# Image Enhancement in Spatial Domain(Point Processing)

Dr. Su Su Maung
CEIT Department
Yangon Technological University

# Outline of Lecture2

- Introduction
- Arithmetic Operations
- Logic Operations
- Histogram Stretching
- Histogram Equalization
- Histogram Matching
- Lookup Tables

# Introduction

- Any image processing operation transforms the grey values of the pixels.
- However, image processing operations may be divided into into three classes based on the information required to perform the transformation. They are:
  - Transforms
  - Neighbourhood processing
  - Point operations

# Transforms

- A transform represents the pixel values in some other, but equivalent form.
- Transforms allow for some very efficient and powerful algorithms.
- In using a transform, the entire image is processed as a single large block.

# Steps in Transform Processing

Input Image

Transform

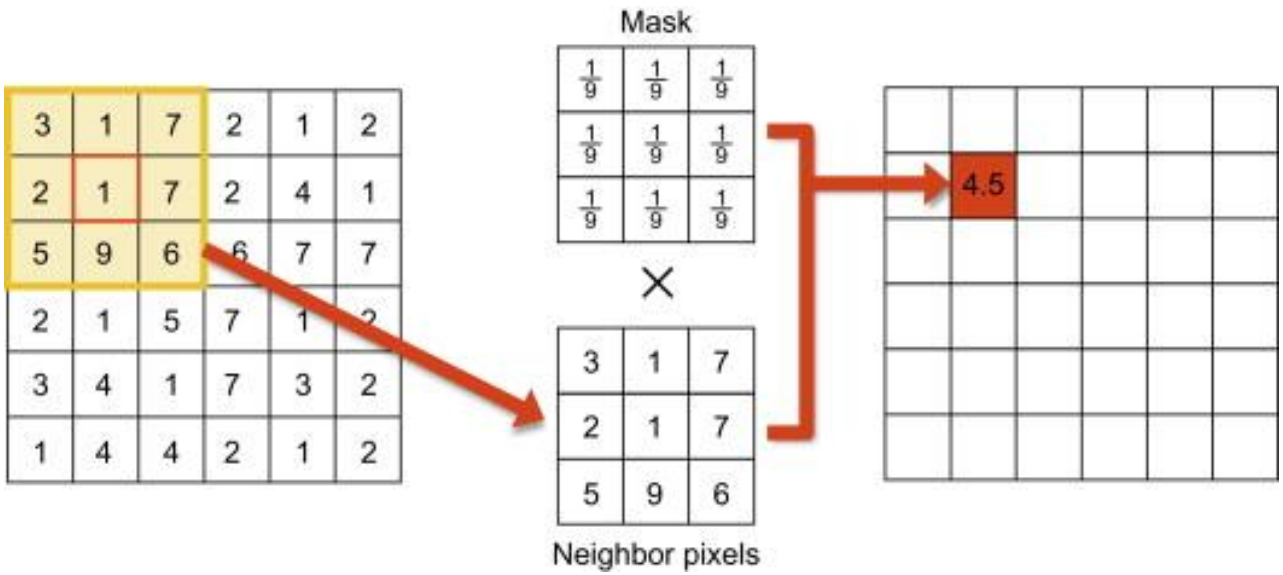
Image Processing Operation

Inverse Transform

Output Image

# Neighbourhood processing

- To change the grey level of a given pixel we need only know the value of the grey levels in a small neighbourhood of pixels around the given pixel.




Neighbourhood processing

# Point operations

- A pixel's grey value is changed without any knowledge of its surrounds.
- Although point operations are the simplest, they contain some of the most powerful and widely used of all image processing operations.
- They are especially useful in image pre-processing, where an image is required to be modified before the main job is attempted.

# Point operations

- Some point operations are
  - Arithmetic operations
  - Logic Operations
  - Histogram stretching (Contrast stretching)
  - Histogram equalization
  - Histogram Matching
  - Lookup  tables

# Arithmetic operations

- Arithmetic operations act by applying a simple function to each grey value in the image. Thus f(x) is a function which maps the range 0 …255 onto itself. Simple functions include adding or subtract a constant value to each pixel or multiplying each pixel by a constant in order to ensure that the results are integers in the 0…255 range.

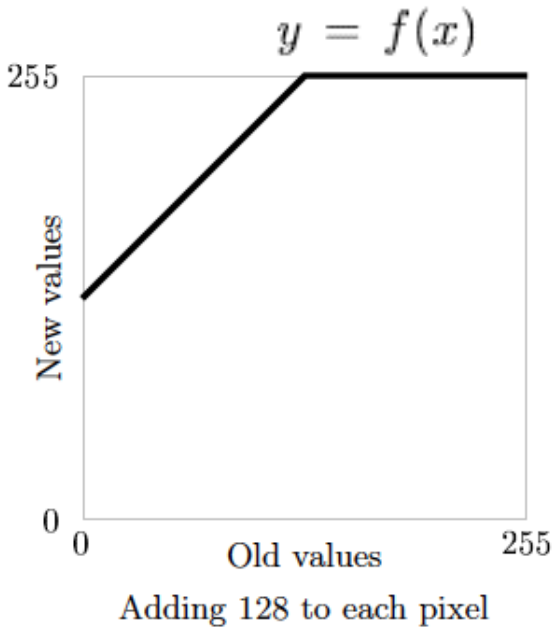
$$y = f(x), \qquad y = x \pm c, \qquad y = Cx$$

- We can do this by first rounding the result to obtain an integer, and then "clipping" the values by setting:

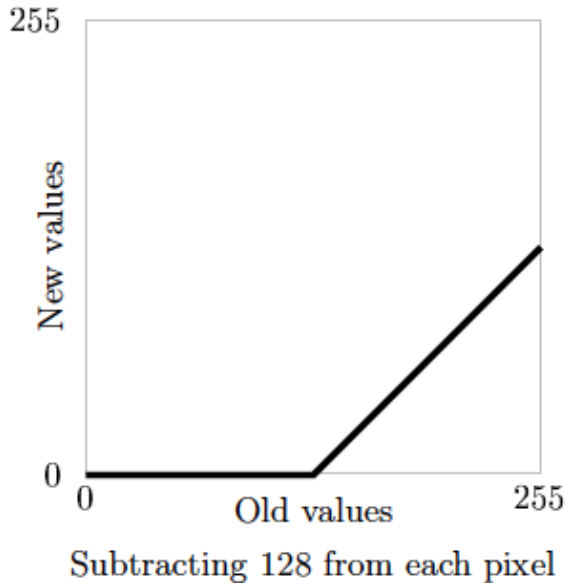
$$y \leftarrow \begin{cases} 255 & if\ y > 255 \\ 0 & if\ y < 0 \end{cases}$$

# Arithmetic operations

- in general adding a constant will lighten an image, and subtracting a constant will darken it.




Adding 128 to each pixel




Subtracting 128 from each pixel

Notice that when we add 128, all grey values of 127 or greater will be mapped to 255. And when we subtract 128, all grey values of 128 or less will be mapped to 0.

