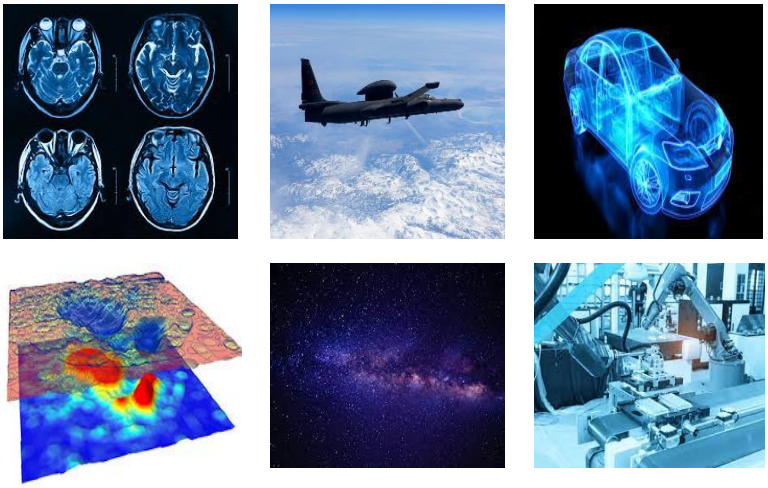
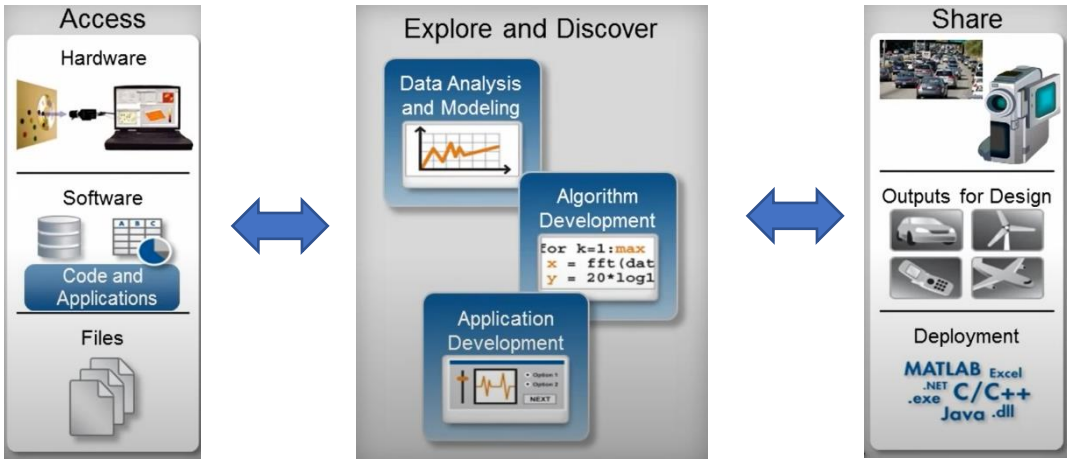
Lecture 10: Image Processing in MATLAB

1. Applications: Image Processing

- Medical imaging
- Surveillance
- Robotics
- Automotive safety
- Consumer electronics
- Geospatial computing
- Machine vision



2. Image processing workflow



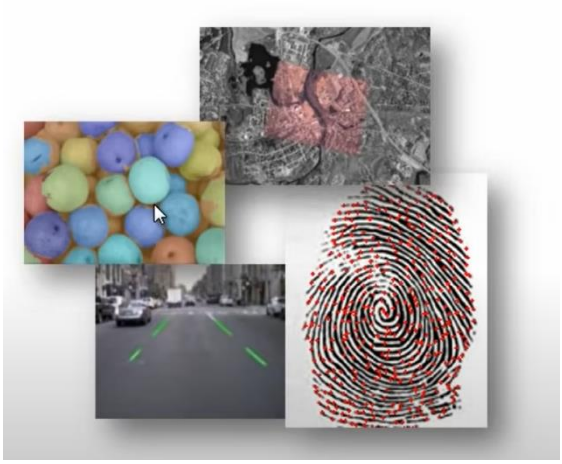
3. Common Image Processing Challenges

- Reading and writing to various file formats
- Create and test algorithms with what-if scenarios
- Identifying causes of algorithm failure
- Visualizing images and intermediate results
- Processing large images with limited memory
- Executing algorithms faster

4. Image Processing Toolbox

Perform image processing, analysis, visualization, and algorithm

- Image display and exploration
- Image enhancement
- Image analysis
- Morphological operations
- Image registration
- Geometric transformation
- ROI-based processing



5. Image Enhancement

What is image enhancement? Pre- and Post-Processing

Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further processing.

