Image Processing and Related Fields

- Signal processing
- Image processing
- Computer/Machine/Robot vision
- Biological vision
- Artificial intelligence
- Machine learning
- Pattern recognition

Computer vision is in parallel to the study of biological vision, as a major effort in the brain study. In this class of *Image Processing and Analysis*, we will cover some basic concepts and algorithms in image processing and pattern classification. The specific topics to be discussed in the course are some subset of these topics.

Applications of Image Processing

Visual information is the most important type of information perceived, processed and interpreted by the human brain. One third of the cortical area of the human brain is dedicated to visual information processing.

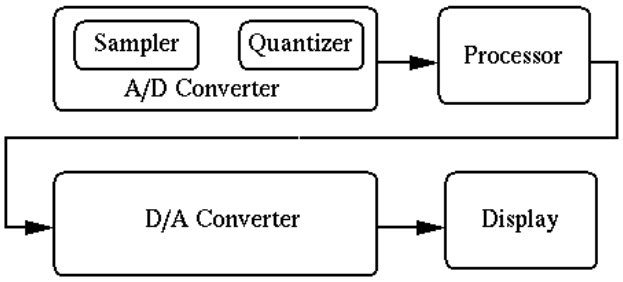
Digital image processing, as a computer-based technology, carries out automatic processing, manipulation and interpretation of such visual information, and it plays an increasingly important role in many aspects of our daily life, as well as in a wide variety of disciplines and fields in science and technology, with applications such as television, photography, robotics, remote sensing, medical diagnosis and industrial inspection.

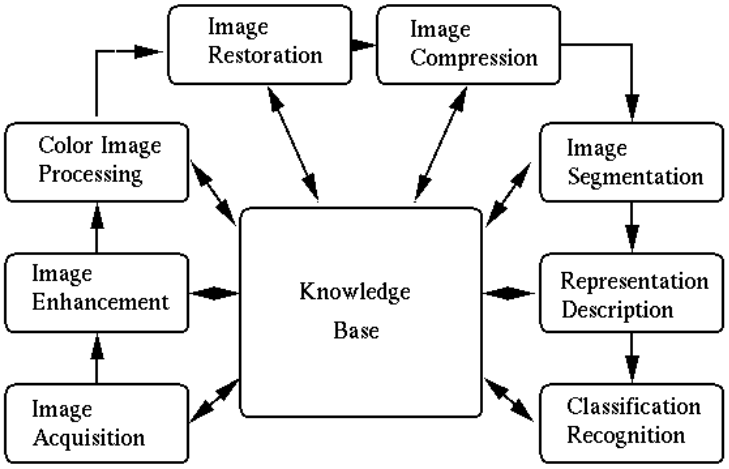
- Computerized photography (e.g., Photoshop)
- Space image processing (e.g., Hubble space telescope images, interplanetary probe images)
- Medical/Biological image processing (e.g., interpretation of X-ray images, blood/cellular microscope images)
- Automatic character recognition (zip code, license plate recognition)
- Finger print/face/iris recognition
- Remote sensing: aerial and satellite image interpretations
- Reconnaissance
- Industrial applications (e.g., product inspection/sorting)

Different Types of Tasks

- **Image acquisition, storage, transmission**: digitization/quantization, compression, encoding/decoding
- **Image Enhancement** and **Restoration**: for improvement of pictorial information for human interpretation, both input and output are in the image form (e.g., the first few application examples above).
- **Image Understanding** and **Image Recognition**: information extraction from images for further computer analysis (e.g., the rest of the application examples above). Input is in image form, but output is some none image representation of the image content, such as description, interpretation, classification, etc.
- **Pre-processing stage of computer vision** of an artificial intelligent system (robots, autonomous vehicles, etc.).

Fundamental Steps in Digital Image Processing





These steps roughly correspond to the visual information processing in the brain.

Visual Perception of Luminance

- **Spectral energy distribution of light source**:

$$L(\lambda), \quad 350\,nm \leq \lambda \leq 780\,nm$$

- **Luminance (intensity)** Light energy reflected by an object:

$$I(\lambda) = \rho(\lambda)\, L(\lambda)$$

where $\rho(\lambda)$ is the reflectivity of the object. I represents the objective physics of the lighting of the object.

- **Image signals:** The light reflected by a 3D object is projected through the lens of the visual system (camera, eye) to become a 2D signal $I(x, y, \lambda)$, which is then detected by the sensors/receptors of the visual system: $f(x, y) = \int_0^\infty I(x, y, \lambda)\, S(\lambda)\, d\lambda$ Here $S(\lambda)$ is the sensitivity (``luminous efficiency'') of the film, the CCD sensors, or the photoreceptors (rods and cones) in the retina. The function of human eye is a bell-shaped function of frequency.

- **Apparent brightness (brightness)**: Brightness is the *perception* or sensation caused by the input light signal. It is a subjective and qualitative attribute of the object being observed, and it depends on the surroundings of an object as well as the luminance. Two objects with different surroundings could have the same luminance but different brightnesses. For example, the screen of a TV set may look gray, but when it is turned on, a black object in the scene may seem darker due to the comparison with the background, e.g., some white objects in the scene. More examples: White'sillusion and Wertheimer-Benary illusion.

- **Contrast:** Assuming the luminance of an object is f and the luminance difference between the object and its surrounding is df, then according to Weber's law, the perceived contrast dp (luminance difference) between the object and its surrounding is dp = df/f = d(ln f) which indicates that at higher level f, larger df is needed to perceive the same contrast at lower level f with a smaller df. In other words, equal increment in ln(f), instead of in f, is perceived to be equally different (equal contrast). Integrating both sides, we get the perceived luminance $p = \int d(ln\, f) = ln\, f + C$ The constant of integration C can be obtained by assuming the perceived luminance is zero p = 0: C = -ln f0, where f0 is the threshold luminance not perceivable. Now we have p = ln(f/f0). The relationship between stimulus f and perception p is logarithmic. Weber's law describes a general phenomenon in human perception. Another example is the difference between different sound frequencies. The difference between C4 (middle C, 261.63 Hz) and C5 (523.25 Hz) is an octave, perceived the same as the difference between C5 and C6 (1046.5 Hz), although the frequency differences between the two pairs are quite different (261.63 Hz. vs. 523.25 Hz).

Color Representation

What Determines the Color?

Along the visible wavelength (350 nm - 780 nm), there are only about 128 fully saturated colors that can be distinguished. It is the energy spectral distribution $P(\lambda)$ of the signal that determines the colors we perceive.