Adding and subtracting a constant

# Image Addition

Image1

Image2

Addition of two images

```
## Addition of two images##
from PIL.ImageChops  import add, subtract, multiply, difference
from PIL import Image
im1 = Image.open("images/image1.jpg")
im2 = Image.open("images/image2.jpg").convert('RGB').resize((im1.width, im1.height))
add(im1, im2).show()
```

# Image Addition (cont.)

Image1

Image2

Addition of two images

Arithmetic operations on an image: addition

# Image Addition (cont.)

Image1

Image2

Addition of two images

Arithmetic operations on an image: addition

# Implementing pixel addition/subtraction by Python commands

```
import matplotlib.pyplot as plt
import skimage.io as io
import numpy as np

img=io.imread('images/rice.tif');
plt.figure()
io.imshow(img)
img=np.float64(img)
img1=np.uint8(np.clip(img+128,0,255))
plt.figure()
io.imshow(img1)
img2=np.uint8(np.clip(img-128,0,255))
plt.figure()
io.imshow(img2)
```

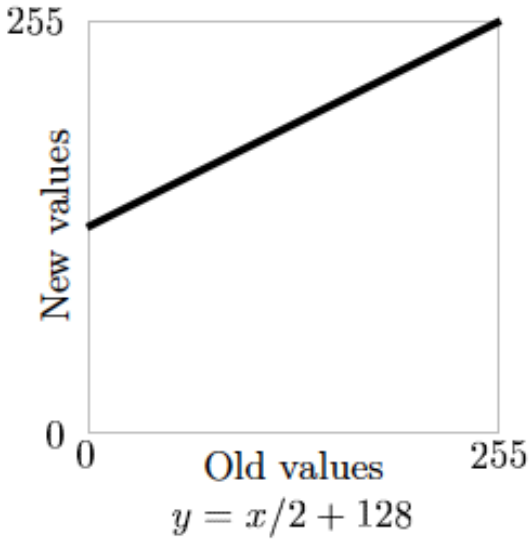
Original Image

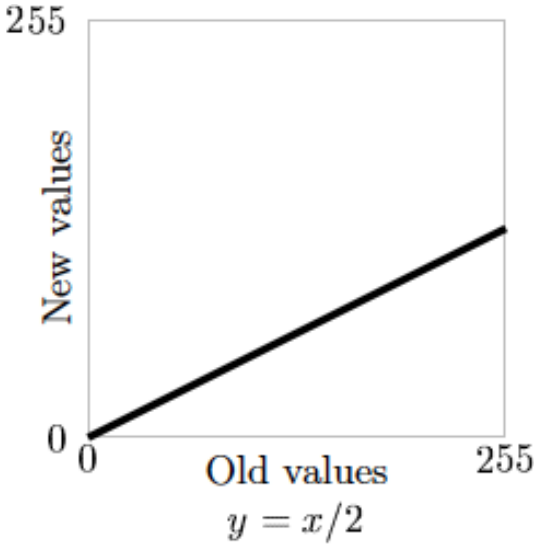
Adding 128

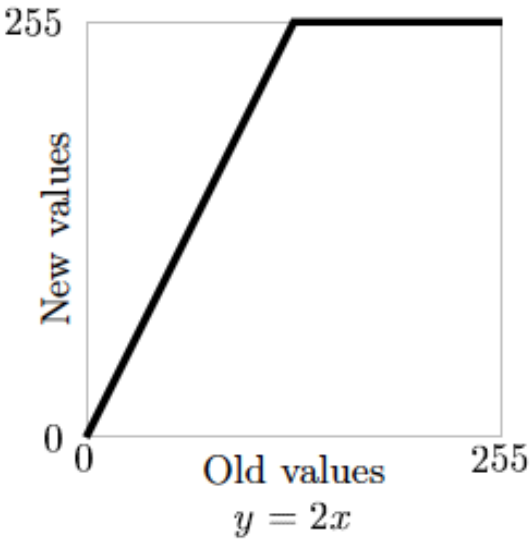
Subtracting 128

# Implementing pixel multiplication by Python commands

- Arithmetic operations on image are performed by dividing and multiplying by 2 to each pixel values.



# Implementing pixel multiplication by Python commands (cont.)

original image

y=x/2

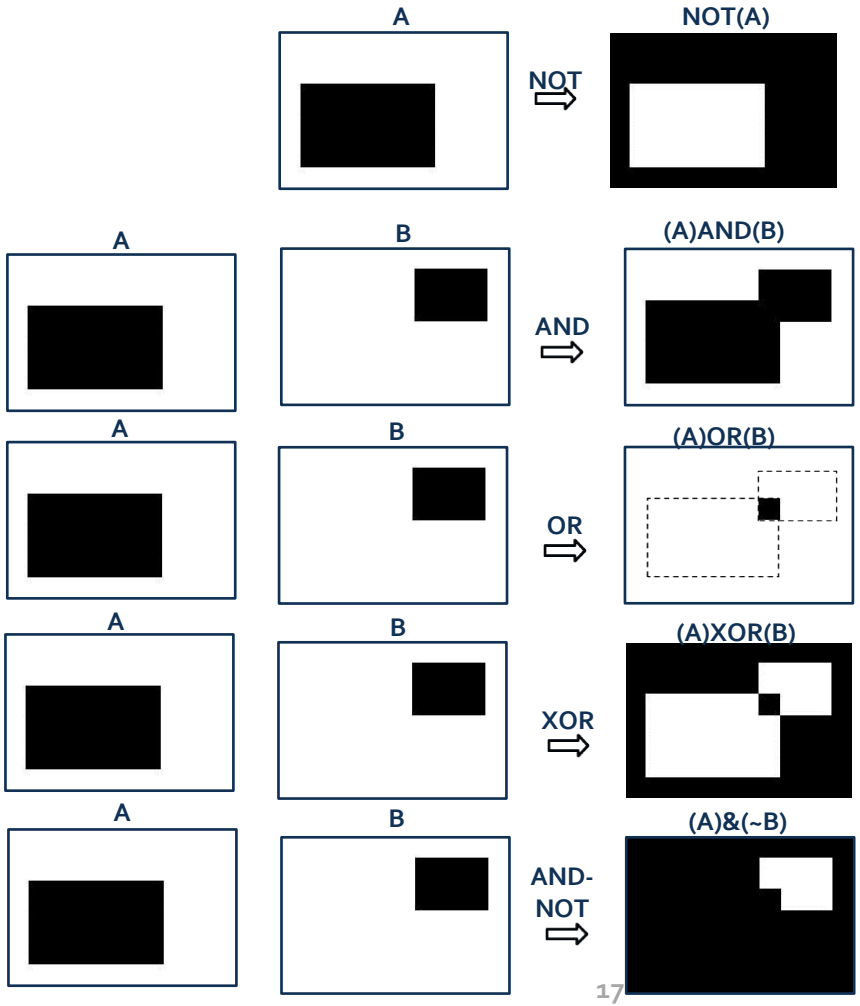
y=2x

y=x/2+128

```
img=io.imread('images/jasmine.jpg');
plt.figure()
io.imshow(img)
img=np.float64(img)
img1=np.uint8(np.clip((img/2),0,255))
plt.figure()
io.imshow(img1)
img2=np.uint8(np.clip((img*2),0,255))
plt.figure()
io.imshow(img2)
img3=np.uint8(np.clip((img/2)+128,0,255))
plt.figure()
io.imshow(img3)
```

Arithmetic operations on an image: multiplication and division

# Logic Operations



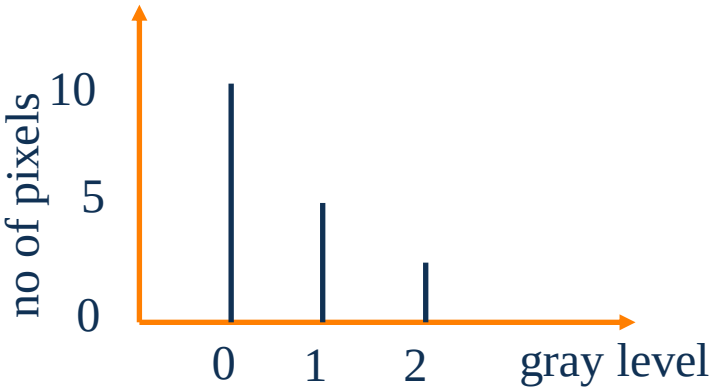

Some logic operations between binary images. Black represents binary 0s and white binary 1s.

