BMB5113
COMPUTER VISION

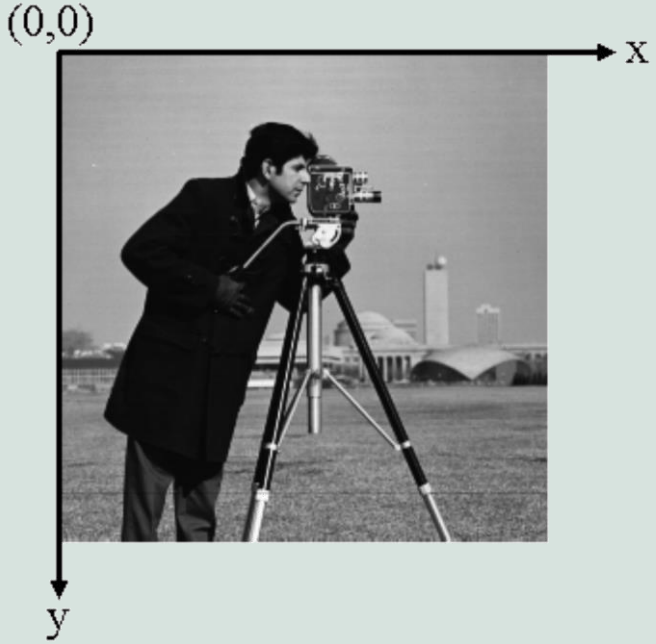
IMAGE PROCESSING I

Digital Images in General

- Image is a function of four variables f(x,y; t, λ)
  - For color image, λ takes three different values corresponding to red, green and blue components
  - t is a time variable for a sequence of frames
  - for constant λ (black and white) and a constant t
    - f becomes a function of two spatial variables
  - **Binary Image :**
    - If there are just two values, e.g. black and white, we usually represent them by 0 and 1.
  - **Gray-scale Image :**
    - If there is just a single plane of color, usually represented in between 0 and 255 (for int/uint) or 0 and 1 (for float or double)

A Digital Image

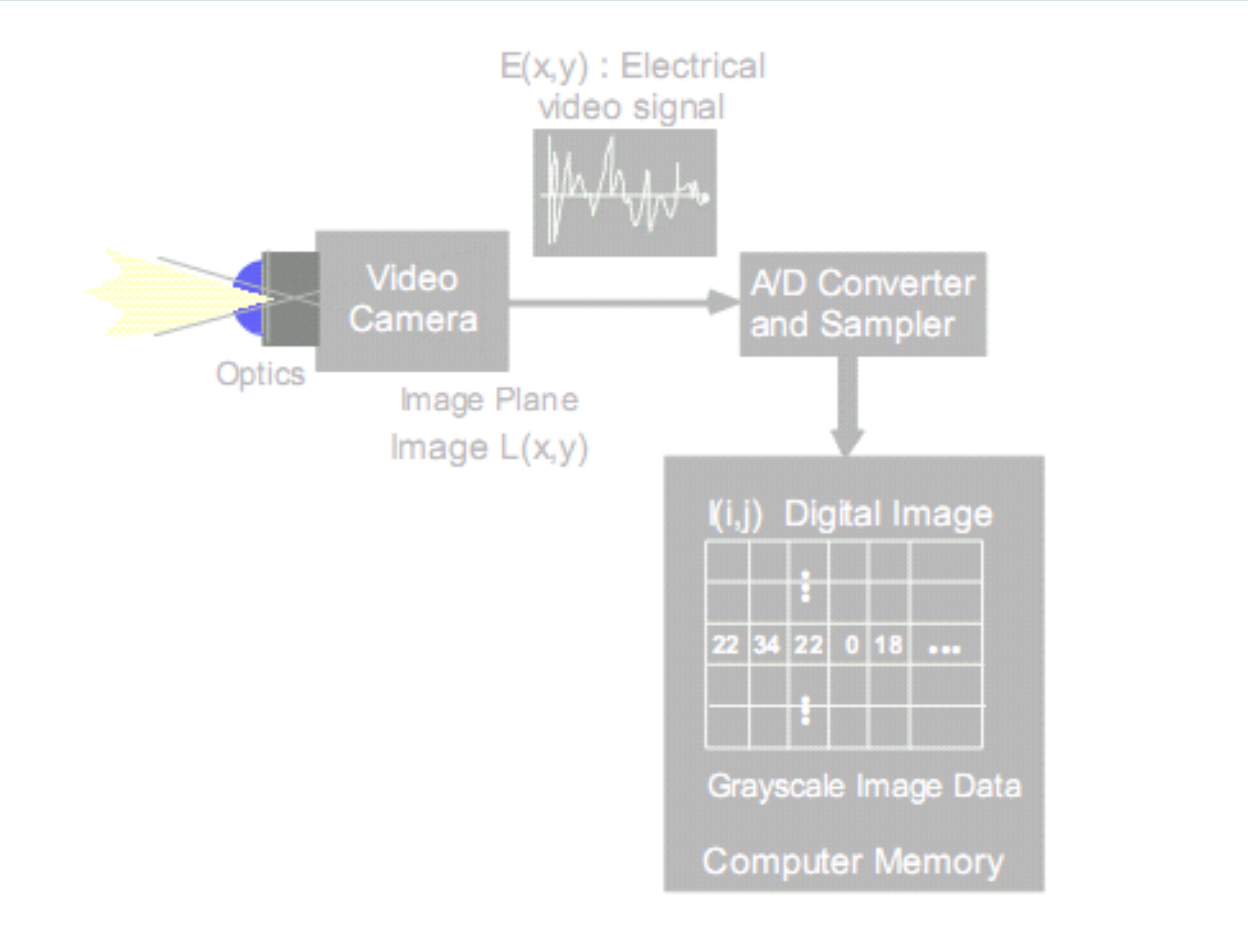
- 2-D function of intensities
  - can be represented as f(x,y)
  - (x,y) → 2-D coordinate pair
  - f(x,y) → the image intensity value at (x,y)