Three Components of Color

Hue: the dominant wavelength, the redness of red, greenness of green, etc.

Saturation: how pure the color is, or how much white is contained in the color. For example, red and royal blue are more saturated than pink and sky blue, respectively.

Luminance: the amount or intensity of light.

Tristimulus Theory

There exist 3 types of cells (cones) in human retina of different response functions (luminous efficiency functions): $S_i(\lambda), \ (i = 1, 2, 3)$. They overlap with each other and peak in the yellow-green, green and blue regions, respectively. The responses of these cells to a signal of intensity $C(\lambda)$ (a ``color'') are therefore

$$r_i(C) = \int S_i(\lambda) C(\lambda) d\lambda \quad (i = 1, 2, 3)$$

The perceived color is determined by the combination of these 3 responses $r_i, \ (i = 1, 2, 3)$. In other words, if two colors $C_1(\lambda)$ and $C_2(\lambda)$ produce the same responses:

$$r_i(C_1) = r_i(C_2) \quad (i = 1, 2, 3)$$

then they are perceived as the same color.

Color Models

There exist many different color models (all composed of three independent variables), for example:

- RGB model: using Red, Green, and Blue as three primaries to represent a color.
- HSV model: using Hue, Saturation, and Value (intensity) to represent a color
- XYZ model (International Commission on Illumination, CIE)

Color Matching

It is possible for different colors, energy distributions, to produce exactly the same visual perception in the human visual system. These colors are said to be matched and are called *metamers*. Two matching colors $C_1(\lambda)$ and $C_2(\lambda)$ can be represented by $C_1(\lambda) \equiv C_2(\lambda)$ Note that in general matching colors do not necessarily have identical energy distributions,

$$C_1(\lambda) \neq C_2(\lambda)$$

Three-Color Theory

Any color can be reproduced by mixing an appropriate set of three primary colors (e.g., CIE X, Y, Z, or red, green, and blue, not unique) with energy distributions $P_k(\lambda), \quad (k = 1, 2, 3)$.

Matching Colors with Primaries

Suppose in order to match a given color $C(\lambda)$ the three primaries need to be mixed in proportions of $\beta_k, \quad (k = 1, 2, 3)$:

$$C'(\lambda) = \sum_{k=1}^{3} \beta_k P_k(\lambda)$$

For the mixed color $C'(\lambda)$ to be perceived the same as the given color $C(\lambda)$, the responses of the three types of cone cells to $C'(\lambda)$ should be the same as those to $C(\lambda)$:

$$r_i(C) = r_i(C') \quad (i = 1, 2, 3)$$

The cone cells' responses to $C(\lambda)$ are

$$r_i(C) = \int S_i(\lambda) C(\lambda) d\lambda \quad (i = 1, 2, 3)$$

and their responses to the matching color $C'(\lambda)$ are

$$r_i(C') \qquad \int S_i(\lambda) C'(\lambda) d\lambda = \int [\sum_{k=1}^{3} \beta_k P_k(\lambda)] S_i(\lambda) d\lambda$$

$$\sum_{k=1}^{3} \beta_k \int S_i(\lambda) P_k(\lambda) d\lambda = \sum_{k=1}^{3} \beta_k a_{ik} \quad (i = 1, 2, 3)$$

where a_{ik} is defined as the response of ith cells to the kth primary:

$$a_{ik} \triangleq r_i(P_k) = \int S_i(\lambda) P_k(\lambda) d\lambda \quad (i, k = 1, 2, 3)$$ which can be found given the cone cells' sensitivities $S_i(\lambda)$ and the three primary colors $P_k(\lambda)$.

For $C'(\lambda)$ to be perceived the same as $C(\lambda)$, we require

$$r_i(C') = \sum_{k=1}^{3} \beta_k a_{ik} = r_i(C) = \int S_i(\lambda) C(\lambda) d\lambda \quad (i = 1, 2, 3)$$

These three equations are called the *color matching equations*. As both a_{ik} and the right-hand side of the equations (available from the given $C(\lambda)$ and $S_i(\lambda)$) are

known, the 3 coefficients $\beta_k \quad (i = 1,2,3)$ can be obtained by solving the 3 color matching equations, and the matching color is produced by mixing the three primaries: $C'(\lambda) = \sum_{k=1}^{3} \beta_k P_k(\lambda) \equiv C(\lambda)$

CIE XYZ Primaries

The Commission Internationale de l'Eclairage (CIE) defined three standard primaries called **X**, **Y**, and **Z**. Any color $C(\lambda)$ can be matched using these primaries with positive weights X(C), Y(C), and Z(C). The *chromaticity* values of a color is defined by its weights for the three primaries normalized by the total energy X+Y+Z:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}$$

so that x+y+z=1. Chromaticity values depend on the hue and saturation of the color, but are independent of the intensity. All visible colors are represented by the points inside an enclosed area in the X+Y+Z=1 plane. And the chromaticity diagram is the projection of this enclosed area on (X,Y) plane.

Image Digitization

A two-dimensional scene can be represented by a 2D function f(x,y) of light intensity at the spatial location (x,y). However, in order for the continuous scene to be represented and processed digitally in a computer, it needs to be digitized. Specifically, the digitization includes the *quantization* of the intensity function value and the *sampling* of the two spatial dimensions. Correspondingly, the digital processing of the image can be classified into intensity (gray level) operations applied to the pixel values and geometric operations in the two spatial dimensions.

Quantization:

The continuous range of light intensity $0 \le x \le G$ received by the digital image acquisition system need be quantized to L gray levels (e.g., $L = 2^8 = 256$). The numbers of gray levels of the following eight images are respectively 256, 128, 64, 32, 16, 8, 4, and 2, respectively.

