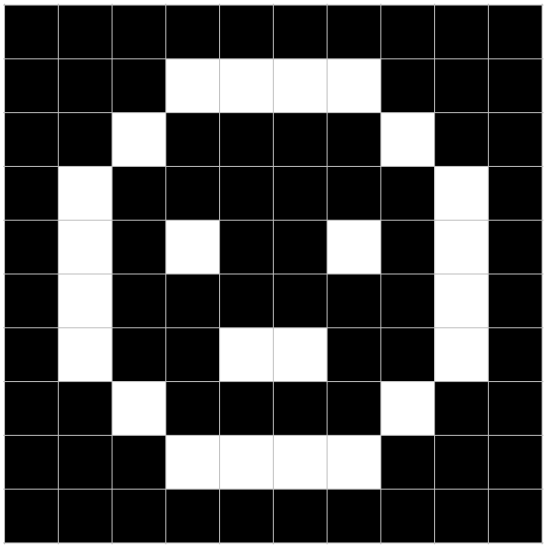
BME 200
Lesson 14 - Bioimaging
Oct 8, 2020

By the end of the lecture students should be able to:
- Describe how an image filter works
- Define kernel and image processing
- Explain how a computer represents bitmapped images
- Calculate spatial resolution
- Calculate a new pixel value given a kernel

I. How do computers make images?
   A. Computers represent images using two different strategies
      1. Bitmapped graphics
      2. Vector graphics
   B. In image analysis bitmapped graphics are more common, so we will spend our time talking about this method.

II. Bitmapped graphics
   A. Computers represent images with numbers. Here's how.
   B. A computer screen is a large grid of tiny squares called pixels. The size of your screen determines how many pixels are in it.
      1. This projector is 640 x 480, or 640 pixels across by 480 pixels down for a total of 307,200 pixels in all.
      2. If you have a newer iPhone (iPhone XR), the screen resolution is 1792 pixels tall by 828 pixels wide, for a total of 1,483,776 pixels.
      3. Your laptop probably has a screen resolution of at least 1440 pixels by 900, for a total of 1,296,000 pixels.
   C. We make a picture by putting a color in each square. The simplest pictures are black and white. Modern displays can show over 16 million (2^{24}) colors per square.
