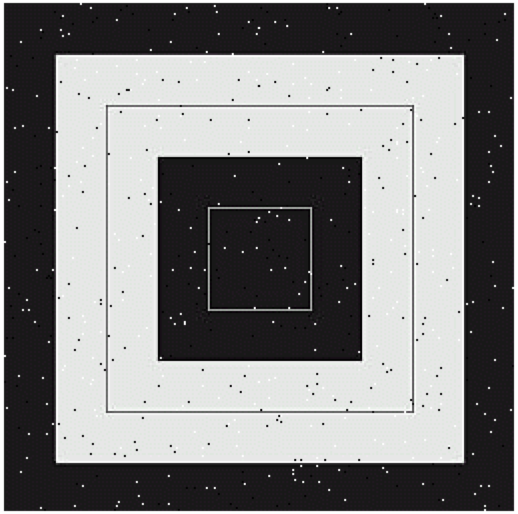
Additional Noise Examples



original



Gaussian noise (σ=20)

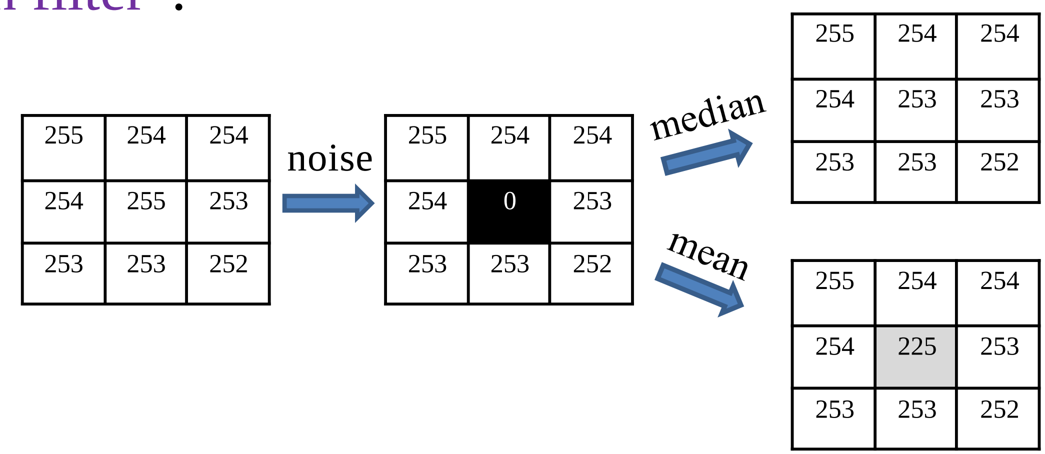


Salt & Pepper noise (p=0.01)

Local Approaches to Denoising

- Local denoising: pixel at x, y will change as a function of its surrounding pixels (called neighborhood, or environment)

- Local means uses average (mean) of neighborhood.
  - a.k.a "smoothing filter".
- Local medians uses median of neighborhood.
  - a.k.a "median filter".



- Which approach would you choose for which noise type?

Denoising by Local Means: Motivation

- If the pixel x, y resides in a smooth portion of the image, the light intensity in its neighborhood is about the same, so averaging will not change it significantly.

- In addition, it is known that averaging m independent random variables decreases standard deviation σ to $\sigma/\sqrt{m}$.

  For example, in a 3x3 environment we get $\sigma/3$.

- So in smooth areas, averaging preserves the signal component of the pixel, yet substantially decreases Gaussian noise contribution.

Denoising by Local Medians: Motivation

- Median is not sensitive to outliers as much as average.

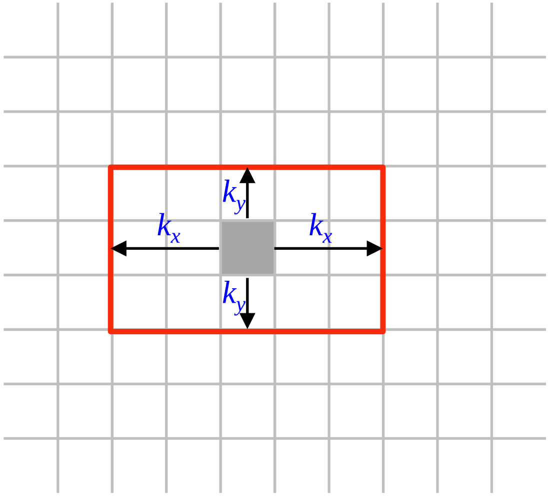
- If the pixel x, y was hit by an extreme noise component (such as in S&P), local median will eliminate it, by replacing it with a value that is more representative to the environment

Neighborhood of a Pixel

- Neighborhood (or environment) of a pixel (x,y) is the set of all pixels close to it. For example, a 3x3 square neighborhood:

$$N_{3x3}(x,y) = \begin{bmatrix} x-1, y-1 & x, y-1 & x+1, y-1 \\ x-1, y & x, y & x+1, y \\ x-1, y+1 & x, y+1 & x+1, y+1 \end{bmatrix}$$

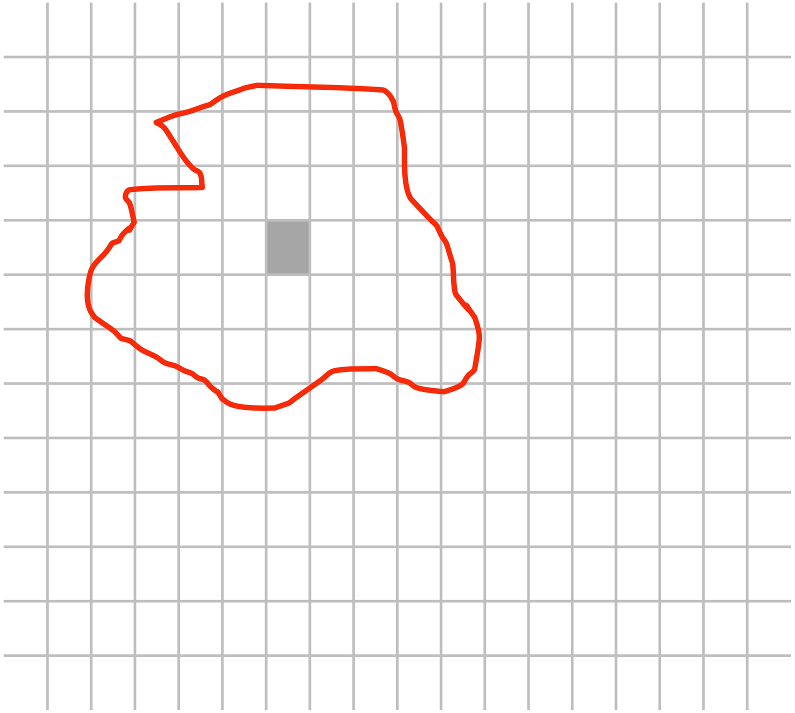
- More generally, a rectangular neighborhood of dimensions (k_x, k_y) is a $(2k_x+1)$-by-$(2k_y+1)$ rectangle.



