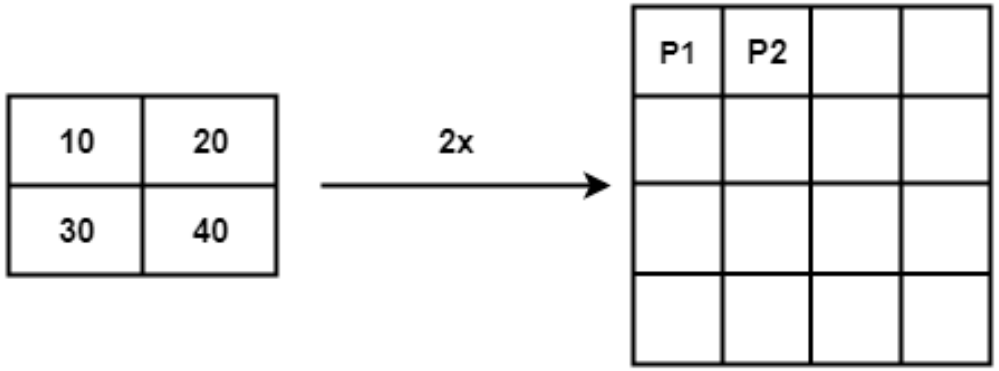
Welcome to Computer Vision

Review: Nearest Neighbor Interpolation

- We assign the unknown pixel to the nearest known pixel.
  - Example: Suppose, we have a 2×2 image and let's say we want to upscale this by a factor of 2 as shown below.



Review: Bi-Linear interpolation

- Let us first review the linear interpolation where more weight is given to the nearest value
  - Assume we want to find the pixel value at locations 1and 2
  - Let us call these points: x and z

$$f(\mathrm{x}) = \frac{x2-x}{x2-x1} \times f(x1) + \frac{x-x1}{x2-x1} \times f(x2)$$
$$= \frac{1}{3} \times 20 + \frac{2}{3} \times 10 = 13.3$$

$$f(\mathrm{z}) = \ldots\ldots\ldots\ldots$$

Image Filtering (spatial domain)

ICS 483 – Computer Vision

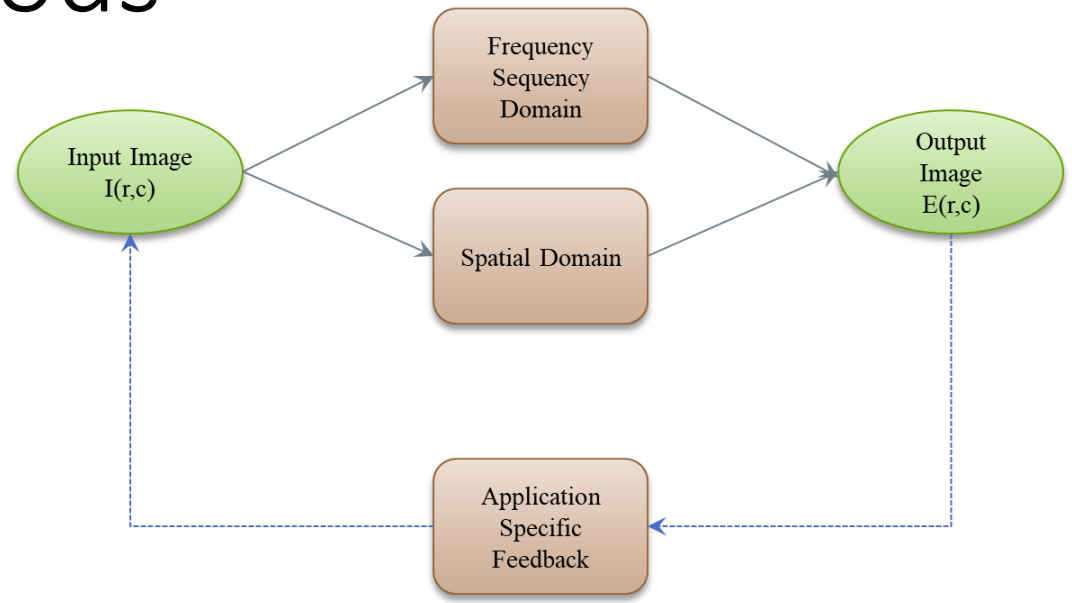
Overview

- Image enhancement
- Histogram processing
- Image filtering (Spatial)
- Image Noise
- Convolution vs. Correlation
- Mathematical morphology
- Image filtering (Frequency)

Image enhancement

- Process an image so that the result will be more suitable than the original image for a specific application.
- The suitableness is up to each application.
- A method which is quite useful for enhancing an image may not necessarily be the best approach for enhancing another image
- Common reasons for enhancement include
  - Improving visual quality,
  - Improving recognition accuracy.

Image enhancement methods



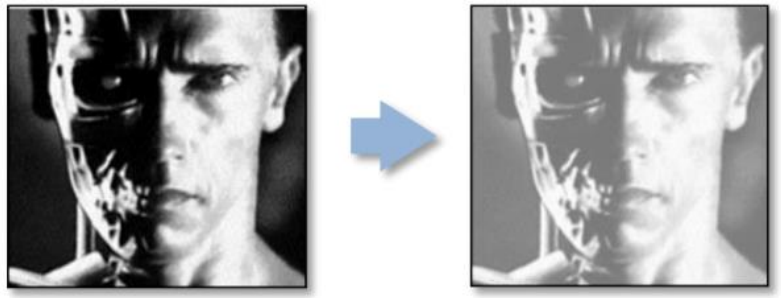
- Enhancement can be done in
  - Spatial domain
  - Frequency domain
- Spatial Domain : (image plane)
  - Techniques are based on direct manipulation of pixels in an image
- Frequency Domain :
  - Techniques are based on modifying the Fourier transform of an image
- There are some enhancement techniques based on various combinations of methods from these two categories.

Image enhancement methods

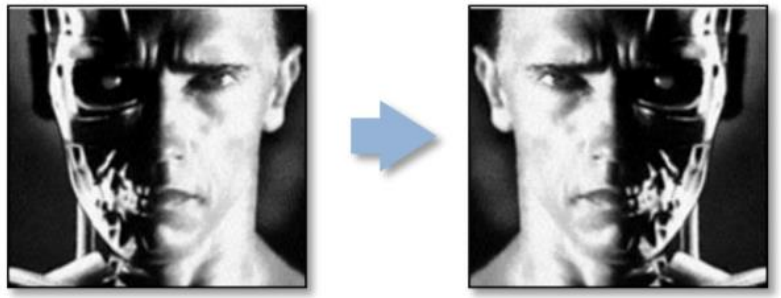
- Spatial domain transformations are:
  - Point operations
  - Filter (Mask) operations

Point operation

- Point operation deals with pixel intensity values individually
- Enhancement at any point depends only on the image value at that point.
- The transformed output pixel value does not depend on any of the neighboring pixel value of the input image
- Point Operation Examples:
  - Image Negative
  - Contrast Stretching
  - Thresholding



$g(x,y) = f(x,y) + 20$



$g(x,y) = f(-x,y)$

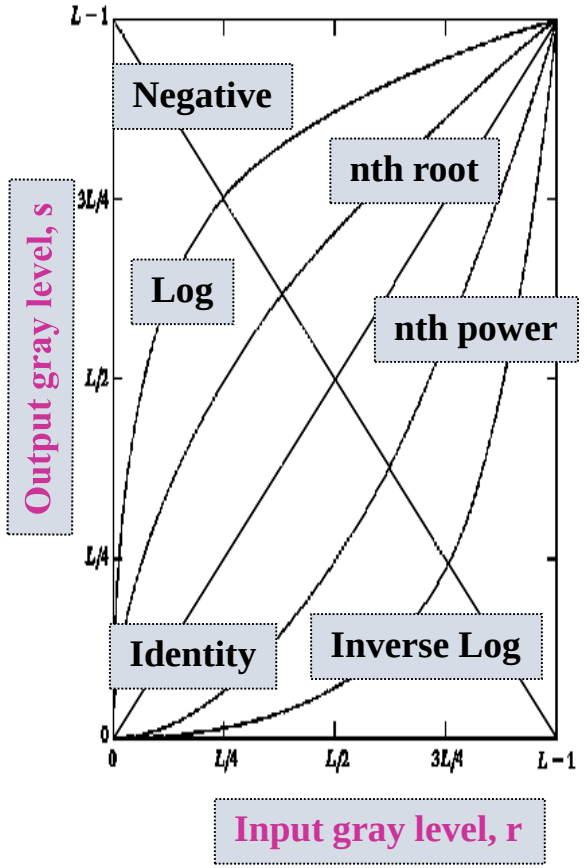
Point operations: Intensity Transformation

- The intensity values are altered using particular transformation techniques as per the requirement.
- Gray-level transformation function

  S=T(r)

  where **r** is the pixels of the input image and **s** is the pixels of the output image.
  T is a transformation function that maps each value of **r** to each value of **s**

- Basic types of transformation functions that are used frequently in image enhancement:
  - Linear, Logarithmic: $s = c \log(r + 1)$, Power-Law: $S = C\, r^{\gamma}$



Some Basic Intensity Transformation Functions

Image Negatives

Log Transformations

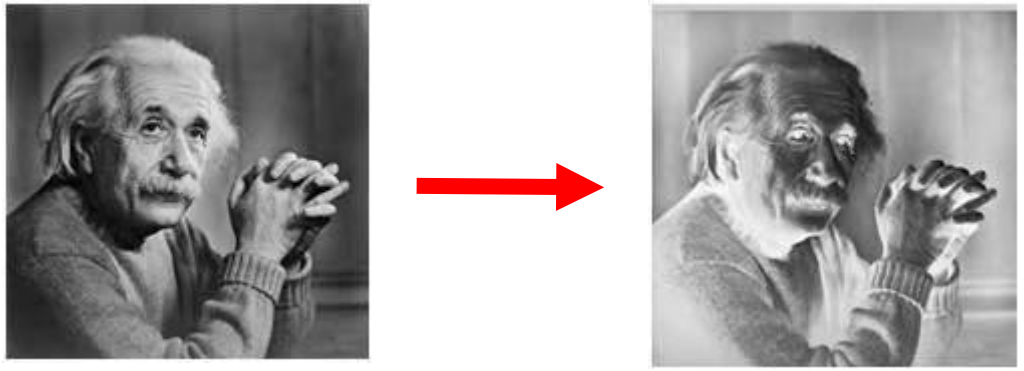
Power-Law(Gamma) Transformations

Piecewise-Linear Transformation Functions

Image Negatives

- Reversing the intensity levels of an image.
- Image negative is produced by subtracting each pixel from the maximum intensity value. e.g. for an 8-bit image, the max intensity value is $2^8 - 1 = 255$
- An image with gray level in the range ***[0, L-1]*** where $L = 2^n$; n = 1, 2…
- Negative transformation : $s = L - 1 - r$

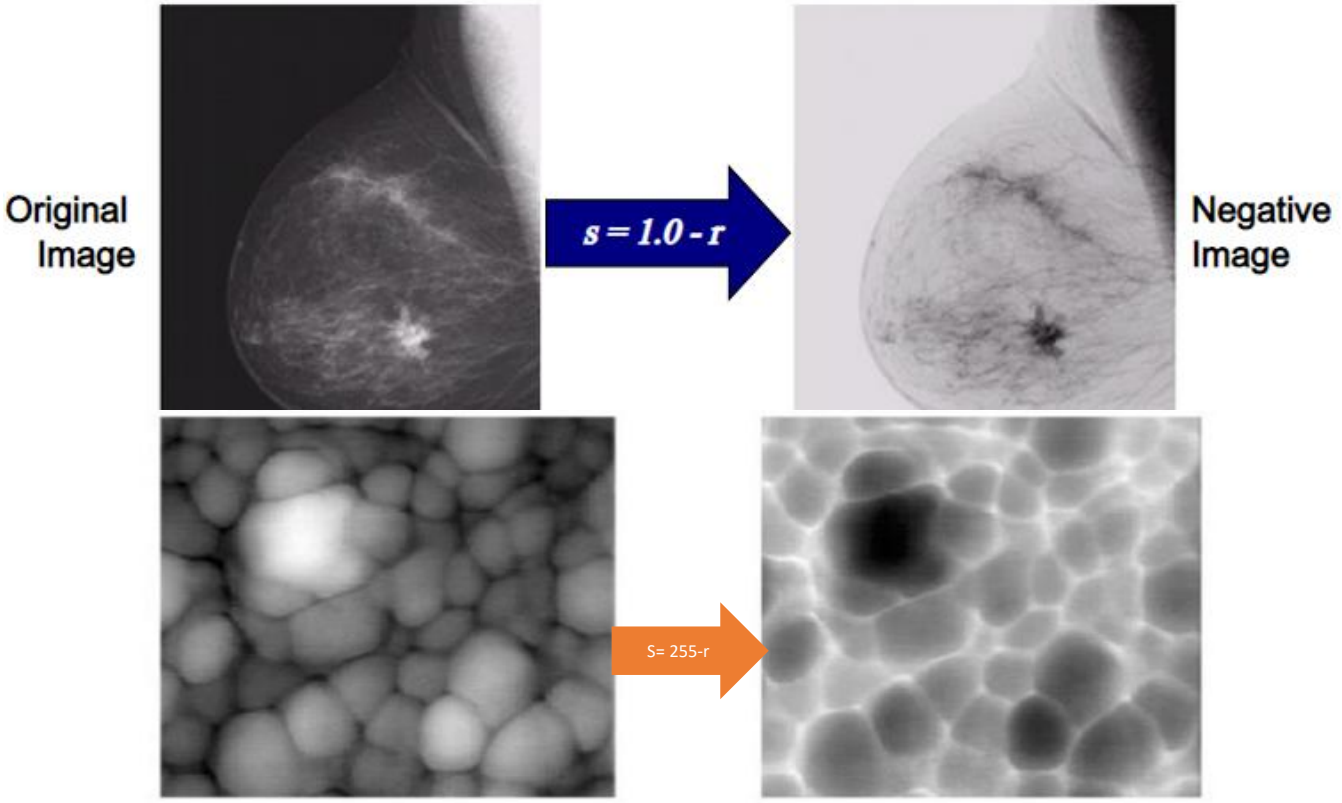
Image Negatives



- Reversing the intensity levels of an image.
- Image negative is produced by subtracting each pixel from the maximum intensity value. e.g. for an 8-bit image, the max intensity value is $2^8 - 1 = 255$
- An image with gray level in the range ***[0, L-1]*** where $L = 2^n$; n = 1, 2…
- Negative transformation : $s = L - 1 - r$

Image Negatives: Applications

- Suitable for enhancing white or gray detail embedded in dark regions of an image.



Some Basic Intensity Transformation Functions

Image Negatives

Log Transformations

Power-Law(Gamma) Transformations

Piecewise-Linear Transformation Functions

Log Transformations

- Log curve maps a narrow range of low gray-level values in the input image into a wider range of output levels.
- c is a constant and $r \geq 0$
- Used to expand the values of dark pixels in an image while compressing the higher-level values.

$$s = c \log(r + 1)$$

Some Basic Intensity Transformation Functions

Image Negatives

Log Transformations

Power-Law (Gamma) Transformations

Piecewise-Linear Transformation Functions

Power-Law (Gamma) Transformations

- Gamma correction function is used to correct image's luminance.
- Power-law transformations: $s=cr^{\gamma}$ **or** $s=c(r+\varepsilon)^{\gamma}$
  - γ : gamma, gamma correction
  - γ **<1** maps a narrow range of dark input values into a wider range of output values [increase brightness]
  - γ **>1** maps a narrow range of bright input values into a wider range of output values [increase darkness]

Power-Law (Gamma) Transformations

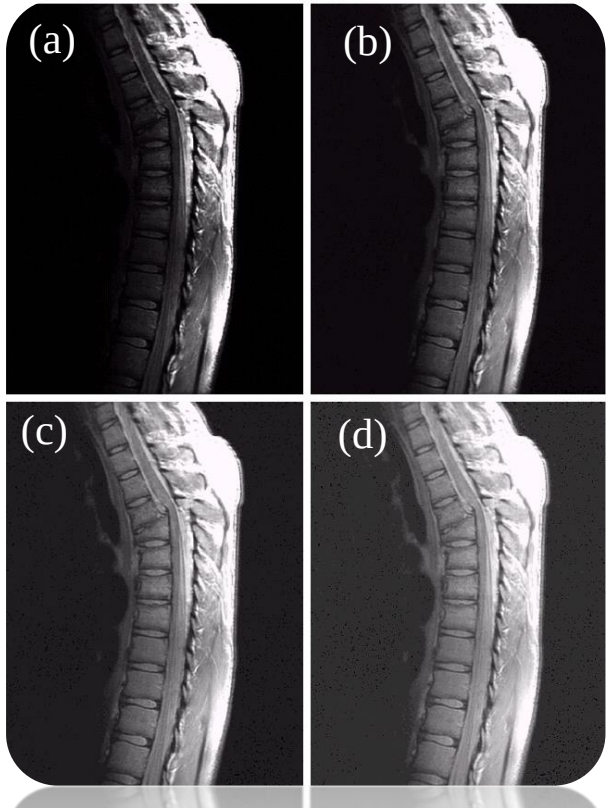
a) A magnetic resonance image (MRI) of an upper thoracic human spine with a fracture dislocation and spinal cord impingement

- The picture is predominately dark
- An expansion of gray levels are desirable ⇨ needs $\gamma < 1$

b) result after power-law transformation with $\gamma = 0.6$, c=1

c) transformation with $\gamma = 0.4$ (best result)

d) transformation with $\gamma = 0.3$ (under acceptable level)



