School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

APPM4058A: Digital Image Processing
Exercise 1

2019-2-6

1 Objectives

- Perform gray level transformation for an image
- Design simple gray level transformation functions given a gray level transformation graph

2 Image processing using Matlab

2.1 Digital image representation

2.2 Reading images

- Matlab: `imread('filename');`, Here, 'filename' is a string containing the complete name of the image file (including the extension). The semicolon at the end of a statement is used for suppressing output.
- `size(f);` or `[M, N] = size(f)` gives the row (M) and column (N) dimensions of an image. Similarly `M = size(f,1);` or `N=size(f,2)` gives the row or column dimensions.
- `whos f` gives additional information about variable `f`.

2.3 Displaying images

- `imshow(f);`
- `imshow(f, [low, high]);` displays as black all values less than or equal to `low`, as white all values greater than or equal to `high`
- `imshow(f, []);` sets variable `low` to the minimum value of array `f` and `high` to its maximum.
- `figure, imshow(f);` displays the image in a new window.

2.4 Writing images

- `imwrite(f, 'filename');` With this syntax, the string contained in filename must include a recognized file format extension.

2.5 Classes

- `double`
- `single`
- `uint8`, `int8`

- `uint16`,`int16`
- `uint32`, `int32`
- `char`
- `logical`

Conversion between classes `B = class_name(A)` or using the following functions.

- `im2uint8` converts input to `uint8`
- `im2uint16`
- `im2double`
- `im2single`
- `mat2gray` converts input to double in the range $[0,1]$
- `im2bw` converts to logical

For example,

```
im = imread('cameraman.png');
>> whos im
Name        Size             Bytes  Class    Attributes

im        256x256            65536  uint8

>> B = double(im);
>> whos B
Name        Size              Bytes  Class     Attributes

B         256x256            524288  double
```

To convert `im` to a double in the range of $[0,1]$, we can use `mat2gray` function.

2.6 Array indexing

- Matlab indexing starts from 1 (not 0).
- `v = [2 4 6 8 10]`

  `v` is a row vector. `w = v.'` gives a column vector (vector transpose); `v(1:3)` gives a range of elements in `v`; Similarly, `v(3:end)`, here `end` indicates the last element in the vector.
- `A=[1 2 3; 4 5 6; 7 8 9]` gives a 3×3 matrix. `A(2, 4)` extracts the element in the 2nd row, 4th column of `A`.
- `T = A(1:3, 1:2])` extracts a sub matrix of `A`. `A(:, 3)` extracts the entire 3rd column of `A`. Find out what will `V = A(:)` do. `sub2ind` and `ind2sub` convert back and forth between row-column subscripts and linear indices.

  ```
  H = hilb(4)

  H =

  1.0000    0.5000    0.3333    0.2500
  0.5000    0.3333    0.2500    0.2000
  0.3333    0.2500    0.2000    0.1667
  0.2500    0.2000    0.1667    0.1429
  ```

```
>> linear_indices = sub2ind(size(H), [1 2 4], [3 4 3])

linear_indices =

9    14    12

>> H(linear_indices)

ans =

0.3333    0.2000    0.1667

>> [r, c]=ind2sub(size(H), linear_indices)

r =

1     2     4


c =

3     4     3
```

2.7 Arithmetic operators

Matlab has two different types of arithmetic operations: matrix arithmetic operations are defined by the rules of linear algebra; and array arithmetic operations are carried out element wise. The period (dot) character (.) distinguishes array operations from matrix operations. For example, $A*B$ indicates matrix multiplication in the traditional sense, whereas $A.*B$ indicates array multiplication, where the output array is the same size as A and B, and each element in the output array is the product of corresponding elements of A and B. That is, $C = A.*B$, then $C(I,J) = A(I,J)*B(I,J)$. For example,

$$A = \begin{bmatrix} a1 & a2 \\ a3 & a4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b1 & b2 \\ b3 & b4 \end{bmatrix} \tag{1}$$

The array product of A and B gives the result

$$A.*B = \begin{bmatrix} a1b1 & a2b2 \\ a3b3 & a4b4 \end{bmatrix} \tag{2}$$

2.8 Relational operators

Relational operators in Matlab are array operators that compare corresponding pairs of elements in arrays of equal dimensions. For example,

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =

1      2      3
4      5      6
7      8      9

>> B = [0 2 4; 3 5 6; 3 4 9]
```

```
B =

0     2     4
3     5     6
3     4     9

>> A == B

ans =

0     1     0
0     1     1
0     0     1

>> A >= B

ans =

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

We see that these operators produce logical arrays with 1s indicating true, and 0s indicating false.