# Histograms

- Given a greyscale image, its histogram consists of the histogram of its grey levels; that is, a graph indicating the number of times each grey level occurs in the image.

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 |

Image with gray level 3

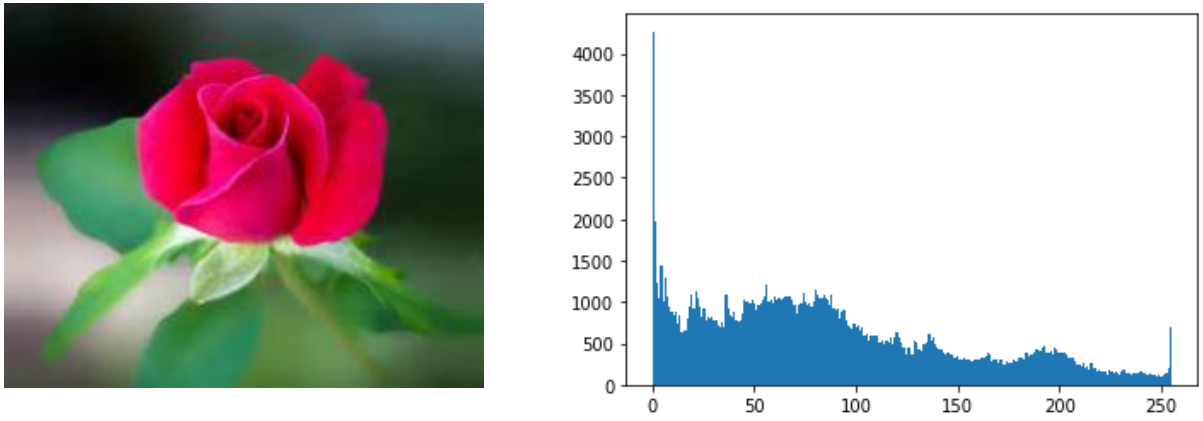



Histogram of the image

# Histograms (cont.)

- We can view the histogram of an image in Python by using the plt.hist function:

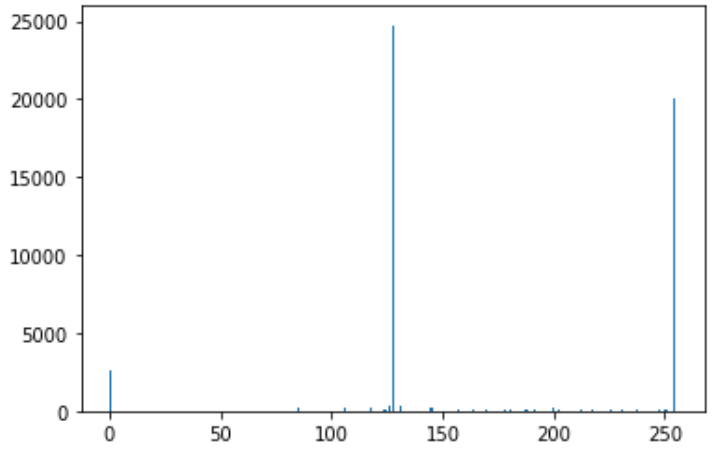
```
import matplotlib.pyplot as plt
img=io.imread('images/redrose.jpg');
plt.figure()
io.imshow(img)
plt.figure()
plt.show(plt.hist(img.flatten(),bins=256))
```




The image redrose.jpg and its histogram

# Histograms (cont.)

- The following image is the image which has only three grey levels. (0, 128 and 255)

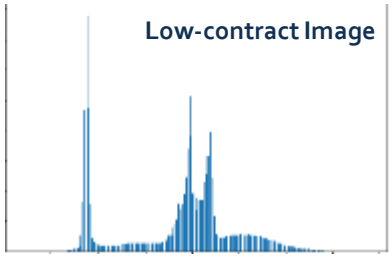


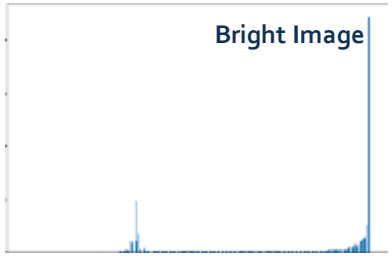

Original image and its histogram

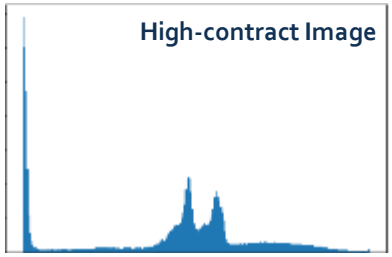
# Histograms (cont.)

- In a dark image, the grey levels (and hence the histogram) would be clustered at the lower end:
- In a uniformly bright image, the grey levels would be clustered at the upper end:
- In a low contracted image, the grey levels would be clustered together in the centre of the histogram
- In a well contrasted image, the grey levels would be well spread out over much of the range:

# Histograms (cont.)


*x*-axis – values of intensities
*y*-axis – their frequencies

# Histograms (cont.)

- Given a poorly contrasted image, we would like to enhance its contrast, by spreading out its histogram. There are two ways of doing this.
  - Histogram stretching (Contrast stretching)
  - Histogram equalization

# Histogram stretching (Contrast stretching)

- We have a low contract image with the histogram shown as below, associated with a table of the numbers $n_i$ of grey values:

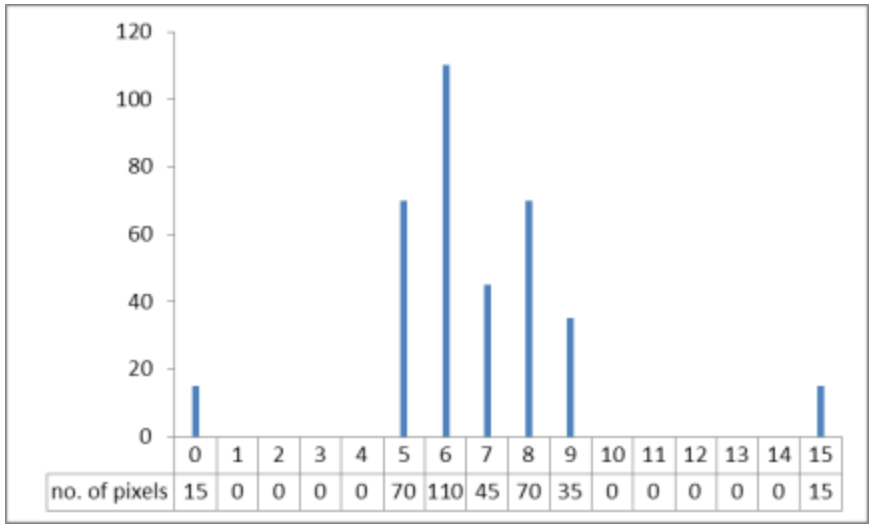
| Grey level i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n_i$ | 15 | 0 | 0 | 0 | 0 | 70 | 110 | 45 | 70 | 35 | 0 | 0 | 0 | 0 | 0 | 15 |

- We can stretch the grey levels in the centre of the range out by applying the piecewise linear function.