When $k_x = k_y = 1$ we get a 3x3 square.

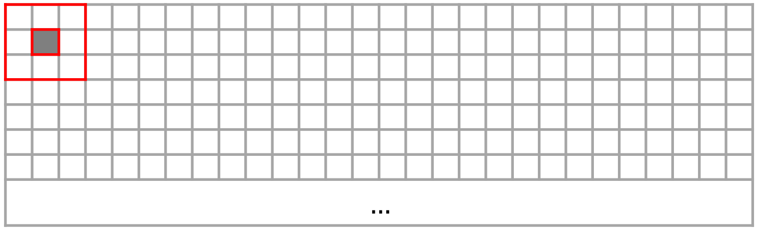
Neighborhood

- Even more generally, a neighborhood of a pixel can take any other shape:



Local Noise Reduction

- In local denoising (both local means and local medians), we visit each pixel and update its value (using some operator on its environment).



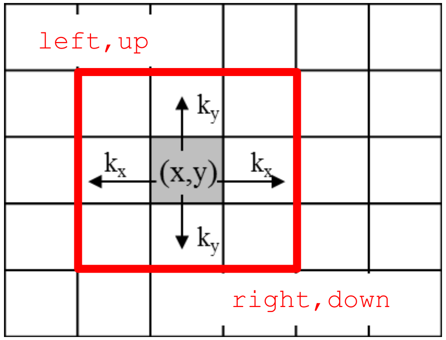
- Note that in the boundaries of the image the environment is smaller. We will use the same operators on the smaller environment.

- The updated pixel values are stored in a separate copy of the image (why?)

Local Operator - Code

```
def local_op(img, op, kx=1, ky=1):
    w,h = img.size
    mat = img.load()
    new_img = img.copy()
    new_mat = new_img.load()

    for x in range(w):
        for y in range(h):
            # 4 corners, do not exceed image boundaries
            left  = max(x-kx, 0)
            up    = max(y-ky, 0)
            right = min(x+kx, w-1)
            down  = min(y+ky, h-1)

            # flatten 2D neighborhood into 1D list
            neighbors_list = [mat[xx,yy] for xx in range(left, right+1) \
                                         for yy in range(up, down+1)]
            # apply op in list and assign result to pixel x,y
            new_mat[x,y] = op(neighbors_list)

    return new_img
```

Default: 3x3 square neighborhood



The operator is applied on the neighboring pixels

Local Means and Local Medians - Code

```
def local_means(img, kx=1, ky=1):
    mean = lambda lst: round(sum(lst)/len(lst))
    return local_op(img, mean, kx, ky)
```

```
def local_medians(img, kx=1, ky=1):
    median = lambda lst: sorted(lst)[len(lst)//2]
    return local_op(img, median, kx, ky)
```

Putting Local Means/medians to the Test

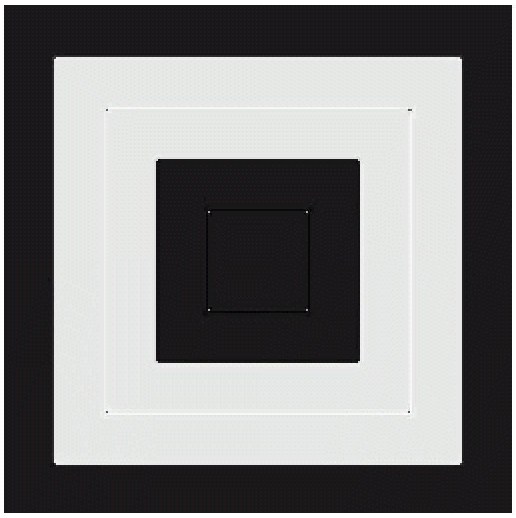
We will explore different local denoising methods on-line in class, and display results back to back with original or each other.

Any conclusions? Which method is better? Where is it better?

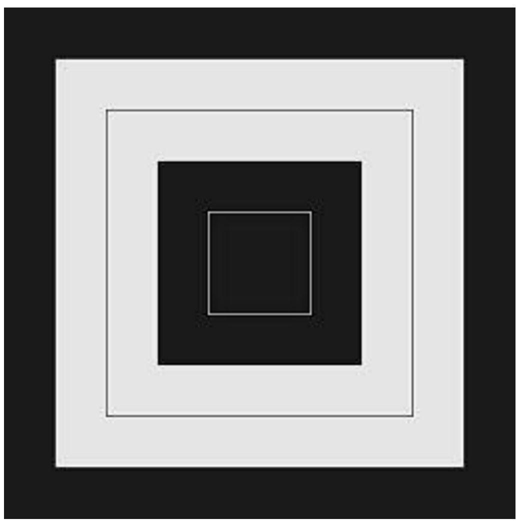
Time (and energy) permitting, we will also explore variants with larger local windows (specifically, $k=2$).