- Noise removal
- Deblurring
- Filtering

Example: In this example, we'll walk through a typical image processing workflow. We'll use MATLAB and Image Processing Toolbox to analyze deforestation in the Amazon rainforest.

Year 2000 Year 2004 Year 2008 Year 2012 Year 2016

Identify the deforested regions in each image, and then calculate their area in pixels. We can then use this scale to convert the area from pixels to square kilometers.

1. Import the one from 2000. When you bring the image into the workspace, the name of the variable will match the name of the file by default or using the command:

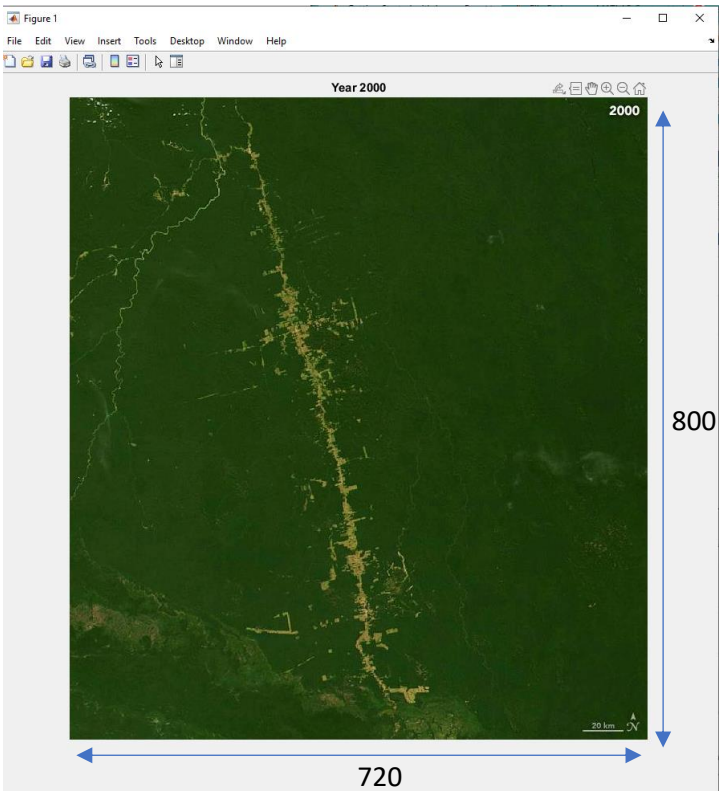
```
year2000 = imread("year2000.jpg");
```