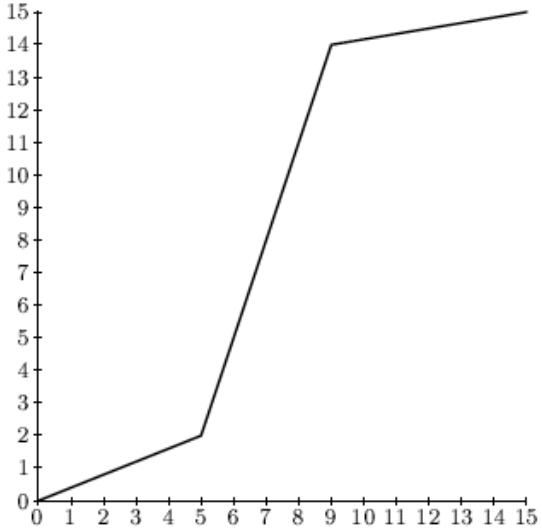
- This function has the effect of stretching the grey levels 5-9 to grey levels 2-14 according to the equation where *i* is the original grey level and *j* its result after the transformation

$$j = \frac{14-2}{9-5}(i-5)+2$$

# Histogram stretching (Contrast stretching)




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| no. of pixels | 15 | 0 | 0 | 0 | 0 | 70 | 110 | 45 | 70 | 35 | 0 | 0 | 0 | 0 | 0 | 15 |



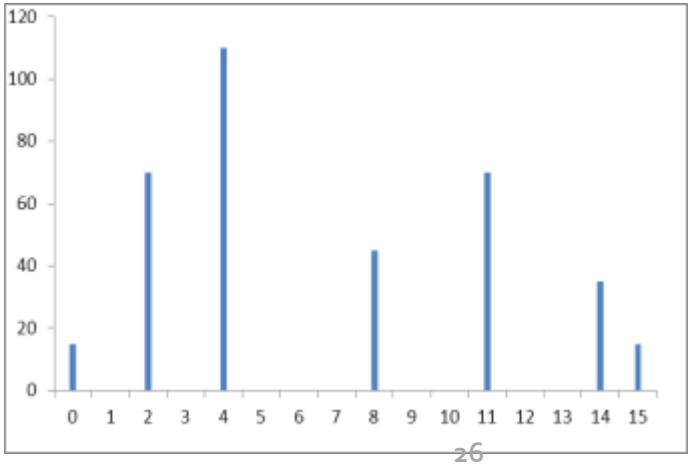

A histogram of a poorly contrasted image, and a stretching function

# Histogram stretching (Contrast stretching)

- Grey levels outside this range are either left alone or transformed according to the linear functions at the ends of the graph above.
- This yields:

| $i$ | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|
| $j$ | 2 | 5 | 8 | 11 | 14 |

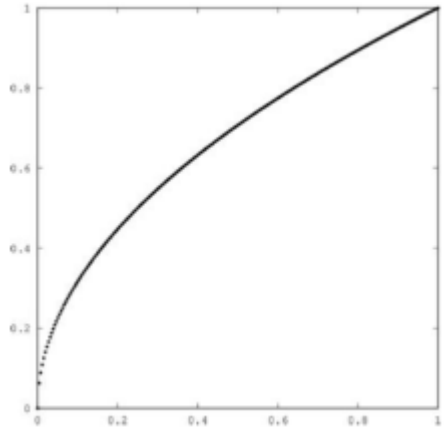
- The corresponding histogram indicates an image with greater contrast than the original.



# Histogram stretching (Contrast stretching)

- To perform histogram stretching in Python the exposure module of skimage method may be used.

```
import matplotlib.pyplot as plt
import skimage.io as io
import skimage.exposure as ex
import numpy as np
img=io.imread('images/rice.tif');
plt.figure()
io.imshow(img)
img1=ex.adjust_gamma(img,0.5)
plt.figure()
io.imshow(img1)
```



The function used in above code

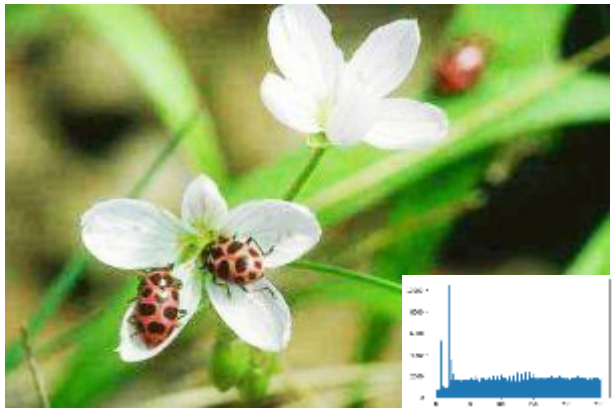
# Histogram equalization

- Histogram stretching requires user input.
- Histogram equalization is an entirely automatic procedure.
- The idea is to change the histogram to one which is uniform; that is that every bar on the histogram is of the same height, i.e. each grey level in the image occurs with the same frequency.
- Histogram equalization is used to adjust the contrast and brightness of the image.

# Histogram equalization (cont.)

Low Contrast

Enhance



High Contrast

# Histogram equalization (cont.)

- Suppose our image has *L* different grey levels 0,1,2,…L-1, and that grey level *i* occurs $n_i$ times in the image. Suppose also that the total number of pixels in the image is *n*

$$n_0 + n_1 + n_2 + \cdots + n_{L-1} = n.$$

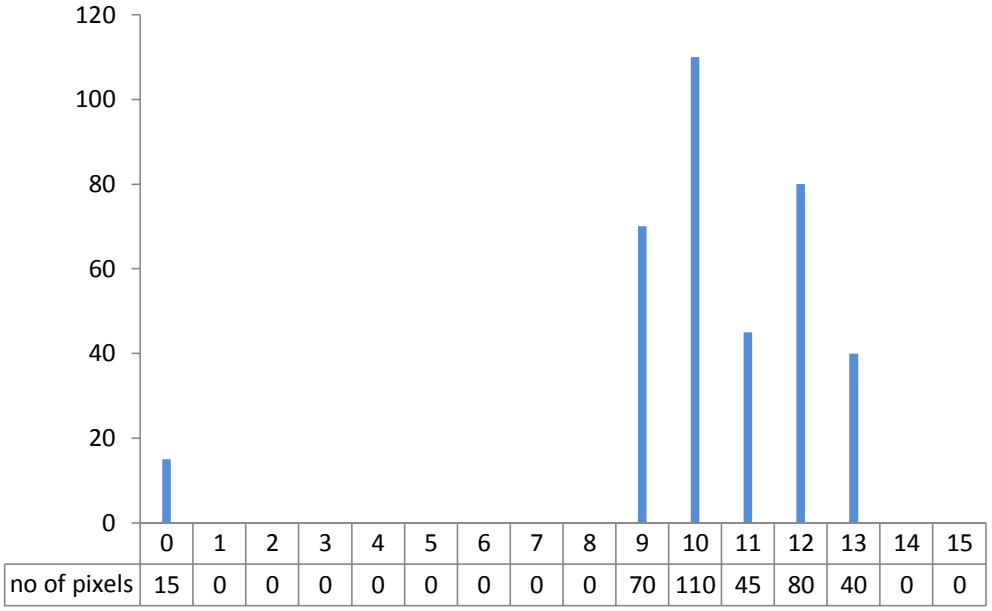
- To transform the grey levels to obtain a better contrasted image, we change grey level *i* to

$$\frac{(n_0 + n_1 + \cdots + n_i)}{n}(L-1)$$

- And this number is rounded to the nearest integer .