Example: Cleaning S&P

local medians (3X3)

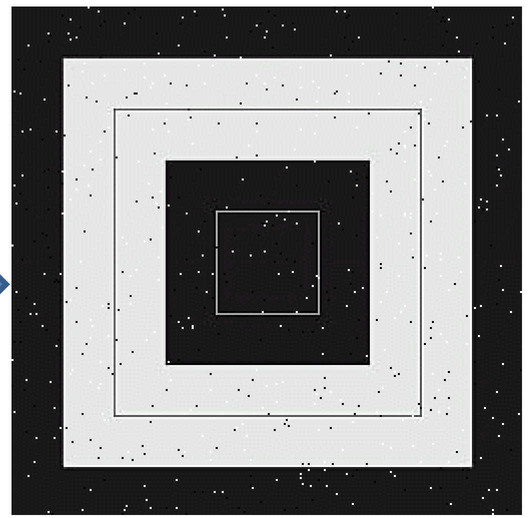


original

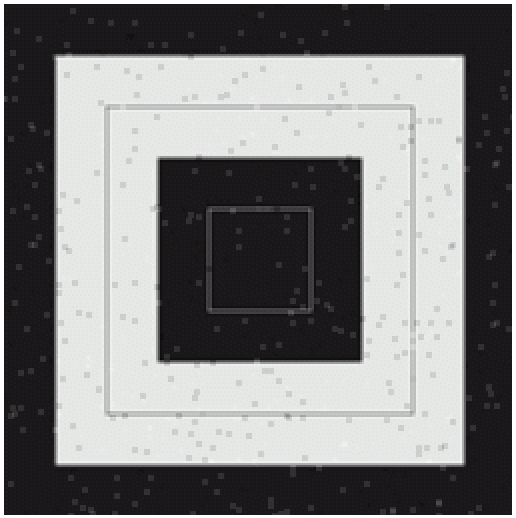


1% S&P

Noisy



local means (3X3)

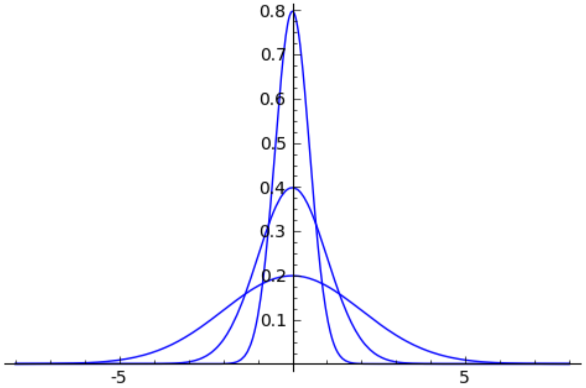


- Local medians eliminates S&P
  but also eliminates fine details (לזרוק את התינוק עם המים)
- Local means blurs image

Local Approaches – Pros and Cons

- Local medians:
  - ☑ Not sensitive to extreme outliers (will reduce S&P noise)
  - ☑ Preserves sharpness of edges
  - ✘ Eliminates small, fine features

- Local means:
  - ☑ Preserves original signal in smooth areas,
    yet substantially decreases Gaussian noise contribution
  - ☑ Reduces SD (σ)
  - ✘ In non-smooth areas blurs the image
  - ✘ Sensitive to extreme outliers



Three Gaussians, with σ = 0.5, 1, 2 (σ = 0.5 is the narrowest).

Time Complexity of Local Means and Local Medians

- Suppose the image dimensions are n-by-m.
- The number of pixels we visit is $O(n \cdot m)$.
- For every such pixel, we either compute the average of the values in the window, or find their median.
- The number of pixels in a window is $(2k+1)^2 = 4k^2 + 4k + 1 = O(k^2)$.
  - Computing averages takes $O(k^2)$.
  - For median, we employed sorting, taking $O(k^2 \, log k^2) = O(k^2 \, log k)$ steps (faster median finding algorithms do exist – wait for the data structures course)

Weighted Local Means

- Uniform averaging over the whole neighborhood, as discussed before, can be expressed as the matrix dot product:

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

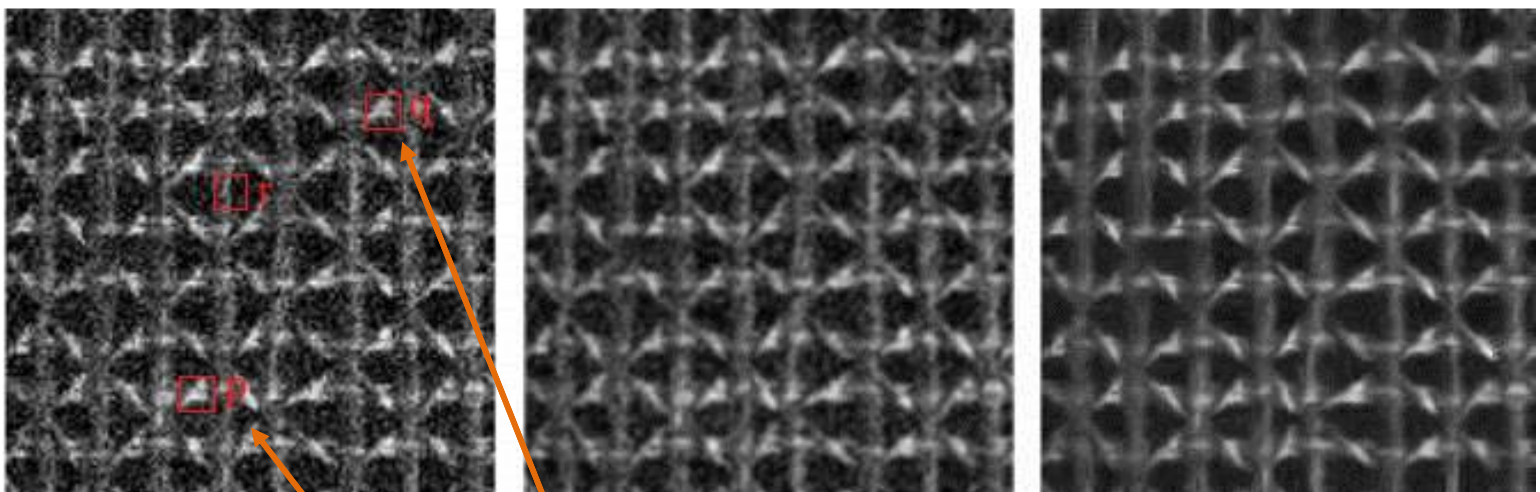
- A common variant puts more weight close to the center, for example:

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

- Other weights matrices (a.k.a filters or masks) are used for various goals.

Non-Local Means
(for reference only)

- Many natural images have a high degree of redundancy. Specifically, this means that for most small windows in the original image, the window has many similar windows in the same image.



Gaussian noise | Local means | Non-local means means

- Windows centered at p and q are similar, but not to the one centered at r.

