seeing more than a pretty picture

Image Analysis
Where’s the fish?
Where’s the fish?
1. What are images?
   - Grayscale
   - Color
   - Movies

2. Getting images into (and out of) MATLAB
   - Imread
   - vidReader

3. Finding Objects
   - Thresholding
   - ROI Segmenting

4. Getting information from objects
   - regionprops
   - Morphological Operations
1. What are images?
   - Grayscale
   - Color
   - Movies

2. Getting images into (and out of) MATLAB
   - imread
   - vidReader

3. Finding Objects
   - Thresholding
   - ROI Segmenting

4. Getting information from objects
   - regionprops
   - Morphological Operations
Part 1. What are Images?

What does the *computer* see when you give it a picture?

In other words, how does a computer know that the blood cell or neuron should be light?

while the background should be dark?
Part 1. What are Images?

Zooming in, we find that images are made up of discrete units of different light intensity.

At this distance, one could say the image resembles a 3x5 Matrix.

All we have to do to convert it into a matrix is assign a number to each shade of grey.

It turns out, this is exactly what computers see.
Part 1. What are Images?

Most images display 255 different intensity values.

This is because most images are saved at 8-bit depth, meaning every pixel has at most 8 bits dedicated to it.

How many different numbers can be stored with 8 bits?
What are color images?

Color images are made by combining either 3 (RGB) or four color channels (CMYK)
Color images are made by combining either 3 (RGB) or four color channels (CMYK).

Individually, however, these channels are separate grayscale images, with values between 0 and 255.

How do these three 2-D matrices combine to form a single color image?
Let’s return to the 3x5 image of the blood vessel:

While it appears to be a single 2-D 3x5 matrix, we know it is actually composed of 3 different, but equally sized 2 dimensional 3x5 matrices:

To create the color image, we combine the three components along a *third dimension*
How do you expect to store movies?
Each frame in a movie has 2 dimension size dimensions that are stored as rows and columns.
Colors are then stored along a third dimension (which always has size of either 3 or 4)
Finally, time, or frame number, is stored along the fourth dimension.
In MATLAB, suppose you have a variable named MyMovie that stores a color movie 30 frames long with 512x512 pixels in each frame.

How would you…..

1. Create a matrix, M1 that stores all of the red pixels values in frame 12?
   What are the sizes of its dimensions?
   
   \[
   M1 = \text{MyMovie}(::;1,12) \\
   \text{size}(M1) = 512x512x1x1
   \]

2. Create a matrix, M2 that stores all of the color values in the first row of frame 20?
   What are the sizes of its dimensions?
   
   \[
   M2 = \text{MyMovie}(1,::;20) \\
   \text{size}(M1) = 512x512x1x1
   \]

3. Create a matrix, M3 that stores the green pixel values across all frames from a single pixel located in the 100th column and 250th row pixels values across all frames?
   What are the sizes of its dimensions?
   
   \[
   M3 = \text{MyMovie}(250,100,2,:.) \\
   \text{size}(M1) = 1x1x1x30
   \]
1. What are images?
   - Grayscale
   - Color
   - Movies

2. Getting images into (and out of) MATLAB
   - imread
   - vidReader

3. Finding Objects
   - Thresholding
   - ROI Segmenting

4. Getting information from objects
   - regionprops
   - Morphological Operations
To import an image, use the `imread` function.

\[
\text{ImageMatrix} = \text{imread('pathname/filename')}
\]

- **ImageMatrix**: Variable that will store the pixel values of your image. Will be either a 2-D or a 3-D matrix.
- **imread('pathname/filename')**: Function. Location of your file. Must be a string or a variable that stores a string.

**REMEMBER:**

While the expression on the right side of the equality sign is your image as a matrix - you must provide a variable on the left side of the equality sign that will store that matrix for future use.
Part 2. Getting Images into MATLAB

Import the image I asked you to download and save it in a variable called ImageMatrix.
To display an image, use the *imshow* function.

**imshow(ImageMatrix)**

- **Function**
  - Input image

The input to imshow must be a 2-D or 3-D matrix or a variable that stores one.

The result is a figure window containing your image.

Why don't we need to include anything on the left side of an equality sign?
Display ImageMatrix with imshow
Unfortunately, there is not a single function that will read a video file into a matrix directly from the file’s location the same way *imread* reads an image.