   D. Imagine a really old computer where the computer screen only has a resolution of 10 pixels by 10 pixels. Here's what the screen would look like:

1. Let's say we want to draw a meh face on our simple computer screen. It might look like this.

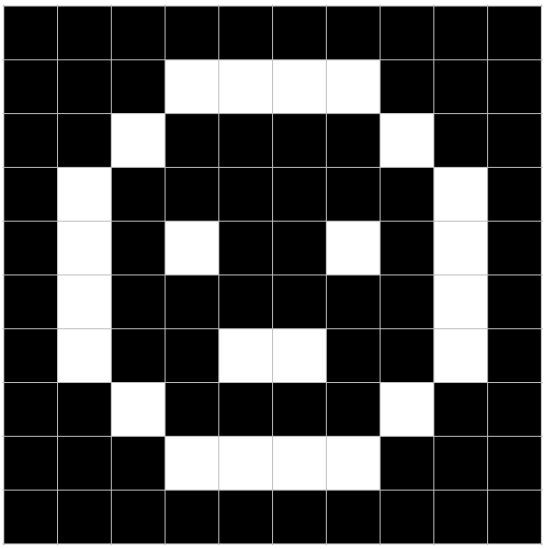


   a) Notice that some of the pixels are black and some are white. Each color can be represented by a number. There are two colors, so we can say black = 0 and white = 1
   b) If I save this picture to a file, it will contain 100 numbers (10 x 10) like this:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0
1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0
```

c) If I hit return after every 10 numbers, here is what the file would look like:

```
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0 0 0
0 0 1 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0
0 1 0 1 0 0 1 0 1 0
0 1 0 0 0 0 0 0 1 0
0 1 0 0 1 1 0 0 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0
```

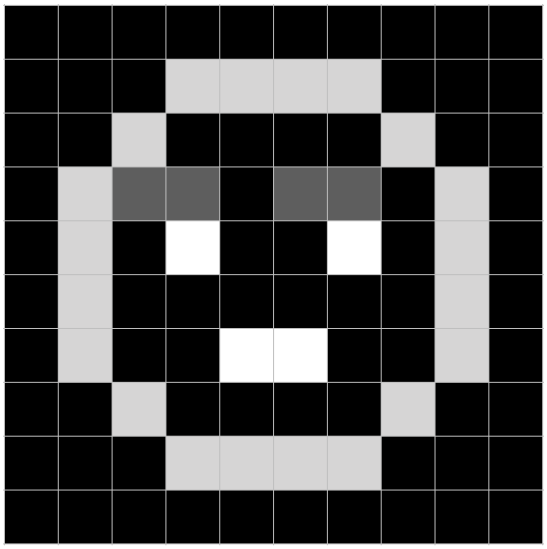


d) Wherever the number is 0, the square is black and wherever the number is 1, the square is white
e) In computer jargon, numbers are made up of **bits**, which is why this is called a *bit-mapped image*
f) So the bits (numbers) determine the colors and their position in the file determines where the color goes
g) This image is a *1-bit image* because only a single 1 and 0 is used for each pixel