Denoising by Non-Local Means (for reference only)

- The non-local (NL) means algorithm (A. Buades, B. Coll, and J. M. Morel, 2005) heavily employs the notion of non-local, similar windows. Given a window centered at (x, y), we search for all windows in the image that are similar to it.

- In other words, we look for all (x', y') such that the "distance" between the windows centered at x, y and x', y' is below some fixed threshold h.

- We compute the weighted average value of all those similar center pixels (including (x, y) itself), with higher weights assigned to windows that are more similar. The corrected value for (x, y) equals this average.

- The method is called non-local since the windows that effect the corrected value are not necessarily in close proximity to (x, y).

- Remark: This is a fairly simplified version of NL means. For reasons of efficiency, one usually scans only a subset of all possible windows.

More on Digital Image Processing

- Common problems:
  - Noise reduction (denoising) - removing noise from an image.
  - Segmentation - partitioning a digital image into segments (e.g. background and foreground)
  - Edge detection – detecting discontinuities in the image
  - Image\video Compression – decrease volume in memory (usually lossy)
  - Tracking – identifying relate objects in subsequent frames of a film
  - Registration - transforming different images into one coordinate system (*e.g*. minor shifts in the camera position in subsequent frames
  - Color correction.

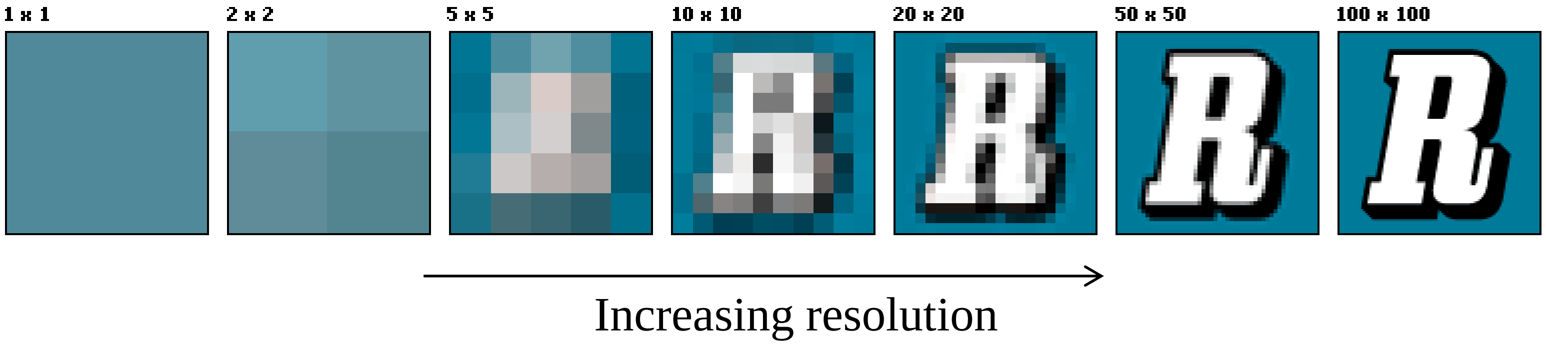
- Typical applications:
  - Machine vision
  - Medical / biological image analysis
  - Face detection
  - Object recognition
  - Augmented reality
  - …

57

Resolution and Pixel Physical Size

- Resolution is the capability of the sensor to observe or measure the smallest object clearly with distinct boundaries.

- Resolution depends upon the physical size of a pixel.
  Higher resolution = lower pixel size.



Source: Wikipedia

Compression and Image Formats

- Digital images with high pixel resolution and bit depth take up lots of computer memory.

- This motivates the need for compressing images.

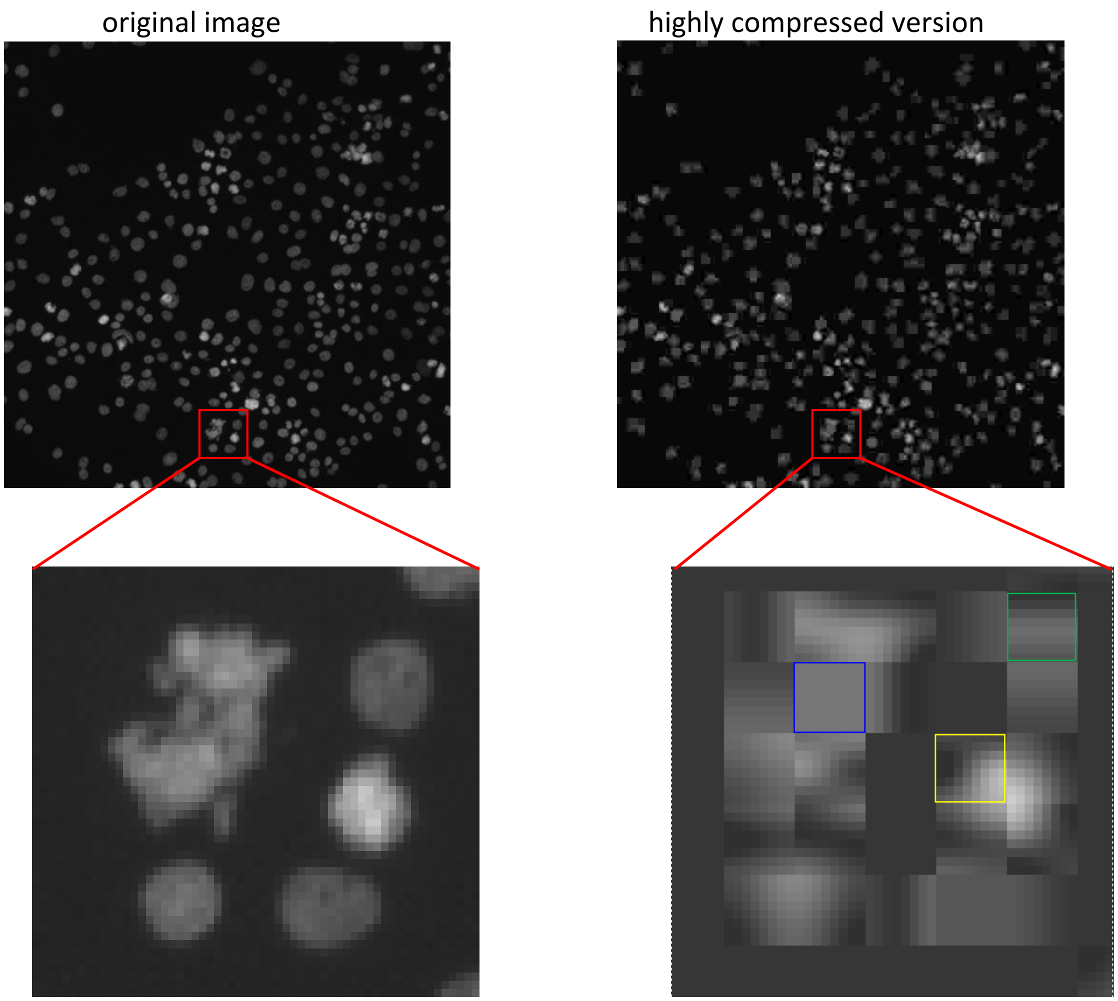
- During compression, some of the information in the image may be lost, in which case the compression is termed lossy. Otherwise, we call it lossless.