Some Basic Intensity Transformation Functions

Image Negatives

Log Transformations

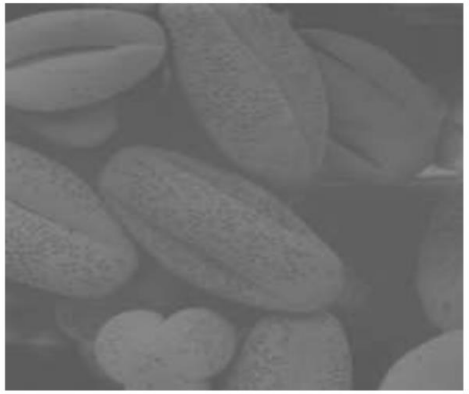
Power-Law (Gamma) Transformations

Piecewise-Linear Transformation Functions

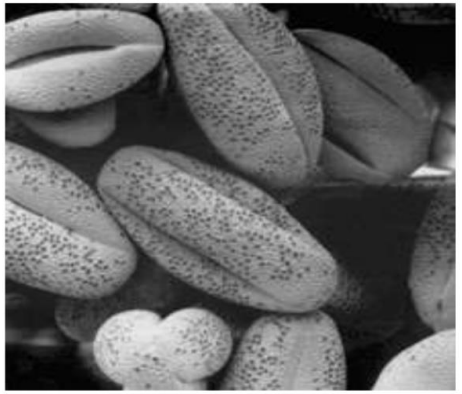
Piecewise-Linear Transformation Functions

- Some commonly used piece-wise linear transformations are:
  - Contrast Stretching
  - Gray-level Slicing
  - Clipping

Contrast Stretching



Original Image

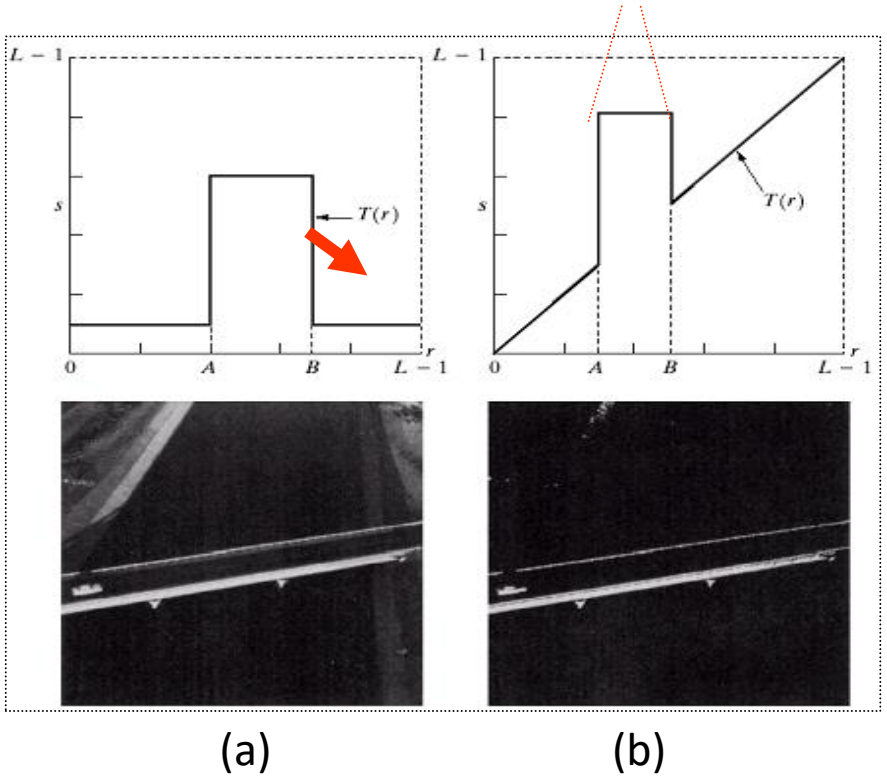


Contrast Enhanced Image

- It is the difference between the intensity values of darker and brighter pixels
- Contrast stretching expands the range of intensity levels in an image.
- Contrast stretching is done in three ways:

1. Multiplying each input pixel intensity value with a constant scalar.
   - Example: s=2*r
2. Using Histogram Equivalent
3. Applying a transform which makes dark portion darker by assigning slope of < 1 and bright portion brighter by assigning slope of > 1.

Gray-level Slicing

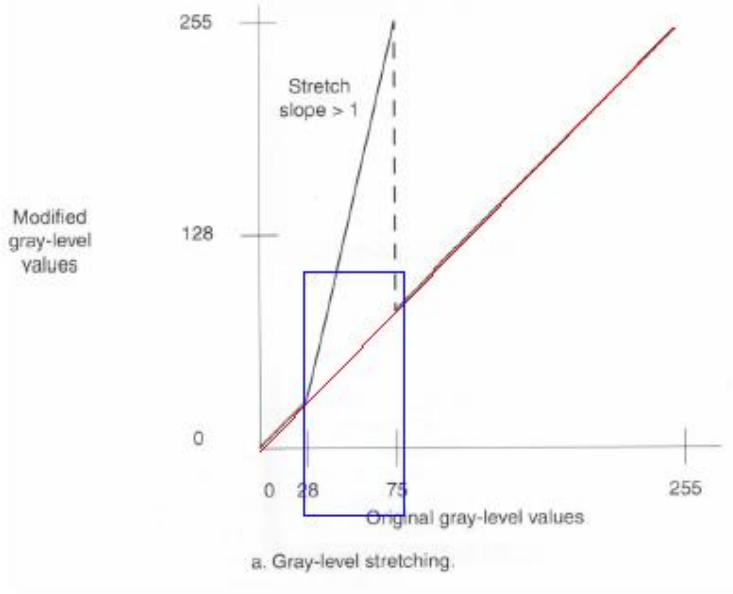


(a) (b)

- Highlighting a specific range of gray levels in an image
- Display a high value of all gray levels in the range of interest and a low value for all other gray levels
  - (a) transformation highlights range [A,B] of gray level and reduces all others to a constant level
  - (b) transformation highlights range [A,B] but preserves all other levels

Gray-level Slicing Example

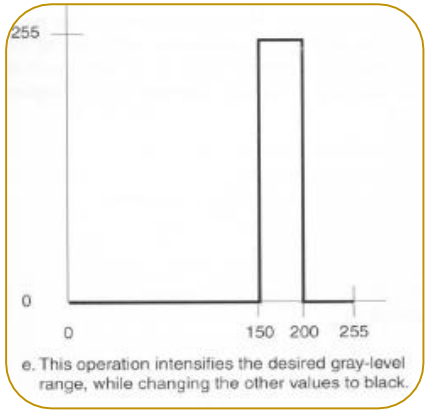
b. Original image.



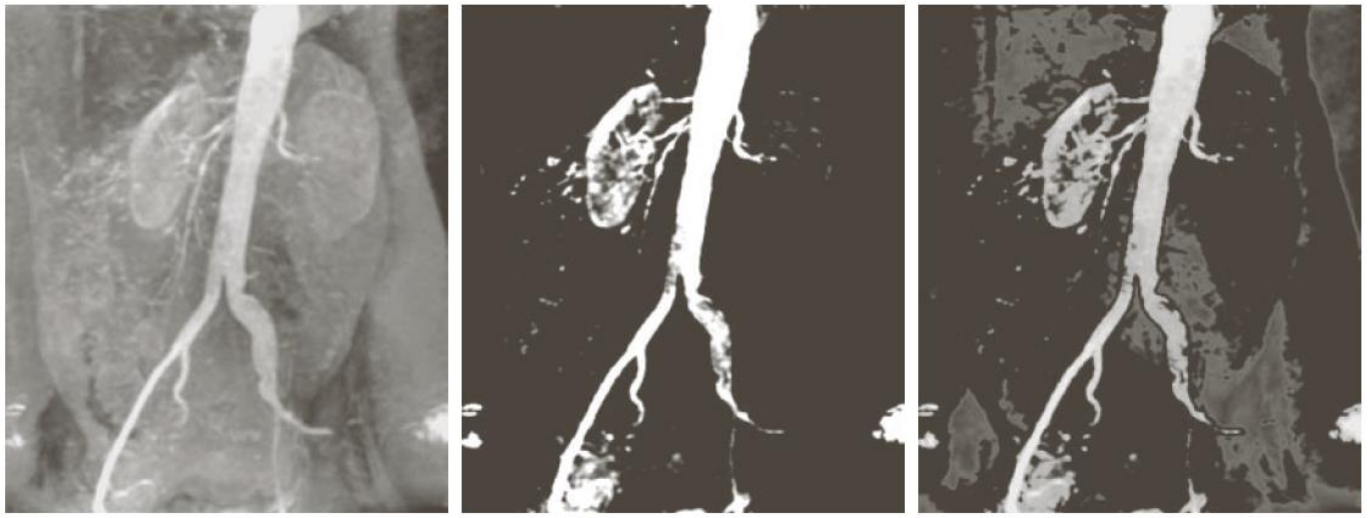
a. Gray-level stretching.

c. Image after modification.

Gray-level Slicing Example

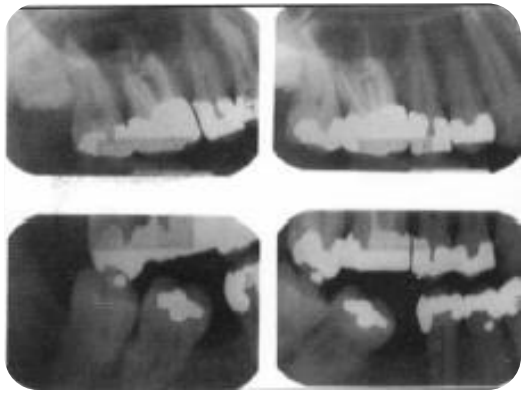


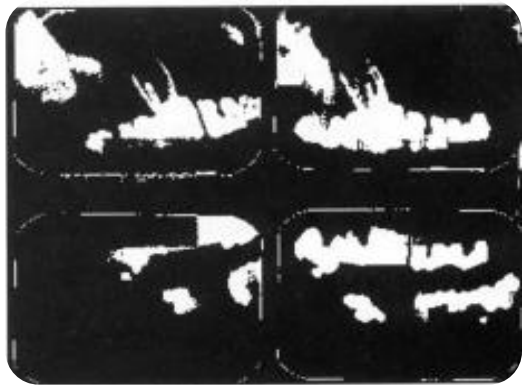
e. This operation intensifies the desired gray-level range, while changing the other values to black.



a b c

FIGURE 3.12 (a) Aortic angiogram. (b) Result of using a slicing transformation of the type illustrated in Fig. 3.11(a), with the range of intensities of interest selected in the upper end of the gray scale. (c) Result of using the transformation in Fig. 3.11(b), with the selected area set to black, so that grays in the area of the blood vessels and kidneys were preserved. (Original image courtesy of Dr. Thomas R. Gest, University of Michigan Medical School.)



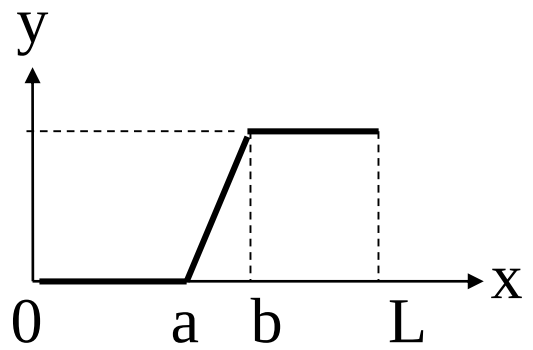


Clipping

- This function truncates all intensities below the optional value Min to Min and all intensities above the optional value Max to Max.

- Clipping can be used to remove unwanted features, noise, or extraneous information from an image.

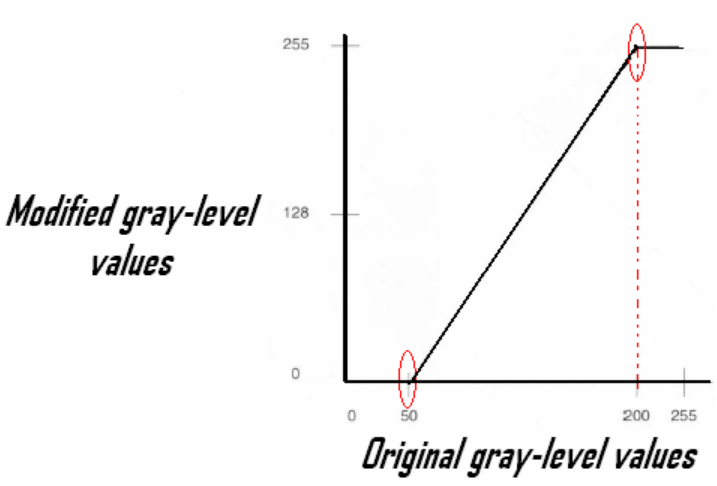
$$y = \begin{cases} 0 & 0 \le x < a \\ \beta(x-a) & a \le x < b \\ \beta(b-a) & b \le x < L \end{cases}$$



$a = 50, b = 150, \beta = 2$

Clipping Example

e. Original image.



f. Image after modification.

Piecewise-Linear Transformation Functions

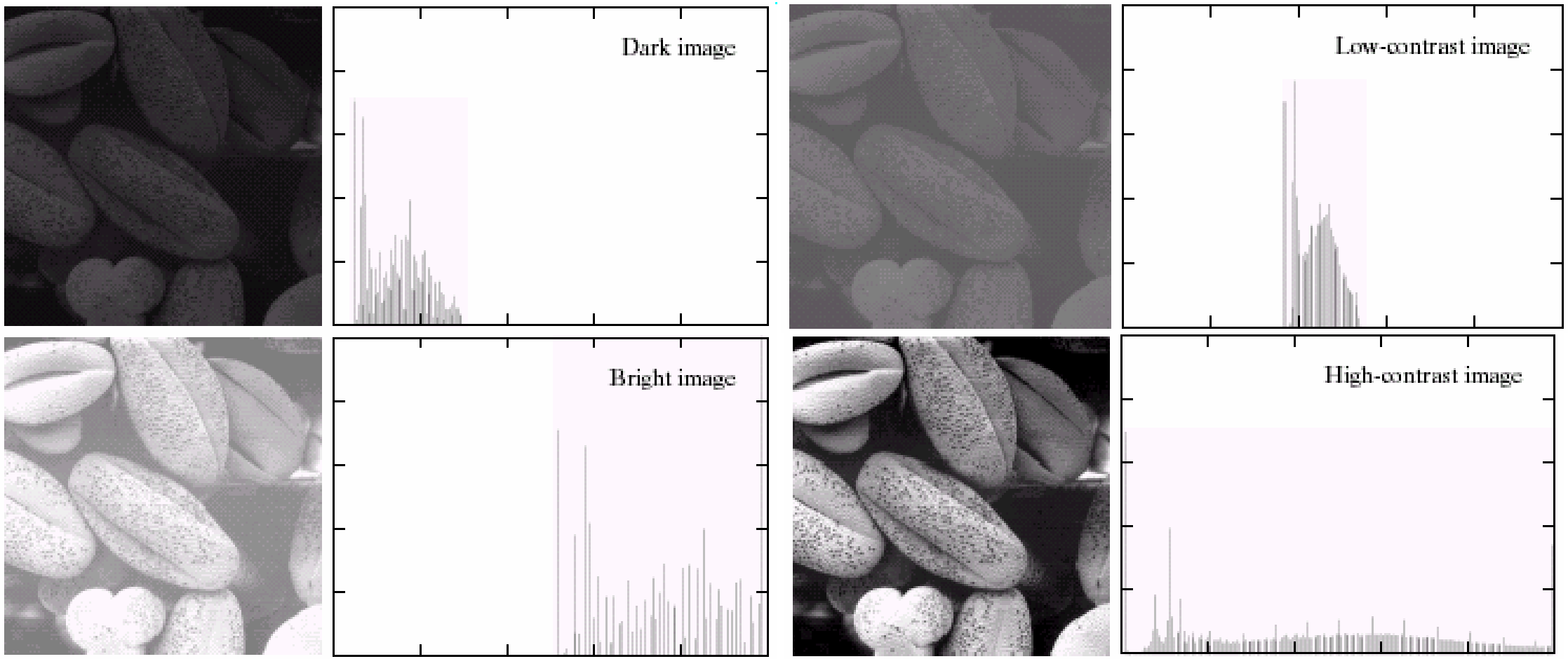
- **Advantage:**
  - The form of piecewise functions can be arbitrarily complex.
  - A Practical Implementation of some important transformations can be formulated only as piecewise functions.
- **Disadvantage:**
  - Their specification requires considerably more user input.

Histogram Processing

- Intuitively, we expect that an image whose pixels tend to occupy the entire range of possible gray levels, tend to be distributed uniformly will have a high contrast and show a great deal of gray level detail.

- It is possible to develop a transformation function that can achieve this effect using histograms.

Histogram processing



Histogram equalization

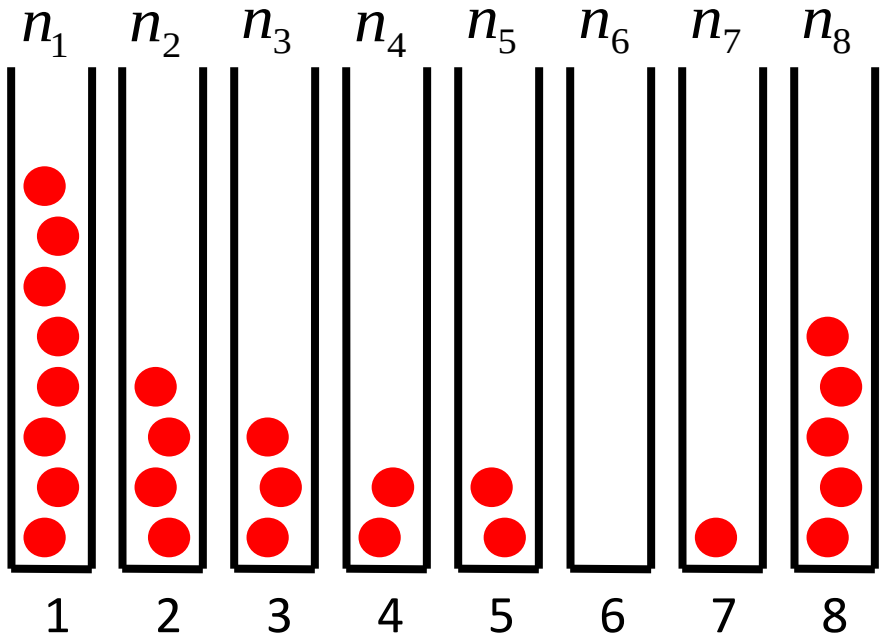
- Histograms are functions describing information extracted from the image.
- In a process called histogram equalization, this information is used to compute a contrast increasing intensity transformation function.
- Histogram equalization transforms the intensity values so that the histogram of the output image approximately matches the flat (uniform) histogram.