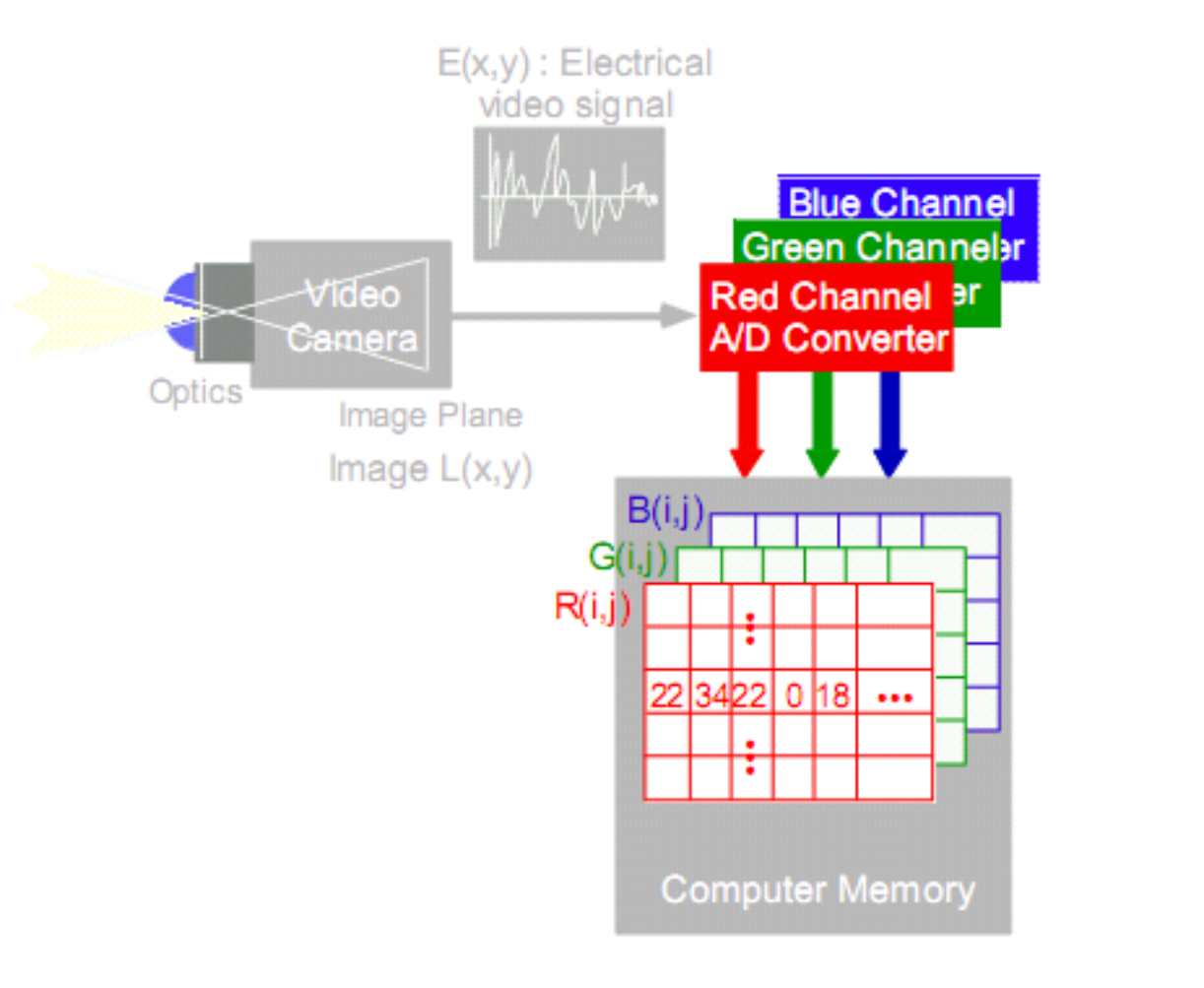
Digitization of An Image

- Filtering
  - For successful reconstruction $f_s \geq 2f_{max}$
- Sampling
  - Taking samples at specific intervals from 2-D spatial plane
- Quantization
  - Adjustment for intensities to have specific values
- Digitization
  - Converting the intensity values to digital form

Grayscale Image Sensing Systems:

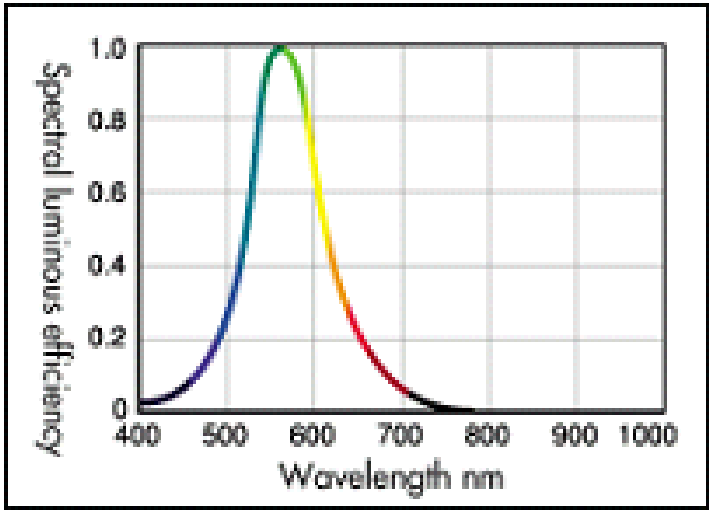


Color Image Image Sensing Systems:



CCD cameras are much more sensitive than the eye

Human Eye



CCD Camera

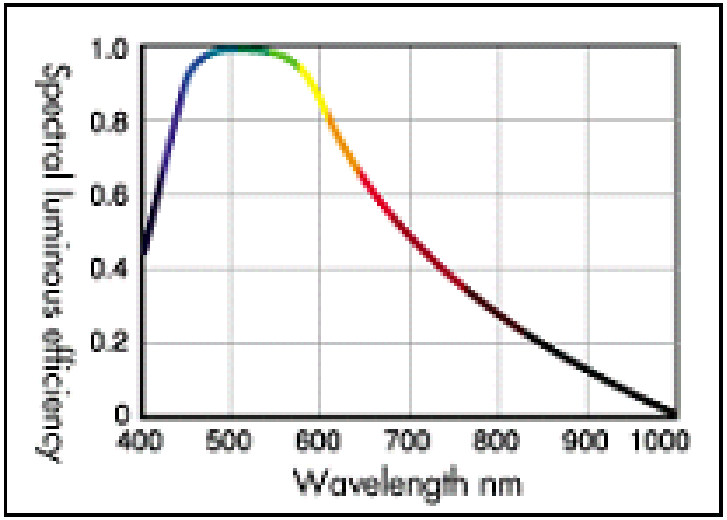


Image Pixels

- After sampling and quantization, a digital image is a rectangular array of integer values.
-