- jpg, tiff, png, bmp, gif etc., differ by the type of compression applied to the original image.

  The bmp format is lossless, while the other formats are lossy (tiff can be both, depending on some parameter settings).

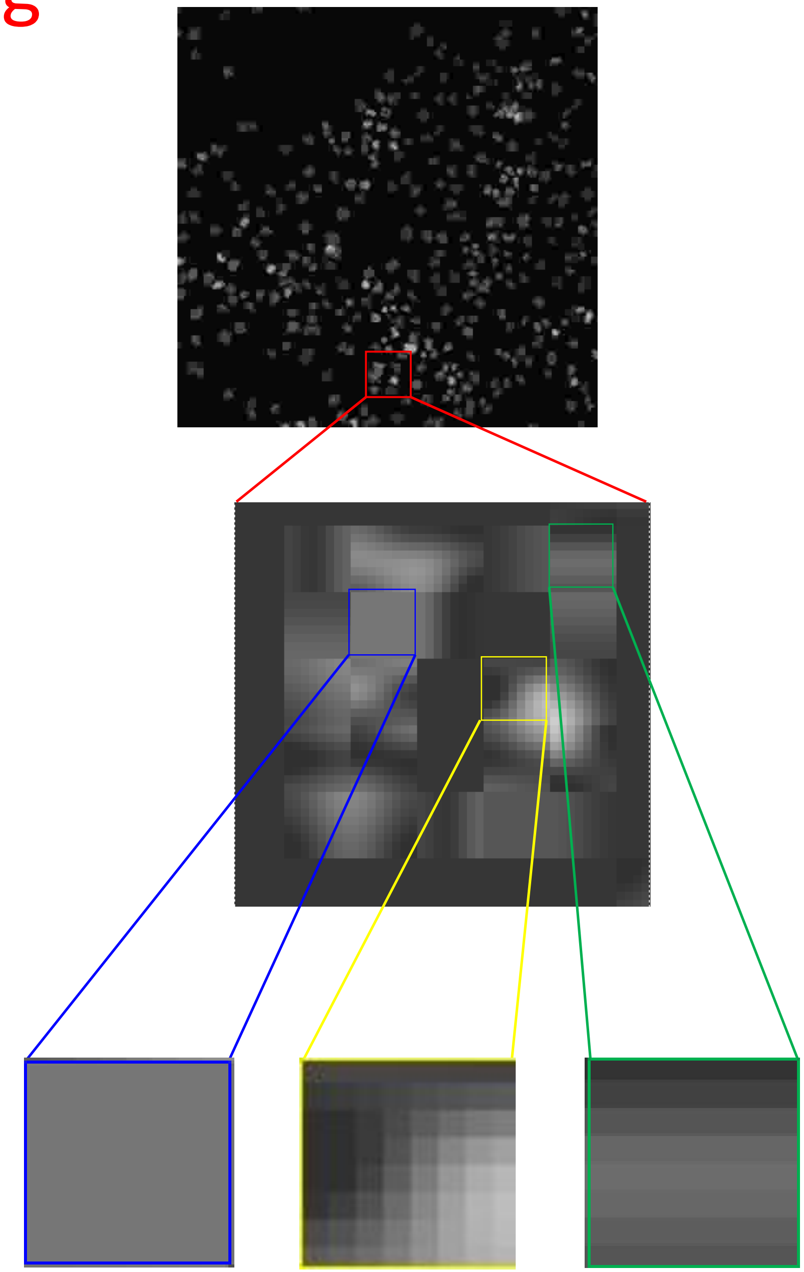
The Example of jpg

- jpg format partitions the image into squares of 8-by-8 pixels.
- Most such squares will exhibit only gradual, moderate changes, especially in smooth areas of the image.
- These gradual changes can be well approximated by far fewer bits than the 8·8·8 = 512 bits in the original representation.
- A factor of 10 (or even more) saving in space can be achieved.



Human HT29 colon-cancer cells.
In the compressed image on the right, In the blue square all pixels are identical. In the green square, pixels only change from top to bottom. In the yellow square, pixels change in both directions.

The Example of jpg



Segmentation

- The process of partitioning a digital image into multiple segments (sets of pixels, also known as superpixels).

Source:
http://www.sonycsl.co.jp/person/nielsen/applets.html

- The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.

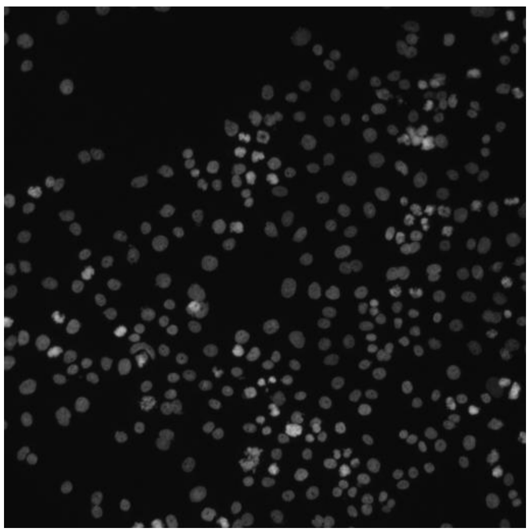
- Image segmentation is critical for many subsequent processes, such as object recognition, shape analysis and tracking. It is typically used to locate objects and boundaries (lines, curves, etc.).