2. Greyscale
   a) The world's not black and white so our pictures shouldn't be either. If we use larger numbers, then we can represent shades of grey
   b) Instead of each pixel only having a value of 1 or 0, we can let it have a number between 0 and 15.
   c) In this scenario, 0 is black, 15 is white, and the numbers in between represent grey (closer to 0 is darker grey; closer to 15 is lighter grey)
   d) To represent 0 - 15, we need a few more bits, like this:

e) If we use this larger range of numbers to create our image, it might look like this:

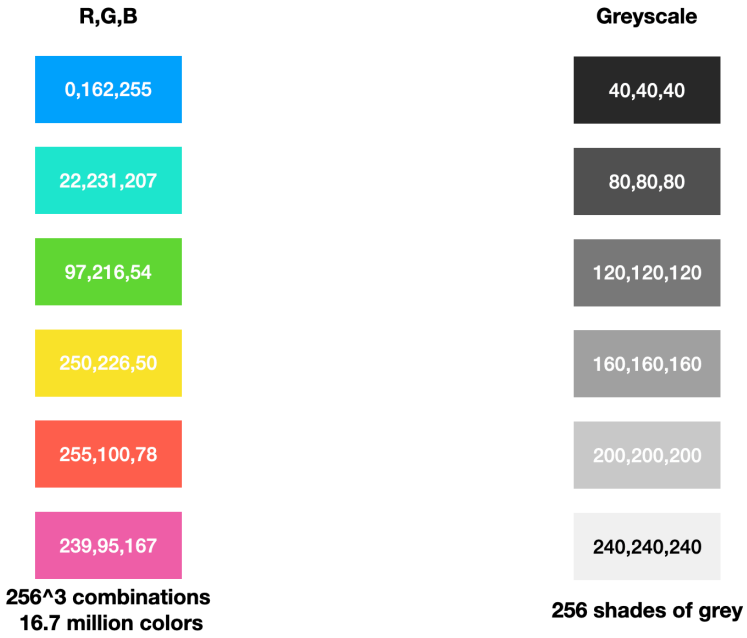
```
00 00 00 00 00 00 00 00 00 00
00 00 00 04 04 04 04 00 00 00
00 00 04 00 00 00 00 04 00 00
00 04 12 12 00 12 12 00 04 00
00 04 00 15 00 00 15 00 04 00
00 04 00 00 00 00 00 00 04 00
00 04 00 00 15 15 00 00 04 00
00 00 04 00 00 00 00 04 00 00
00 00 00 04 04 04 04 00 00 00
00 00 00 00 00 00 00 00 00 00
```



f) And so on…

3. Colors
   a) Most displays make colors by mixing red, blue, and green
   b) Specify a value of each from 0-255 and you get over 16 million color combinations
   c) Show some red, blue, green, grey, tangerine
   d) This lets you create complex images, such as video game characters

e) You can use three numbers to specify colors, or greyscale

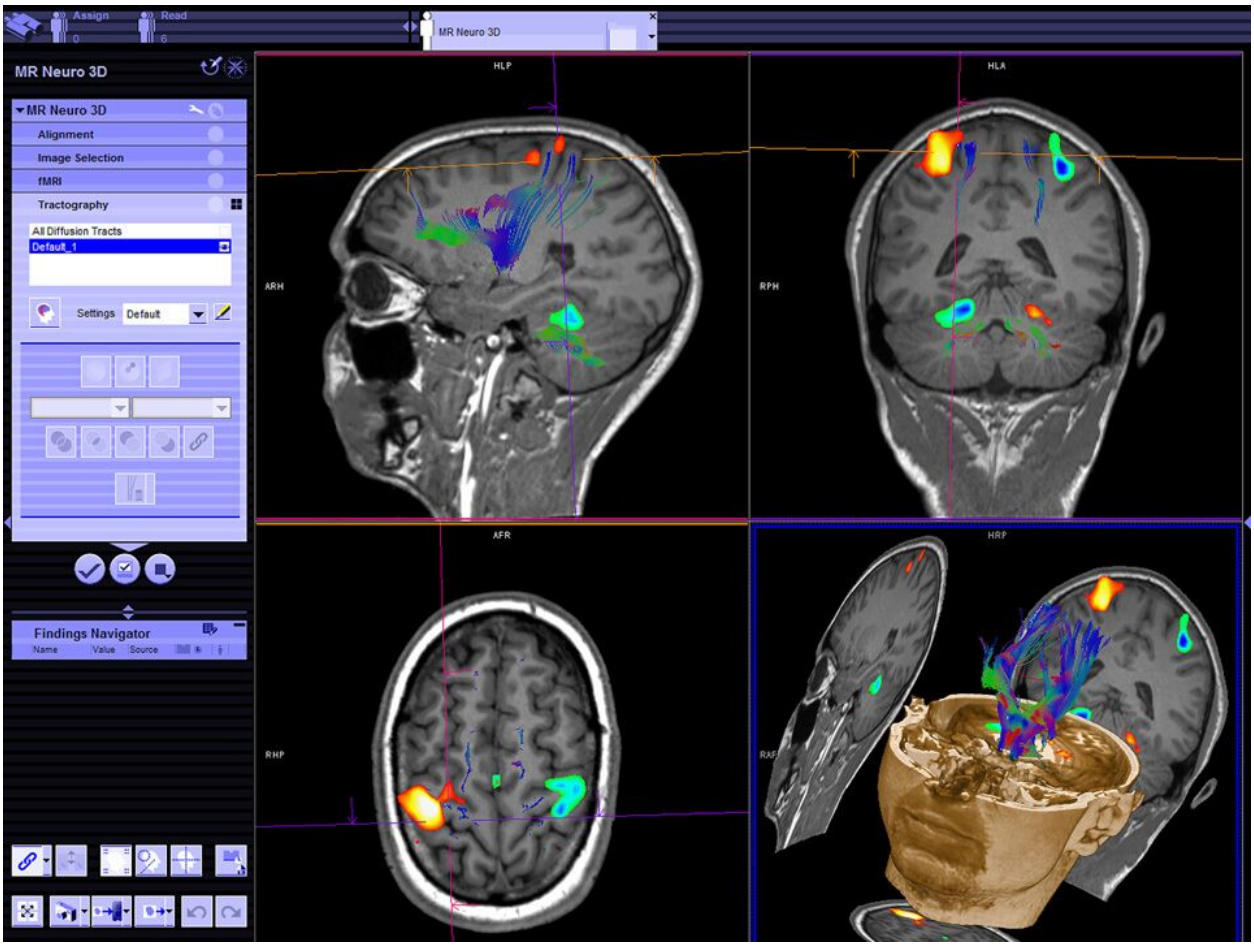


f) Student problem. How many shades of grey can you have if you use 8 bits in your image?