Histogram function

- The histogram function is defined over all possible intensity levels.
  - For each intensity level, its value is equal to the number of the pixels with that intensity.

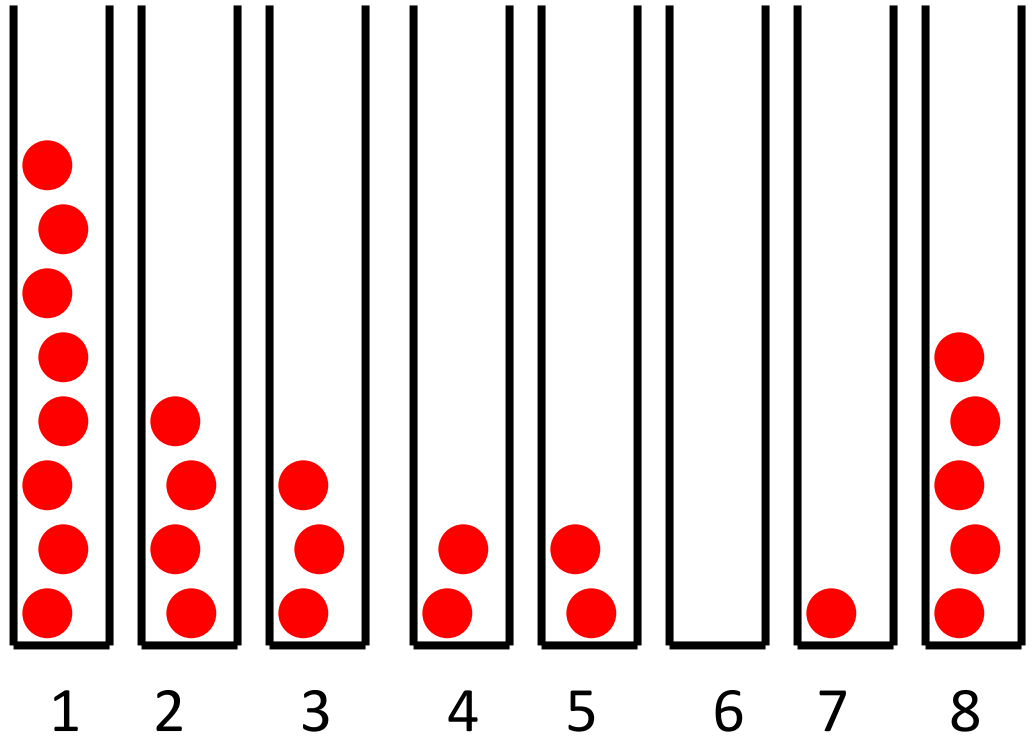
$$h(r_k) = n_k$$

$$h(r_1) = 8$$
$$h(r_2) = 4$$
$$h(r_3) = 3$$
$$h(r_4) = 2$$
$$h(r_5) = 2$$
$$h(r_6) = 0$$
$$h(r_7) = 1$$
$$h(r_8) = 5$$



Histogram function example

1	8	4	3	4
1	1	1	7	8
8	8	3	3	1
2	2	1	5	2
1	1	8	5	2



Normalised histogram function

- The normalized histogram function is the histogram function divided by the total number of the pixels of the image:

$$p(r_k) = \frac{h(r_k)}{n} = \frac{n_k}{n}$$

- It gives a measure of how likely is for a pixel to have a certain intensity. That is, it gives the probability of occurrence the intensity.

- The sum of the normalized histogram function over the range of all intensities is 1.

Normalised histogram function example

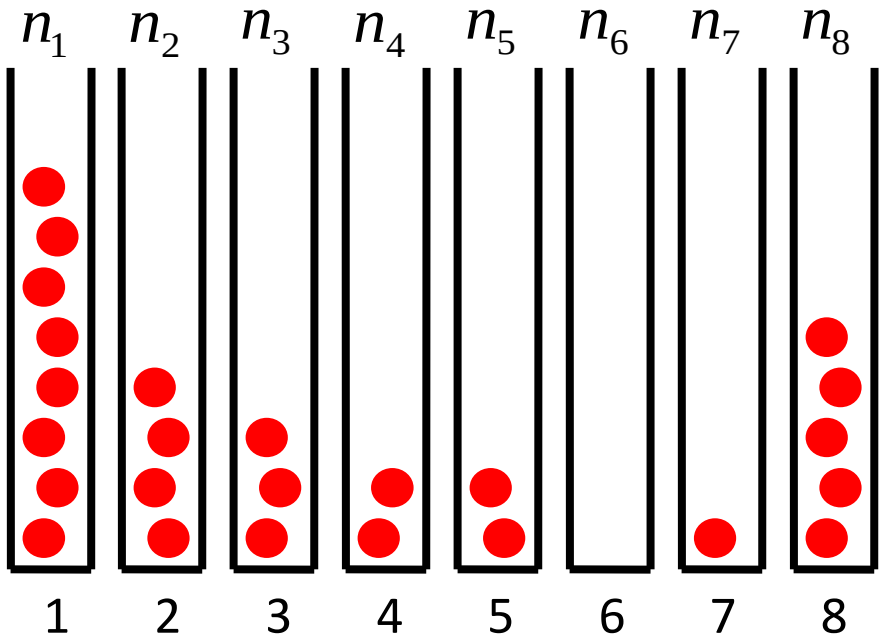
1	8	4	3	4
1	1	1	7	8
8	8	3	3	1
2	2	1	5	2
1	1	8	5	2

$$h(r_1) = 8$$
$$h(r_2) = 4$$
$$h(r_3) = 3$$
$$h(r_4) = 2$$
$$h(r_5) = 2$$
$$h(r_6) = 0$$
$$h(r_7) = 1$$
$$h(r_8) = 5$$

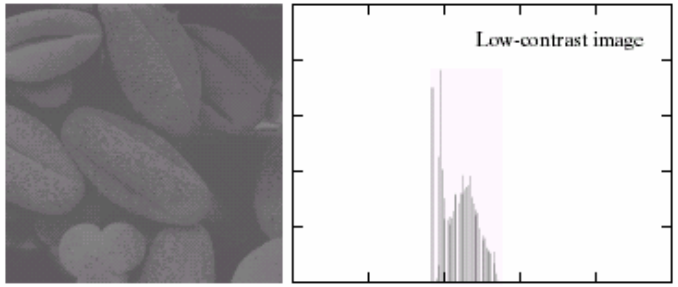
Histogram function

$$p(r_1) = 8/25 = 0.32$$
$$p(r_2) = 4/25 = 0.16$$
$$p(r_3) = 3/25 = 0.12$$
$$p(r_4) = 3/25 = 0.08$$
$$p(r_5) = 2/25 = 0.08$$
$$p(r_6) = 0/25 = 0.00$$
$$p(r_7) = 1/25 = 0.04$$
$$p(r_8) = 5/25 = 0.20$$

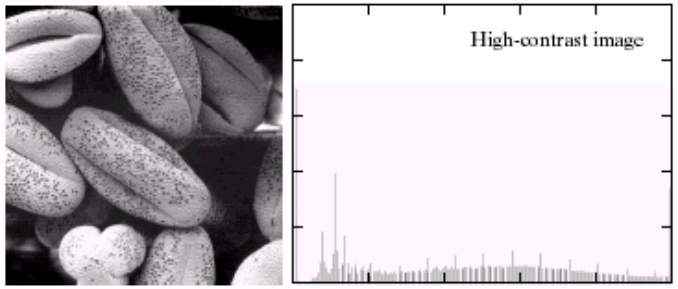
Normalized histogram function

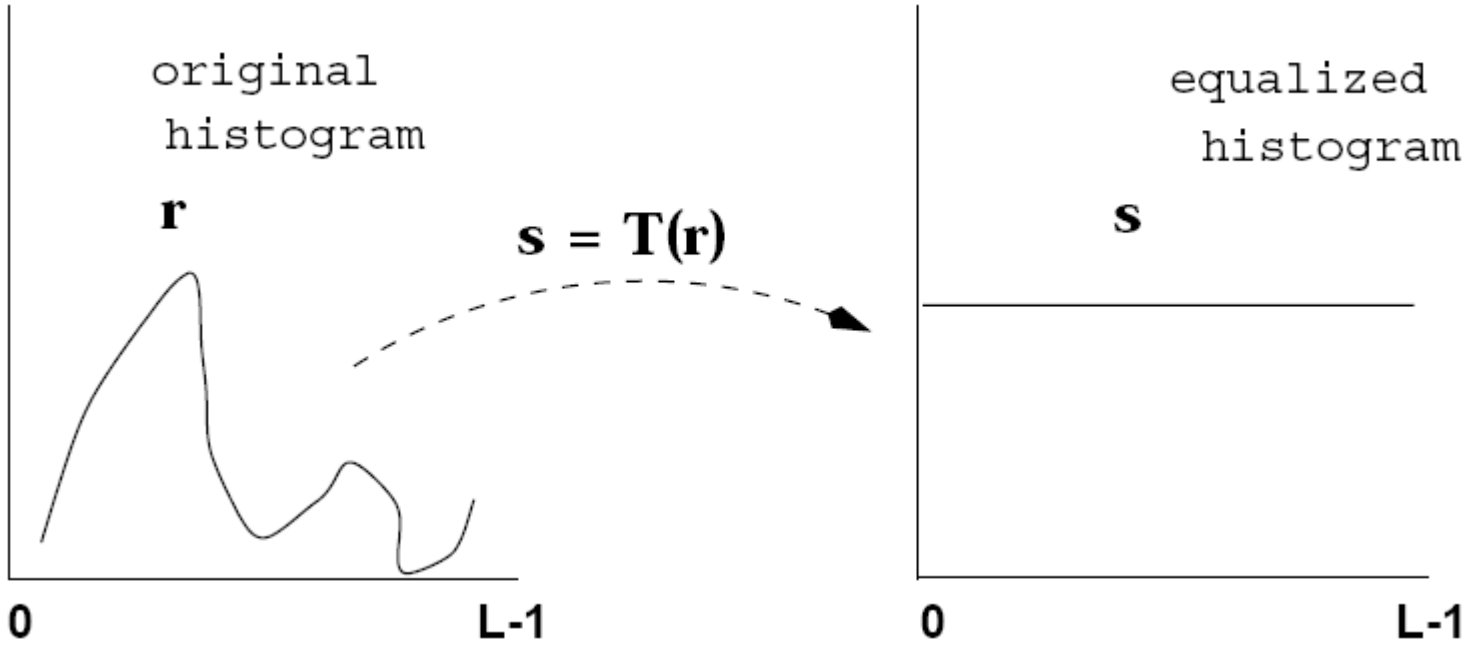


Histogram Equalization



- The main idea is to redistribute the gray-level values uniformly.





Histogram equalization algorithm

- Let $r_k, k = 1,2,...,m$ be the intensities of the image, and let $p(r_k)$ be its normalized histogram function.

- The intensity transformation function for histogram equalization is:

$$T(r_k) = \sum_{j=1}^{k} p(r_k)$$

That is, we add the values of the normalized histogram function from 1 to k to find where the intensity r_k will be mapped.

- Notice that the range of the equalized image is the interval [0,1].

Histogram equalization example

- Histogram equalization algorithm

0	0	1	0	2	0
1	0	7	7	7	0
0	7	0	0	7	0
1	0	0	7	2	0
0	0	7	1	0	1
1	0	7	7	7	0

frequencies

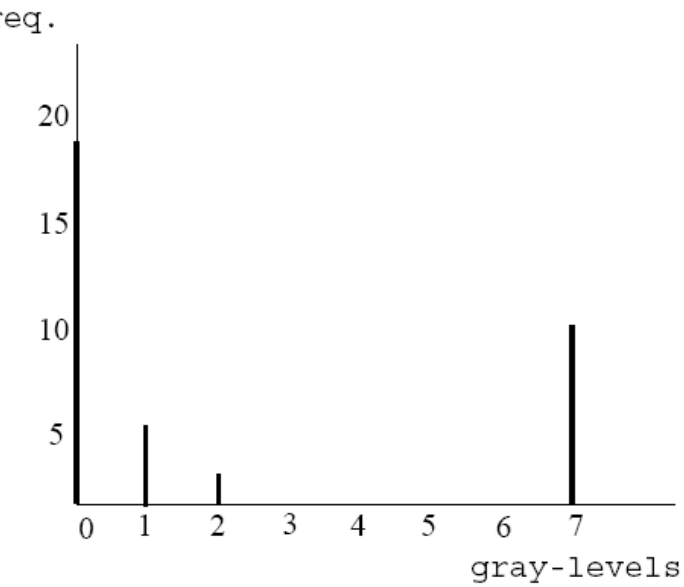
f(0)= 18

f(1)= 6

f(2)= 2

f(3)=f(4)=f(5)=f(6) = 0

f(7) = 10



- Divide frequencies by total number of pixels to represent as probabilities.

$$p_k = n_k / N$$

$$P(0) = \frac{f(0)}{36} = \frac{1}{2} \qquad P(1) = \frac{f(1)}{36} = \frac{1}{6}$$

$$P(2) = \frac{f(2)}{36} = \frac{1}{18} \qquad P(3) = P(4) = P(5) = P(6) = 0$$

$$P(7) = \frac{f(7)}{36} = \frac{5}{18}$$

Histogram equalization example

$$T(r_k) = \sum_{j=1}^{k} p(r_k)$$

$$P(0) = \frac{f(0)}{36} = \frac{1}{2} \qquad P(1) = \frac{f(1)}{36} = \frac{1}{6}$$

$$P(2) = \frac{f(2)}{36} = \frac{1}{18} \qquad P(3) = P(4) = P(5) = P(6) = 0$$

$$P(7) = \frac{f(7)}{36} = \frac{5}{18}$$

- Compute intensity transformation function

$T(r_0) = \sum_{j=0}^{0} p_r(r_k) = p_r(r_0) = \frac{1}{2} = 0.5$

$T(r_1) = \sum_{j=0}^{1} p_r(r_k) = \frac{1}{2} + \frac{1}{6} = 0.5 + 0.17 = 0.66$

$T(r_2) = \sum_{j=0}^{2} p_r(r_k) = \frac{1}{2} + \frac{1}{6} + \frac{1}{18} = 0.5 + 0.17 + 0.06 = 0.72$

$T(r_3) = T(r_4) = T(r_5) = T(r_6) = 0.72$

$T(r_7) = \sum_{j=0}^{7} p_r(r_k) = 0.72 + \frac{5}{18} = 1.01$

- Multiple intensity transformation function by intensity level – 1

$S_0 = (L-1) * T(r_0) = 7*0.5 = \lfloor 3.5 \rfloor = 3$

$S_1 = (L-1) * T(r_1) = 7*0.66 = \lfloor 4.6 \rfloor = 4$

$S_2 = (L-1) * T(r_2) = 7*0.72 = \lfloor 5.04 \rfloor = 5$

$S_3 = S_4 = S_5 = S_6 = 7*0.72 = \lfloor 5.04 \rfloor = 5$

$S_7 = (L-1) * T(r_7) = 7*1.01 = \lfloor 7.01 \rfloor = 7$

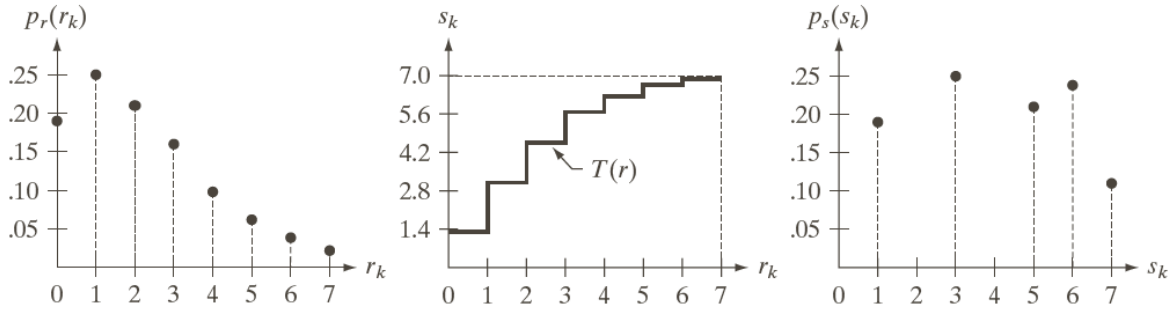
Histogram equalization (another example)

Q: Given a greyscale image (3-bit) with 64x64 resolution. Do histogram equalization given the following information.

$S_0 = T(r_0) = 7 \sum_{j=0}^{0} p_r(r_k) = 7 * p_r(r_0) = 7*0.19 = \lfloor 1.4 \rfloor = 1$

$S_1 = T(r_1) = 7 \sum_{j=0}^{1} p_r(r_k) = 7 * 0.44 = \lfloor 3.08 \rfloor = 3$

-
-
-



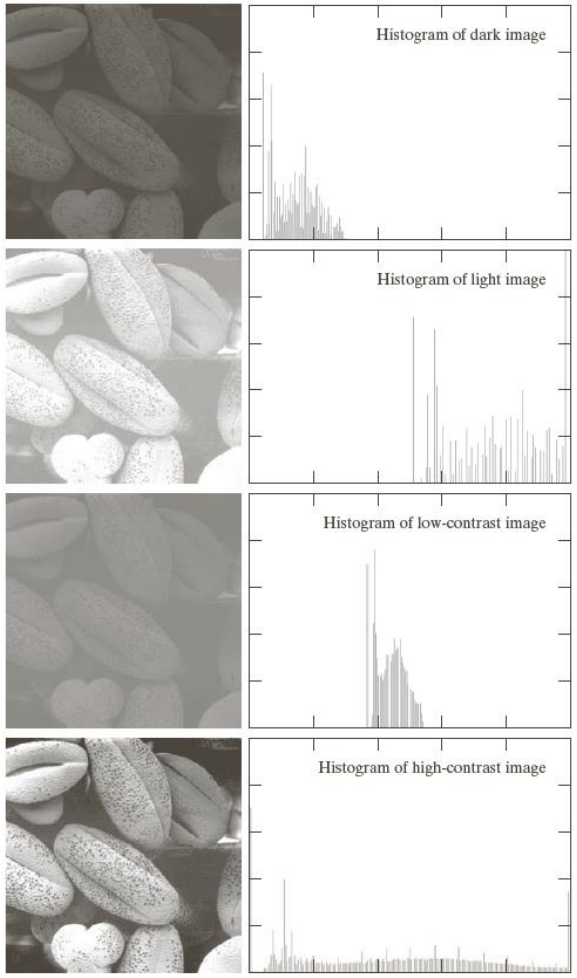
a b c Input histogram Equalized histogram

FIGURE 3.19 Illustration of histogram equalization of a 3-bit (8 intensity levels) image. (a) Original histogram. (b) Transformation function. (c) Equalized histogram.