- Examples: locating tumors or anatomical structures in medical images; face detection; identifying objects in satellite images (roads, forests, crops, etc.).

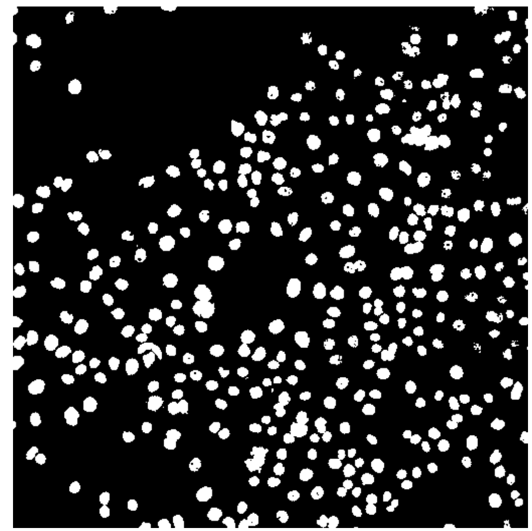
Binary Segmentation by Thresholding

- Simplest segmentation method: Apply a threshold to turn a gray-scale image into a binary image (BW).

- Assumes the image contains two classes of pixels denoted foreground and background, and these two classes have distinct, different light intensities.



Human HT29 colon-cancer cells
http://www.broadinstitute.org/bbbc/image_sets.html



Binary segmentation, threshold = 40

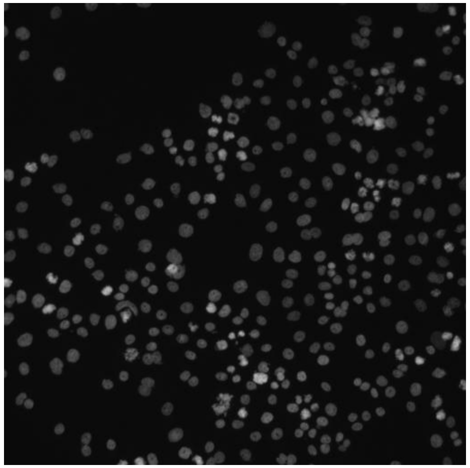
- Generally, one can apply more than one threshold, creating >2 segments

Picking a Threshold

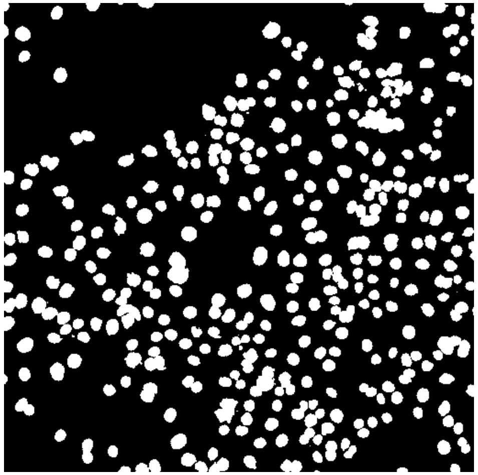
- The key is to select the appropriate threshold
- Which one is the best here?

- When the threshold is too low (20 in this case) areas in the image where cells are densely populated become bulbs.
- When it is too high (60) some cells are lost (those whose brightness was low in the original image).

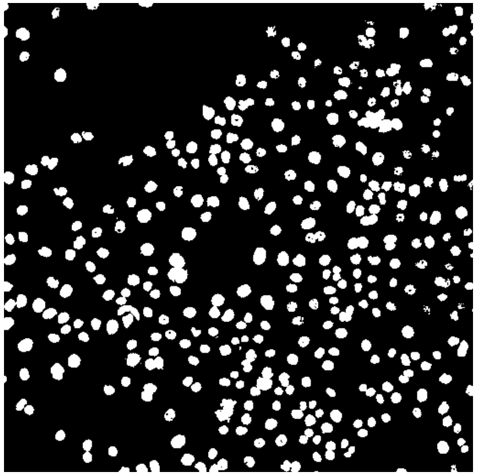
Original



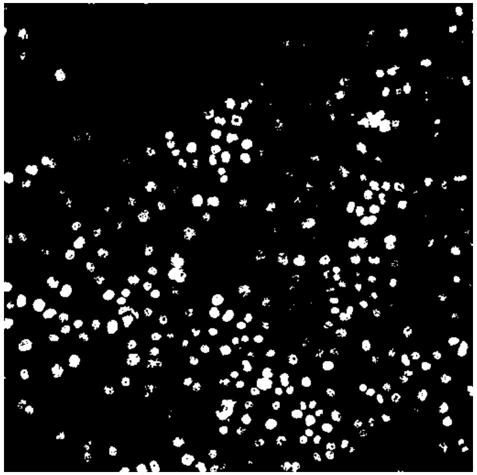
Threshold = 20



Threshold = 40



Threshold = 60

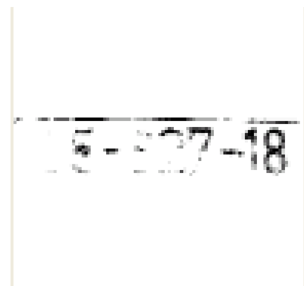


Binary Segmentation – another example

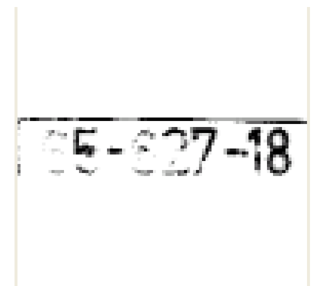
- Below are the results of binary segmentation with increasing thresholds (out_20 for example uses threshold 20).