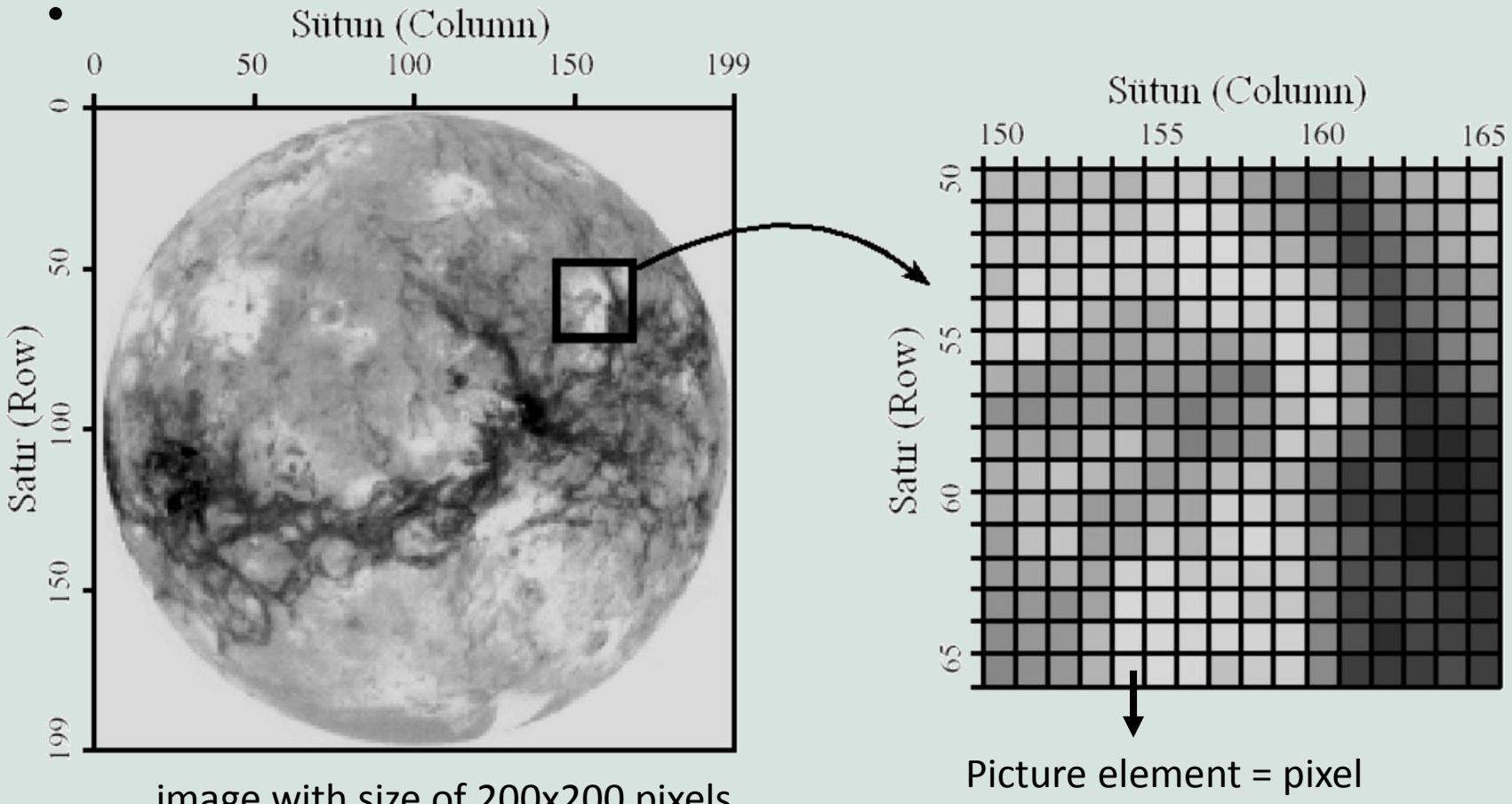
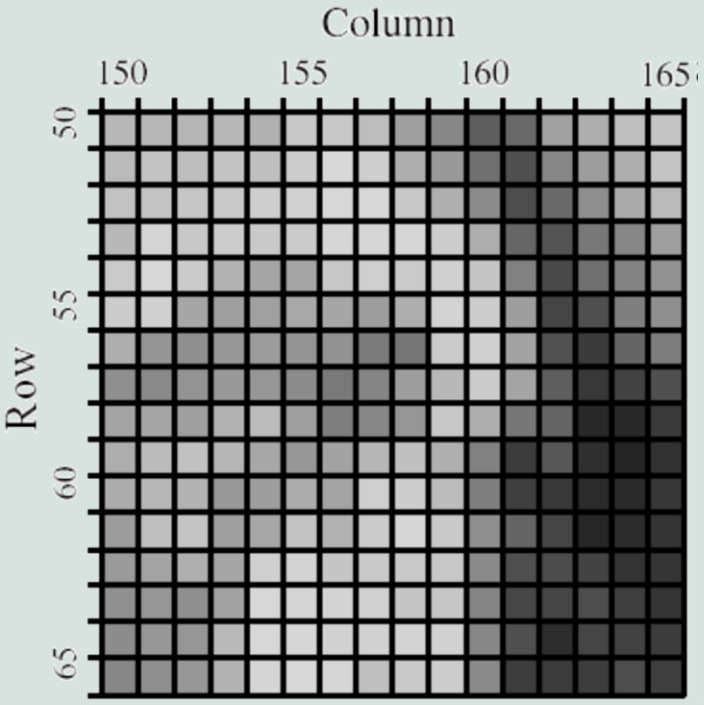


image with size of 200x200 pixels

Pixel Intensity Values

- Intensity values are
  - non-negative
  - called gray levels



Column

Row	150	155	160	165
50	183 183 181 184 177 200 200 189	159 135 94 105 160 174 191 196		
	186 195 190 195 191 205 216 206	174 153 112 80 134 157 174 196		
	194 196 198 201 206 209 215 216	199 175 140 77 106 142 170 186		
	184 212 200 204 201 202 214 214	214 205 173 102 84 120 134 159		
55	202 215 203 179 165 165 199 207	202 208 197 129 73 112 131 146		
	203 208 166 159 160 168 166 157	174 211 204 158 69 79 127 143		
	174 149 143 151 156 148 146 123	118 203 208 162 81 58 101 125		
	143 137 147 153 150 140 121 133	157 184 203 164 94 56 66 80		
60	164 165 159 179 188 159 126 134	150 199 174 119 100 41 41 58		
	173 187 193 181 167 151 162 182	192 175 129 60 88 47 37 50		
	172 184 179 153 158 172 163 207	205 188 127 63 56 43 42 55		
	156 191 196 159 167 195 178 203	214 201 143 101 69 38 44 52		
65	154 163 175 165 207 211 197 201	201 199 138 79 76 67 51 53		
	144 150 143 162 215 212 211 209	197 198 133 71 69 77 63 53		
	140 151 150 185 215 214 210 210	211 209 135 80 45 69 66 60		
	135 143 151 179 213 216 214 191	201 205 138 61 59 61 77 63		