# Histogram equalization (cont.)

- Suppose a 4-bit greyscale image has the histogram indicating poor contrast.




| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| no of pixels | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70 | 110 | 45 | 80 | 40 | 0 | 0 |

# Histogram equalization (cont.)

| Grey Level i | $n_i$ | $\sum n_i$ | $(1/24)\sum n_i$ | Rounded value |
|---|---|---|---|---|
| 0 | 15 | 15 | 0.63 | 1 |
| 1 | 0 | 15 | 0.63 | 1 |
| 2 | 0 | 15 | 0.63 | 1 |
| 3 | 0 | 15 | 0.63 | 1 |
| 4 | 0 | 15 | 0.63 | 1 |
| 5 | 0 | 15 | 0.63 | 1 |
| 6 | 0 | 15 | 0.63 | 1 |
| 7 | 0 | 15 | 0.63 | 1 |
| 8 | 0 | 15 | 0.63 | 1 |
| 9 | 70 | 85 | 3.65 | 4 |
| 10 | 110 | 195 | 8.13 | 8 |
| 11 | 45 | 240 | 10 | 10 |
| 12 | 80 | 320 | 13.33 | 13 |
| 13 | 40 | 360 | 15 | 15 |
| 14 | 0 | 360 | 15 | 15 |
| 15 | 0 | 360 | 15 | 15 |

# Histogram equalization (cont.)

| Original grey level i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Final grey level j | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 8 | 10 | 13 | 15 | 15 | 15 |

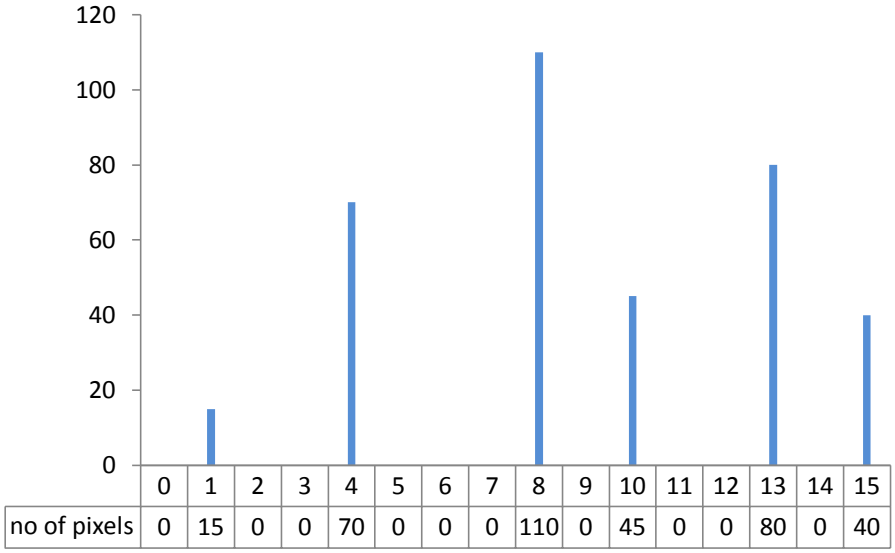



Figure 3. The histogram of figure after equalization

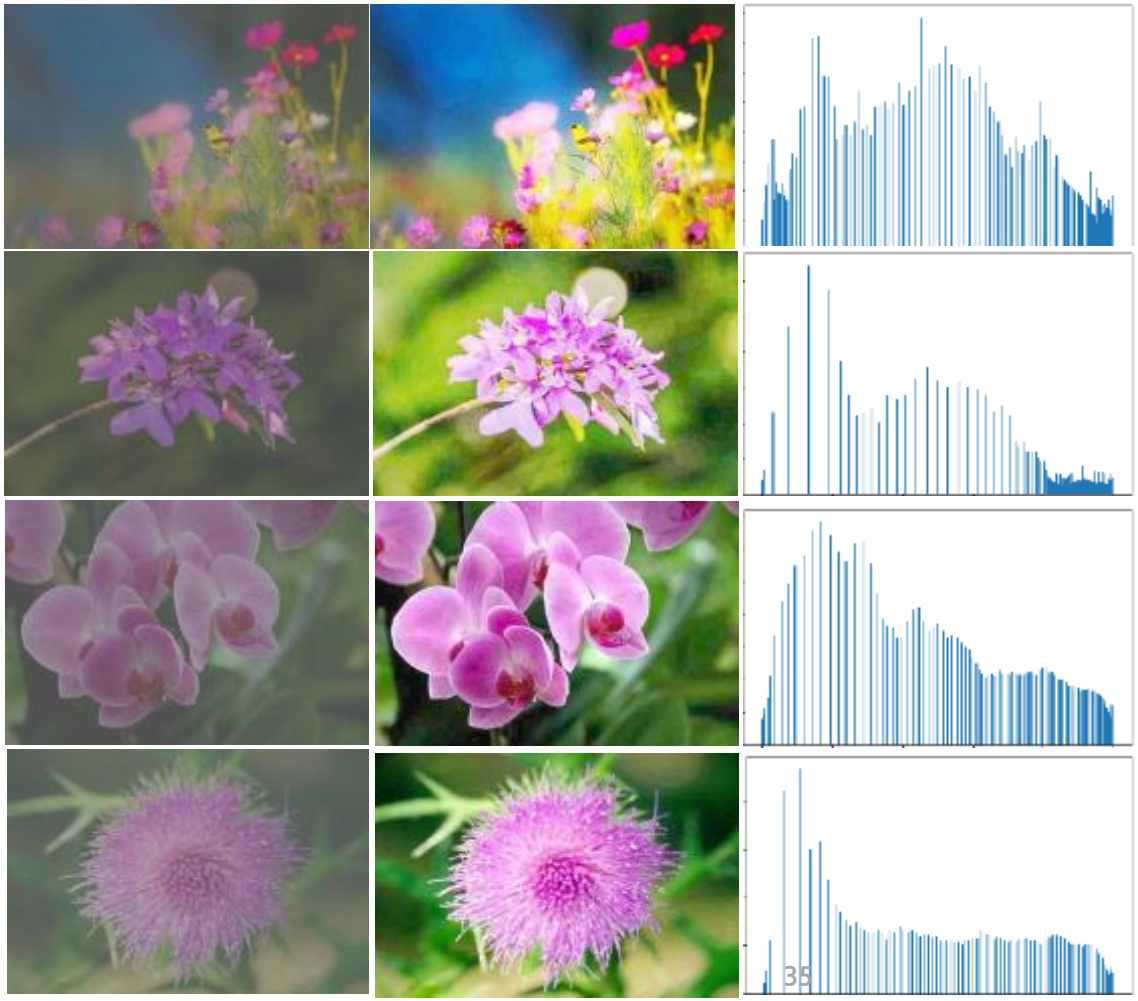
# Histogram equalization (cont.)

- To apply histogram equalization in Python, use the equalize_hist function of skimage.exposure; for example:

```
img=io.imread('images/rice.tif');
img1=ex.equalize_hist(img)
plt.figure()
io.imshow(img1)
plt.figure()
plt.show(plt.hist(img1.flatten(),bins=256))
```

- Applies histogram equalization to the low contract image, and produces the resulting histogram.

# Histogram Equalization (cont.)



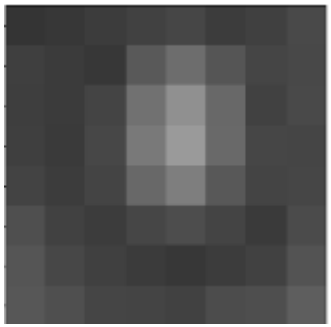
Left column: original images.