Instead, importing videos is a **two** step process:
Unfortunately, there is not a single function that will read a video file into a matrix directly from the file’s location the same way `imread` reads an image.

Instead, importing videos is a two step process:

\[
\text{VidObject} = \text{VideoReader('pathname/filename')}
\]

- **Variable that will store the Video Object.** Acts as a translator between the file and MATLAB so MATLAB can understand the video. Contains information including frame rate, bit depth, etc..
- **Function**
- **Location of your file.**
  - Must be a string or a variable that stores a string
Instead, importing videos is a two step process:

\[
\text{VidObject} = \text{VideoReader('pathname/filename')}
\]

Variable that will store the Video Object. Acts as a translator between the file and MATLAB so MATLAB can understand the video. Contains information including frame rate, bit depth, etc..

Function
Location of your file.
Must be a string or a variable that stores a string

\[
\text{VidMatrix} = \text{read(VidObject)}
\]

3-D or 4-D matrix containing the pixel values for every frame
Function
Object created by VideoReader
Import the movie I asked you to download and save it in a variable called VidMatrix
Roadmap

1. What are images?
   - Grayscale
   - Color
   - Movies

2. Getting images into (and out of) MATLAB
   - imread
   - vidReader

3. Finding Objects
   - Thresholding
   - ROI Segmenting

4. Getting information from objects
   - regionprops
   - Morphological Operations
There are two types of pixels in every image:

1. Pixels that are interesting
2. Pixels that are not
There are two types of pixels in every image:

1. Pixels that are interesting
2. Pixels that are not

How do we separate the two?
There are two types of pixels in every image:

1. Pixels that are interesting
2. Pixels that are not

**How do we separate the two?**

Separating an image into interesting vs uninteresting pixels is equivalent to converting the image matrix into a logical matrix.
There are two types of pixels in every image:

1. Pixels that are interesting
2. Pixels that are not

How do we separate the two?

Separating an image into interesting vs uninteresting pixels is equivalent to converting the image matrix into a logical matrix

Therefore, separating pixels can be done by coming up with a clever logical question that is:

true for interesting pixels and
false for uninteresting pixels
One of the most popular questions for separating pixels is to ask whether they are either less than or greater than some constant value.

This is known as thresholding.
One of the most popular questions for separating pixels is to ask whether they are either \textit{less than} or \textit{greater than} some constant value.

This is known as \textit{thresholding}.

To query a single number, all we need to do is:

\begin{align*}
L1 &= 5 < 15 \\
    &\rightarrow L1 = 1
\end{align*}
One of the most popular questions for separating pixels is to ask whether they are either less than or greater than some constant value.

This is known as **thresholding**.

To query a single number, all we need to do is:

\[ L1 = 5 < 15 \rightarrow L1 = 1 \]

To query each element of a matrix, we use the exact same syntax!

\[ L2 = [20, 12; 11, 15] < 15 \rightarrow L2 = [0,1;1,0] \]
One of the most popular questions for separating pixels is to ask whether they are either *less than* or *greater than* some constant value.

This is known as *thresholding*.
Test Case

Create a binary image from the first image you imported.

Then display the binary image with imshow.

Notice these imperfections.

In some cases it may be important to clean an image - we will discuss how in a few slides.
Alternatively, one can ask whether pixels are contained within some boundary

This is commonly referred to as **ROI segmentation**

Boundaries can be determined **manually** or **automatically**

For instance, one could manually select an ROI from our volleyball image:
Alternatively, one can ask whether pixels are contained within some boundary.

This is commonly referred to as **ROI segmentation**.

Boundaries can be determined **manually** or **automatically**.

For instance, one could manually select an ROI from our previous image:

But, how do we do this?
To manually select an ROI, use the *roipoly* function.

```matlab
BinaryImage = roipoly(ImageMatrix)
```
To manually select an ROI, use the `roipoly` function.

\[
\text{BinaryImage} = \text{roipoly}(\text{ImageMatrix})
\]

Variable that will store a binary matrix the same size as `ImageMatrix`, where ones correspond to pixels that were inside the ROI you drew, and zeros were outside

Function

Matrix that stores the pixel intensities of your image
To manually select an ROI, use the *roipoly* function.

\[
\text{BinaryImage} = \text{roipoly(} \text{ImageMatrix} \text{)}
\]

- **Variable** that will store a binary matrix the same size as *ImageMatrix*, where ones correspond to pixels that were inside the ROI you drew, and zeros were outside.
- **Function**
- **Matrix** that stores the pixel intensities of your image.