Picture element = pixel

Spatial Resolution

256x256 pixel

72 dpi

128x128 pixel

36 dpi

64x64 pixel

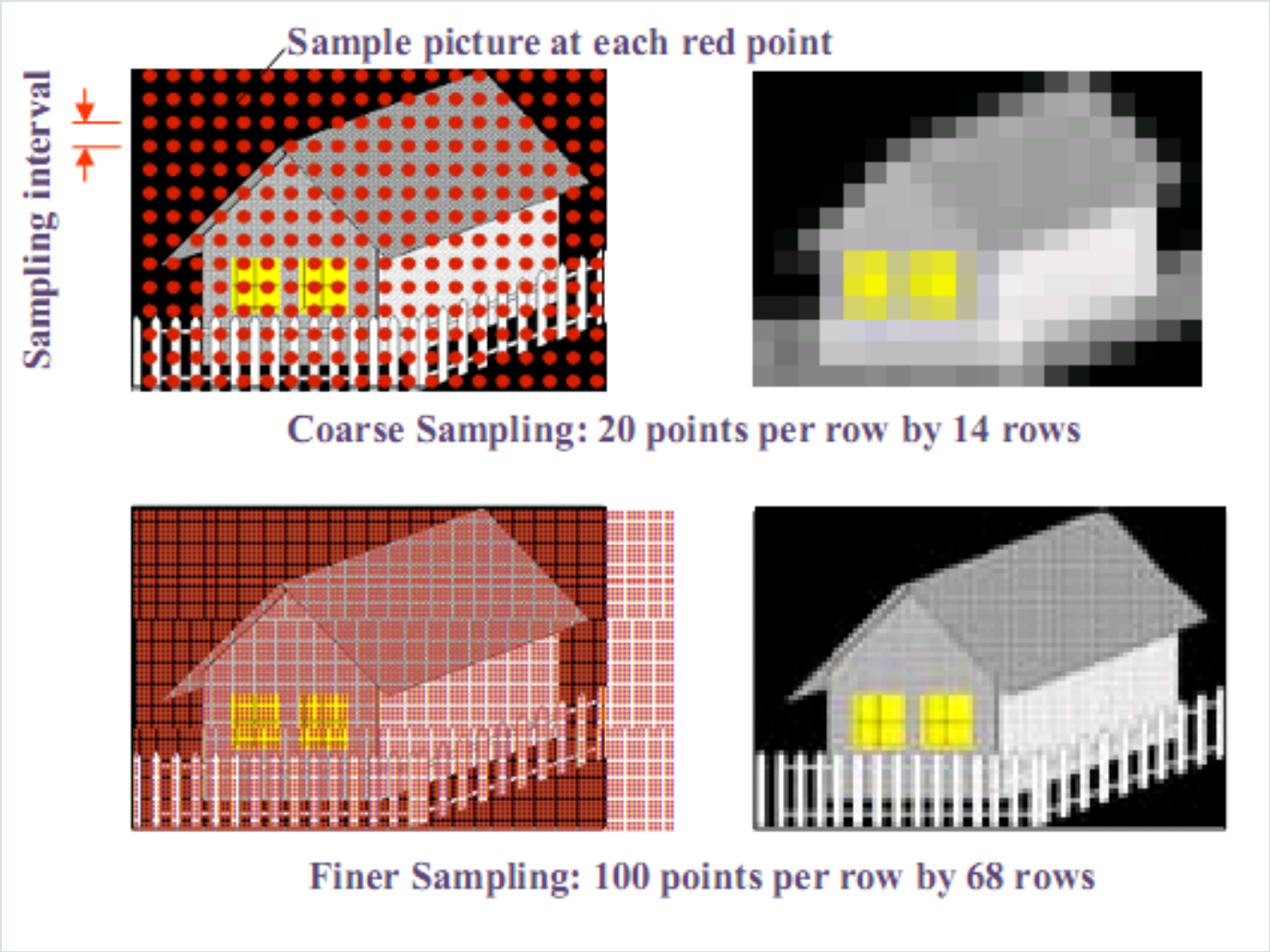
18 dpi

32x32 pixel

9 dpi

- Images with same physical dimensions (3.6’’x3.6’’)
- But with different resolutions
- Resolution = $\frac{\#\ of\ pixels}{unit\ physical\ dimension}$
- Resolution units
  - dpi: dots per inch
  - ppi: pixels per inch

Sampling (Resolution)



Bit depth of an Image

L = 8 bit

L = 7 bit

L = 6 bit

L = 5 bit

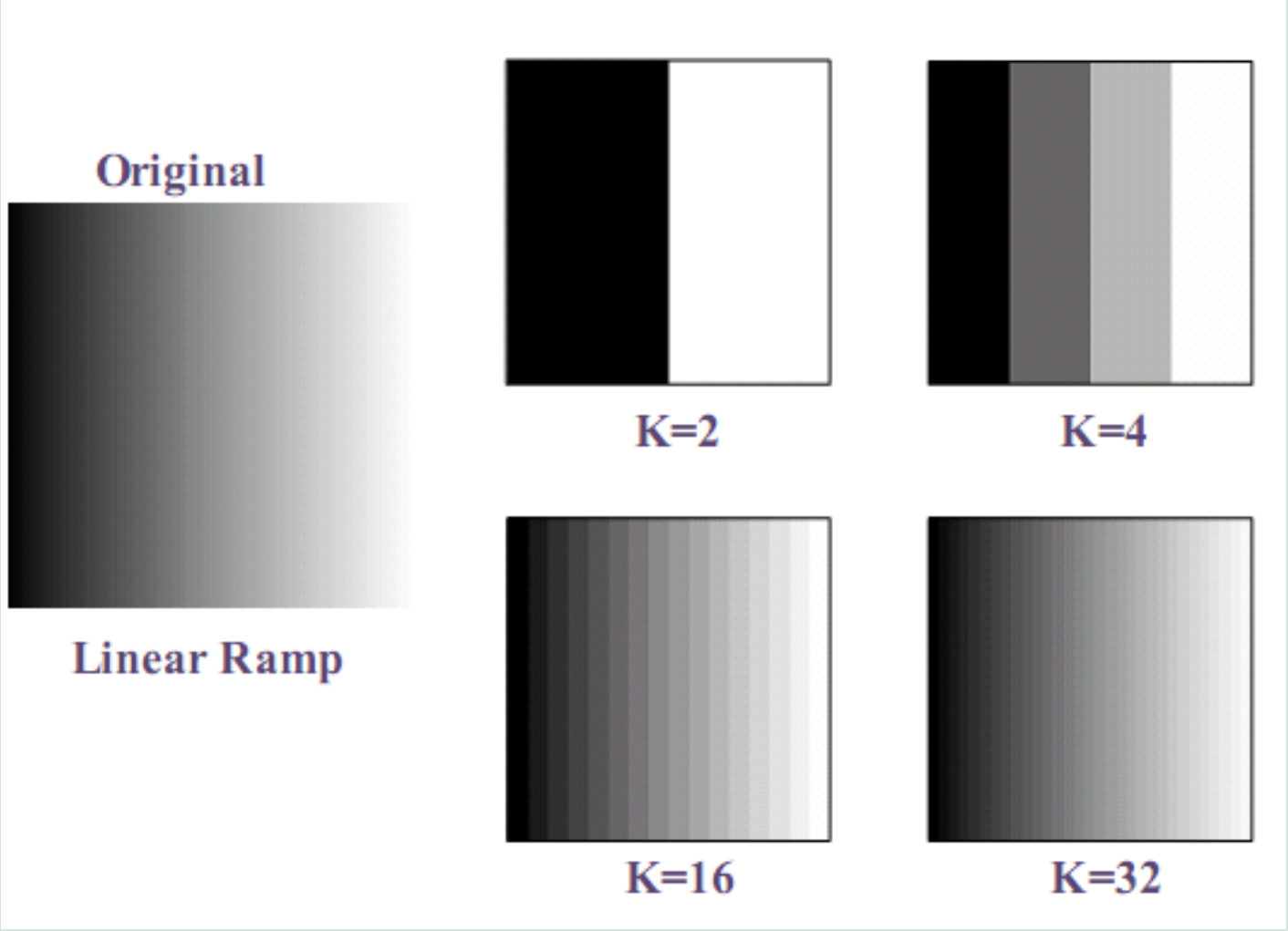
L = 4 bit

L = 3 bit

L = 2 bit

L = 1 bit

Grayscale Quantization Level:



Color Image Quantization Level

K=2 (each color)

K=4 (each color)

VSCode Setup for Building Programs in bm5113 Environment

- After environment setup arrange VSCode json files
  - "terminal.integrated.defaultProfile.windows": "Command Prompt",
  - "python.condaPath": "C:\\ProgramData\\Anaconda3\\Scripts\\conda.exe"
- Run the following update
  - conda update anaconda-navigator
  - conda update navigator-updater