- **Uniform distribution**

  Define L+1 boundaries $t_k = kG/L = k\Delta, \quad (k = 0, 1, \cdots, L)$

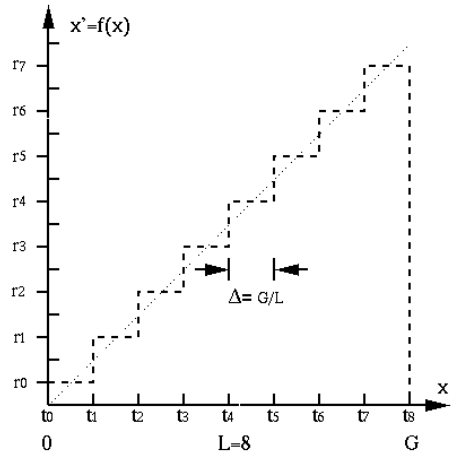
  $\Delta \triangleq G/L$

  where . And define the L discrete gray levels to represent the L intervals:

  $$r_k = t_k + \Delta/2 = k\Delta + \Delta/2 = (k\Delta + k\Delta + \Delta)/2 = (t_k + t_{k+1})/2$$

  Then the quantization can be defined as a function

  $$x' = f(x) = r_k, \quad \text{iff } t_k \le x \le t_{k+1}$$



- **Mean square error optimization**

  Define mean square error of the quantization process as

  $$\mathcal{E} = E[(x - x')^2] = \int_{-\infty}^{\infty} (x - x')^2 p(x) dx = \sum_{i=0}^{L-1} \int_{t_i}^{t_{i+1}} (x - r_i)^2 p(x) dx$$

  where $p(x)$ is distribution of input intensify x. The optimal quantization in terms of

t_k and r_k can be found by minimizing $\mathcal{E}$, by solving

$$\frac{\partial \mathcal{E}}{\partial t_k} = \frac{\partial \mathcal{E}}{\partial r_k} = 0$$

This method requires $p(x)$ to be known. The previous quantization is optimal when $p(x)$ is a uniform distribution. When $p(x)$ is not uniform, more gray levels will be assigned to the gray scale regions corresponding to higher $p(x)$.

- **Contrast equalization**

  The perceived contrast is a function of the intensity. Specifically, we perceive the same contrast between the object and its surrounding if $\frac{\Delta f}{f} \approx \frac{df}{f} = d(ln\, f) = constant$ where f is the intensity and $\Delta f \approx df$ is the intensity difference, the absolute contrast. For example,

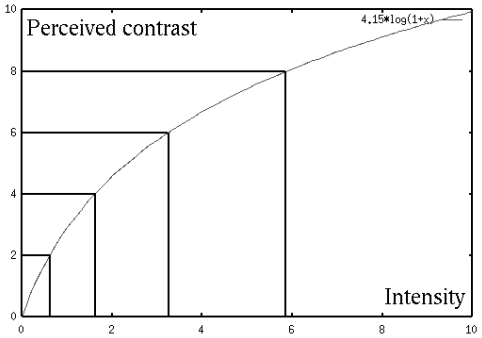
  $$\frac{20-10}{10} = \frac{200-100}{100}$$

  $$\Delta f = 200 - 100 = 100$$

  i.e., a high contrast of at a high absolute intensity f = 100 is perceived the same as a much lower contrast of $\Delta f = 20 - 10 = 10$ at a low absolute intensity f = 10. In other words,, we are less sensitive to contrast when the intensity f is high. As another example, consider the perceived brightness of a 3-way light bulb with 50, 100 and 150 Watts (with the assumption that the brightness is proportional to the power consumption). The perceived contrast between 50 and 100 is higher than that between 100 and 150 as $(100-50)/50 > (150-100)/100$. Consequently, the perceived contrast can be defined as a logarithmic function of the intensity:

  $$y = f(x) = a\ log_e(1+x)$$

  As shown in the figure, to perceive the same contrast, larger intensity difference is needed for higher intensity regions than lower ones.

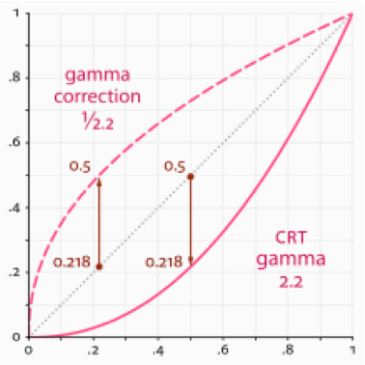


To most efficiently use the limited number of gray levels available, we can allocate more gray levels in the low intensity region where our eye is more sensitive to

contrast) than in high intensity region.

Gamma correction

In the image acquisition process, nonlinear mapping may occur in various stages. For example, in the camera system, the in-coming light intensity may be nonlinearly mapped to the film or digital recording sensors, in the cathode ray tube (CRT), the applied voltage may be nonlinearly mapped to the brightness of the CRT display, and in the biological visual system, the in-coming light intensity is nonlinearly perceived by retina and the visual cortex of the brain. To compensate for all such nonlinear mappings, the following power function that relates the input x to the output y can be considered: $y = Ax^{\gamma}$ where the ranges of both the input and output are normalized so that $0 \le x, y, \le 1$. Here A is a constant scaling factor, and γ is a parameter that characterizes the nonlinearity. Obviously when $\gamma = 1$, y is linearly related to x. Otherwise, we have a nonlinear mapping. As an example, the nonlinear CRT mapping modeled by $y = x^{2.2}$ can be corrected by another nonlinear mapping $z = y^{1/2.2} = (x^{2.2})^{1/2.2} = x$, as shown below:



Spatial sampling

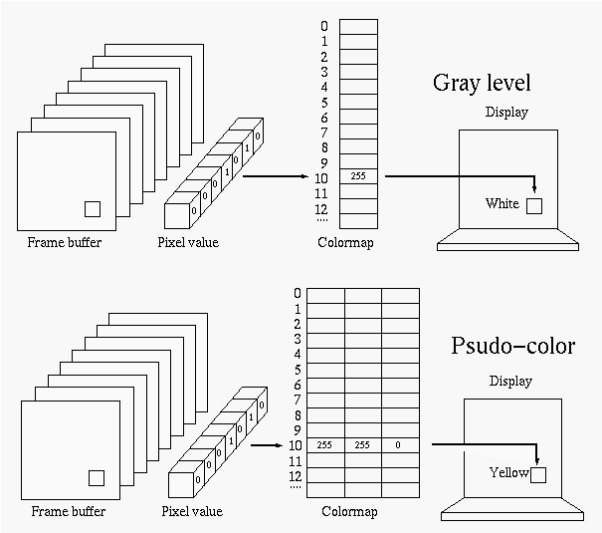
Also, the continuous two-dimensional image space needs to be sampled by the digital image acquisition system to form a raster, a 2D array of pixels (picture-elements) in rows and columns. Same as in 1D case, the sampling theorem also applies her, with the only difference that the sampling is carried out in two spatial dimensions, instead of one temporal dimension.