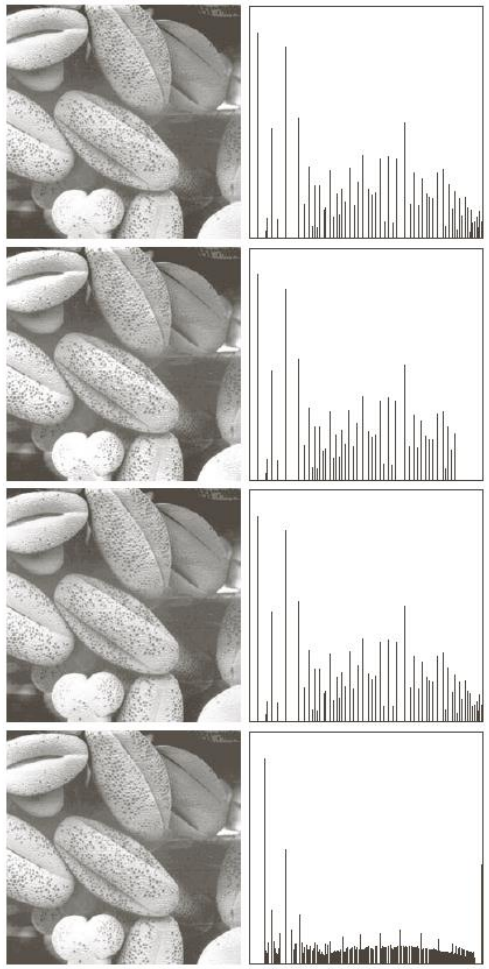
r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Click for More Details

Histogram equalization examples



Original images and histograms



Equalized images and histograms

Histogram equalization examples

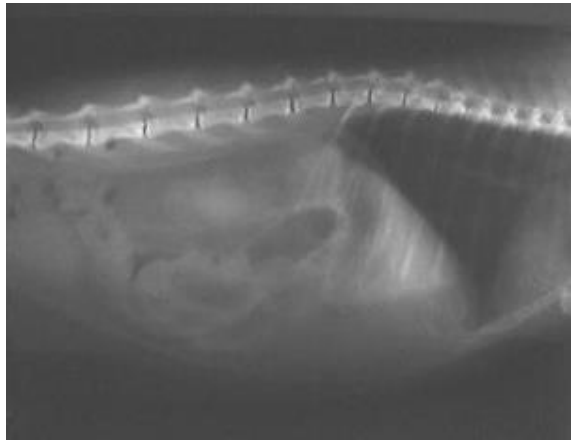
Before

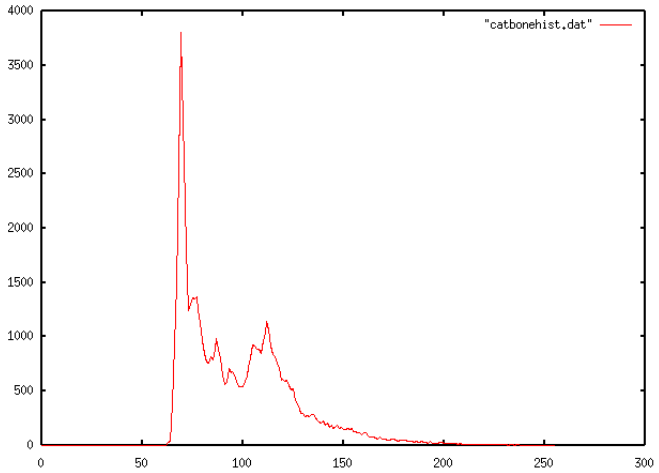
After

Source: https://medium.com/@animeshsk3/back-to-basics-part-1-histogram-equalization-in-image-processing-f607f33c5d55

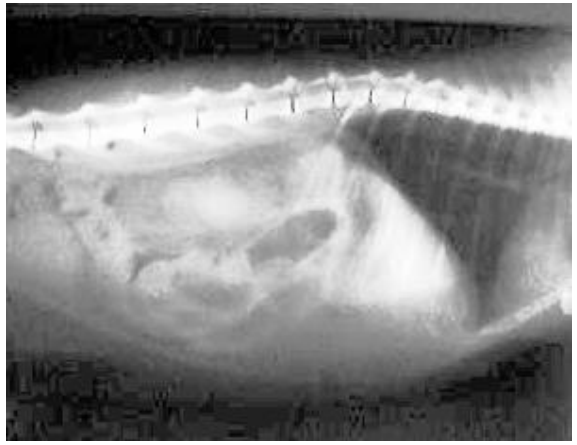
Histogram equalization examples

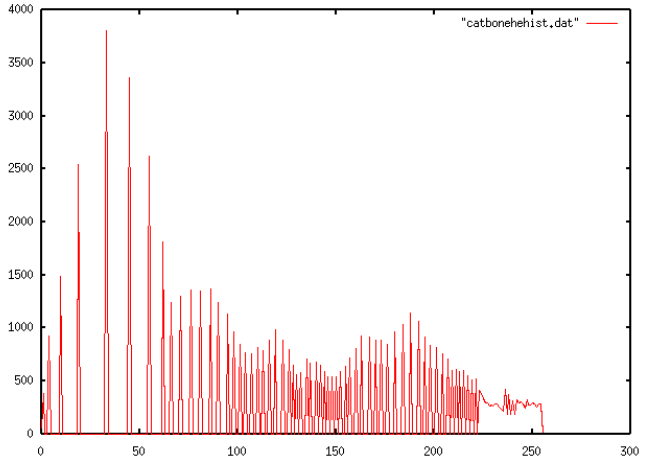
Low contrast





High contrast

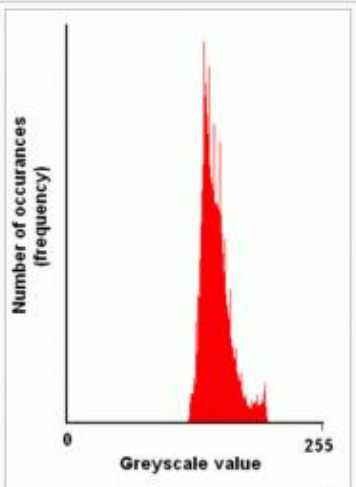




Histogram equalization examples

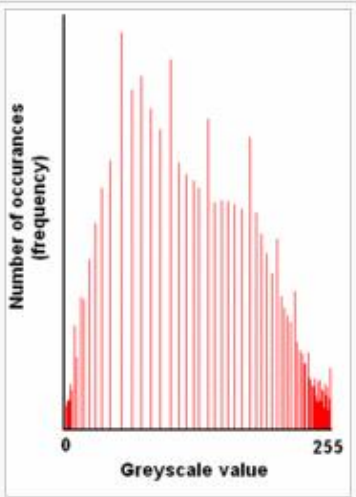
Adapted from Wikipedia

An unequalized image



Corresponding histogram

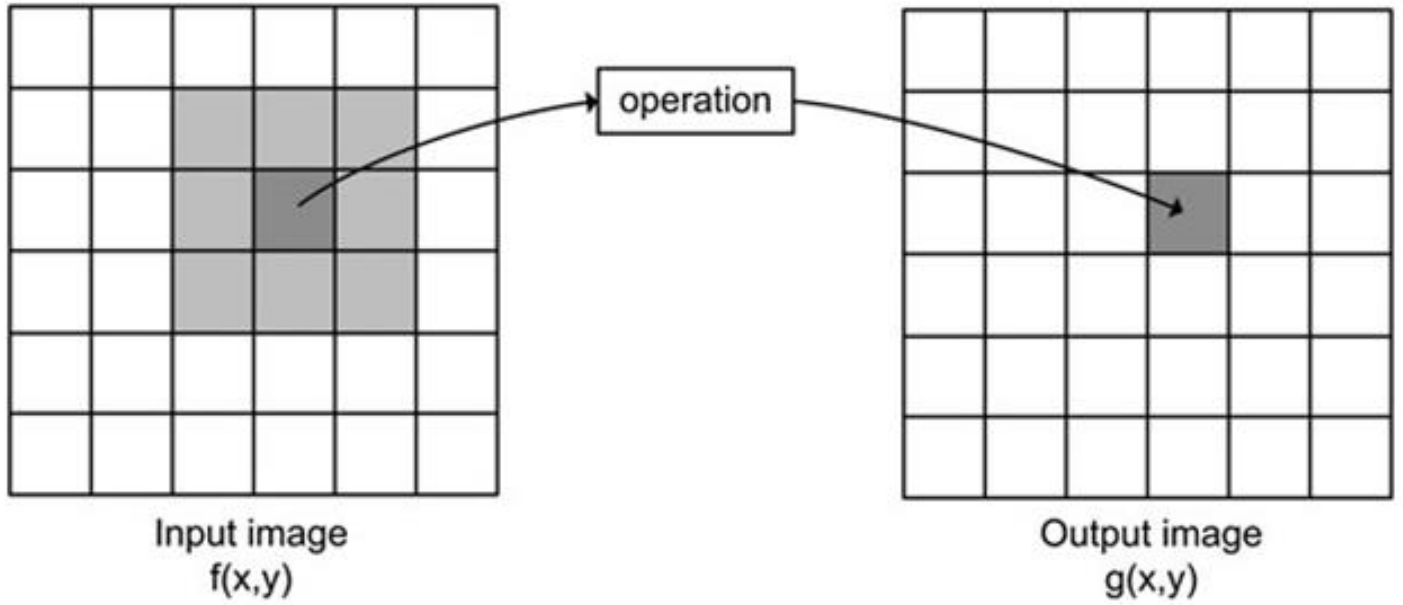
Same image after histogram equalization



Corresponding histogram

Filter (Mask) Operations

Spatial domain filtering



- Capabilities of point operations are limited
- Filters: combine pixel's value + values of neighbors

What Point Operations Can’t Do

- Combining multiple pixels needed for certain operations:
  - Enhance an image, e.g., denoise, Blurring, Sharpening.
  - Extract information, e.g., texture, edges.
  - Detect patterns, e.g., template matching.

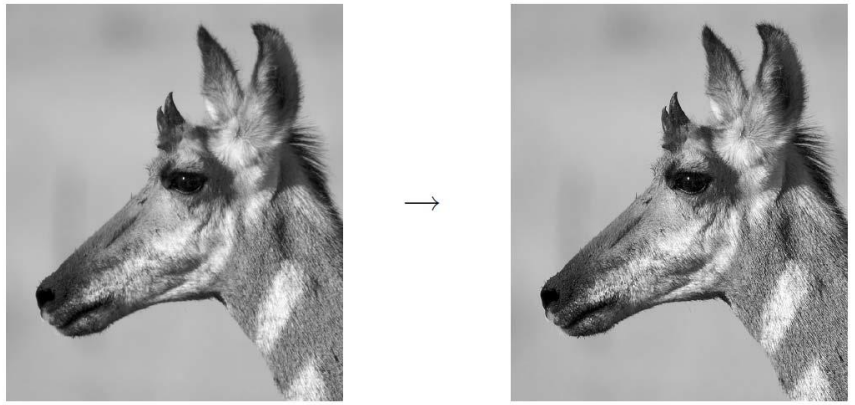


Image Noise

Noise in image , is any degradation in an image signal , caused by external disturbance while an image is being sent from one place to another place via satellite , wireless and network cable .

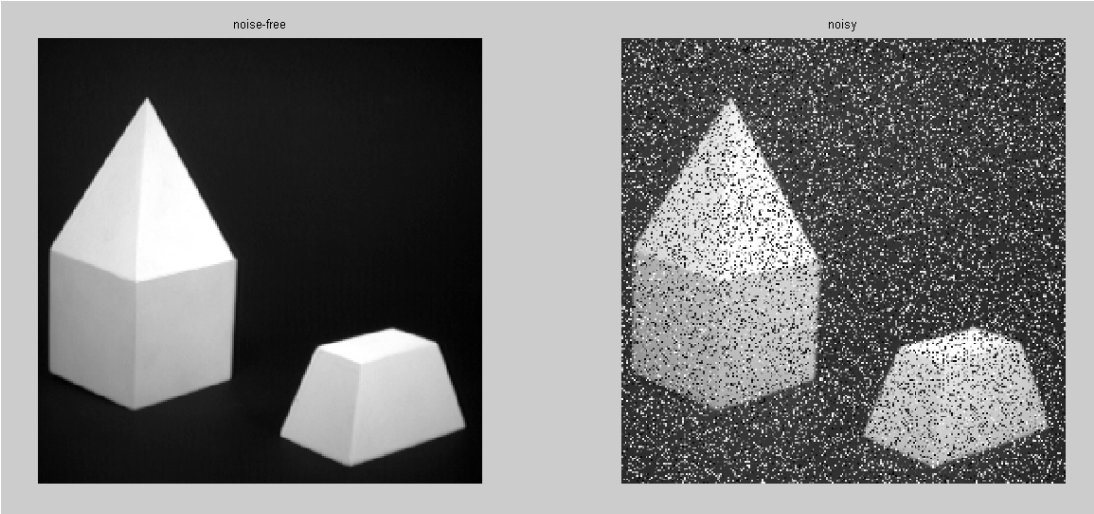
- Noise sources
  - Light Variations
  - Camera Electronics
  - Surface Reflectance
  - Lens

- Let f(i,j) be the true pixel values and n(i,j) be the noise added to the pixel (i,j)
  - Additive noise: $f^{\wedge}(i,j) = f(i,j) + n(i,j)$
  - Multiplicative noise: $f^{\wedge}(i,j) = f(i,j) * n(i,j)$

Image processing is useful for noise reduction

Common types of noise

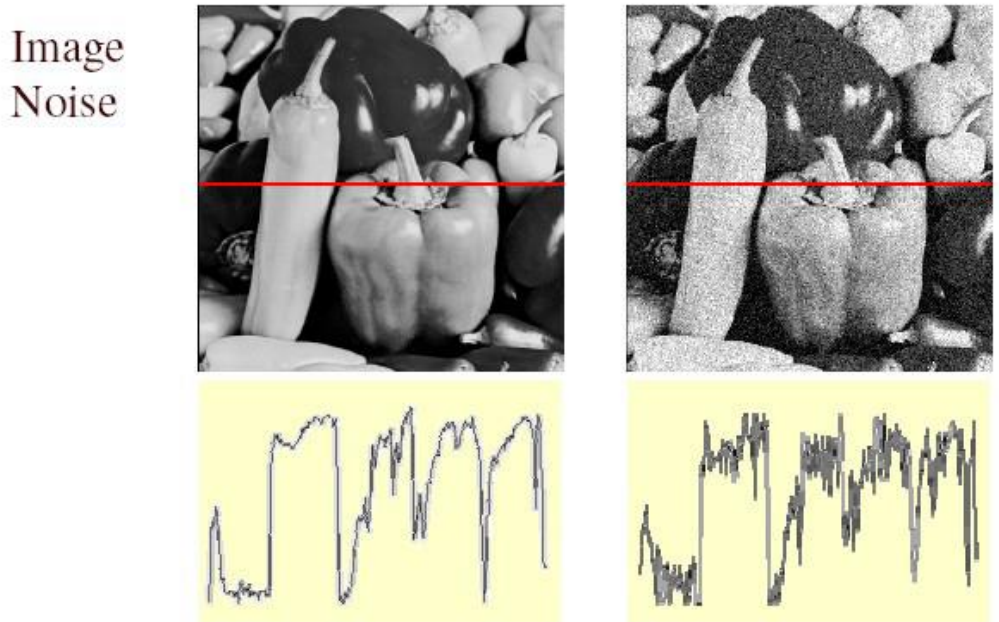
- Salt and pepper noise: contains random occurrences of black and white pixels



https://in.mathworks.com/matlabcentral/fileexchange/22141-impulse-noise-addition

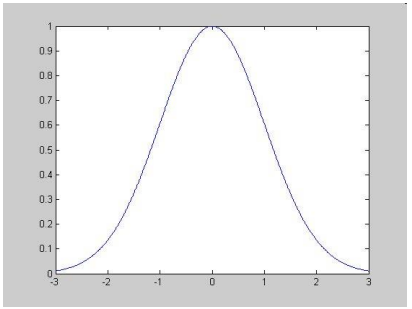
Common types of noise

- Gaussian noise: variations in intensity drawn from a Gaussian normal distribution

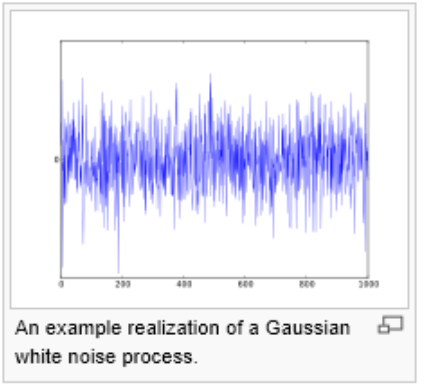


$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$



$$n(i,j) = e^{\frac{-x^2}{2\sigma^2}}$$



An example realization of a Gaussian white noise process.

Generate noise—Python example

```
import numpy as np
import cv2

puppy = cv2.imread('puppy.jpg')

def addNoise(img):
    dst = np.empty_like(img)
    noise = cv2.randn(dst, (0,0,0), (20,20,20))
    pup_noise = cv2.addWeighted(img, 0.5, noise, 0.5, 30)

addNoise(puppy)
```

Original Image

Original Image + Noise

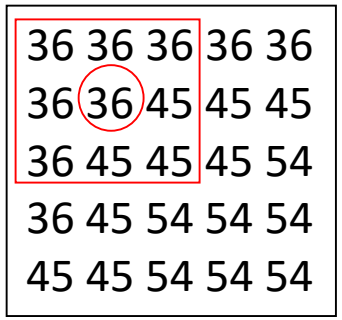
More Details: https://medium.com/analytics-vidhya/gaussian-blurring-with-python-and-opencv-ba8429eb879b

How to remove noise (denoising)?

