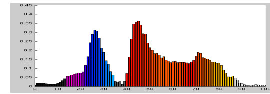


Digital Image Processing (750474)

Lecture 5

Outline of the Lecture

- **Image Types**
- **Converting between data classes and image types**
- **Converting images using IPT Function**
- **Matlab image Arithmetic Functions**
- **Array indexing**

Image Types

- The toolbox supports **four** types of images:
  - Intensity Image.
  - Binary Images.
  - Indexed Images.
  - RGB Images.

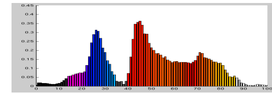
Intensity Images: (Gray scale Images)

- An intensity image is a **data matrix** whose values have been scaled to represent intensities.

Allowed Classes	Range
uint8	0 - 255
uint16	0 - 65535
double	[0 - 1]

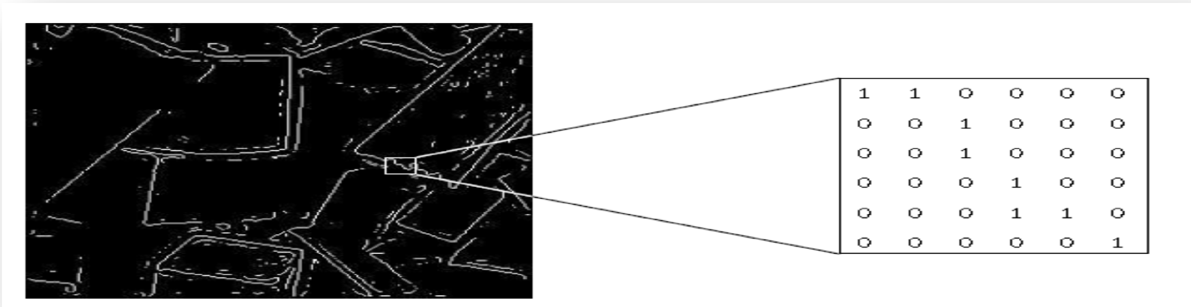
230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

Intensity Images



Binary Images:

- **Logical** array containing only **0s** and **1s**, interpreted as **black** and **white**, respectively. In matlab, by convention, BW is a variable Binary image.



Binary Images

- If the array contains 0s and 1s whose values are of data class **different from logical** (for example uint8), it is not considered a binary image in Matlab.

Conversion a numeric array to binary:

1. To convert, we use function logical.

```
>> x = [ 0   1   1   0   1   0];
>> y = logical (x);
```

If **x** contains other than **0s** and **1s**, the logical function **converts all nonzero values to logical 1s.**

2. Using relational and logical operators we can create a logical array.

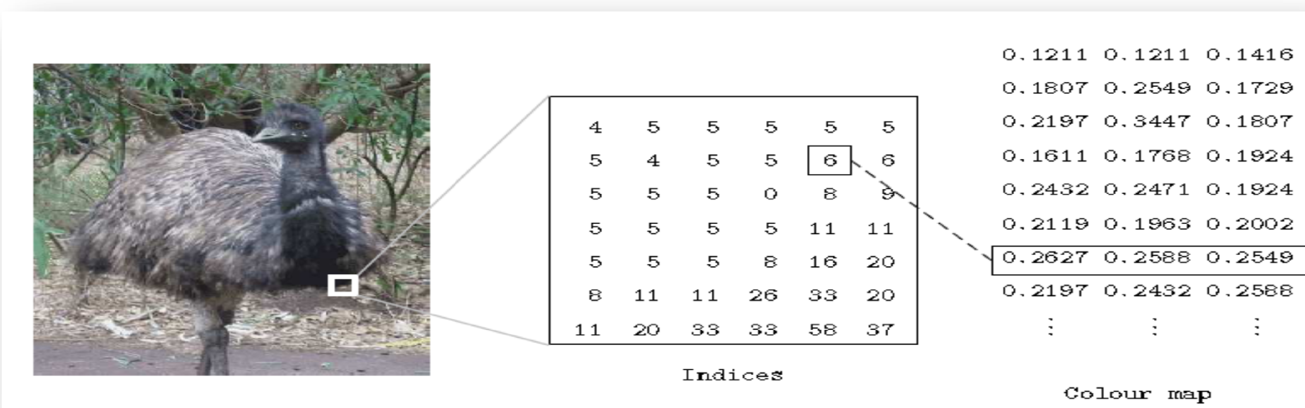
- To test if an array is logical, we use islogical function:

```
>> islogical (y);% returns 1 if y is a logical array; otherwise it returns 0;
```

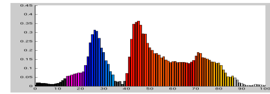
- **logical** array can be converted to numeric arrays using the data class conversion functions.