Center column: corresponding histogram equalized images.

Right column: histograms of the images in the center column

# Histogram Equalization (cont.)

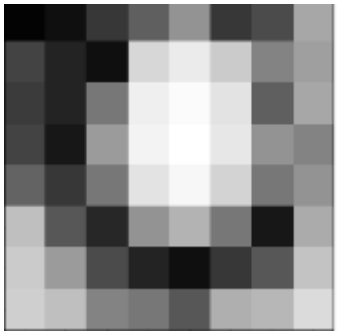
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 52 | 55 | 61 | 66 | 70 | 61 | 64 | 73 |
| 63 | 59 | 55 | 90 | 109 | 85 | 69 | 72 |
| 62 | 59 | 68 | 113 | 144 | 104 | 66 | 73 |
| 63 | 58 | 71 | 122 | 154 | 106 | 70 | 69 |
| 67 | 61 | 68 | 104 | 126 | 88 | 68 | 70 |
| 79 | 65 | 60 | 70 | 77 | 68 | 58 | 75 |
| 85 | 71 | 64 | 59 | 55 | 61 | 65 | 83 |
| 87 | 79 | 69 | 68 | 65 | 76 | 78 | 94 |



8 x 8 small gray image

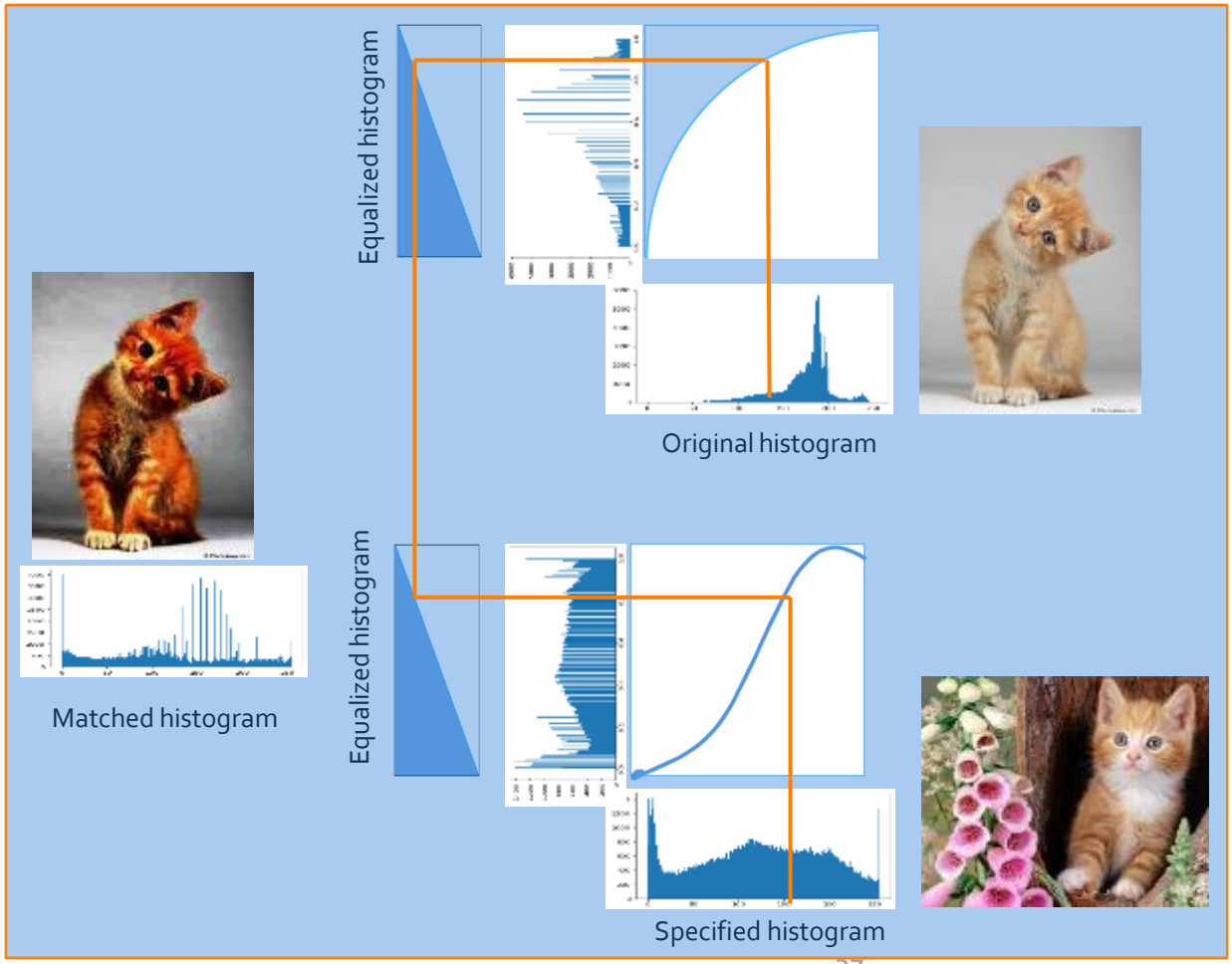
```
import skimage.exposure as ex
import numpy as np
import skimage.io as io
img=np.array([[52,55,61,66,70,61,64,73],
        [63,59,55,90,109,85,69,72],
        [62,59,68,113,144,104,66,73],
        [63,58,71,122,154,106,70,69],
        [67,61,68,104,126,88,68,70],
        [79,65,60,70,77,68,58,75],
        [85,71,64,59,55,61,65,83],
        [87,79,69,68,65,76,78,94]])
imgnew=np.uint8(ex.equalize_hist(img)*255)
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 15 | 55 | 95 | 147 | 55 | 75 | 167 |
| 67 | 35 | 15 | 215 | 235 | 203 | 131 | 159 |
| 59 | 35 | 119 | 239 | 251 | 227 | 95 | 167 |
| 67 | 23 | 155 | 243 | 255 | 231 | 147 | 131 |
| 99 | 55 | 119 | 227 | 247 | 211 | 119 | 147 |
| 191 | 87 | 39 | 147 | 179 | 119 | 23 | 171 |
| 203 | 155 | 75 | 35 | 15 | 55 | 87 | 195 |
| 207 | 191 | 131 | 119 | 87 | 175 | 183 | 219 |



Histogram equalized image

# Histogram Matching or Specification




Histogram matching or histogram specification

- The transformation of an image so that its histogram matches a specified histogram of target image.
- The idea is to calculate equalized histograms of input and specified images.
- Do mapping from original histogram to specified histogram through equalized histogram.

# Histogram Matching or Specification(cont.)

- Perform the Histogram Matching

**Original Image**

| Grey level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| No. of Pixels | 8 | 10 | 10 | 2 | 12 | 16 | 4 | 2 |

**Desired Image**

| Grey level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| No. of Pixels | 0 | 0 | 0 | 0 | 20 | 20 | 16 | 8 |

# Histogram Matching or Specification(cont.)

- Histogram Equalization (Input Image)

| Grey level | No. of Pixels ($n_k$) | PDF ($n_k$/N) | CDF | (L-1)*CDF | $H_k$ |
|---|---|---|---|---|---|
| 0 | 8 | 0.13 | 0.13 | 0.91 | 1 |
| 1 | 10 | 0.16 | 0.29 | 2.03 | 2 |
| 2 | 10 | 0.16 | 0.45 | 3.15 | 3 |
| 3 | 2 | 0.03 | 0.48 | 3.36 | 3 |
| 4 | 12 | 0.18 | 0.66 | 4.62 | 5 |
| 5 | 16 | 0.25 | 0.91 | 6.37 | 6 |
| 6 | 4 | 0.06 | 0.97 | 6.79 | 7 |
| 7 | 2 | 0.03 | 1.0 | 7 | 7 |
| | 64 | 1 | | | |

# Histogram Matching or Specification(cont.)

- Histogram Equalization (Target Image)

| Grey level | No. of Pixels ($n_k$) | PDF ($n_k$/N) | CDF | (L-1)*CDF | $S_k$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 |
| 4 | 20 | 0.31 | 0.31 | 2.17 | 2 |
| 5 | 20 | 0.31 | 0.62 | 4.34 | 4 |
| 6 | 16 | 0.25 | 0.87 | 6.09 | 6 |
| 7 | 8 | 0.13 | 1.0 | 7 | 7 |
| | 64 | 1 | | | |

# Histogram Matching or Specification(cont.)

- Mapping