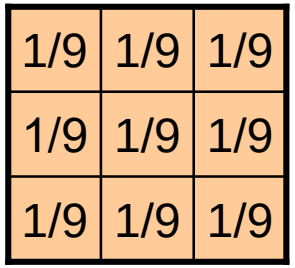
- Filtering: Use filters to remove noise

Filtering Operations Use "filters"

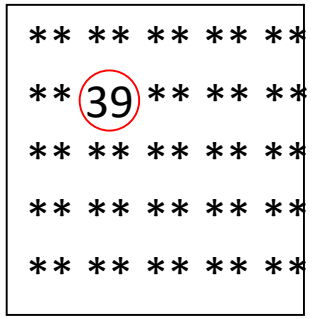
- Filters (masks) operate on a neighborhood of pixels.
- The sub-image is called a filter (or mask, kernel, template, window).
- The values in a filter sub-image are referred to as coefficients, rather than pixels.
  - A mask of coefficients is centered on a pixel.
  - The mask coefficients are multiplied by the pixel values in its neighborhood and the products are summed.
  - The result goes into the corresponding pixel position in the output image.



Input Image



filter



Output Image

Simple Neighbourhood Operations

- Some simple neighborhood operations include:
  - Min: Set the pixel value to the minimum in the neighborhood
  - Max: Set the pixel value to the maximum in the neighborhood
  - Average: Set the pixel value to the average of the neighborhoods
  - Median: The median value of a set of numbers is the midpoint value in that set (e.g. from the set [1, 7, 15, 18, 24] 15 is the median). Sometimes the median works better than the average

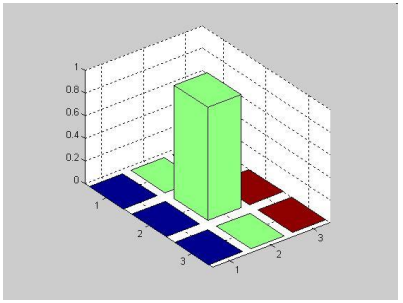
Image Filtering

- Image filtering is used to:
  - Remove noise
  - Sharpen contrast
  - Highlight contours
  - Detect edges
  - Other uses?

- Image filters - linear or nonlinear.
  - Linear filters are also know as convolution filters
  - Nonlinear operations such as median filter.

What linear filters can do? Examples - 1

*



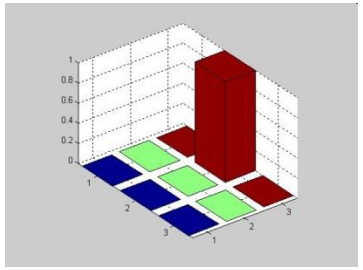
0	0	0
0	1	0
0	0	0

=

Filtered (no change)

Filtering Examples - 2

*



0	0	0
1	0	0
0	0	0

=

Key properties of linear filters

- Linearity: filter(f1 + f2) = filter(f1) + filter(f2)
- Shift invariance: same behavior regardless of pixel location.
  - The value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood

    filter(shift(f)) = shift(filter(f))

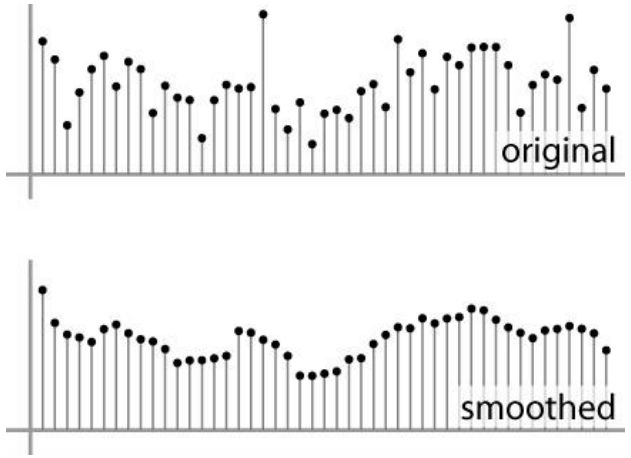
  - Any linear, shift-invariant operator can be represented as a convolution
- Commutative: a * b = b * a

Key properties of linear filters

- Associative: a * (b * c) = (a * b) * c
  - Often apply several filters one after another: (((a * b1) * b2) * b3)
  - This is equivalent to applying one filter: a * (b1 * b2 * b3)
- Distributes over addition: a * (b + c) = (a * b) + (a * c)
- Scalars factor out: ka * b = a * kb = k (a * b)
- Identity: unit impulse e = [0, 0, 1, 0, 0], a * e = a
- Differentiation: $\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$

Average filtering

- One of the simplest spatial filtering operations we can perform is a smoothing operation (remove sharp features)
  - Simply average all of the pixels in a neighborhood around a central value
  - Especially useful in removing noise from images
  - Also useful for highlighting gross detail
  - Moving average in 1D:

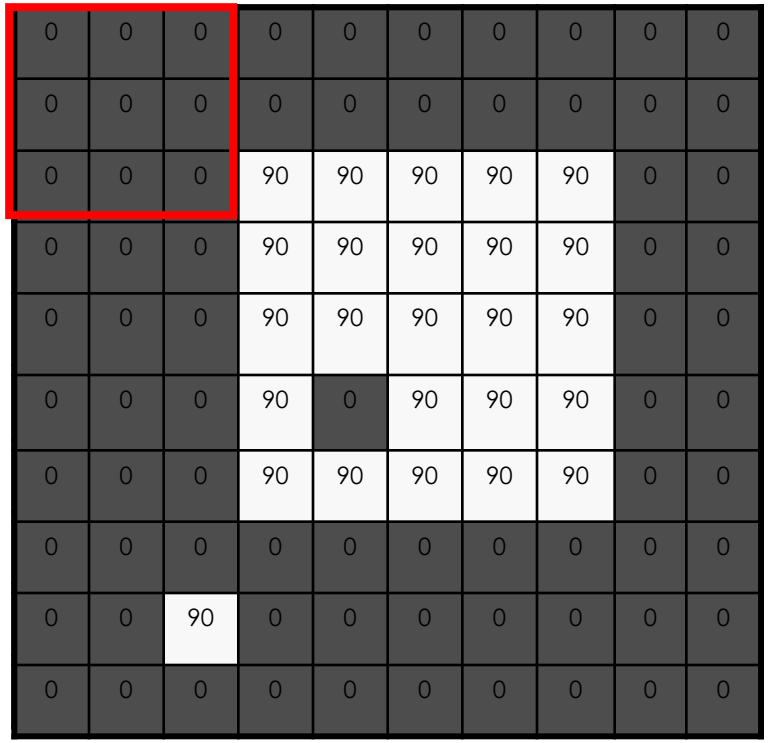


1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

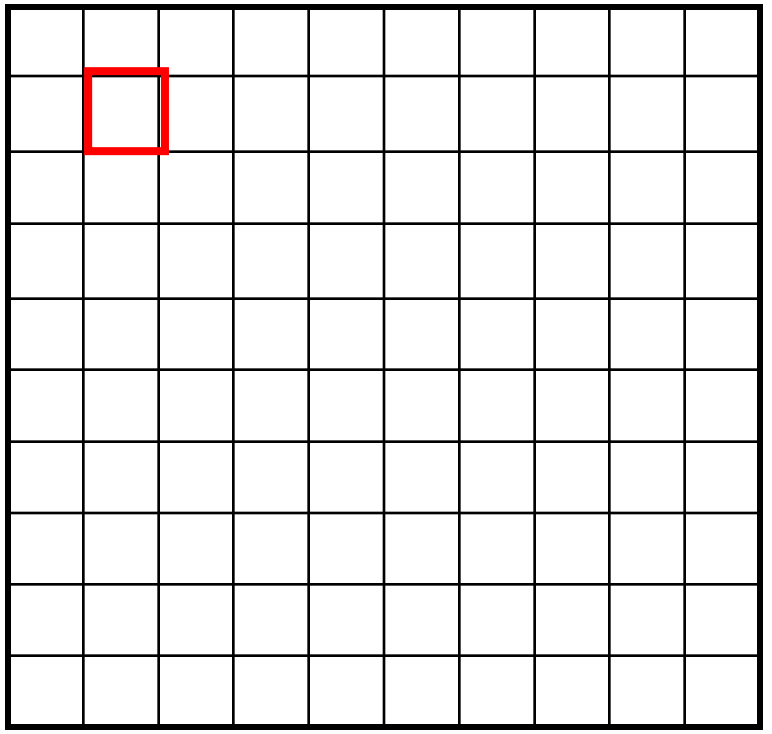
Average filter

Average filtering: Example

$f[.,.]$



$g[.,.]$



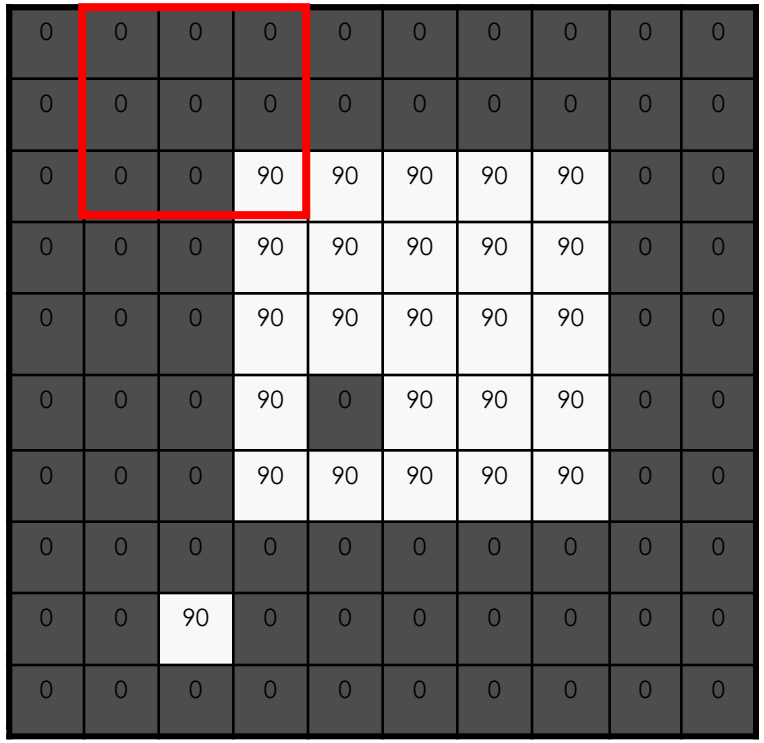
$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

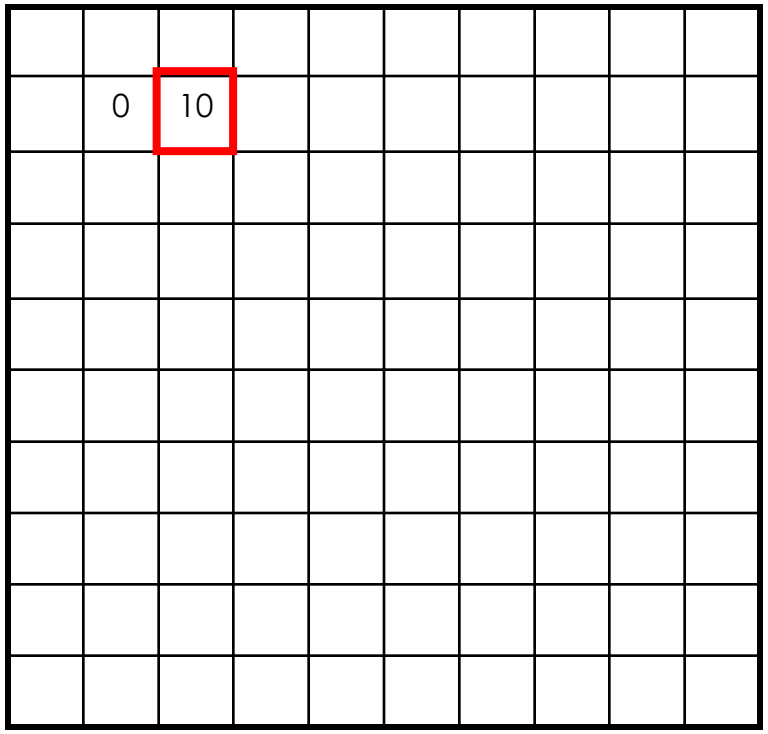
$$g[m,n] = \sum_{k,l} h[k,l]\ f[m+k, n+l]$$

Average filtering: Example

$f[.,.]$



$g[.,.]$



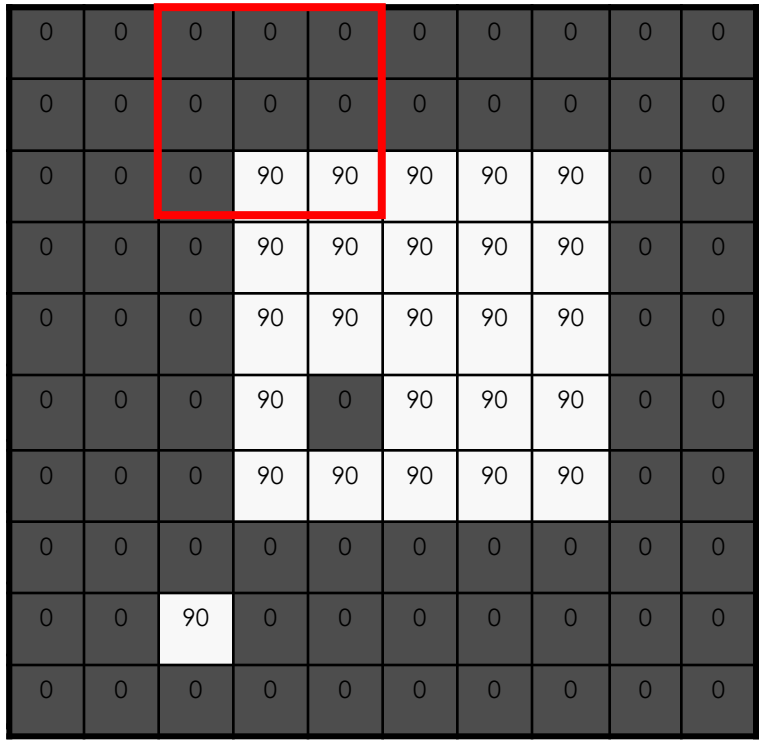
$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

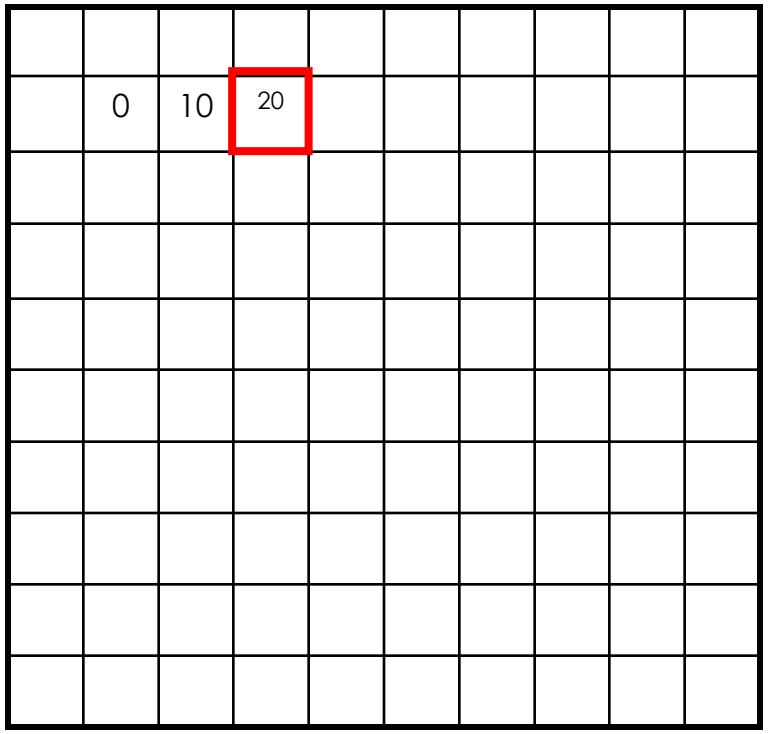
$$g[m,n] = \sum_{k,l} h[k,l]\ f[m+k, n+l]$$

Average filtering: Example

$f[.,.]$



$g[.,.]$



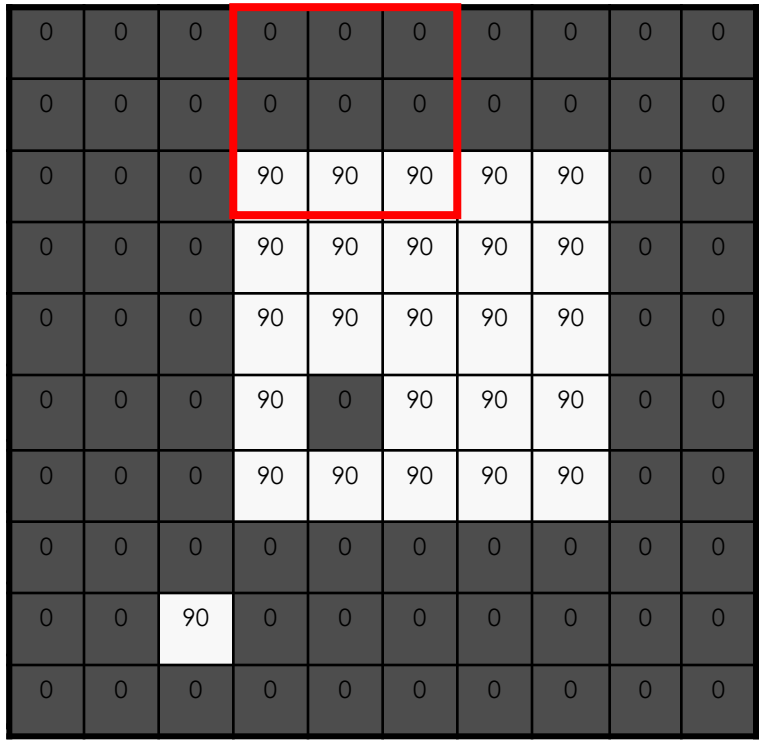
$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