out_20.bmp

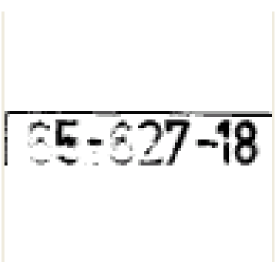
out_40.bmp



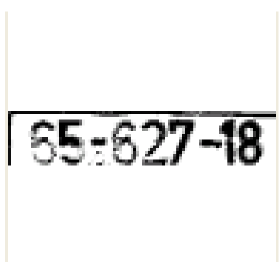
out_60.bmp



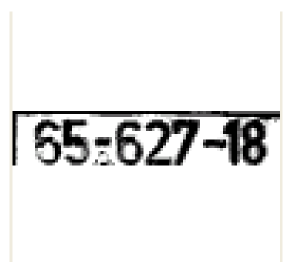
out_80.bmp



out_100.bmp



out_120.bmp



out_140.bmp

out_160.bmp

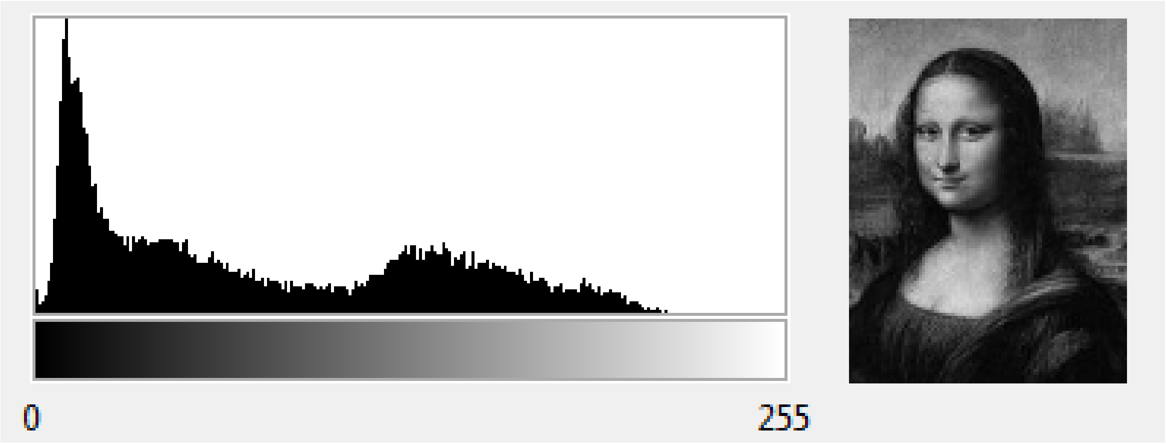
out_180.bmp

Otsu Threshold

- A good threshold for segmentation:
  - minimizes differences within each segment, and
  - maximizes differences between segments.

- Otsu's method finds such optimal threshold.

- Uses image histogram: grey level values distribution.
  - x-axis – grey hues
  - y-axis – number of pixels with a particular hue

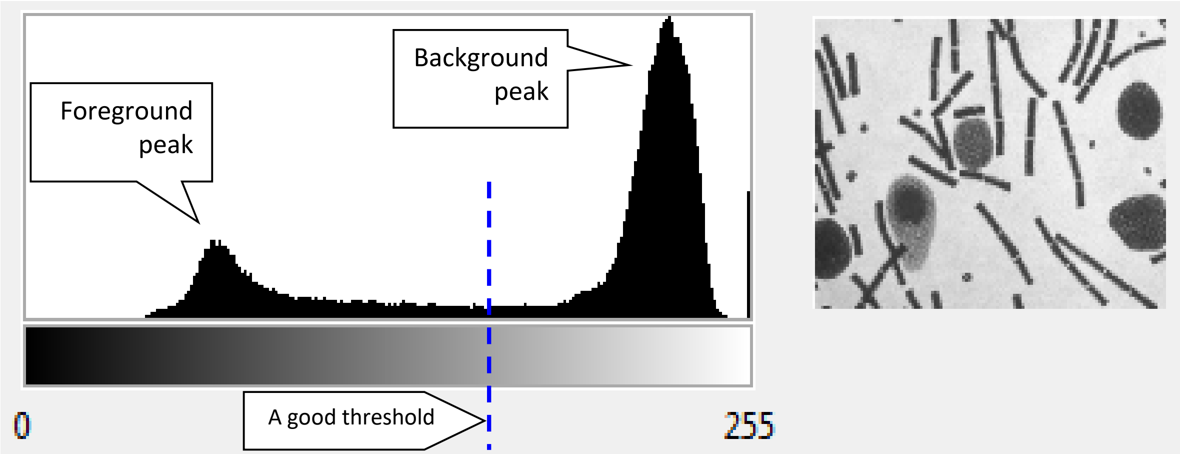


Otsu Threshold

- Otsu's method relies on the assumption that the foreground and the background of the image differ substantially in their brightness.

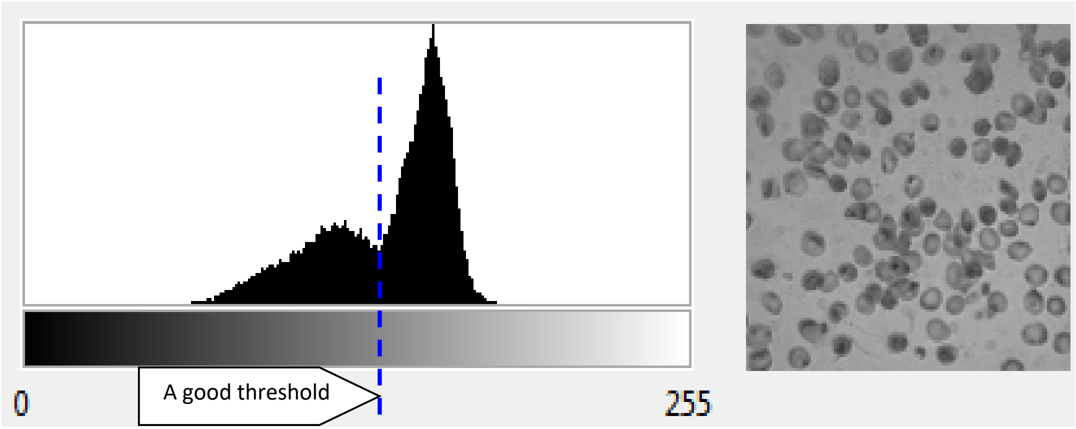
- This assumption is not true in many cases, as in the Mona Lisa example.

- However, when this assumption holds, there are expected to be two peaks in the gray values of an image's histogram (such image histograms are called bi-modal).
- In this case the lowest mid-point between these two peaks would be a good choice for a threshold.