Python Image Functions: Read

- Reading an image:

```
from PIL import Image
import numpy as np
im = np.array(Image.open('./jpg/car1.jpg'))
```

- Converting image to grayscale after reading

```
pil_im = Image.open('empire.jpg').convert('L')
```

- Row and column dimensions of an image:

```
print(im.shape)
[M,N,C] = im.shape
```

Python Image Functions: Display

- Displaying an image:

```
import matplotlib.pyplot as plt

imgplot = plt.imshow(im, cmap='gray')
```

  - cmap: the colormap used to display image
  - by default ‘viridis’
  - vmin=0, vmax=255 parameters can be used when displaying
  - Interactive displaying of the pixel values possible

Python Image Functions: Write

- Writing an image:

```
pil_img = Image.fromarray(im)
pil_img.save('./jpg/car1_copy.jpg')
```

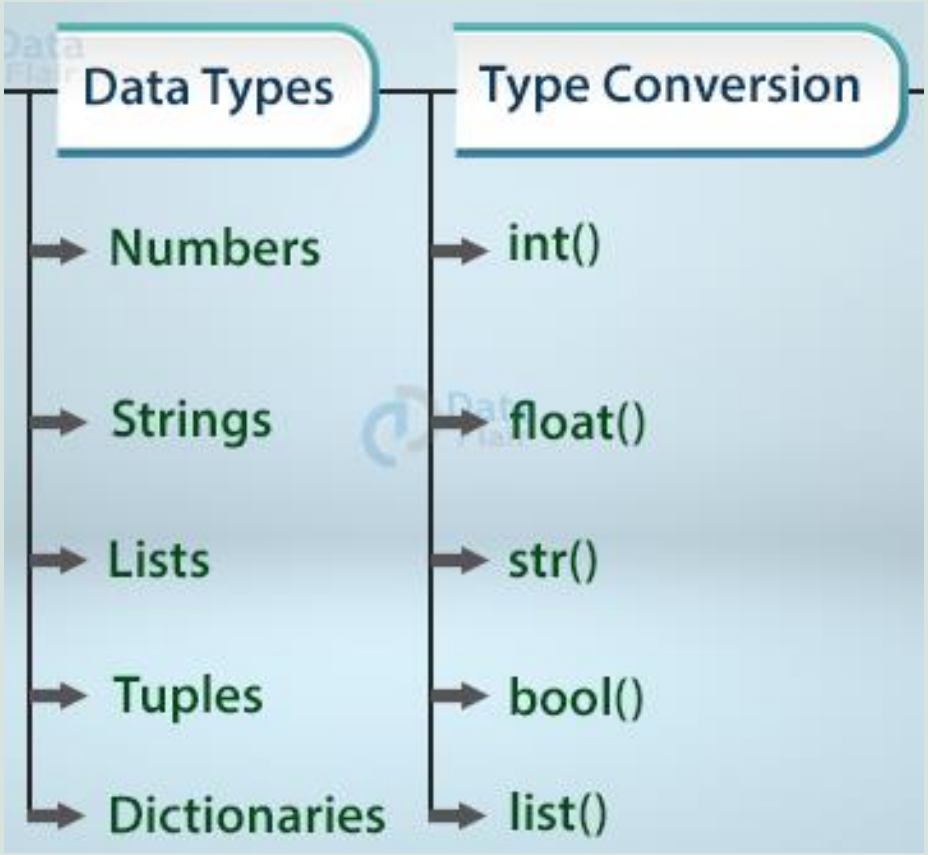
- Minimum and maximum values of the image

```
print(im_f[:,:,:].min())
print(im_f[:,:,:].max())
```

Python Variable Types

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Python Conversion of Types



Python Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Python Comparison Operators

- When
  - a=3
  - b=5

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Bitwise Operators

- When a=0011 1100 b=0000 1101

Operator	Description	Example
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Logical Operators

- When a=1 and b=1

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership and Identity Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Python Image Operator Functions

- from PIL import Image, ImageMath
- abs(*image*)
  - Absolute value.
- convert(*image*, *mode*)
  - Convert image to the given mode.
  - The mode must be given as a string constant.
- float(*image*)
  - Convert image to 32-bit floating point.
  - This is equivalent to convert(image, “F”).
- int(*image*)
  - Convert image to 32-bit integer. This is equivalent to convert(image, “I”).
  - Note that 1-bit and 8-bit images are automatically converted to 32-bit integers if necessary to get a correct result.
- max(*image1*, *image2*)
  - Maximum value.
- min(*image1*, *image2*)
  - Minimum value.

Python
Generating and Processing an Image

```
img = np.zeros([100,100,3],dtype=np.uint8)
# generate a new black image
img = Image.new( 'RGB', (2000,2000), "black")
# generate the pixel map
pixels = img.load()
for i in range(img.size[0]):
        # for every pixel:
        for j in range(img.size[1]):
        #do some stuff that requires i and j as parameter
                if pixels[i,j] == (255, 0, 0):
                        pixels[i,j] = (0, 0 ,0)
```

Python Special Variables

- import math

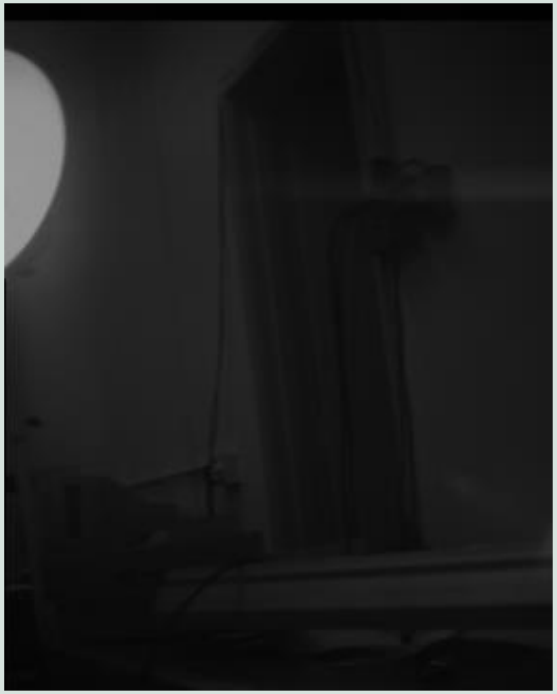
  math.nan

  math.pi

- import sys

  sys.float_info.epsilon

Image Enhancement



Dark image

Image after enhancement

Pixelwise Operations: Illumination Adjustment

- General form: g(x,y)=T[f(x,y)]
- Add to or subtract from image intensities f(x,y) a constant number b