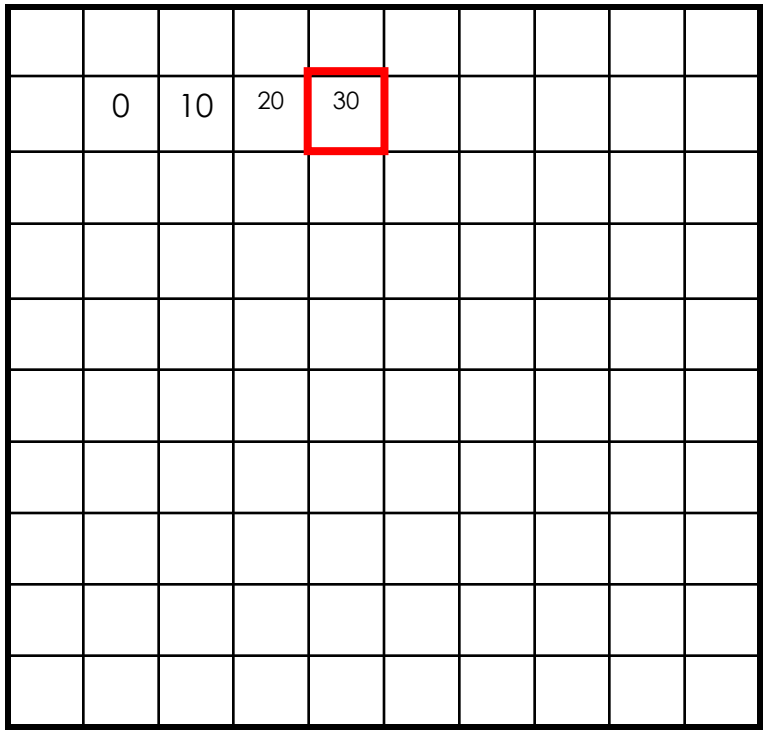
$$g[m,n] = \sum_{k,l} h[k,l]\ f[m+k, n+l]$$

Average filtering: Example

$f[.,.]$



$g[.,.]$



$h[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$g[m,n] = \sum_{k,l} h[k,l]\ f[m+k, n+l]$$

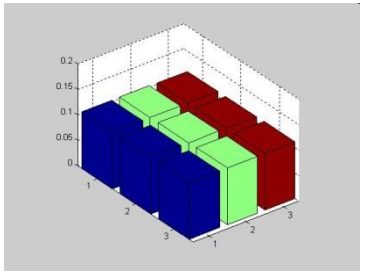
Average filtering: Example

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Average filtering: Example 2

*

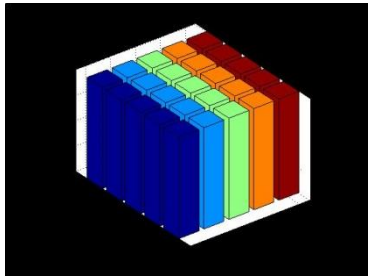


1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

Average filtering: Example 3

$*\frac{1}{25}$



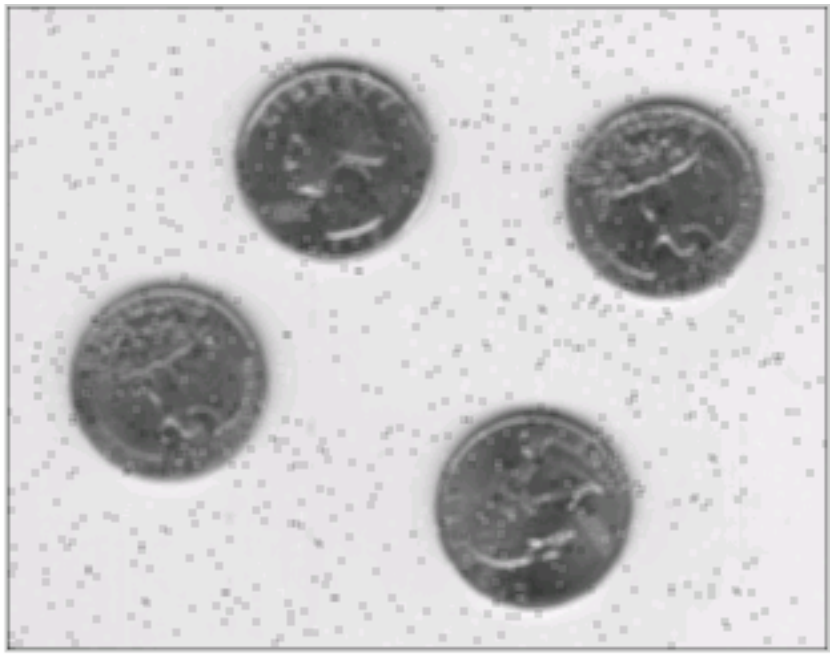
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

$=$

Noise removing using average filters

- On the left is an image containing a significant amount of salt and pepper noise.
- On the right is the same image after processing with an average filter.

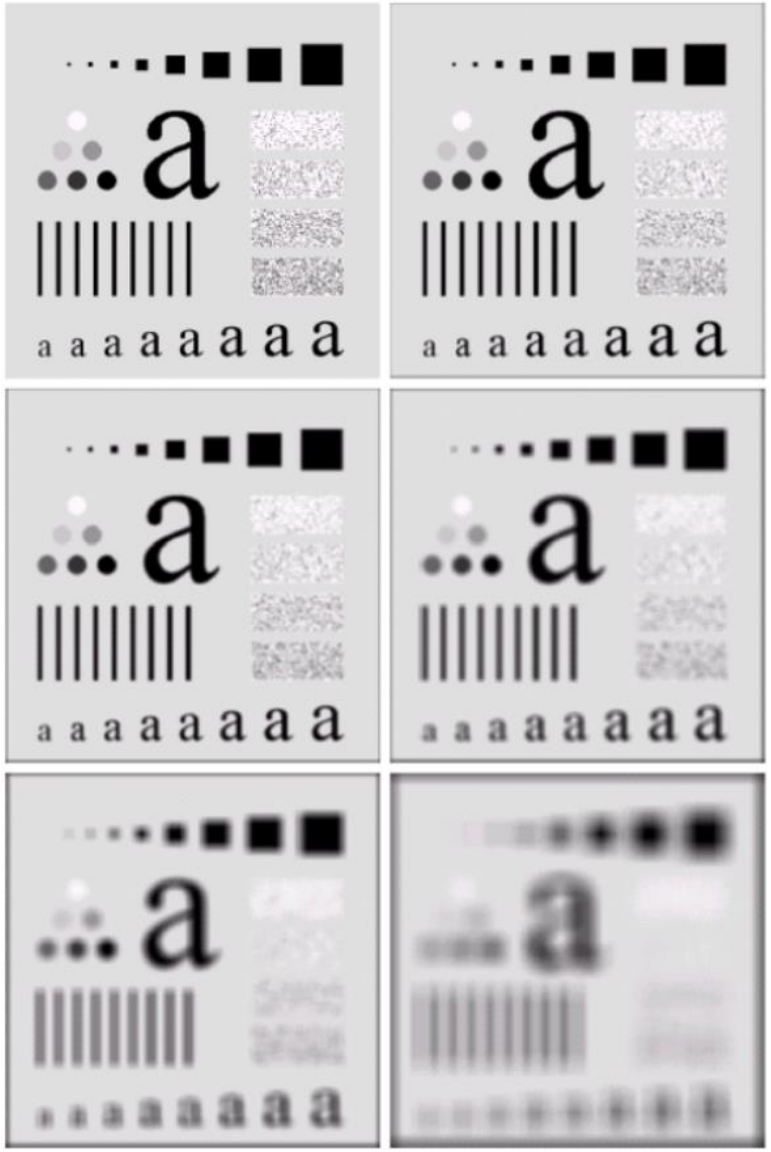
Original Image



Modified Image

Image Smoothing: Example

- The image at the top left is an original image of size 500*500 pixels
- The subsequent images show the image after filtering with an averaging filter of increasing sizes: 3, 5, 9, 15 & 35
- Notice how detail begins to disappear



Weighted Smoothing Filter

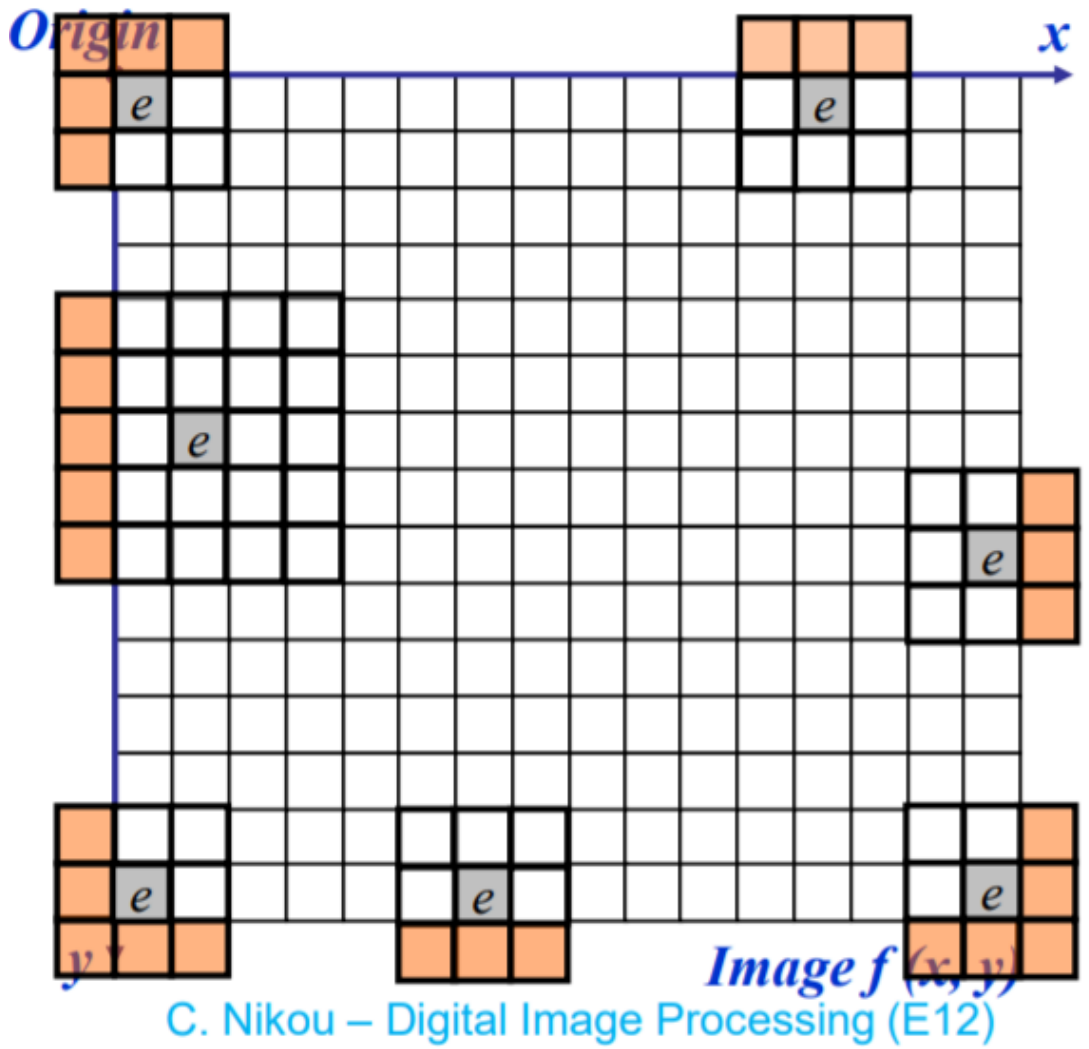
- More effective smoothing filters can be generated by assigning different pixels in the neighborhood different weights in the averaging function
  - Pixels closer to the central pixel are more important
  - Reduce smoothing effect
  - Often referred to as a weighted averaging

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

Boundary issues

- At the edges of an image we are missing pixels to form a neighborhood

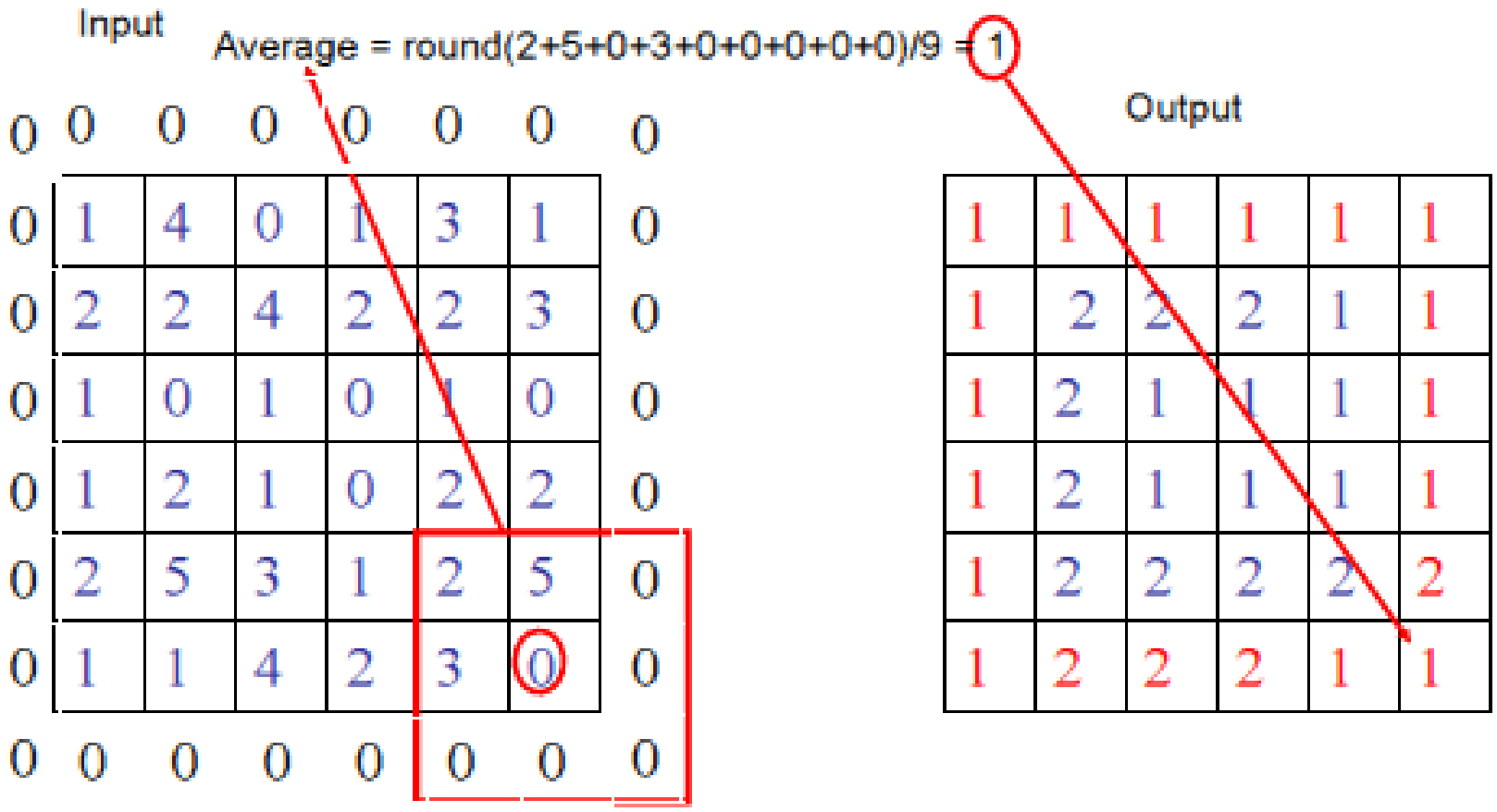


C. Nikou – Digital Image Processing (E12)

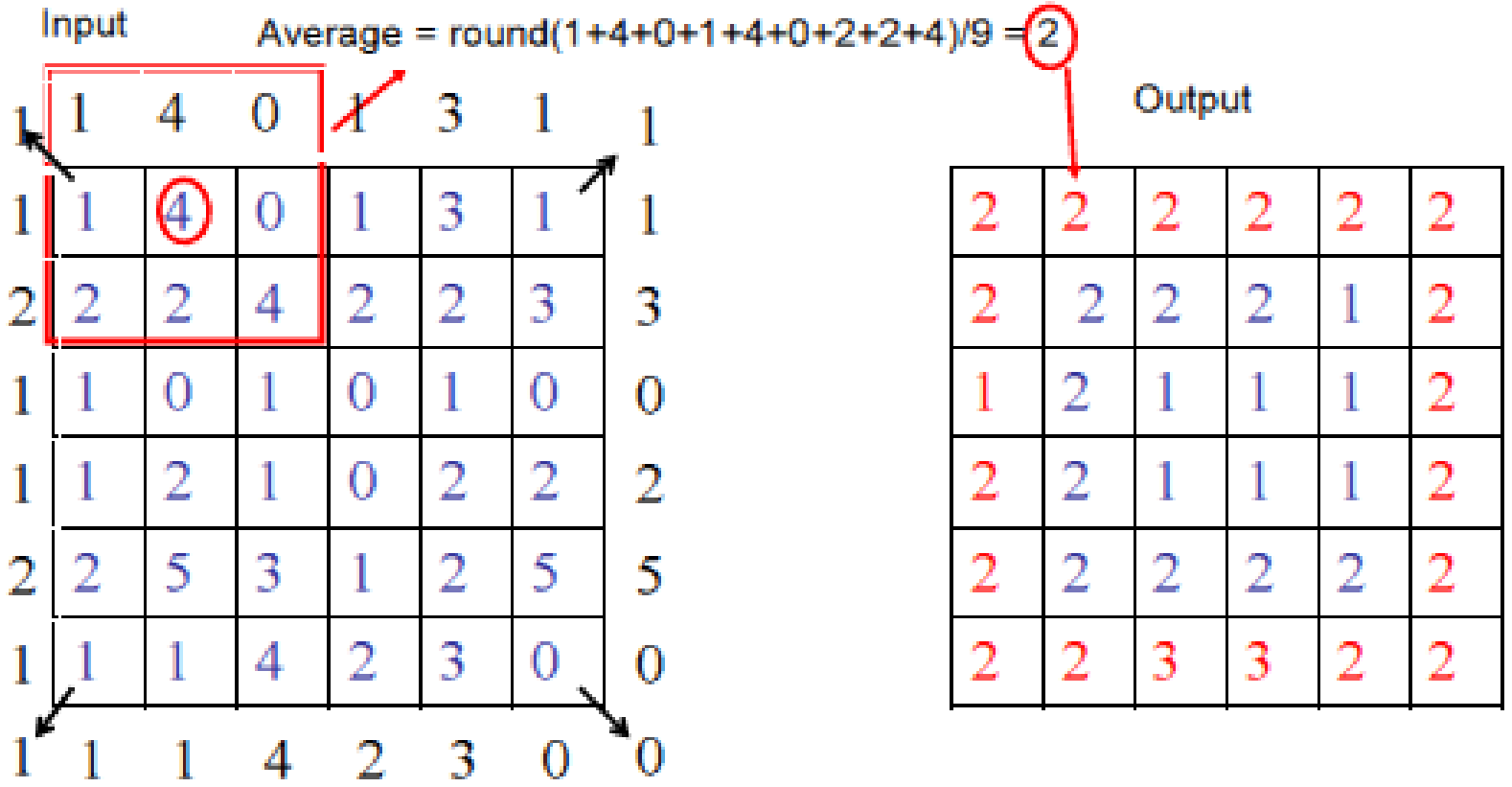
Boundary issues

- There are a few approaches to deal with missing edge pixels:
  - Pad the image (padding)
    - Typically with either all white or all black pixels
  - Replicate border pixels
  - Truncate the image
  - Allow pixels wrap around the image
    - Can cause some strange image artefacts