2.9 Flow control

The flow control statements in Matlab are similar to the

1. The 'if ... end' structure

   - if ... end
   - if ... else ... end
   - if ... elseif ... else ... end

   For example, we want to write a function that computes the average intensity of an image. We also want our function to be able to work with both vector and image inputs.

   ```
   function av = avearge(A)
   %AVERAGE Computes the average value of an array.
   % AV = AVERAGE(A) computes the average value of input A,
   % which must be a 1-D or 2-D array.

   % Check the validity of the input
   if ndims(A) > 2
     error('The dimensions of the input cannot exceed 2.')
   end

   % Compute the average
   av = sum(A(:))/length(A(:));
   ```

2. The 'for ... end' loop. For example,

   ```
   count = 0;
   ```

```
for k = 0:2:20
  count = count + 1;
end
```

The 2 in `0:2:20` is the loop increment. If it is omitted, i.e., `0:20`, it is taken to be 1.

3. The ’while … end’ loop. For example,

```
a = 10;
b = 5;
while a
  a = a-1;
  while b
    b = b-1;
  end
end
```

4. The ’switch’ statement for example,

```
switch newclass
  case ’uint8’
    g = im2uint8(f);
  case ’uint16’
    g  = im2uint16(f);
  case ’double’
    g = im2double(f);
  otherwise
    error(’Unknown or incoreect image calss.’)
end
```

2.10 Matlab M-files

Matlab M-files can be scripts of statements, or functions that can accept input arguments, or can output one or more outputs. An example M-file is provided in the `exercise1` folder.

3 Image manipulation and processing using Python

3.1 Basic image manipulation

- Opening and writing to image files

```
import matplotlib.pyplot as plt
img = plt.imread(’cameraman.png’)
img.dtype
img.shape
plt.gray()
plt.imshow(img)
plt.show()
plt.axis("off")
```

- `plt.axis("off")` removes the axis and the ticks
- `img.dtype` gives the type of the image
- `img.shape` gives the size of the image

-
  ```
  from scipy import misc
  img = misc.imread('cameraman.png')
  type(img)
  img.shape, img.dtype
  ```
- `img.max()`, `img.min()`, `img.mean()` gives maximum, minimum, and mean values (statistical information) in image `img`.
- `numpy` functions `flipud()`, `rotate(img, deg)` gives flipping upside down and rotation (geometrical transformation) of `deg` degree.

4 Problems

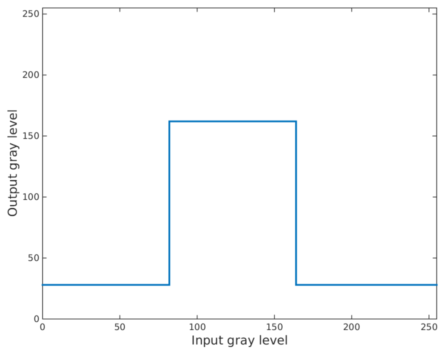
1. Suppose that in a binary image you have k white pixels in a total of N pixels. Find the mean and variance of the image.
2. Under what circumstances would you perform a contrast stretching for an image?
3. What is the full dynamic range of an N-bit gray scale image? Implement a transformation that reverses, i.e., 0 becomes L-1, 1 becomes L-2, and so on, the intensity levels of an image (L is the total number of gray levels in the image). Test your implementation for image 'pollen'.
4. The image shown in Table 1 has pixels with dynamic range 0 to 15.
   (a) Find the brightness of the image and use it to identify two pixels as outliers.
   (b) Dealing with the outliers, apply a contrast stretch to the image.