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

Original

B=-50

B=+50

Pixelwise Operations: Contrast Adjustment

- Multiply image intensities f(x,y) a constant number a

$$g(x,y) = af(x,y)$$

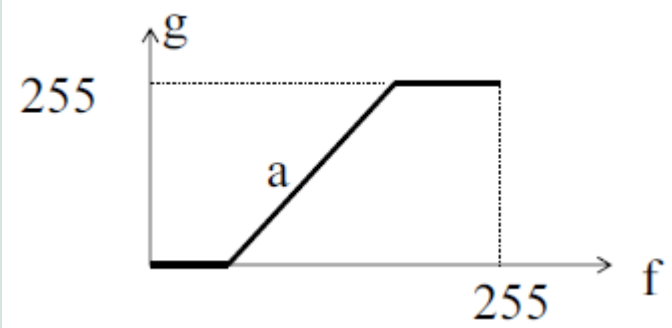
Original

a <1

a >1

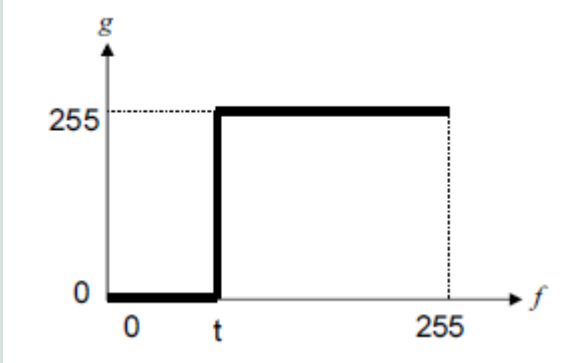
Pixelwise Operations: Illumination & Contrast Adjustment

$$g(x,y) = af(x,y) + b$$



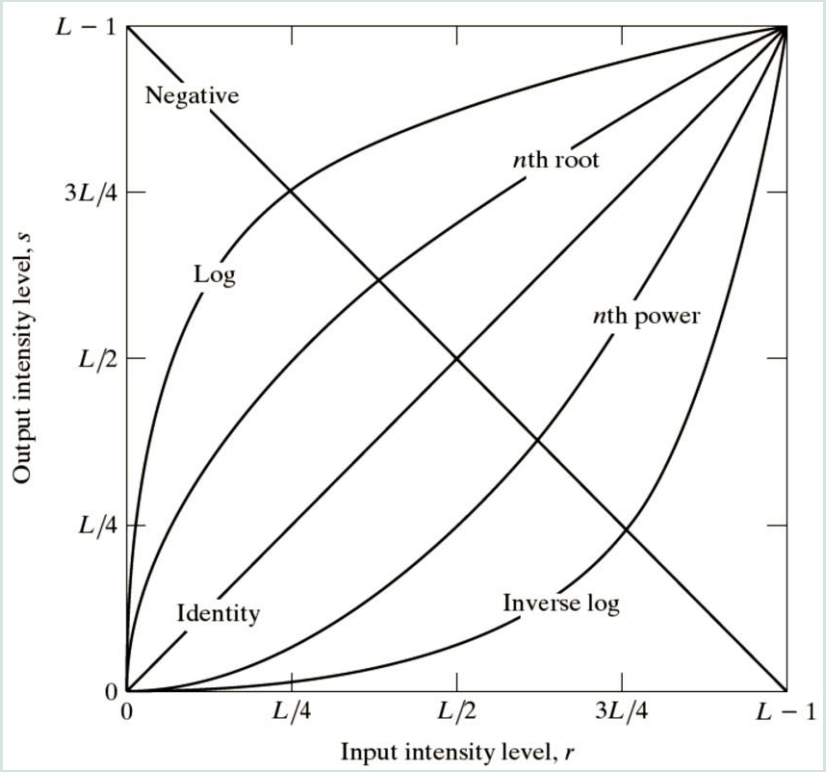
Pixelwise Operations: Simple Thresholding

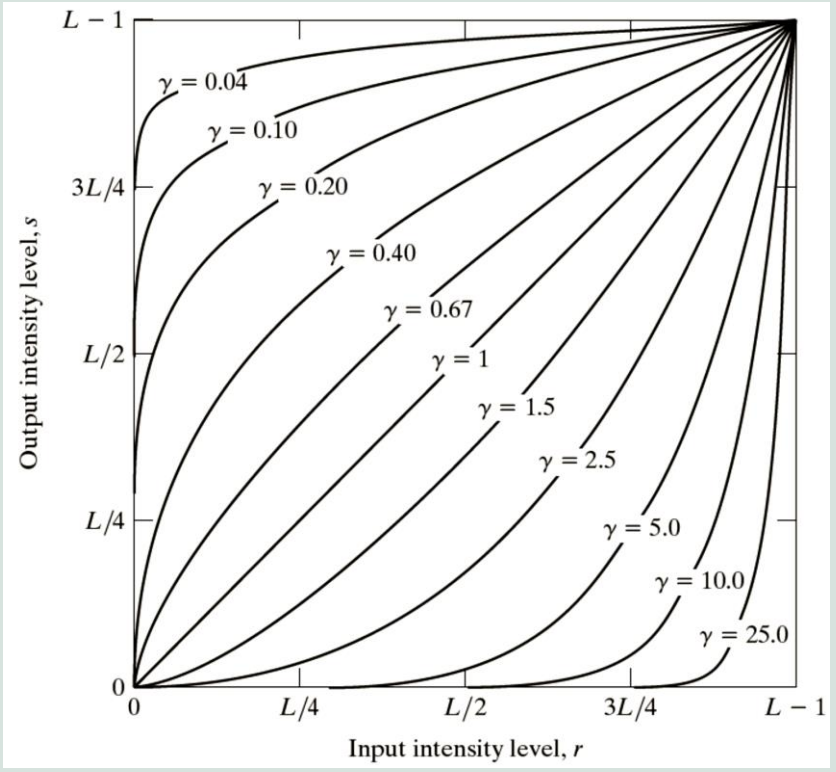
$g(x,y) = f(x,y) \geq t$



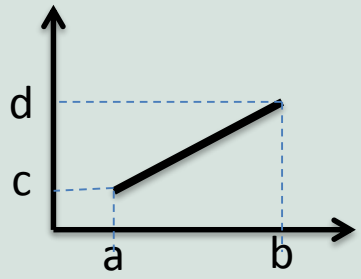
Pixelwise Operations: Logarithmic and Gamma Transformations

- Logarithmic transformations g= c*log(1+f(x,y))
- Gamma corrections g= c*f(x,y)$^{\gamma}$