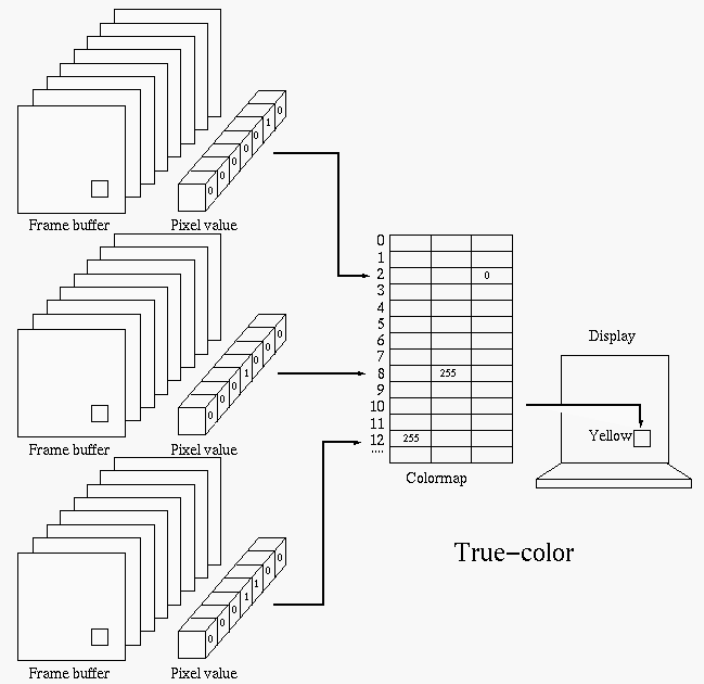
Color and pseudo-color images

A color image is usually represented by three functions of space. In most color formats, the three functions are for three *primary colors* such as red, green and blue $f_r(x,y)$, $f_g(x,y)$, and $f_b(x,y)$, or some other three parameters such as *intensity*, *hue* and *saturation*, $f_i(x,y)$, $f_h(x,y)$, and $f_s(x,y)$.

Sometimes artificial colors can be assigned to a gray level image to better distinguish visually the different gray levels.

The display of gray level, pseudo-color and true-color images on a monitor screen through color-map (color lookup table) is illustrated below.





Neighbors and Connectivities

As digital image is quite different from a continuous scene. As a digital image is no longer isotropic, some concepts intuitive in continuous world, such as neighbor, connectivity, distance, need to be carefully defined for digital images.

Neighbors of Pixel

There are two different ways to define the neighbors of a pixel p located at (x, y):

- **4-neighbors**

  The 4-neighbors of pixel p, denoted by $N_4(p)$, are the four pixels located at (x-1, y), (x+1, y), (x, y-1) and (x, y+1), there are, respectively, above (north), below (south), to the left (west) and right (east) of the pixel p.

- **8-neighbors**

  The 8-neighbors of pixel p, denoted by $N_8(p)$, include the four 4-neighbors and four pixels along the diagonal direction located at (x-1, y-1) (northwest), (x-1, y+1) (northeast), (x+1, y-1) (southwest) and (x+1, y+1) (southeast).

x−1,y−1	x−1,y	x−1,y+1
x,y−1	x,y	x,y+1
x+1,y−1	x+1,y	x+1,y+1

Connectivity

In a binary (black and white) image, two neighboring pixels (as defined above) are *connected* if their values are the same, i.e., both equal to 0 (black) or 255 (white).

In a gray level image, two neighboring pixels are connected if their values are close to each other, i.e., they both belong to the same subset of similar gray levels: $p \in V$ and $q \in V$, where V is a subset of all gray levels in the image.

Specifically, the connectivity can be defined as one of the following:

- **4-connected** Two pixels p and q are 4-connected if they are 4-neighbors and $p \in V$ and $q \in V$;
- **8-connected** Two pixels p and q are 8-connected if they are 8-neighbors and $p \in V$ and $q \in V$;
- **mixed-connected** Two pixels p and q are mix-connected if
  p and q are 4-connected, **or**
  p and q are 8-connected **and** not 4-connected through a third pixel
  ($N_4(p) \cap N_4(q) \not\subset V$)
- The second condition states that if p and q are 8-connected and they are also 4-connected through a third pixel, the tighter 4-connectivity through a third pixel is preferred and therefore p and q are no longer considered as 8-connected.

Two pixels at p at (x, y) and q at (u, v) not 4, 8, or mix-connected can still be connected through a path composed of a sequence (chain) of pixels

$$(x_0, y_0) = (x, y), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1}), (x_n, y_n) = (u, v)$$

with all neighboring pixels (x_i, y_i) and (x_{i+1}, y_{i+1}) 4, 8, or mix-connected.

Example:

The upper-right pixel and the lower-left pixel are 8 and mix-connected, but they are not 4-connected:

0 0 1
0 1 0
1 0 0

The upper-right pixel and the lower-left pixel are 4, 8 and mix-connected:

0 1 1
0 1 0
1 1 0

Distances

Any *distance metric* D(p, q) between pixels p and q must satisfy:

- $D(p, q) \geq 0, \quad (D(p, q) = 0 \text{ iff } p = q)$;
- $D(p, q) = D(q, p)$;
- $D(p, q) \leq D(p, r) + D(r, q)$.

where r is an arbitrary pixel.
Specifically, the distance between pixels p at (x, y) and q at (u, v) can be defined by one of the following:

- **Euclidean distance**

$$D_E(p, q) = \sqrt{(x-u)^2 + (y-v)^2} = [|x-u|^2 + |y-v|^2]^{1/2}$$

- **City-block distance**

$$D_4(p, q) = |x-u| + |y-v| = [|x-u|^1 + |y-v|^1]^{1/1}$$

- **Chess-board distance**

$$D_8(p,q) = max\{|x-u|,|y-v|\} = [|x-u|^{\infty} + |y-v|^{\infty}]^{1/\infty}$$

From these definitions we see that a general distance definition is

$$D(p,q) = [|x-u|^{L} + |y-v|^{L}]^{1/L}$$

where L can take any value between 1 and ∞. When L is small (e.g., 1), contributions of the two dimensions are treated equally, but when L is large (e.g., toward ∞), the dimension with larger contribution is more emphasized. Note that other types of distance metrics can also be used.

The D_E distance in digital image approximates the actual Euclidean distance in continuous situation.

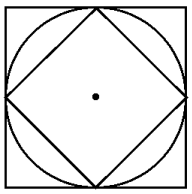
The numbers in the following array show the D_4 distances to the pixel in the center. Note that all 4-neighbors have distance 1.

4 3 2 3 4
3 2 1 2 3
2 1 0 1 2
3 2 1 2 3
4 3 2 3 4

The numbers here are the D_8 distances to the pixel in the center. Note that all 8-neighbors have distance 1.