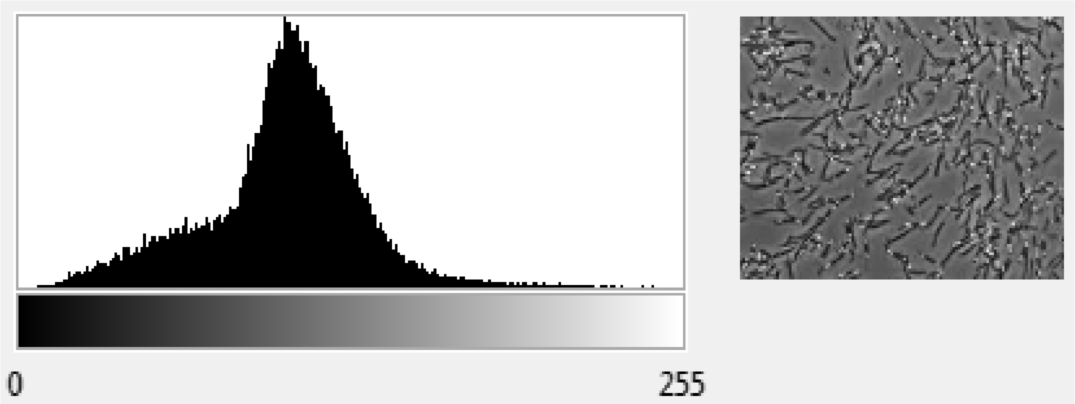


Otsu Threshold

- When the difference between foreground and background are less sharp, the peaks may be partly overlapping:



- When the image histogram is not bi-modal, Otsu's method will be inapplicable:



Otsu's Formula

For every threshold *t* denote:

back – number of background pixels (<= t)
fore – number of foreground pixels (> t)

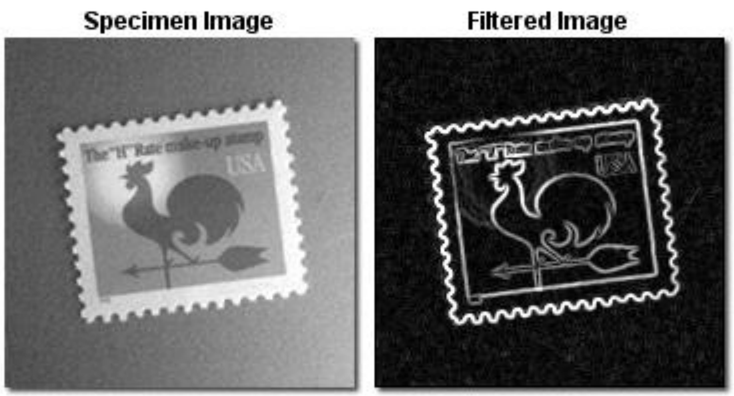
mean_back – mean value of the background pixels
mean_fore – mean value of the foreground pixels

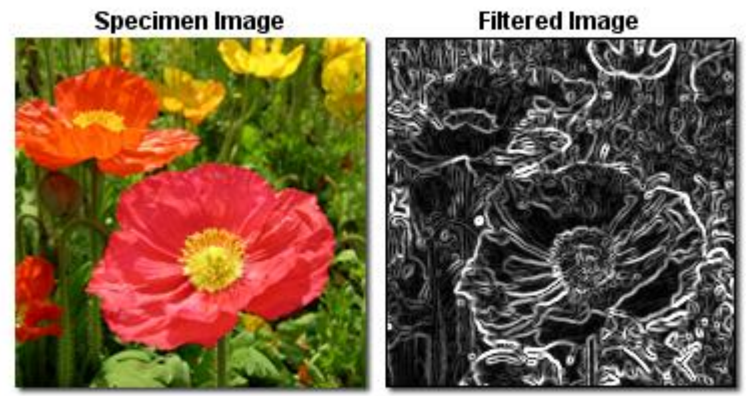
var_between(t) = *back * fore * (mean_back - mean_fore)*^{2}

- Otsu threshold is the one that maximizes the *var_between* among all possible thresholds *t*.
- What is the effect of the difference between the means?
- What is the effect of the relative sizes of the background and foreground?

Edge Detection

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

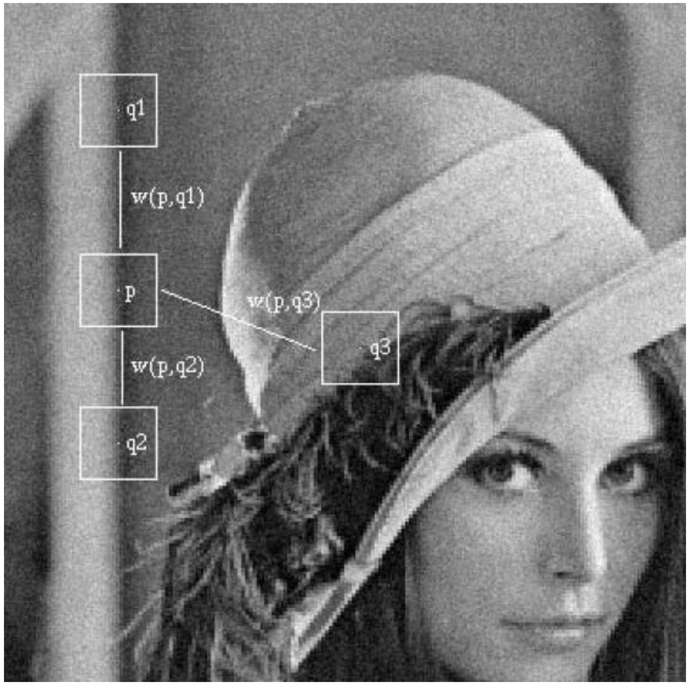




images extracted using Sobel filter from:

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

Some non-CS issues



From Wikipedia: Lenna or Lena is the name given to a standard test image widely used in the field of image processing since 1973. It is a picture of Lena Sderberg, shot by photographer Dwight Hooker, cropped from the centerfold of the November 1972 issue of Playboy magazine. Given the nature of the image and its source, several academics have criticized its continued use in scientific publications and higher education as both sexist and unprofessional.

The course staff joins this view. We do our best to avoid objectification of women in the course or the course material.