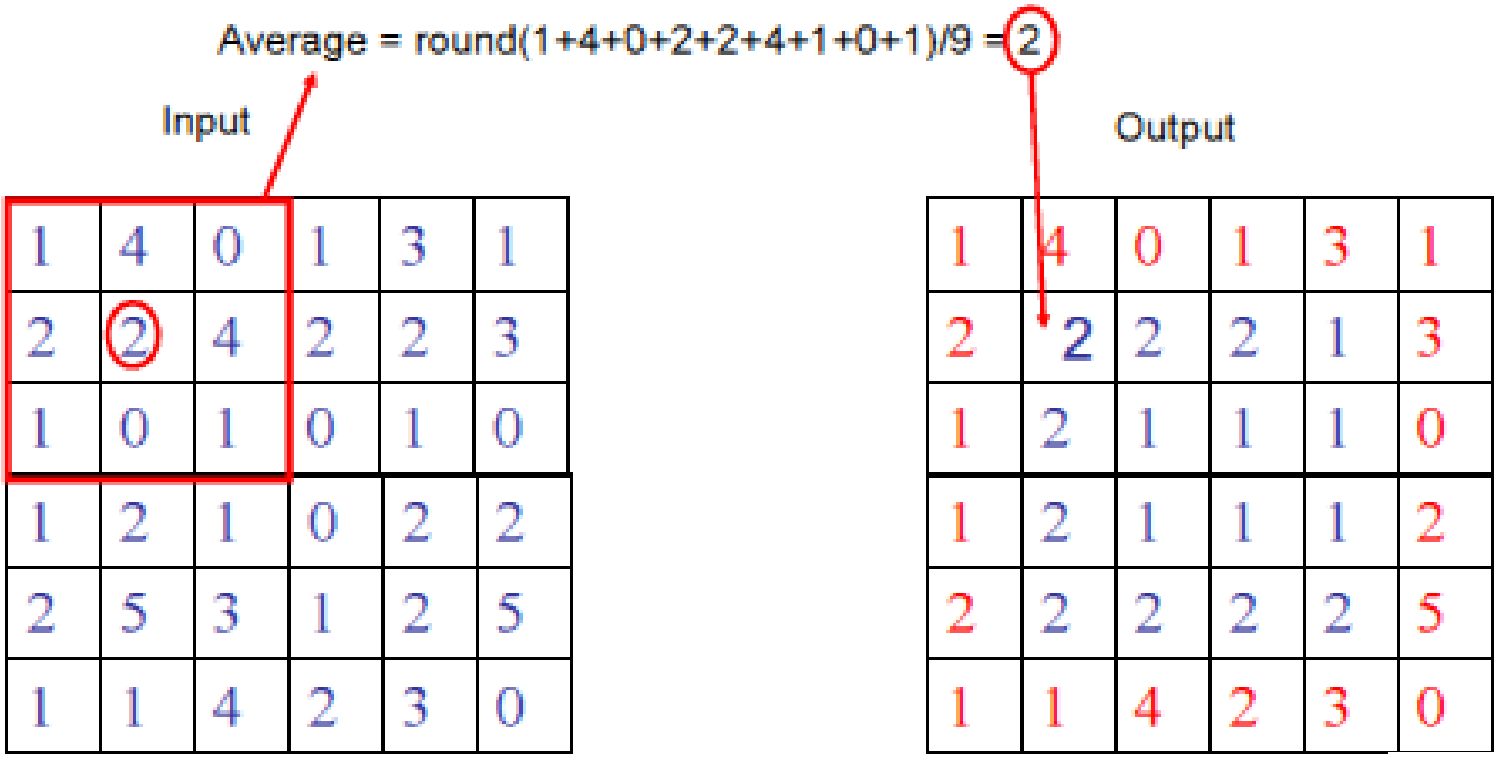
Average filtering example with border padding



Average filtering example with border replication

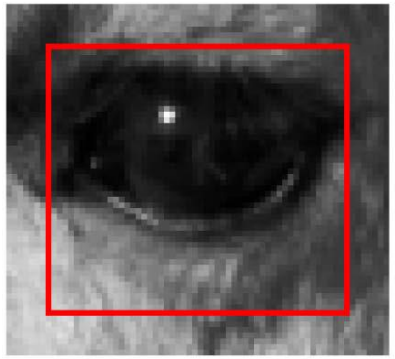


Average filtering example with border truncation

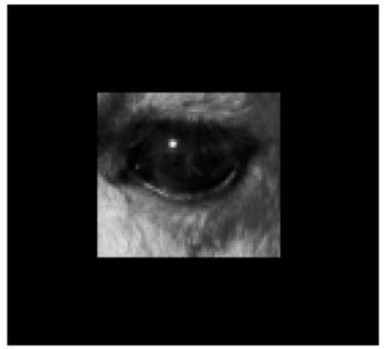


Average filtering examples

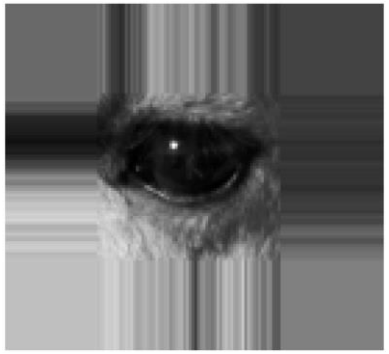
Filtered Image:
Truncate (crap)



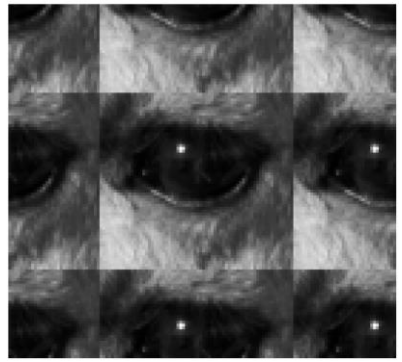
Filtered Image:
Zero Padding



Filtered Image:
Replicate



Filtered Image:
Wrap



Average filter—Python example

```
import cv2
import numpy as np

 # read the image
image = cv2.imread('imagePath.jpg')
# apply the 3x3 average filter on the image
kernel = np.ones((3,3),np.float32)/9
processed_image = cv2.filter2D(image,-1,kernel)
# display image
cv2.imshow(Average Filter Processing', processed_image)
# save image to disk
cv2.imwrite('processed_image.png', processed_image)
```

More examples: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html

Gaussian Filtering

- A Gaussian kernel gives less weight to pixels further from the center of the window

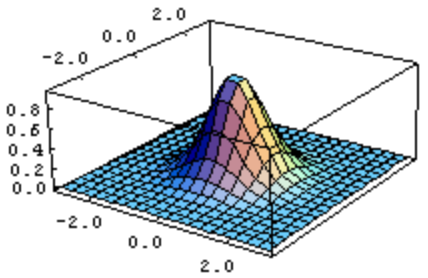
$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

$H[u, v]$

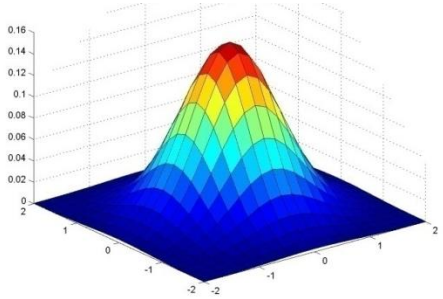
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

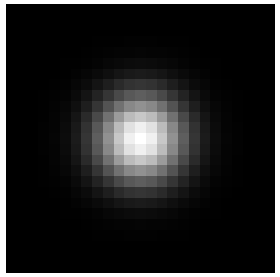


- This kernel is an approximation of a Gaussian function.

Gaussian Filtering

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

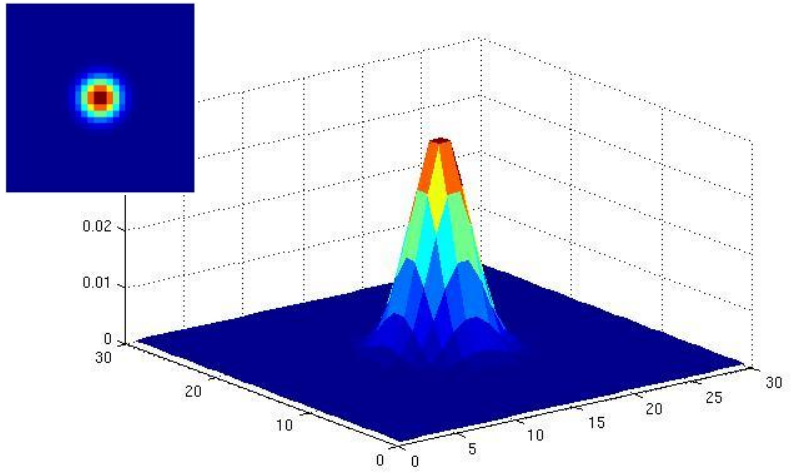




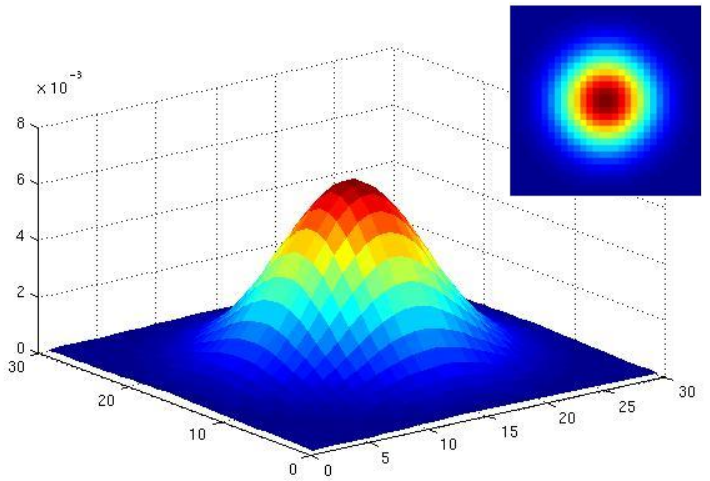
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, σ = 1

Parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing.



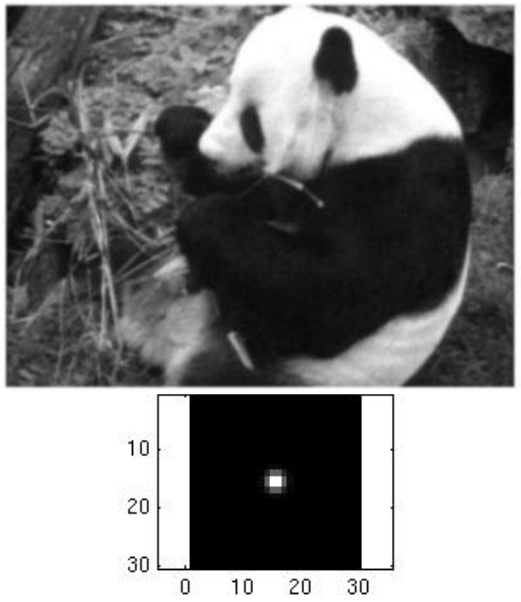
σ = 2 with 30 x 30 kernel

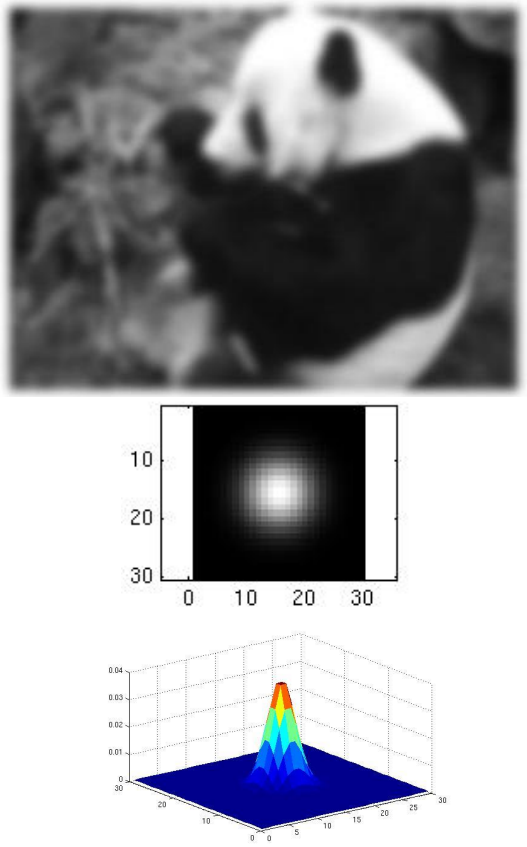


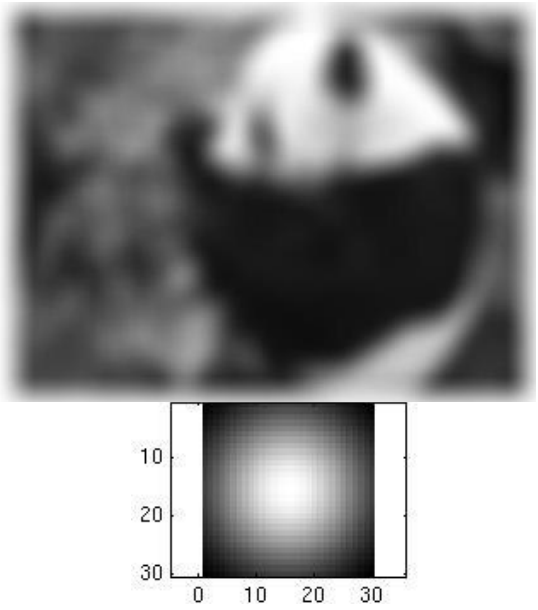
σ = 5 with 30 x 30 kernel

Smoothing with a Gaussian

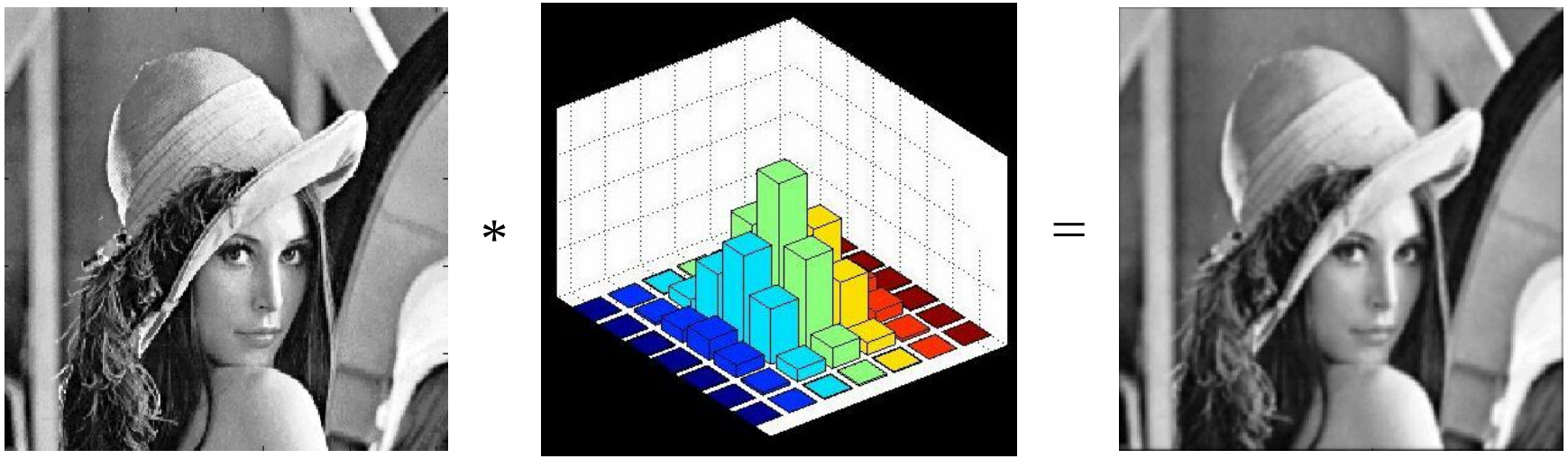
- Parameter σ is the "scale" / "width" / "spread" of the Gaussian kernel, and controls the amount of smoothing.