2 2 2 2 2
2 1 1 1 2
2 1 0 1 2
2 1 1 1 2
2 2 2 2 2

The following figure shows the iso-distance contours composed of all points having equal distance to the center point. The circle is for Euclidean distance, the square is for the D_8 distance, the diamond is for the D_4 distance.



Distance between two connected pixels can be defined as the number of hops from one pixel to the next along the shortest path connecting the two pixels, according to the definition of connectivity (4, 8, or mix-connected).

The upper-right pixel is 8 and mix-connected to the lower-left pixel with a D_8 distance 2:

0 0 1
0 1 0
1 0 0

The upper-right pixel is 4 and mix-connected to the lower-left pixel with a D_4 distance 4:

0 1 1
0 1 0
1 1 0

Gray levels and histogram

The *histogram* is of essential importance in terms of characterizing a given image, and it is a global description of the appearance of the image. The histogram h[i] (i = 0, …, 255) is the probability of an arbitrary pixel to have gray level i, which can be approximated as:

h[i]=(Number of pixels of gray level i)/(Total number of pixels)

The cumulative density function is defined as:

$$H[j] = \sum_{i=0}^{j} h[i], \qquad (j = 0, 1, \cdot, 255)$$

Here is the code for finding the histogram of a given image:

```
for (i = 0; i < glevel; i++) H[i] = h[i] = 0;
for (i = 0; i < M; i++)
 for(j = 0; j < N; j++){
  k = img[i][j];
  h[k]++;
 }
for (i = 0; i < glevel; i++) {
 h[i] = h[i]/M/N;
 for (j = 0; j <= i; j++) H[j] = H[j] + h[j];
}
```

where ^{glevel} is the number of gray levels (256 for a 8-bit image) and note that as the density function, the histogram satisfies:

$$\sum_{i=0}^{glevel-1} h[i] = H[glevel-1] = 1$$

For a gray level image to be properly displayed on screen, its pixel values have to be within a proper range. For a 8-bit digital image there are $2^8 = 256$ (from 0 to 255) gray levels. However, after applying certain processing operations to the input image, the gray levels of the resulting image are no longer necessarily within the proper range for display. In this case rescaling of the image is needed:

$$y = f(x) = 255\ \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x_{min} and x_{max} are, respectively, the minimum and maximum pixel values in the image. The rescaling can be implemented by the following code:

```
min = LARGE; max = -min;
for(i = 0; i < M; i++)
 for(j = 0; j < N; j++){
  if(img[i][j] < min) min = img[i][j];
  if(img[i][j] > max) max = img[i][j];
 }
scale = 255.0/(max - min);
for(i = 0; i < M; i++)
 for(j = 0; j < N; j++){
  img[i][j] = scale * (img[i][j] - min);
 }
```

where $LARGE$ is some large number (e.g., the largest floating point number representable in the computer) known to be greater than the highest pixel value.

Image Scaling and Rotation

Enlargement: The size of a given image can be easily enlarged integer multiple times (2, 3, etc.) by repeating each of the pixels in the image. For example, a 2 by 2 image can be doubled by

$$f_{2\times 2} = \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \Rightarrow f_{4\times 4} = \begin{bmatrix} 1 & 1 & 3 & 3 \\ 1 & 1 & 3 & 3 \\ 4 & 4 & 5 & 5 \\ 4 & 4 & 5 & 5 \end{bmatrix}$$

Obviously the drawback of this simple method is that it is not flexible in terms of the scaling factor, and the resulting image is likely to look blocky.
This replication can be implemented equivalently by this two-step procedure:

- **Zero interlacing**

$$f_{2\times 2} = \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \Rightarrow f'_{4\times 4} = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Convolution** with kernel

$$H_{2\times 2} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$ to get

$$f'_{4\times 4} * H = f_{4\times 4}$$

An obvious problem of enlargement by replication is that the resulting image looks blocky, which can be avoided by using linear interpolation:

$$f_{2\times 2} = \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \Rightarrow f_{4\times 4} = \begin{bmatrix} 1 & 2 & 3 & 1.5 \\ 2.5 & 3.25 & 4 & 2 \\ 4 & 4.5 & 5 & 2.5 \\ 2 & 2.25 & 2.5 & 1.25 \end{bmatrix}$$

This operation is called *bilinear interpolation* (two-dimensional linear interpolation) which can be implemented equivalently by this two-step procedure:

- **Zero interlacing**

$$f_{2\times 2} = \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix} \Rightarrow f'_{4\times 4} = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Convolution** with kernel

$$H_{3\times 3} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

to get

$$f'_{4\times 4} * H_{3\times 3} = f_{4\times 4}$$

Note that the convolution assumes zero pixels outside the image. The resulting image looks smooth instead of blocky.

Reduction: Image size can be easily reduced by subsampling, e.g., getting rid of every other pixel in each row and column:

$$f_{4\times4} = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 99 & 3 & 8 \\ 1 & 4 & 3 & 2 \\ 3 & 2 & 1 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 4 \\ 1 & 3 \end{bmatrix} \text{ or } \begin{bmatrix} 2 & 3 \\ 4 & 2 \end{bmatrix} \text{ or } \begin{bmatrix} 2 & 3 \\ 3 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 99 & 8 \\ 2 & 4 \end{bmatrix}$$

In any of the 4 possible subsampling cases, three fourths of the information contained in the original image is lost. A better way (better model of eye) is to find the average of a 2x2 neighborhood as the resulting pixel:

$$\begin{bmatrix} 1 & 2 & 4 & 3 \\ 2 & 99 & 3 & 8 \\ 1 & 4 & 3 & 2 \\ 3 & 2 & 1 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 26 & 4.5 \\ 2.5 & 2.5 \end{bmatrix}$$

Again, this operation can be implemented in a two-step process:

- **Regional averaging** by convolving with

$$H_{2\times2} = \frac{1}{4}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

to get

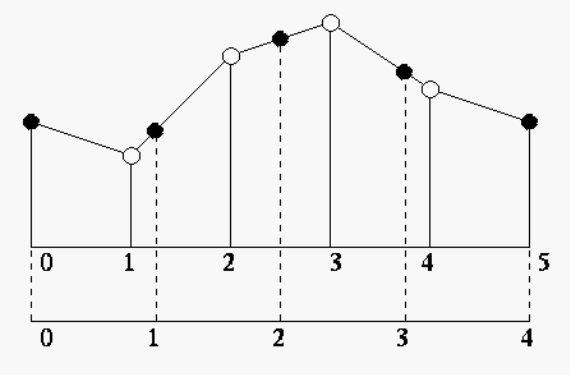
$$f_{4\times4} * H_{2\times2} = f'_{4\times4} = \begin{bmatrix} 26 & 27 & 4.5 & 2.75 \\ 26.5 & 27.23 & 4.0 & 2.5 \\ 2.5 & 2.5 & 2.5 & 1.5 \\ 1.25 & 0.75 & 1.25 & 1.0 \end{bmatrix}$$

- **Subsampling** $f'_{4\times4}$ to get

$$\begin{bmatrix} 26.5 & 4.5 \\ 2.5 & 2.5 \end{bmatrix}$$

Arbitrary resizing

It is obviously more desirable to arbitrarily resize a given image (enlarge or reduce the image proportionally or non-proportionally). We first consider converting a one-dimensional m-sample input signal $\{x_i, (i = 0, 1, \cdots, m-1)\}$ into an n-sample output $\{y_j, (j = 0, 1, \cdots, n-1)\}$, where n may be either smaller or greater than m.