| Grey Scale | H | S | Map |
|---|---|---|---|
| 0 | 1 | 0 | 4 |
| 1 | 2 | 0 | 4 |
| 2 | 3 | 0 | 5 |
| 3 | 3 | 0 | 5 |
| 4 | 5 | 2 | 6 |
| 5 | 6 | 4 | 6 |
| 6 | 7 | 6 | 7 |
| 7 | 7 | 7 | 7 |

| Original grey level i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Final grey level j | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 7 |

# Histogram Matching or Specification(cont.)

Modified Image

| Grey Level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| No. of Pixels | 0 | 0 | 0 | 0 | 18 | 12 | 28 | 6 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 2 | 4 | 5 | 5 |
| 0 | 0 | 1 | 1 | 2 | 4 | 5 | 5 |
| 0 | 0 | 1 | 1 | 2 | 4 | 5 | 6 |
| 0 | 1 | 1 | 2 | 2 | 4 | 5 | 6 |
| 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 2 | 2 | 3 | 4 | 5 | 6 |
| 4 | 4 | 4 | 4 | 4 | 4 | 5 | 7 |
| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 7 |

Input Image

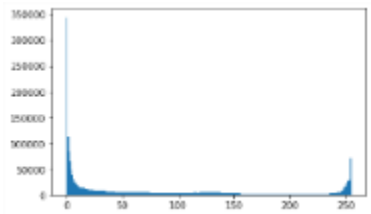
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 5 | 6 | 6 | 7 |
| 4 | 4 | 4 | 4 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 |

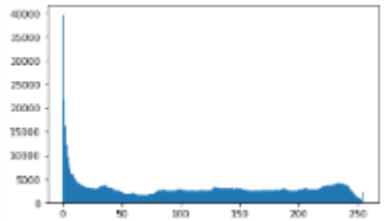
Target Image

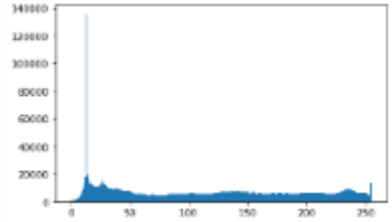
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 5 | 6 | 6 | 6 |
| 4 | 4 | 4 | 4 | 5 | 6 | 6 | 6 |
| 4 | 4 | 4 | 4 | 5 | 6 | 6 | 7 |
| 4 | 4 | 4 | 5 | 5 | 6 | 6 | 7 |
| 4 | 4 | 4 | 5 | 5 | 6 | 6 | 7 |
| 5 | 5 | 5 | 5 | 5 | 6 | 6 | 7 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 |
| 6 | 6 | 6 | 6 | 6 | 6 | 6 | 7 |

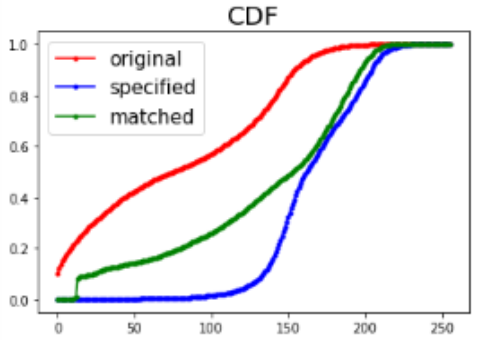
Modified Image

# Histogram Matching or Specification(cont.)










First column : original images and its histogram.
Second column : specified image and its histogram.
Third column : matched image and its histogram.
Fourth column : cumulative distribution of images

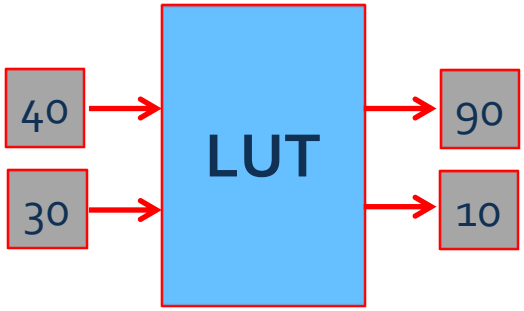
# Lookup tables

- Point operations can be performed very effectively by the use of a lookup table, known more simply as an LUT.
- For operating on images of type uint8, such a table consists of a single array of 256 values, each value of which is an integer in the range 0…255.
- Then our operation can be implemented by replacing each pixel value by the corresponding value in the table.

# Lookup tables (cont.)

| | | | |
|---|---|---|---|
| 40 | 40 | 40 | 40 |
| 40 | 30 | 30 | 40 |
| 40 | 30 | 30 | 40 |
| 40 | 40 | 40 | 40 |

Low Contract




| | | | |
|---|---|---|---|
| 90 | 90 | 90 | 90 |
| 90 | 10 | 10 | 90 |
| 90 | 10 | 10 | 90 |
| 90 | 90 | 90 | 90 |

High Contract

Contract is changed by changing pixel values.

# Lookup tables (cont.)

- LUT corresponding to division by 2 can be reconstructed:

| Index | 0 | 1 | 2 | … | 252 | 253 | 254 | 255 |
|---|---|---|---|---|---|---|---|---|
| LUT | 0 | 0 | 1 | … | 125 | 126 | 127 | 127 |

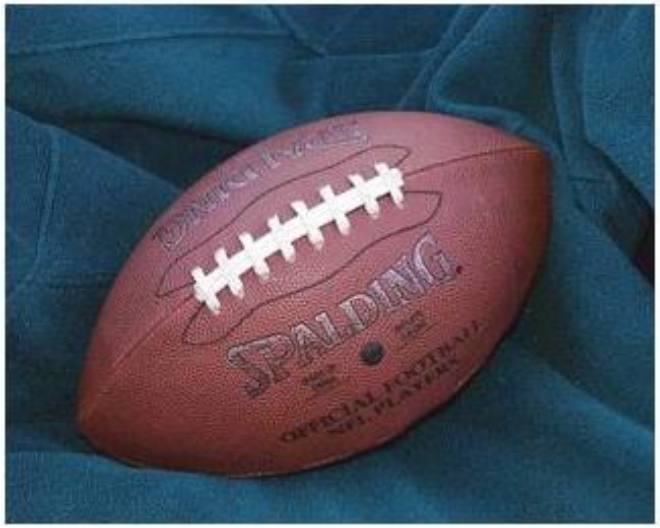
- A pixel with value 2 will be replaced with 1; a pixel with value 253 will be replaced with value 126.

- The above lookup table (division by 2) can be applied by the simple command T[im] in Python ,i.e. T is lookup table and im is input image.

```
T=np.uint8(np.arange(256))/2
img1=np.uint8(T[img])
```

# Lookup tables (cont.)