Indexed Image

- An indexed image consists of an **array** and a **colormap matrix**.
  - The pixel values in an array are **direct indices** into a colormap.
  - Each pixel has a value which does not give its **color** (as for an RGB image), but an **index** to the color in the map.



Indexed Image



By convention in matlab:

- Variable X refer to array, variable map refer to the colormap.
- The array of class logical, uint8, uint16, single or double.
- The colormap matrix is an m-by-3 array of class double (values in [0 1] range).
- Each row of map specifies the red, green and blue components of a single color.
- The color of each image pixel is determined by using corresponding value of X as an index into map.
- A colormap is often stored with an indexed image and is automatically loaded with image when using imread function.

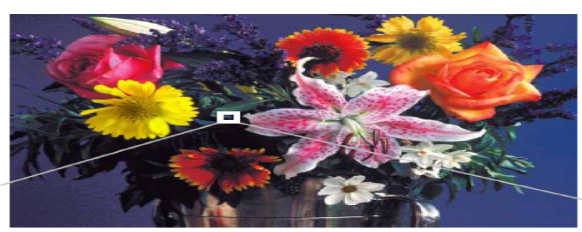
Relationship between values in the image matrix and the colormap	
class	**Range of colormap**
single, double	**1 through p, p is the length of the colormap, Value 1- first row, 2 second, etc**
logical, uint8 or uint16	**value 0 points to the first row, value 1 points to the second, and so on**

RGB color Image (true color image)

- True color images, require a **three-dimensional array** (m-by-n-by-3) of class uint8, uint16, single or double whose pixel values specify intensity values.

class	**range**
single, double	**[0,1]**
uint8	**[0 255]**
uint16	**[0 65535]**

- Unlike an indexed image, however, these intensity values are stored directly in the image array, not indirectly in a colormap.
- m and n are the numbers of **rows** and **columns** of pixels in the image, and the **third dimension** consists of **three planes**, containing red, green, and blue intensity values.
- For example**:** to determine the color of the pixel **(112, 86)**
    - Look at the RGB triplet stored in **(112, 86, 1:3).** Suppose (112, 86, 1) contains the value 0.1238, (112, 86, 2) contains 0.9874 and (112, 86, 3) contains 0.2543
        - The color of the pixel at (112, 86) is: 0.1238   0.9874   0.2543



49	55	56	57	52	53
58	60	60	58	55	57
58	58	54	53	55	56
83	78	72	69	68	69
88	91	91	84	83	82
69	76	83	78	76	75
61	69	73	78	76	76

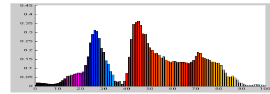
Red

64	76	82	79	78	78
93	93	91	91	86	86
88	82	88	90	88	89
125	119	113	108	111	110
137	136	132	128	126	120
105	108	114	114	118	113
96	103	112	108	111	107

Green

66	80	77	80	87	77
81	93	96	99	86	85
83	83	91	94	92	88
135	128	126	112	107	106
141	129	129	117	115	101
95	99	109	108	112	109
84	93	107	101	105	102

Blue



Converting between data classes and image types

- Matlab expects operands in numeric **computation** to be of double.
- When you **store** an image, you should store it as uint8 image, since this requires far less memory than double.
- When you are **processing** an image, you should **convert it to double**, to convert, we use 2 methods:

1) Type casting

✓ **Convert from one data type to another:** B= data_class_name (A)

Example (1):

```
>> B = double (A)
```

Example (2):

```
>> D = uint8 (c);
```

- If **c** is an array of class **double**, in which all values are **[0 – 255]** (possible fractional value).
- If an array of class **double** has any values **outside** the range **[0 255],** matlab converts to **0** all values that are **less than 0**, and converts to **255** all values that are **greater than 255.**
- Numbers **in between** are converted to integers by **discarding their fractional parts**.

✓ **Converting any of the numeric data classes to logical**

- Results in an array with logical 1s in location where the input array has **nonzero** values and logical 0s where the input array contains 0s.

2) Converting between image classes and types.

✓ Perform necessary **scaling** to convert between image classes and types.

a) im2uint8 (x) detects input data class and scales to allow recognition of data as valid image data.

Example: Convert an image named x from double to uint8.

- Consider the following **2*2** image f of class **double**.

```
>> f= [-0.5        0.5; 0.75        1.5]
```

- Performing the **conversion**

```
>> g= im2uint8 (f)
```

ans
% g= 0 128
% 191 255

b) im2double (x) converts **x** input to class **double** in range **[0 1]**, unless input is of class **double**, **no effect**.