The method is a two-step process of linear interpolation:

- **Convert indices:** Represent each index $0 \le j \le n-1$ for the output as a floating point number p in the range of $0 \le i \le m-1$ for the input:

  $$\frac{p}{m-1} = \frac{j}{n-1}, \quad \text{i.e.} \quad p = j\frac{m-1}{n-1}$$

  The two integer neighbors of p can be found as its floor and ceiling:

  $$i = \lfloor p \rfloor \le p \le \lceil p \rceil = i + 1$$

  where $\lfloor p \rfloor$ and $\lceil p \rceil$ represent, respectively, the floor and the ceiling of p, i.e., the largest integer smaller than p and the smallest integer larger than p.

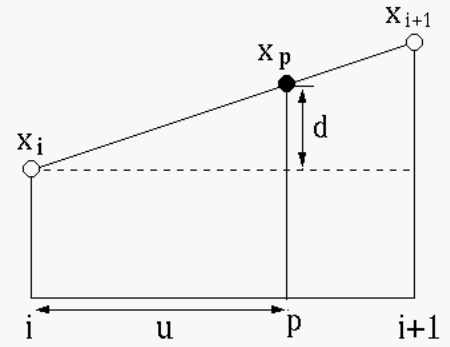
  $$0 \le (u = p - i) \le 1$$

- **Re-sampling:** Find the fraction _____ and note, as shown in the figure,

  $$\frac{d}{u} = \frac{x_{i+1} - x_i}{(i+1) - i} = x_{i+1} - x_i$$

  Now the jth value y_j of the output can be found to be interpolation:

  $$y_j = x_p = x_i + d = x_i + u(x_{i+1} - x_i)$$



The above 1D linear interpolation can be generalized to 2D bilinear interpolation for image resizing.

- **Convert indices:** Similar to the 1D case, we first convert the indices (k, l) of each point in the output image into (p, q) in the range of the input image. Then the corresponding fractions u and v in both dimensions can be found:

$$i = \lfloor p \rfloor,\ i+1 = \lceil p \rceil,\ u = p - i$$

$$j = \lfloor q \rfloor,\ j+1 = \lceil q \rceil,\ v = q - j$$

- **Re-sampling:** Find pixel value x(p,q) as the bilinear interpolation of its four neighbors in the input image, whose gray level values are represented by a, b, c, and d for simplification of the notation:

$$\begin{cases} a = x_{i,j} \\ b = x_{i+1,j} \\ c = x_{i,j+1} \\ d = x_{i+1,j+1} \end{cases}$$

The bilinear interpolation is carried out in two levels of linear interpolations. First we find the interpolation of a, b and c, d:

$$\begin{cases} e = a + u(b-a) \\ f = c + u(d-c) \end{cases}$$

Then we find y(k,l) = x(p,q) as the linear interpolation of e and f:

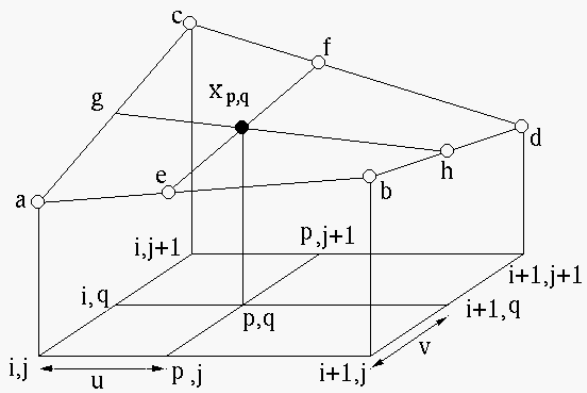
$$y_{k,l} = x_{p,q} = e + v(f-e) = (1-u-v+uv)a + u(1-v)b + v(1-u)a + uvd$$

or, equivalently, we could first find

$$\begin{cases} g = a + v(c-a) \\ h = b + v(d-b) \end{cases}$$

and then find the output pixel:

$$y_{k,l} = x_{p,q} = g + u(h-g) = (1-u-v+uv)a + u(1-v)b + v(1-u)a + uvd$$



Arbitrary rotation

Rotating the input image x by an angle ϕ is equivalent to rotating the output image y by an angle $-\phi$. For the indices (k, l) of each pixel in y we find their position in x:

$$p = \cos(\phi)k + \sin(\phi)l; \qquad q = -\sin(\phi)k + \cos(\phi)l$$

This rotation is about the origin of the image, the top left corner of the image. If it is desired that the rotation is the center (cx, cy) of the image, then

$$p = [\cos(\phi)(k-x0)+\sin(\phi)(l-y0)]+x0; \qquad q = [-\sin(\phi)(k-x0)+\cos(\phi)(l-y0)]+y0$$

Then we find the interpolation value x(p,q) for each pixel y(k,l) of the output image the same way as in the arbitrary scaling discussed above.

