"Data classes"

The data classes supported by MatLab and IPT for representing pixel values in the following Table:

type	class Name	Description	Range	Bytes/ per element
numeric	double	floating-point numbers	$(+10^{308} - 10^{308})$	8
	uint 8	Unsigned 8-bit integers	(0 - 255)	1
	uint 16	Unsigned 16-bit integers	(0 - 65535)	2
	uint 32	unsigned 32-bit integers	(0 - 4294967295)	4
	int 8	signed 8-bit integers	(-128 - 127)	1
	int16	signed 16-bit integers	(-32768, 32767)	2
	int32	signed 32-bit integers	(-2147483648 - 2147483647)	4
	Single	floating point numbers	$-10^{38} - 10^{38}$	4
Character	Char	Characters, Unicode	.	2
Boolean	logical	Logical Data	0 or 1	1

The frequently Data classes that encountered in image processing are double, uint 8 and logical.

* logical arrays are created by using function Logical or by using relational operators.

Image Types

the toolbox supports four types of images:

- Intensity Images
- Binary Images
- Indexed Images
- RGB Images

• Intensity Images: (Grayscale Images)

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

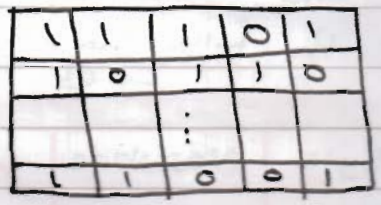
allowed class	range
uint 8	0 - 255
uint 16	0 - 65535
double	[0 - 1]

• Binary Images:

logical array containing only 0s and 1s, interpreted as black and white, respectively.

in Matlab, by convention,

BW is a variable for Binary image.



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:

To convert, we use function logical.

```
>> A = [0 1 0 1 1 1];
>> B = logical(A);
```

if A contains other than 0s and 1s. The logical function converts all nonzero values to logical 1s.

- Using relational and logical operators we can create a logical array.
- To test if an array is logical, we use **islogical** function:

```
>> islogical(B); % returns 1 → if B 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.

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 cooresponding 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.
" you can use any colormap that you choose, not only the default "

- Relationship between values in the image matrix and the colormap :

class	range of colormap
Single, double	1 throug P, P is the length of the colormap, value 1 - first row, 2 second, ...
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)

TrueColor 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]

RGB Array Data Types

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

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 require 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)

Convert the Array A to type of data_class_name

Example ①. B = double (A)

Example ②

D = uint8 (C);

a- if c is an array of class double, in which all values are [0-255] (possible fractional value)

b- 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 convert to 255 all values that are greater than 255.

c- 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.

" Convert an image named X from double to uint8.

Example 1: ⇒ 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) %      g =    0    128
                               191   255
```

b) im2double(X)

Convert X input to class double in range [0,1], Unless input is of class double, no effect.

" Convert an image named X from uint8 to double.

Example: consider the class uint8 image.

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

Performing the conversion.

```
» g = im2double(h);
      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 ⇒ convert to zero.
- Values > Amax ⇒ " " 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 and maximum values in X.

d) im2bw (X, T)

- Convert 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)
    g =    0        0.3333
           0.6667   1.0000
```

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

```
» gb = im2bw ( g , 0.6)   ⇒
    gb =   0   0
           1   1
```

Converting Images

Matlab Command	operation
dither(·)	. Binary from grayscale . Indexed from RGB
gray2ind(·)	convert between intensity format to indexed format.
ind2gray(·)	convert between indexed to intensity
ind2rgb	indexed → rgb
mat2gray()	regular matrix → intensity, Create a grayscale intensity image from data in a matrix by scaling the data.
rgb2gray()	RGB to intensity
rgb2ind()	RGB to indexed
im2bw	intensity to binary

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 x 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, last Column)
>> E = A([1 3],[2 3]); % using vectors to index
                     into a matrix.
```