III. Resolution of medical images

A. These concepts are used to create medical images. For example

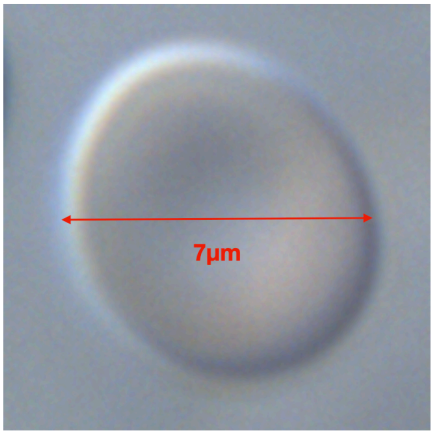
1. Medical imaging



2. Imaging of cells

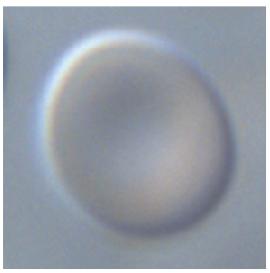
B. 2D images are made of *pixels*
   1. Each pixel represents a square area, and each side of the square is a distance
C. 3D images are made of 3D pixels called *voxels*
   1. A voxel represents a finite volume in the body
D. In medical imaging, the voxels and pixels represent a distance of a real thing
   1. For example, if you know how large each voxel is, and you see that a tumor is 10 voxels in size, then you can get a rough idea of how big it is
E. Likewise, you can estimate cell sizes from computerized images
F. Student problem. Assume in an image of a red blood cell, the cell is approximately circular with a diameter of 200 pixels (see below). Given that the average diameter of a red blood cell is ~ 7 µm (7×10^{-6} m), how large is each pixel?

230 x 230 px image

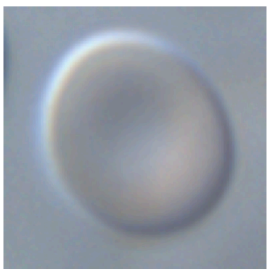


G. How does resolution affect the image?

200 x 200



100 x 100

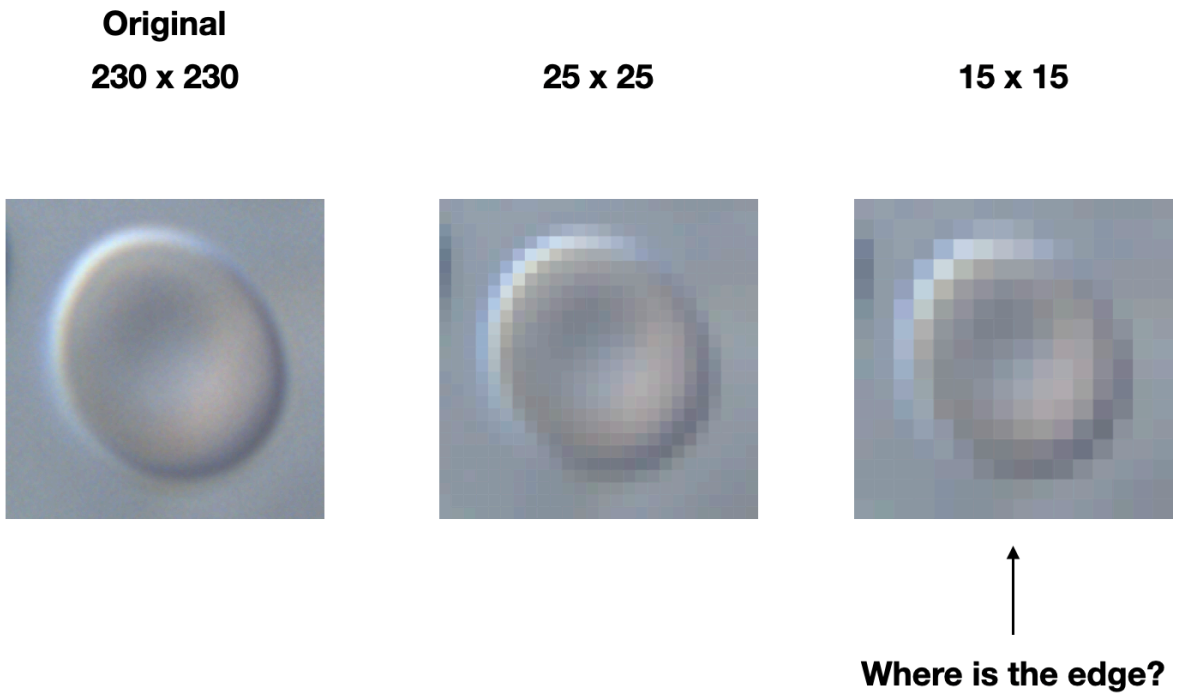


50 x 50

25 x 25

1. Main point: Lower res = more blocky