2. To display an image in MATLAB, you can use the command `imshow:`

```
imshow(year2000)
```

3. Add a simple title to our image.

```
title("Year 2000")
```



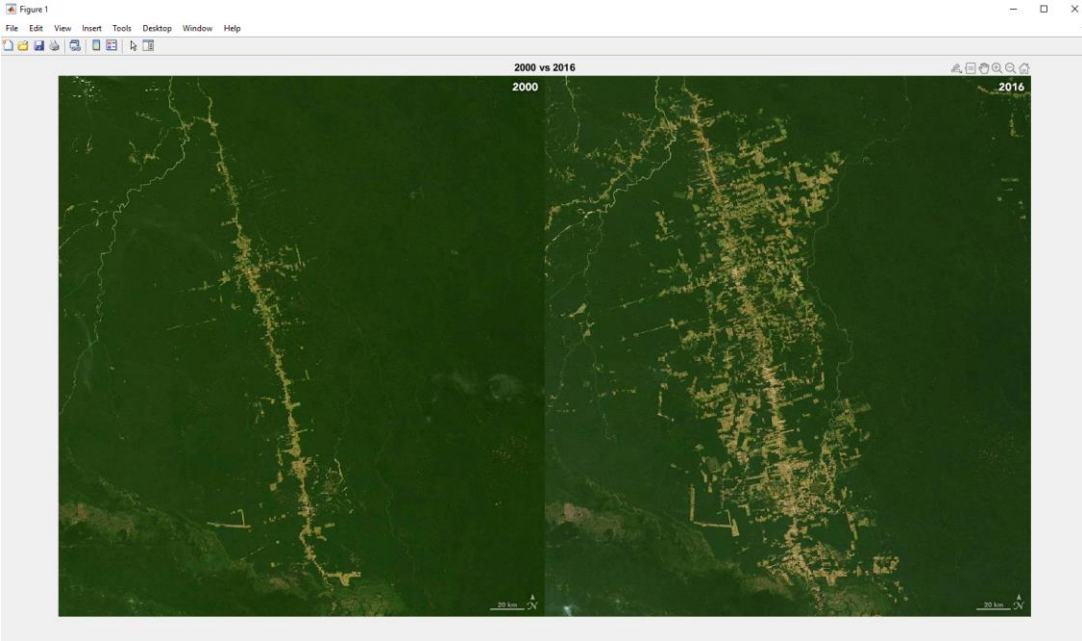
This image has a height of 800 pixels and a width of 720 pixels. In MATLAB, this means we have an array with 800 rows and 720 columns. Color images are 3D arrays because we need to store the red, green, and blue values for each pixel. These values are integers that range from 0 to 255.

4. Let's compare 2000 to 2016. You can use the function `imread` to directly import images into the workspace. To display these images together, use `imshowpair` with the montage option.

```
year2016 = imread("year2016.jpg");
imshowpair(year2000,year2016,"montage")
```

5. Add a simple title to our images.

```
title("2000 vs 2016")
```



We can already see that the deforested area is larger in 2016.

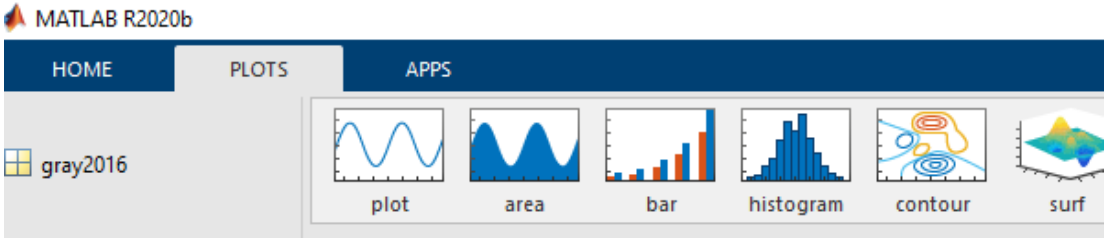
6. To compare the two more accurately, we'll need to isolate the deforested regions. But to compare the two pictures more accurately, we'll need to isolate the deforested regions. In image processing, this is called **segmentation**. The deforested regions appear to be brighter. So we can use this brightness or intensity for segmentation.
   It's easier to visualize intensities in a grayscale image using `im2gray` to convert an image to grayscale. Grayscale images are 2D arrays because we only need one value, the intensity for each pixel.

```
gray2016 = im2gray(year2016);
```

One way to examine these intensities is to create an intensity histogram.