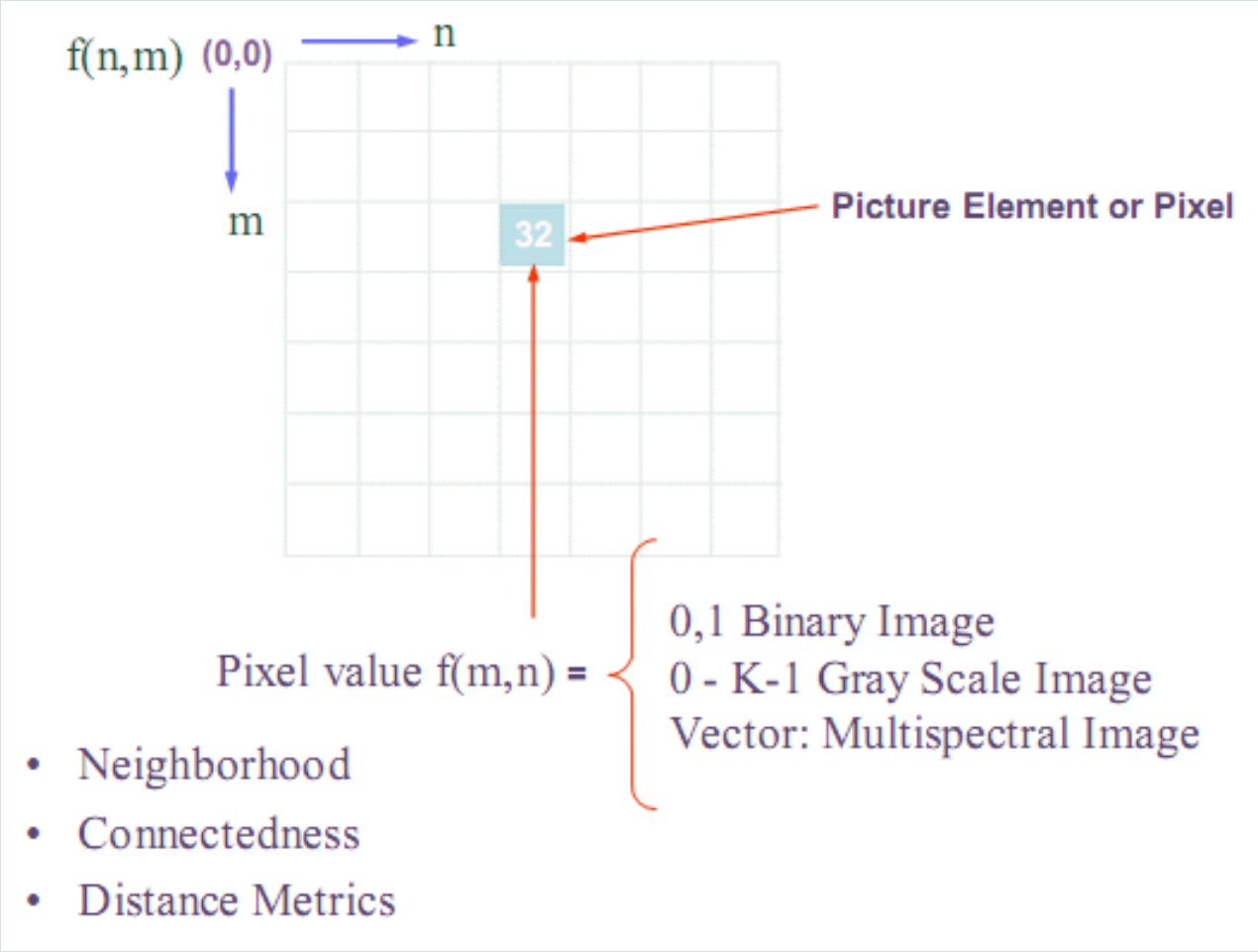


Python Adjusting Image Intensities




```
def imadjust(x,a,b,c,d,gamma=1):
    # Similar to imadjust in MATLAB.
    # Converts an image range from [a,b] to [c,d].
    # The Equation of a line can be used for this transformation:
    # y=((d-c)/(b-a))*(x-a)+c
    # However, it is better to use a more generalized equation:
    # y=((x-a)/(b-a))^gamma*(d-c)+c
    # If gamma is equal to 1, then the line equation is used.
    # When gamma is not equal to 1, then the transformation is not
    linear.
    y = (((x - a) / (b - a)) ** gamma) * (d - c) + c
    return y
```

Pixel-Related Concepts



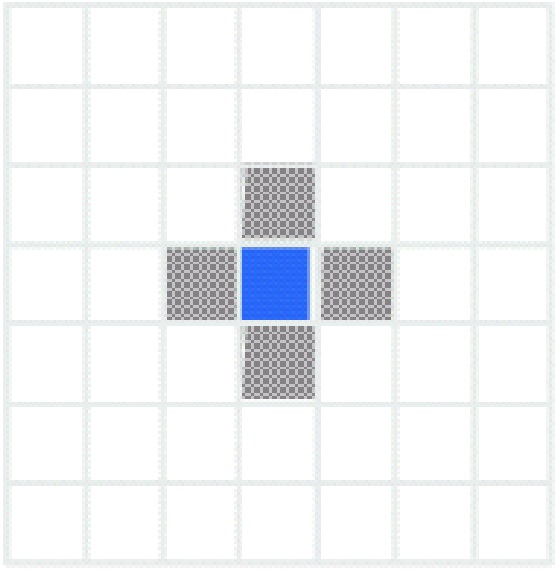
Pixel Neighborhood

- Except on borders of the array, any point (m,n) has 8 neighbor pixels

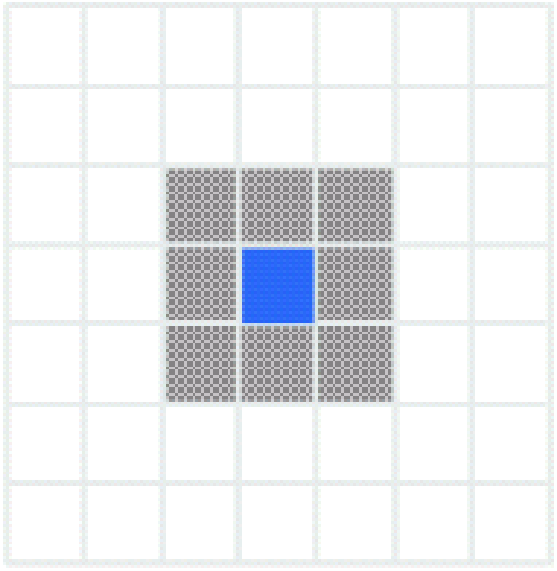
$$\begin{bmatrix} (m-1,n-1) & (m-1,n) & (m-1,n+1) \\ (m,n-1) & (m,n) & (m,n+1) \\ (m+1,n-1) & (m+1,n) & (m+1,n+1) \end{bmatrix}$$

- Note that diagonal neighbors $\sqrt{2}$ units away from (m,n) while horizontal and vertical neighbors are only 1 unit away.