H. But, this brings up an interesting point. Where is the edge?



1. The smallest feature we can resolve is known as the *spatial resolution*
2. Put another way, if we have something smaller than the resolution of a single pixel, we won't be able to clearly make it out
3. We can never know the actual distance if our resolution is too low
4. The best we can do is guess to the nearest pixel
   a) For example, assume each pixel is 0.6 µm and the thing we want to measure is 7 µm across. The best we can say is 7.2–6.6 µm because we are limited by the width of a single pixel (0.6 µm).

IV. Image Processing

A. In many fields that use images, scientists and engineers have developed a large number of tools to *process* the images.

B. These tools are algorithms that enhance the image in some way to make some feature more obvious or visible
   1. This is also known as *filtering*

C. How filtering works
   1. Define a *kernel* or *filter* that describes how you want to manipulate the image
      a) A kernel is just a matrix of numbers, like this:

b) This particular kernel is called an *averaging* or *smoothing* filter because it smooths out pixel values within the image

2. Then, go to each pixel within the image and calculate the filtered/processed value
   a) Formula for convolution

3. Example
   a) Consider this setup:

Corner of image						Kernel for Smoothing filter				New pixel value				
0	255	0			*	1/9	1/9	1/9	=	0	255	0		
255	0	255				1/9	1/9	1/9		255	113	255		
0	255	0				1/9	1/9	1/9		0	255	0		

   b) Multiply each square in the image by each element in the matrix to get the new pixel value:

c) Student problem. Let's fill out the image a little more. Find the new pixel value

Find the new value for the pixel with a blue number				
0	255	0	255	
255	0	255	0	
0	255	0	255	

*

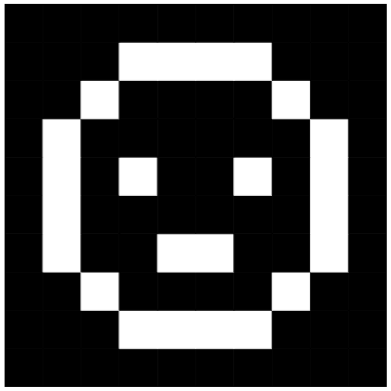
Kernel for Smoothing filter		
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

New pixel value				
0	255	0	255	
255	0	?	0	
0	255	0	255	

d) If we run our smoothing filter on our meh face, this is what happens:

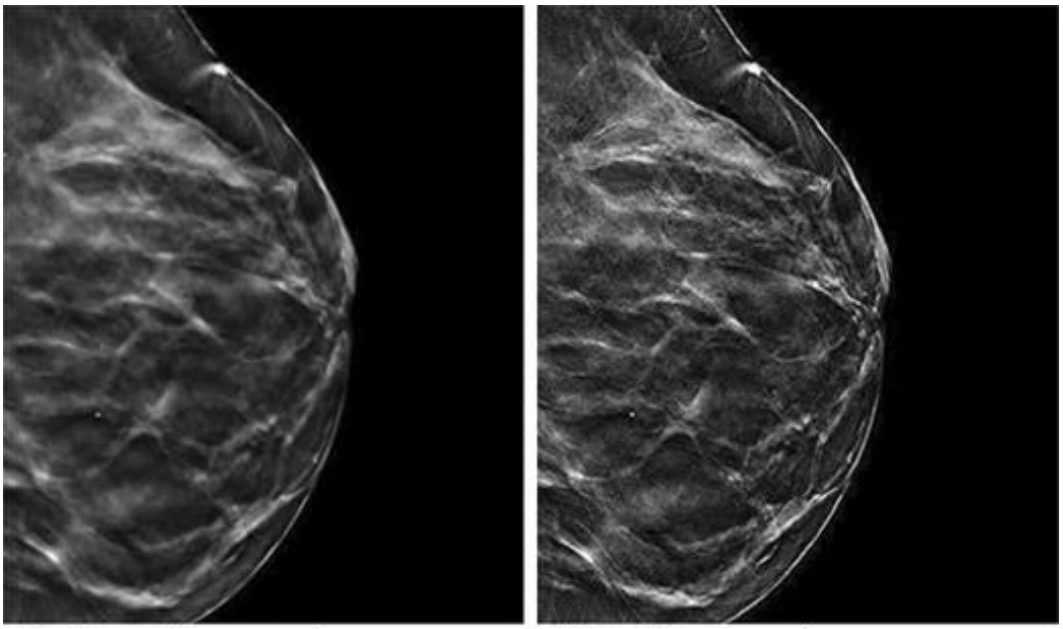
Before



After

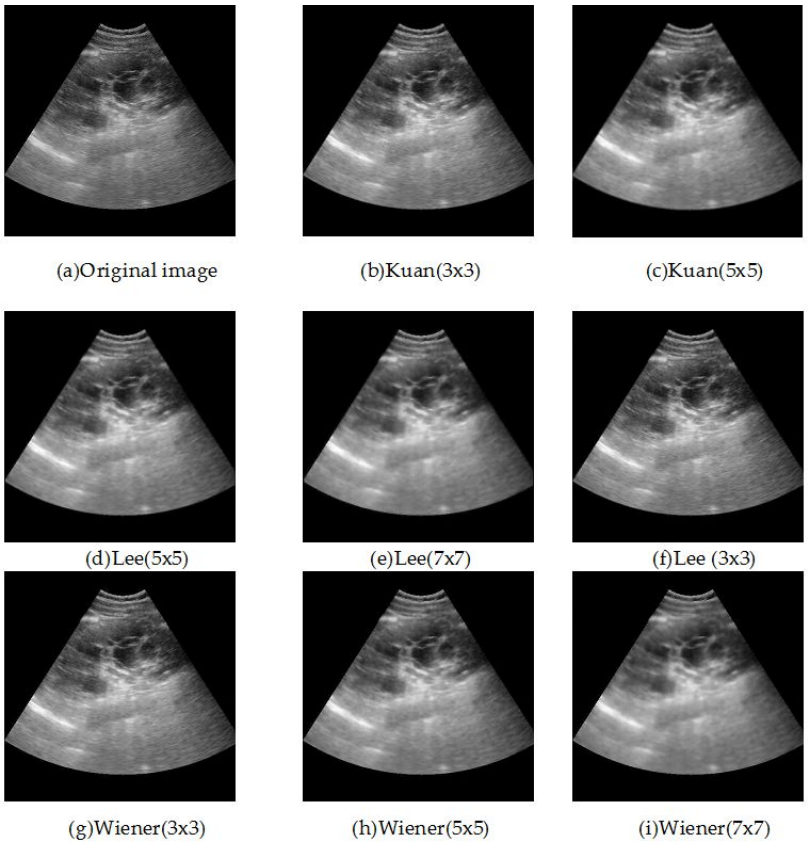
e) Examples of other types of filters
   (1) Edge detection (make edges more visible)
   (2) Sharpen
   (3) Blur

f) Example of a sharpening filter used on a mammogram



https://bit.ly/2GD9ArB

g) Example of smoothing filters on ultrasound images



https://bit.ly/2GzOxWV

h) Example of an edge detection filter used on an x-ray

https://bit.ly/2StGCxd