```
T=np.uint8(np.arange(256))/2
img=io.imread('football.jpg');
plt.figure()
io.imshow(img)
img1=np.uint8(T[img])
plt.figure()
io.imshow(img1)
```




original image

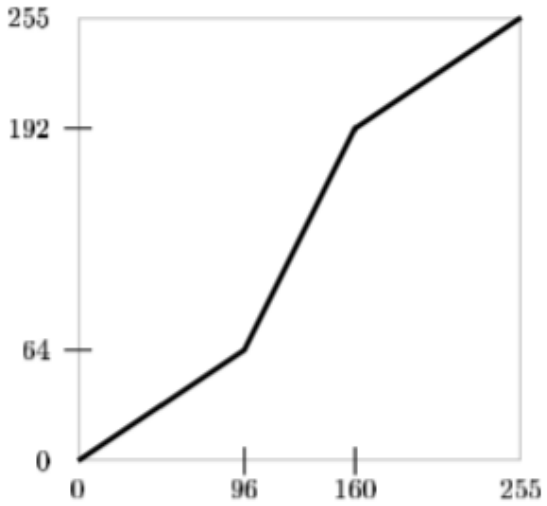

Image using LUT table

# Lookup tables (cont.)

- Contrast stretching function
  - Create a LUT to implement the contrast stretching function shown in figure. Givens are the equations of the three lines used for lookup table.

$$y = \frac{64}{96}x$$
$$y = \frac{192 - 64}{160 - 96}(x - 96) + 64$$
$$y = \frac{255 - 192}{255 - 160}(x - 160) + 192$$

$$y = 0.6667x$$
$$y = 2x - 128$$
$$y = 0.6632x + 85.8947$$

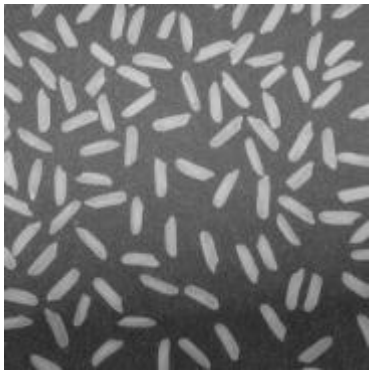



Contract stretching function

# Lookup tables (cont.)

- Python code for contrast stretching function

```
import matplotlib.pyplot as plt
import skimage.io as io
import numpy as np
#create table for contrast stretching function
t1=np.uint8(np.arange(97))*0.6667
t2=2*np.uint8(np.arange(97,161))-128
t3=0.6632*np.uint8(np.arange(161,256))+85.8947
T=np.concatenate((t1,t2,t3))
# apply table to image
img=io.imread('images/rice.tif');
plt.figure()
io.imshow(img)
img1=np.uint8(T[img])
plt.figure()
io.imshow(img1)
```



Original Image

Image using LUT table

# Assignment

The following small image has grey values in the range 0 to 15. Compute the grey level histogram and the mapping that will equalize this histogram. Produce an 8x8 grid containing the grey values for the new histogram-equalized image.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 3 | 2 | 12 | 15 | 15 | 17 | 15 |
| 9 | 7 | 5 | 2 | 5 | 9 | 15 | 15 |
| 6 | 5 | 1 | 1 | 4 | 11 | 18 | 19 |
| 7 | 4 | 1 | 0 | 7 | 11 | 12 | 17 |
| 17 | 6 | 12 | 12 | 17 | 19 | 19 | 17 |
| 11 | 7 | 15 | 15 | 18 | 18 | 17 | 15 |
| 9 | 5 | 7 | 11 | 15 | 12 | 15 | 15 |
| 5 | 3 | 1 | 4 | 6 | 9 | 11 | 11 |

# Assignment (cont.)

The following small image has grey values in the range 0 to 7. Perform histogram matching for image1 to match histogram of desired image2. Produce an 8x8 grid containing the grey values for the new histogram-specified image.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 2 | 4 | 3 | 5 | 5 |
| 5 | 5 | 1 | 2 | 3 | 4 | 5 | 5 |
| 1 | 0 | 2 | 0 | 4 | 4 | 4 | 6 |
| 1 | 1 | 4 | 2 | 2 | 4 | 5 | 6 |
| 4 | 4 | 1 | 0 | 2 | 0 | 5 | 5 |
| 4 | 1 | 1 | 2 | 0 | 5 | 5 | 5 |
| 4 | 1 | 0 | 0 | 2 | 5 | 6 | 7 |
| 4 | 1 | 2 | 2 | 5 | 5 | 7 | 6 |

Image 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4 | 7 | 5 | 5 | 5 | 5 | 6 | 6 |
| 5 | 4 | 4 | 4 | 5 | 4 | 6 | 6 |
| 5 | 5 | 4 | 5 | 4 | 5 | 5 | 7 |
| 5 | 4 | 4 | 4 | 5 | 4 | 6 | 7 |
| 5 | 4 | 4 | 4 | 5 | 4 | 6 | 7 |
| 5 | 5 | 4 | 4 | 5 | 4 | 6 | 7 |
| 5 | 6 | 6 | 6 | 4 | 4 | 6 | 7 |
| 5 | 6 | 6 | 6 | 6 | 6 | 7 | 7 |

Image 2

# Python code

```
##histogram equalization for gray image##
import numpy as np
from matplotlib import pyplot as plt
from skimage.color import rgb2gray
import skimage.io as io
img=io.imread('images/lowcontractflowers.jpg')
img=np.int32(255*rgb2gray(img))
imgnew=img.copy()
l,w=img.shape
n=l*w
L=img.max()
bin=np.zeros(L+1)
binnew=np.zeros(L+1)
for i in range(l):
   for j in range(w):
      for k in range(L+1):
         if img[i][j]==k :
            bin[k]=bin[k]+1
for k in range(L+1):
  binnew[k]=(np.sum(bin[0:k+1])*(L/n)).round()
for i in range(l):
   for j in range(w):
      for k in range(L+1):
         if imgnew[i][j]==k :
            imgnew[i][j]=binnew[k]
            break
```

# Python code (cont.)

```
def goto(line) :
    global lineNumber
    lineNumber=line

def find_hist_eq(img,L):
    l,w=img.shape
    n=l*w
    L=img.max()
    bin=np.zeros(L+1)
    binnew=np.zeros(L+1)
    for i in range(l):
        for j in range(w):
            for k in range(L+1):
                if img[i][j]==k :
                    bin[k]=bin[k]+1
    for k in range(L+1):
        binnew[k]=(np.sum(bin[0:k+1])*(L/n)).round()
    return binnew
```

```
## Histogram Matching ###
from skimage import data
img1= data.coffee()
img2 = data.chelsea()
imgnew=img1.copy()
l,w=img1.shape
L1=img1.max()
L2=img2.max()
bin1=np.zeros(L1+1)
bin2=np.zeros(L2+1)
bin3=np.zeros(L2+1)
bin1=find_hist_eq(img1,L1)
bin2=find_hist_eq(img2,L2)
j=0
i=0
c=L1
while i <=c:
    while j <= 7:
        if bin1[i]==bin2[j]:
            bin3[i]=j
            break
        j=j+1
    if j==8:
        bin1[i]=bin1[i]+1
        j=0
        i-=1
        goto(41)
    i=i+1
```

```
for p in range(8):
    for q in range(8):
        for r in range(8):
            if imgnew[p][q]==r :
                imgnew[p][q]=bin3[r]
                break
```

# Next Week Lecture (Week3)

- Lecture3:Image Enhancement in Spatial Domain (Neighborhood Processing)

Thank You