2	5	7	2
3	2	14	5
6	4	15	4
7	3	3	6

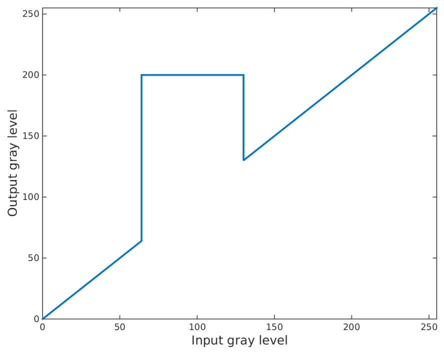
Table 1: A 4×4 gray scale image

5. Perform contrast stretching for images 'pollen', 'aerial_washedout', 'skeleton', and 'fractured_spine'.
6. Highlighting a specific range of gray levels in an image is often desired. Applications include enhancing features such as masses of water in satellite imagery and enhancing flaws in X-ray images. Two approaches often serve as the basic themes.
   - Display a high value for all gray levels in the range of interest and a low value for all other gray levels.
   - Brightens the desired range of gray levels but preserves the background and gray-level tones in the image.

   Implement the above two approaches using two transformation functions illustrated in Fig. 1. Test your implementation on image 'kidney' using different highlighting ranges.
7. Instead of highlighting gray-level ranges, highlighting the contribution made to total image appearance by specific bits might be desired. If an image is composed of 8-bit intensity levels, then bit plane 7 contains the most significant bits, and bit plane 0 contains the least significant bits. The higher order bits contain the majority of the visually significant data. The other bit planes contribute to more subtle details in the image. For example, bit plane 7 can be obtained by processing the input image with a threshold function that maps all levels in the image between 0 and 127 to one level, and maps all levels between 129 to 255 to another. In this case, the result is a binary image.



(a)



(b)

Figure 1: (a) This transformation highlights a certain range of gray levels and reduces all others to a constant level. (b) This transformation highlights a certain range but preserves all other levels.

Implement a bit-plane slicing transformation that capable of producing all the individual bit planes of an 8-bit monochrome image. (For example, a transformation function with the property $T(r) = 0$ for r in the range $[0, 127]$, and $T(r) = 255$ for r in the range $[128, 255]$ produces an image of the 7th bit plane in an 8-bit image.) Test your implementation for image 'fractal-iris.tif'.

8. *Implement a function `[fout, revertclass] = tofloat(f)` that converts an input image `f` to floating point. If `f` is a `double` or `single` image, then `fout` equals `f`. Otherwise, `fout` equals `im2single(f)`. output `revertclass` can be used to convert back to the same class as `f`. Here, the idea is to convert an input image to `single`, perform operations using `single` precision, and then convert the output image to the same class as the input.

5 A few image handling functions

- Reading images: `f=imread('filename')`
- Displaying images: `imshow(f)`
- Writing images: `imwrite(f,'filename')`

5.1 Functions imadjust and stretchlim

1. Function `imadjust` is the basic Image Processing Toolbox function for intensity transformations of gray-scale images. It has the general syntax

   ```
   g=imadjust(f,[low_in high_in],[low_out high_out],gamma)
   ```

   This function maps the intensity values in image f to new values in g, such that values between `low_in` and `high_in` map to values between `low_out` and `high_out`. Values below `low_in` and above `high_in` are clipped; that is, values below `low_in` map to `low_out`, and those above `high_in` map to `high_out`. The values for `low_in, low_out, high_in, high_out, gamma` are between 0 and 1. Parameter gamma specifies the shape of the curve that maps the intensity values in f to create g. If gamma is less than 1, the mapping is weighted toward higher (brighter) output values. If gamma is greater than 1, the mapping is weighted toward lower (darker) output values. If it is omitted from the function argument, gamma defaults to 1 (linear mapping).
2. Function `stretchlim` is used to obtain the lower and higher limit, i.e., `low_in, high_in` that can be used to achieve contrast stretching.

   ```
   Low_High=stretchlim(f)
   ```

6 Other resources

- https://rklein.me/ip/
- https://www.python-course.eu/python_image_processing.php
- http://scipy-lectures.org/advanced/image_processing/
- http://scipy-lectures.org/packages/scikit-image/index.html#scikit-image