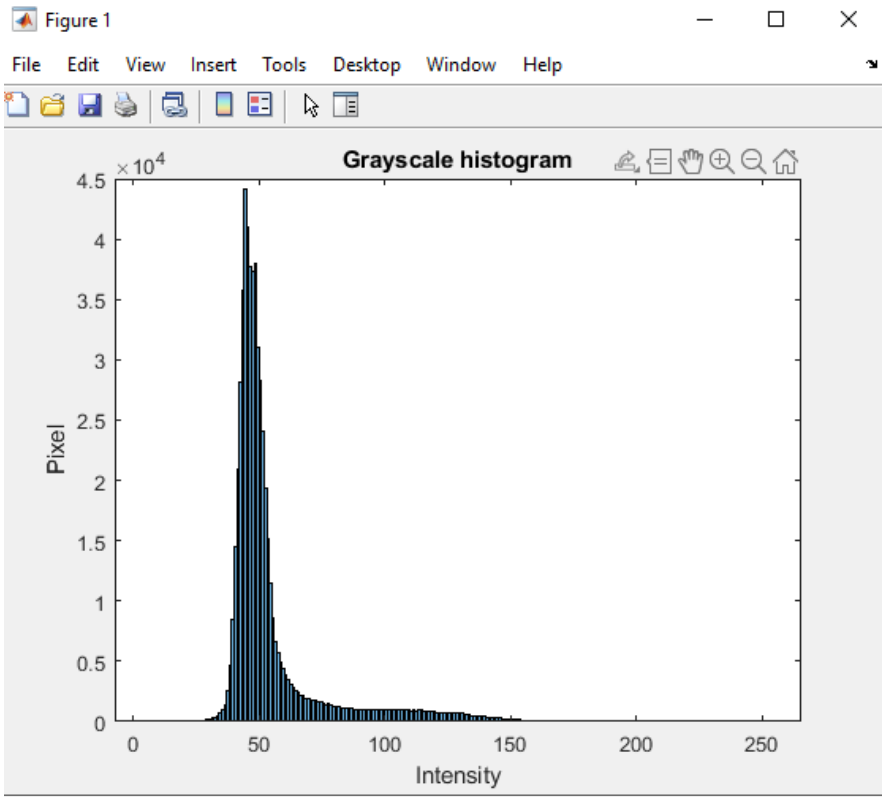
```
imshow(gray2016)
title("2016 in grayscale")
```

Or we can use the PLOT > Histogram to get the histogram of the specified picture.



7. To add a title and axis labels for the histogram, we can use this command

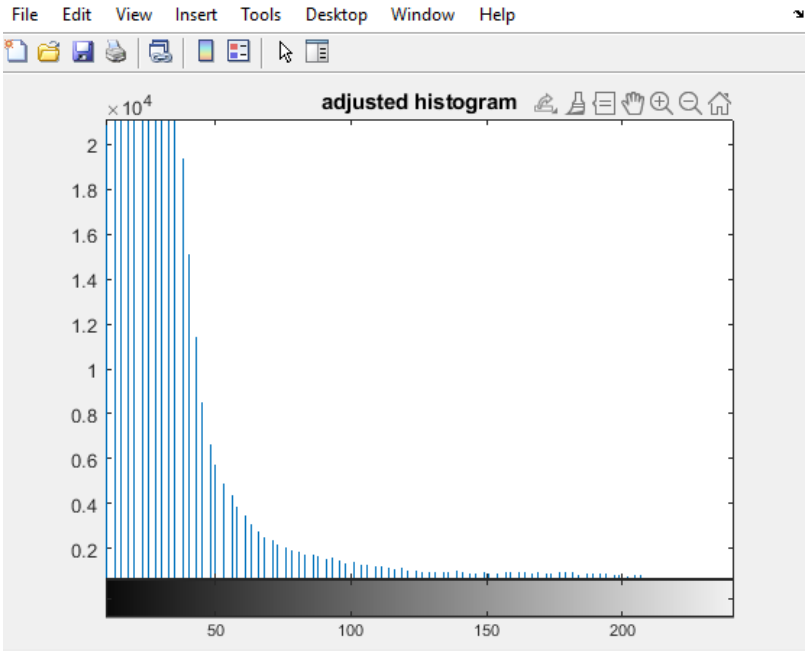
```
title("Grayscale histogram")
xlabel('Intensity')
ylabel('Pixel')
```



8. Most of the intensities are clustered on the left side, making it hard to isolate the bright, deforested regions from the darker surrounding areas. We'll want to adjust this histogram. So we use more pixels that are darker and brighter, which should increase the contrast of the image.

   The function `imadjust` spreads the pixels out to match the intensity range better while still maintaining the original shape. Comparing the original and adjusted images, the deforested re gions are brighter and more prominent. Now, we need to decide which pixels belong to the deforested areas. One way is to threshold the intensity values.

```
adj2016 = imadjust(gray2016);
imhist(adj2016)
title("Adjusted histogram")
```



9. Comparing the original and adjusted images, the deforested regions are brighter and more prominent. Now, we need to decide which pixels belong to the deforested areas. One way is to threshold the intensity values.

```
imshowpair(gray2016,adj2016,"montage")
title("Before and after adjustment")
```