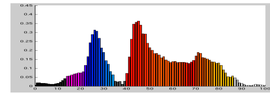
Example: Consider the class uint8 image.

```
>> h = uint8 ([25      50; 128     200]);
```

- Performing the **conversion**

```
>> g = im2double (h);
```

ans
g= 0.0980 0.1961
0.4706 0.7843



c) mat2gray (x, [Amin Amax]) takes arbitrary **double** array input and scaled to range **[0 1].**

- **Values < Amin** function converts them to **zero**.
- **Values < Amax** function converts them to **1**.
  - Convert an **arbitrary array of double** to an array of class **double scaled to the range [0 1]**.

```
>> mat2gray (x)  % sets the values of Amin and Amax to the actual
                 %minimum values in x.
```

d) im2bw (x, T) converts **intensity image** (input matrix) to a **binary image**, anything less than T output set to **0**, otherwise output set to 1.

- T = [0, 1]
- Output is logical.

```
>> Im2bw (x)  % T = 0.5 (default).
```

Example: Convert the following double image **f= [1 2; 3 4]** to binary such that values 1 and 2 become 0 and the other two values become 1.

Solution:

- First we convert it to the range **[0 1]**

```
>> g= mat2gray (f)
      ans
      g= 0        0.3333
         0.6667   1.000
```

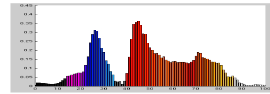
- Then we convert it to **binary** using a **threshold (0.6)**

```
>> gb= im2bw (g, 0.6)
      ans
      gb= 0     0
          1     1
```

Converting images using IPT Functions

Converting images using IPT Functions

Matlab Command	operations
dither (.)	**Gray scale to Binary images.**
	RGB to Indexed images.
gray2ind (.)	**Intensity to indexed images.**
ind2gray (.)	**Indexed to intensity images.**
ind2rgb ( )	**indexed to RGB images.**
mat2gray ( )	**Regular matrix to intensity images, create a gray scale intensity image from data in a matrix by scaling the data.**
rgb2gray ( )	**RGB to intensity images.**
rgb2ind ( )	**RGB to indexed images.**
im2bw ( )	**intensity to binary images.**



Examples:

```
>> y = ind2gray(x,map);
>> [y,map] = gray2ind(x);
>> y = rgb2gray(x);
>> y = gray2rgb(x);
>> [y,map] = rgb2ind;
>> y = ind2rgb(x,map);
```

Matlab image Arithmetic Functions

Matlab image Arithmetic Functions		
Toolbox (IPT)	**Matlab**	**Description**
imadd (A,B)	**A+B**	**Adding two images**
imsubtract (A,B)	**A-B**	**Subtracting two images**
immultiply (A,B)	**A.*B**	**Multiply two images**
imdivide (A,B)	**A./B**	**Divide two images (the values are rounded to the nearest integer, not truncated like true integer arithmetic)**

Example: Reading a true color image into Matlab:

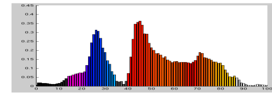
```
>> I = imread ('football jpg');
>> class (I) % uint8
>> size (I) % 250   320   3
>> figure
>> image (I) ;
>> title ('some title');
>> xlabel ('some text');
>> i(231,100,:)
        % ans (:, :, 1) = 48
        % ans (:, :, 2) = 37
        % ans (:, :, 3) =41
>> i= double (i) /255;
>> i (231, 100, :)
        % ans (:, :, 1) = 0.1882
        % ans (:, :, 2) = 0.1451
        %ans (:, :, 3) = 0.1608
>> class (i) % double
```

Array indexing

a) vector indexing

Examples:

```
>> v = [ 1  3  5  7  9] % row vector declaration
>> v(2)
% access the second element of the v.
>> w = v'
```



```
% row vector is converted to a column vector using
% the transpose operator (')
>> v(1:3)
% To access blocks of elements, we use matlab's colon notation.
% Access first three elements of v.
>> v(2:4)  % access the second through the fourth.
>> v(3:end)  % all element from third to the last.
>> v(:)  % produce a column vector.
>> v(1: end)  % produce a row vector.
>> v(1:2:end)
% start at 1, count up by 2 and stop when the count reaches the last.
>> v(end:-2:1)
% started at last, decreased by 2, and stopped at the first element.
```

b) Matrix indexing:

```
>> A = [1 2 3; 4 5 6; 7 8 9]  % 3*3 matrix declaration.
>> A(2, 3)  %  access element in a matrix (2-row, 3-column).
>> C = A (:, 3)  % (all rows, third columns).
>> T = A (1:2, 1:3)  % extract the top two rows.
>> A(end, end)  % Get the last element (last row, lost column).
>> E = A ([1 3], [2 3]); % using vectors to index into a matrix.
```