This will open an interactive figure window that will allow you to click points to create a polygon that surrounds your ROI.

Once you close your polygon, you right click and select *create mask* to generate the binary image that will be stored in the output variable (BinaryImage, here).
Use `roipoly` to create a mask around the indicated cell in frame 1 of your movie.

Display the resulting binary image with `imshow`.

Extract the average value of the pixels in your cell at frame 1.

Create a for loop to go through every nth frame and store the average pixel value at that frame into the nth element of a vector.

Plot these values.
1. What are images?
   ○ Grayscale
   ○ Color
   ○ Movies

2. Getting images into (and out of) MATLAB
   ○ imread
   ○ vidReader

3. Finding Objects
   ○ Thresholding
   ○ ROI Segmenting

4. Getting information from objects
   ○ regionprops
   ○ Morphological Operations
Now that we have identified pixels that are part of an interesting object(s), we must decide what kind of information we would like to extract.

This depends on what our initial goal was.

If we set out to track a moving organism, such as a fish, or paramecia, then we would want to find parameters of position, such as the location of objects within our image or their angular orientations.

Alternatively, if we wanted to generate a time course of changes in a cell’s fluorescence, then we would want to find the average value of all the pixels in our object at each frame.
To extract information from Binary Objects, use the `regionprops` function.

\[
\text{PropertyStructure} = \text{regionprops(BinaryImage, \text{"Property1"})}
\]

| Variable that will store a binary matrix the same size as ImageMatrix, where ones correspond to pixels that were inside the ROI you drew, and zeros were outside | Function | Matrix that stores the binary status of your each pixel | A string that identifies which property you would like to extract for each object |
Suppose our BinaryMatrix is:

![BinaryMatrix Image]

We want to extract the Area and Centroid of all objects in the BinaryMatrix inside of a Data Structure called Property Structure.

PropertyStructure = regionprops(BinaryImage, 'Area', 'Centroid')

What is PropertyStructure?
PropertyStructure(1)
  PropertyStructure(1).Area: 1
  PropertyStructure(1).Centroid: [2, 2]

PropertyStructure(2)
  PropertyStructure(2).Area: 9
  PropertyStructure(2).Centroid: [3, 6]
Use `regionprops` to generate a structure that contains the centroids of all of the objects from the binary image of your fish.

*Note, you will need to use a single color dimension*

How many objects did `regionprops` detect?

There are four *real* objects, yet `regionprops` found 6?

How do we get rid of imposters?
We need to clean our image

To do this we will use *Morphological Operations* a type of *Image Convolution*

Morphological operations are a family of functions that change the value of each pixel based on the values of the surrounding *neighborhood* of pixels
We can use the function $imopen$ to remove Object pixels that aren’t inside a neighborhood of other Object pixels.
Or we can use the function `imclose` to connect proximal but separated object pixels.
Before we can use a morphological operation, however, we must define the neighborhood that will be used to determine the fate of every pixel.

To do this, use the `strel` function:

\[
\text{Neighborhood} = \text{strel}('\text{Shape}', \text{Size})
\]

- **Variable that will contain the object that defines your pixel neighborhood for `imclose`**
- **Function**
- **String that defines the shape of your neighborhood (i.e. 'square' or 'disk')**
- **Number or variable that indicates the size of your neighborhood**

For example, \(\text{Neighborhood} = \text{strel}('\text{square}', 3)\) means that we will examine the pixels in a 3x3 matrix around a given pixel \(X\) to determine pixel \(X\)'s fate.
### CleanMatrix = imopen(BinaryImage, Neighborhood)

| Variable that will store another binary matrix, this time without the noise |
| Function |
| Matrix that contains our binary values |
| Variable where neighborhood information is stored |

### CleanMatrix = imclose(BinaryImage, Neighborhood)

| Variable that will store another binary matrix, with disconnected points connected! |
| Function |
| Matrix that contains our binary values |
| Variable where neighborhood information is stored |
Test Case

Clean your image of a fish using a 3x3 square neighborhood – how many objects will you have now?