Gaussian filtering: Example



Gaussian Smoothing

Gaussian vs Averaging

Gaussian Smoothing

Smoothing by Averaging

Gaussian filter—Python example

```
import cv2
import numpy as np

 # read the image
image = cv2.imread(‘imagePath.jpg’)
# apply the 5x5 Gaussian filter on the image
processed_image = cv2.GaussianBlur(image,(5,5),1)
# display image
cv2.imshow(Gaussian Filter Processing', processed_image)
# save image to disk
cv2.imwrite('processed_image.png', processed_image)
```

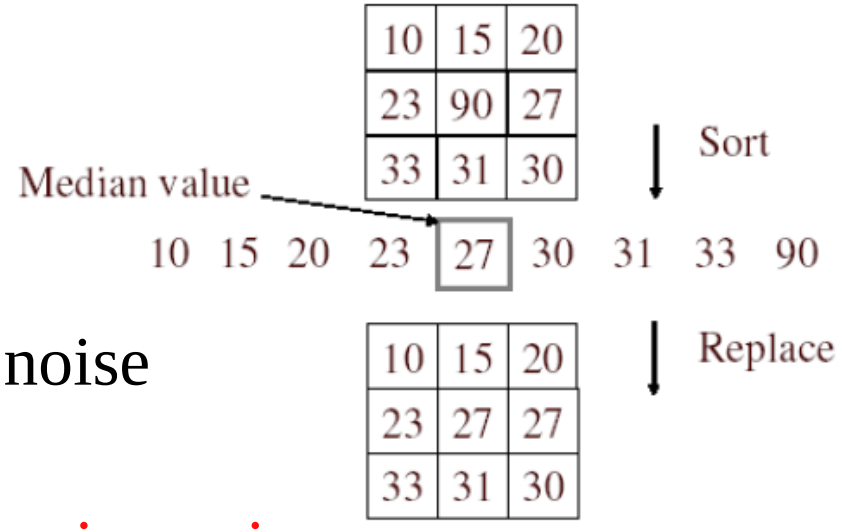
More examples: https://medium.com/analytics-vidhya/gaussian-blurring-with-python-and-opencv-ba8429eb879b

Median filters (non-linear)

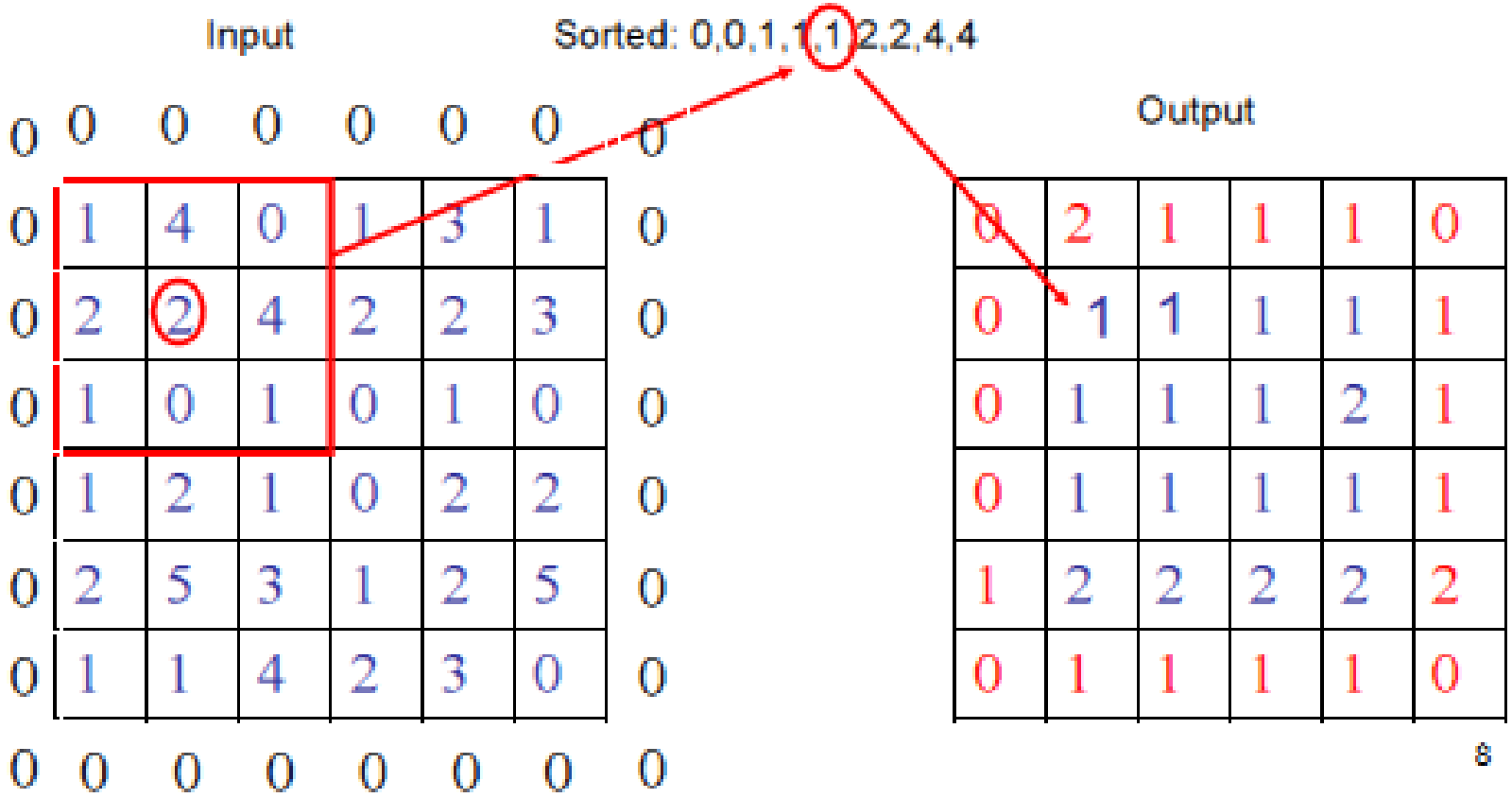
- Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

Median filters (non-linear)

- A Median Filter operates over a window by selecting the median intensity in the window.
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- It is not as efficient at averaging away regular Gaussian noise

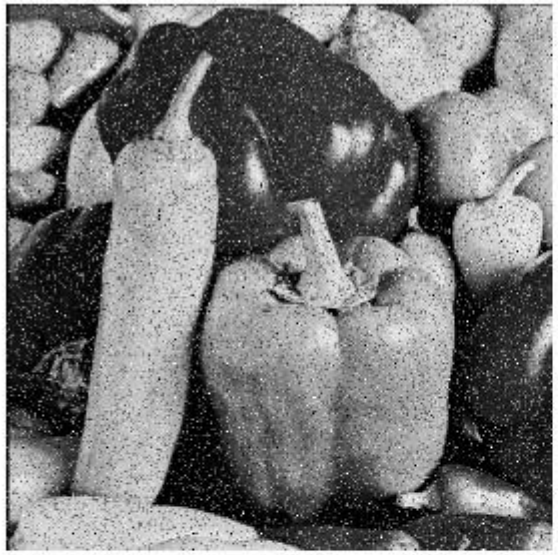


Median filters example

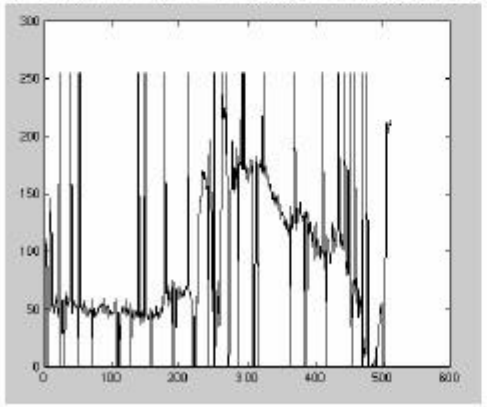


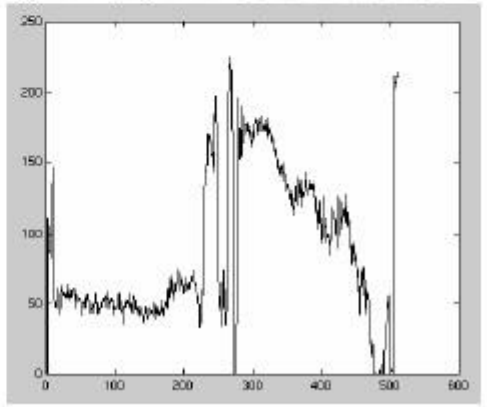
Median filters example

Original Image



Modified Image

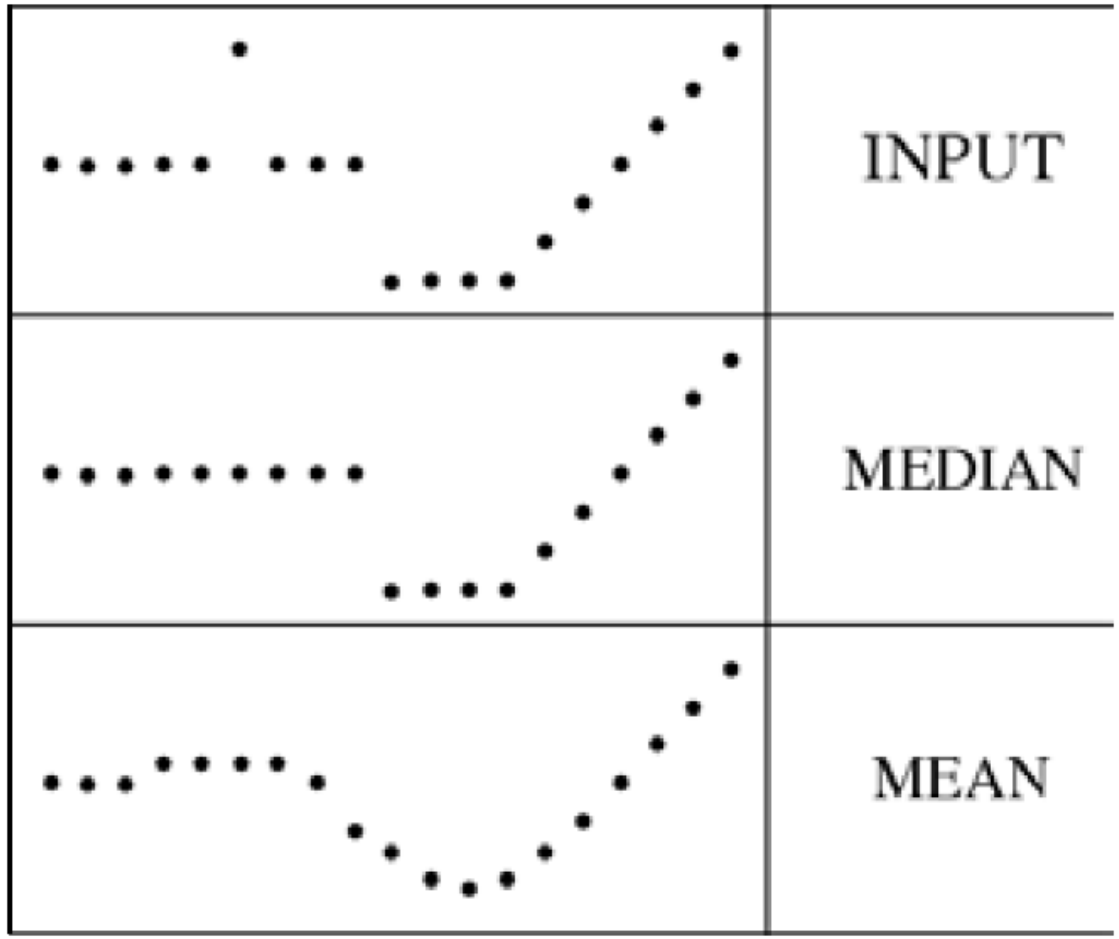




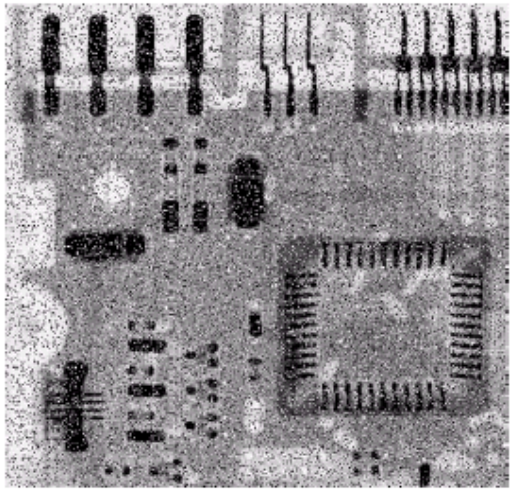
Plots of a row of the image

Median filters

- Median filter is edge preserving



Averaging Filter vs Median Filter Example



Original Image With Noise

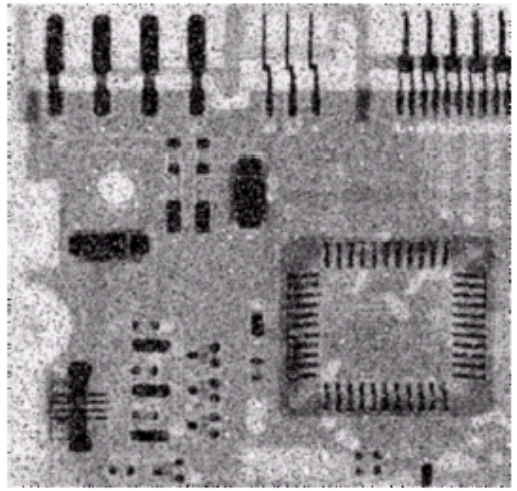


Image After Averaging Filter

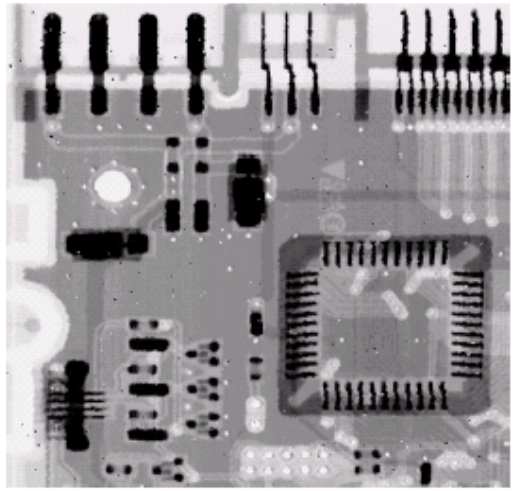
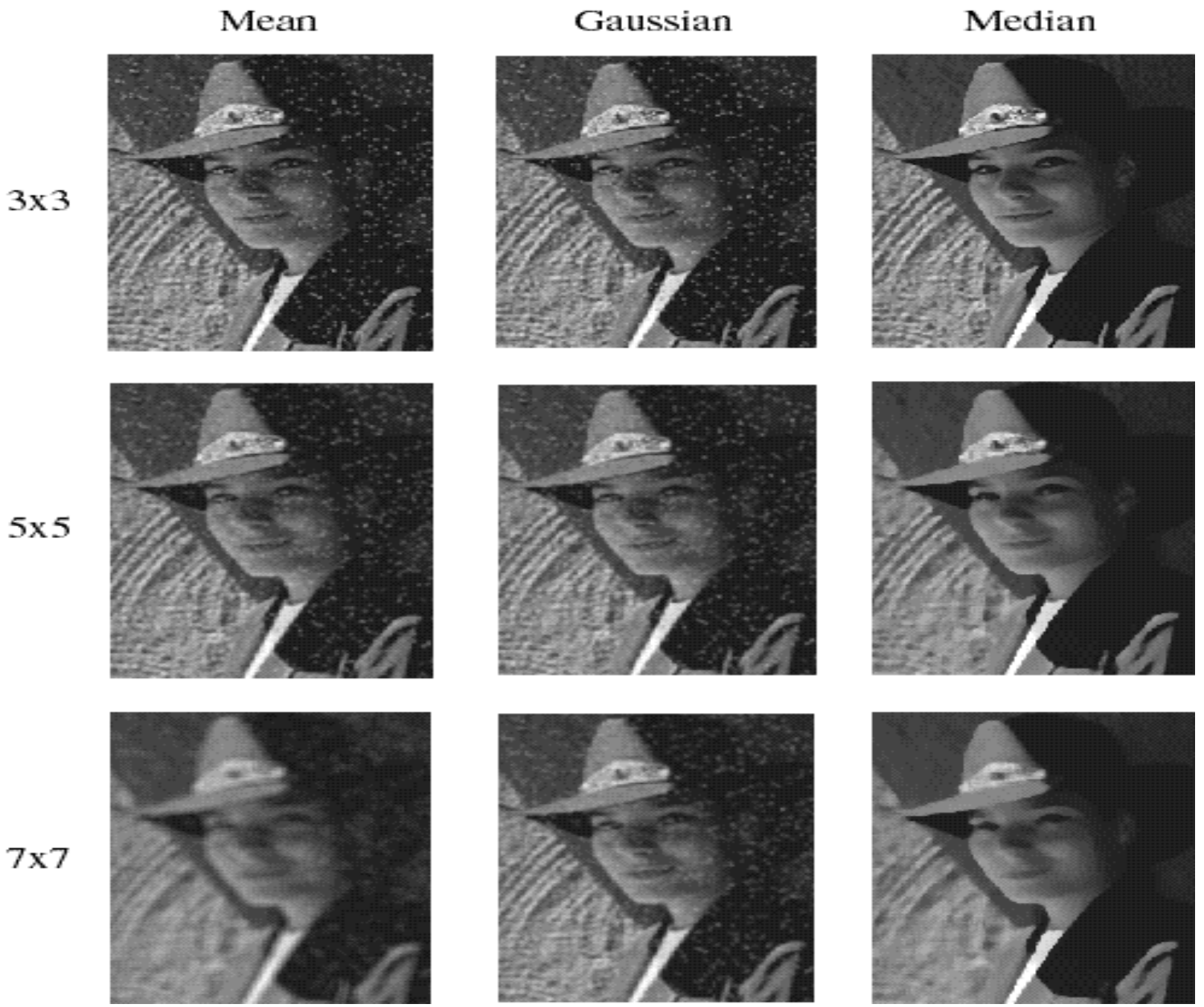


Image After Median Filter

Averaging Filter vs Median Filter Example



Acknowledgments

- Slides have been used from:
  - Forsythe and Ponce: Computer Vision: A Modern Approach
  - Rick Szeliski's book: Computer Vision: Algorithms and Applications
  - Lectures of Prof. Aaron Bobick, School of Interactive Computing, Georgia Tech.
  - Lectures of Dr. James Hays, Brown university.
  - Lectures of Dr. Selim Aksoy, Bilkent University
  - https://www.cs.auckland.ac.nz/~rklette/TeachAuckland.html/775/ (R. Klette. Concise Computer Vision, Springer-Verlag, London, 2014.)
  - http://www.cs.uoi.gr/~cnikou/Courses/Digital_Image_Processing/
  - https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14
  - https://www.cs.cmu.edu/~16385/
  - http://6.869.csail.mit.edu/fa19/
  - ICCV2019 Tutorial: Dr. Michael S. Brown