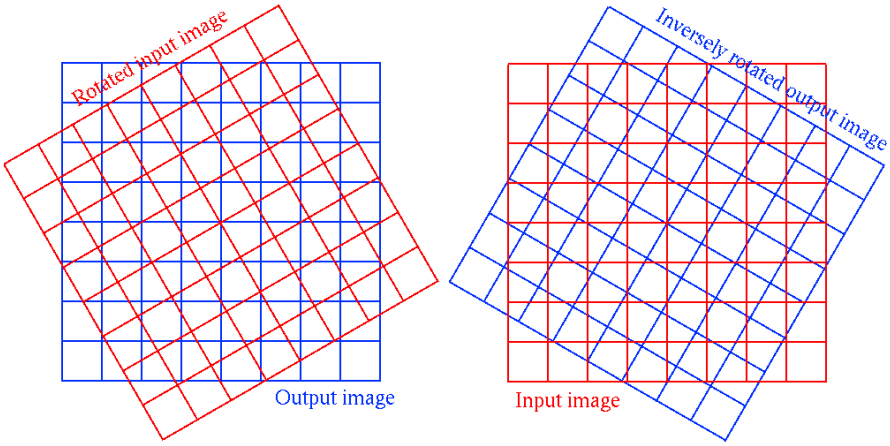
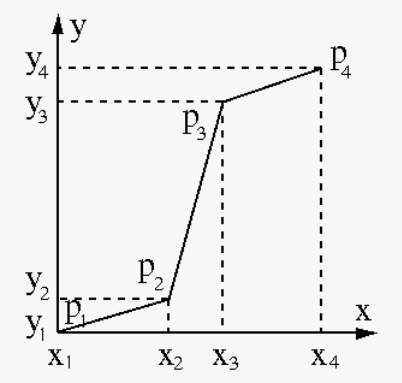


Image Enhancement by Contrast Transform

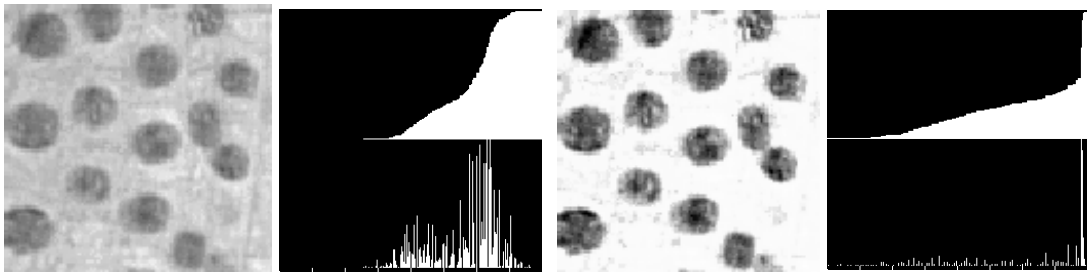
The appearance of an image can be modified according to various needs by a gray level mapping function Y = f(x), where x = x[m,n] is a pixel in the input image and y = y[m,n] is the corresponding pixel in the output image. This mapping function can be specified in different ways, such as a piecewise linear function, or based on the histogram of the input image.

The *histogram* of an image shows the distribution of the pixel values in the image over the dynamic range, typically from 0 to $2^8 - 1 = 255$ for a 8-bit image. The ith item of the histogram is $hist[i] = n_i/N$ (i = 0…255) represents the probability of a randomly chosen pixel has the gray level i, where n_i is the number of pixels of gray level i, and N is the total number of pixels in the image.

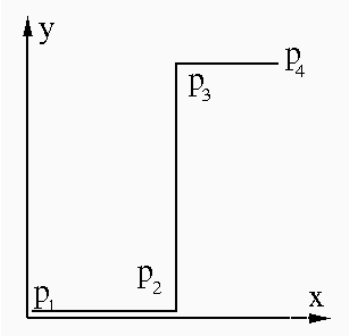
- **Piecewise linear mapping:** A mapping function can be specified by a set of n break points $(x_i, y_j), i = 1, 2, \cdots, n$, with neighboring points connected by straight lines, such as shown here:



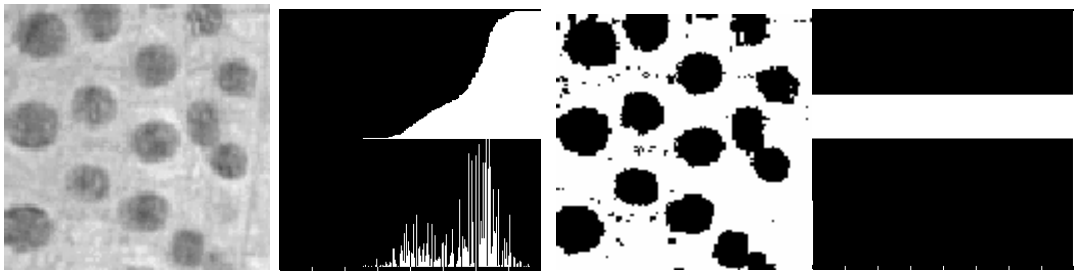
For example, on the left of the image below is a microscopic image of some onion cells. Piecewise linear mapping is applied to stretch the dynamic range for the cells (dark) and to compress the background (bright).



- **Thresholding:**



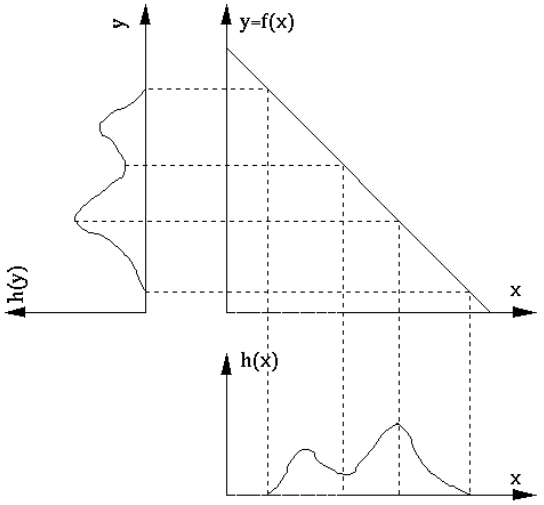
As a special case of piecewise linear mapping, thresholding is a simple way to do image segmentation, in particular, when the histogram of the image is bimodal with two peaks separated by a valley, typically corresponding to some object in the image and the background. A thresholding mapping maps all pixel values below a specified threshold to zero and all above to 255.



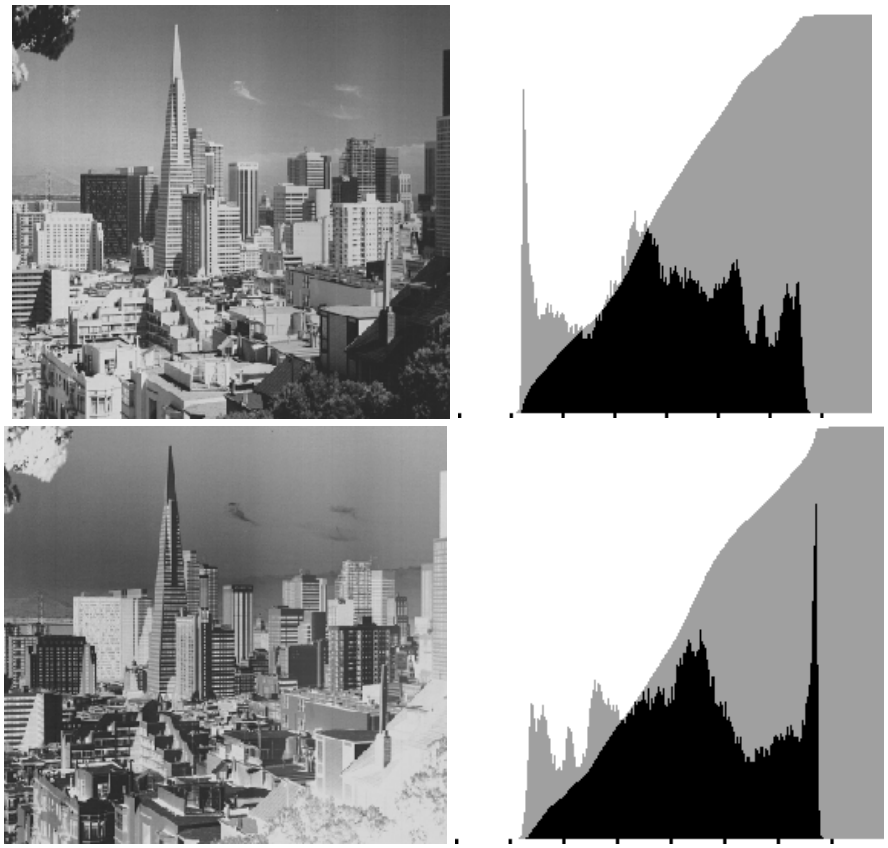
- **Negative image:**

$$y = f(x) = L - 1 - x$$

This mapping is shown below which generates the negative of the input image:



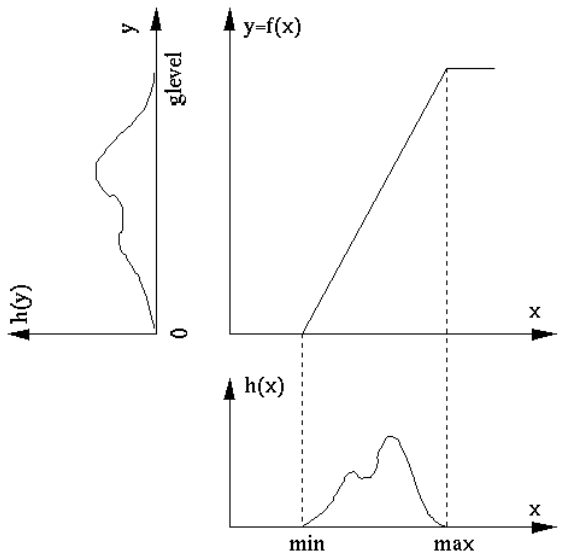
Example:



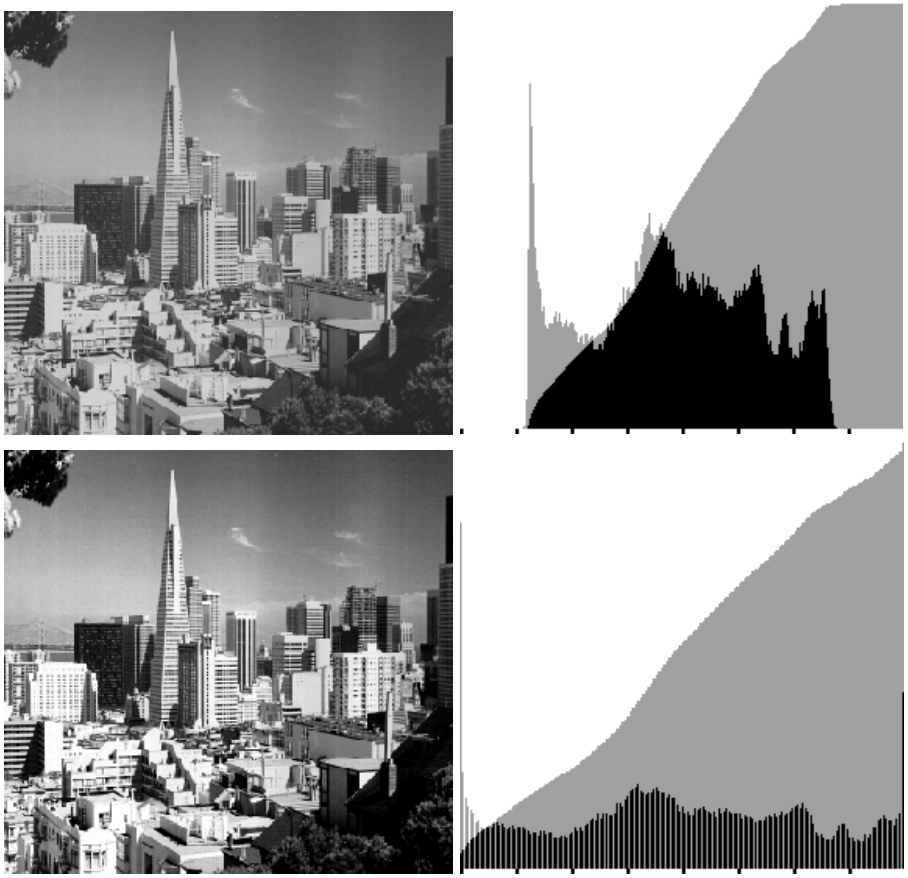
- **Min-max linear stretch:**

$$y = f(x) = \begin{cases} 0 & x < min \\ \frac{x-min}{max-min}(L-1) & min \leq x \leq max \\ L-1 & x > max \end{cases}$$

This is a piecewise linear mapping between the input and output images of three linear segments with slopes 0 for x < min, (L-1)/(max-min) > 1 for min < x < max, and 0 for x > max. The greater than 1 slope in the middle range stretches the dynamic range of the image to use all gray levels available in the display.



Example:



- **Linear stretch based on histogram:**

  If in the image there are only a small number of pixels close to minimum gray level 0 and the maximum gray level L-1 = 255, and the gray level of most of the pixels are concentrated in the middle range (gray) of the histogram, the above linear stretch method based on the minimum and maximum gray levels has very limited effect (as the slope (L-1)/(max-min) is very close to 1). In this case we can push a small percentage (e.g., 3%, 5%) of gray levels close to the two ends of the histogram toward 0 and L-1.