Pixel Connectedness

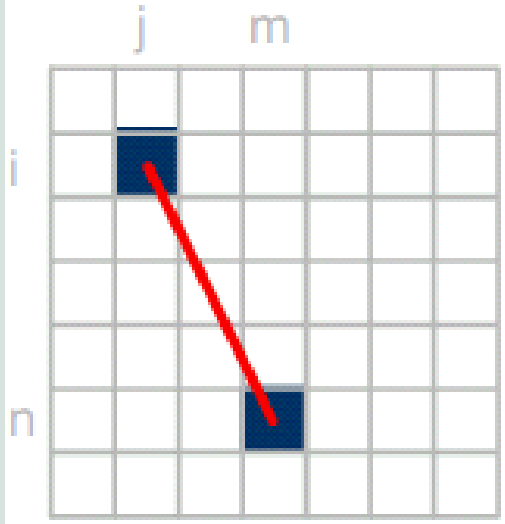


Four Neighbor



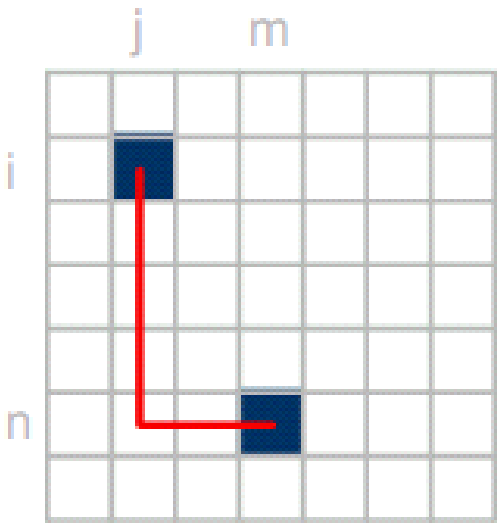
Eight Neighbor

Distance Metrics



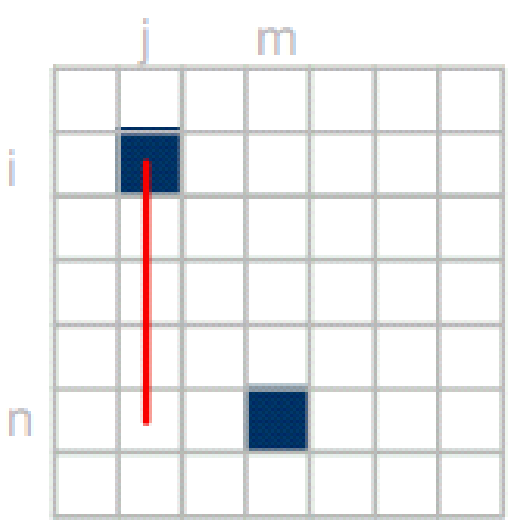
Euclidean Distance

$$= \sqrt{(i-n)^2 + (j-m)^2}$$



City Block Distance

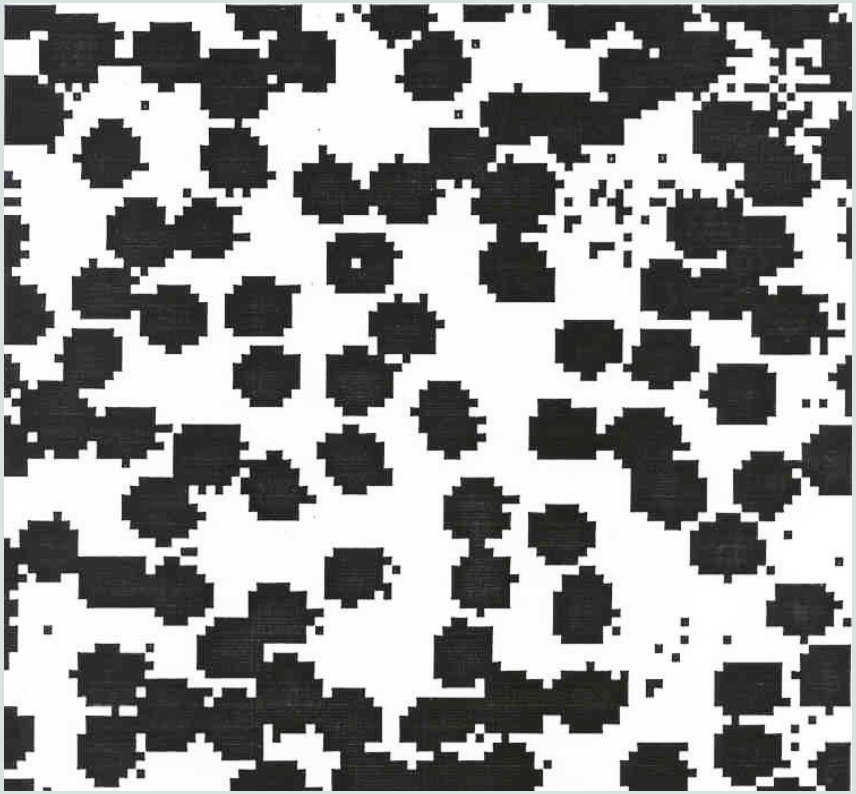
$$= |i-n| + |j-m|$$



Chessboard Distance

$$= \max[\ |i-n|, |j-m|\ ]$$

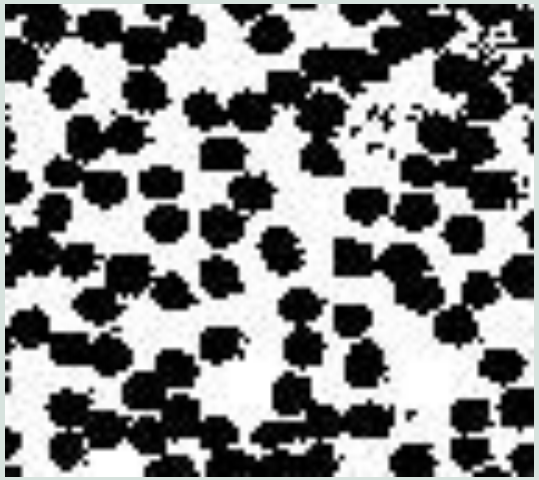
Noise Removal:
Red Blood Cell Image



- Many blood cells are separate objects
- Many touch – bad!
- Salt and pepper noise from thresholding

Removing Salt-and-Pepper Noise

- Change pixels all of whose neighbors are different:
  - see hole filled at bottom
- Delete objects that are tiny relative to targets: see some islands removed at right



Neighborhood (Spatial) Operations for Enhancement

- Linear Filtering:
  - low-pass filter masks
- Nonlinear filtering:
  - median, max, min, and other order-statistic operators
- Morphological filters

Linear filtering

- Convolution with a filter mask is a weighted sum of pixel intensities

5	5	5	5	5	5	5
5	5	5	5	5	5	5
5	5	5	5	5	5	5
0	0	0	0	5	5	5
0	5	0	0	0	5	0
5	0	0	5	0	0	0
0	0	0	0	0	0	0

Input

1/5 x

0	1	0
1	1	1
0	1	0

Mask
(Kernel)

Mean Filter

5	5	5	5	5	5	5
5	5	5	5	5	5	5
4	4	4	4	5	5	5
1	2	1	2	3	5	4
2	1	1	1	2	2	2
2	2	1	1	1	1	0
1	0	0	1	0	0	0

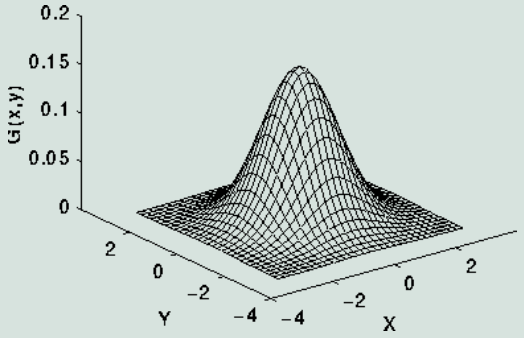
Output

Linear Filtering: Lowpass Filters

- Lowpass filters reduce noise,…
- But smoothes edges
- Mean filter is a LPF
- Gaussian Filter is a better LPF

LP Filters: Gaussian

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



Gaussian Filters are separable: you can filter rows then columns…

.006	.061	.242	.383	.242	.061	.006

or, apply 2D convolution:

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

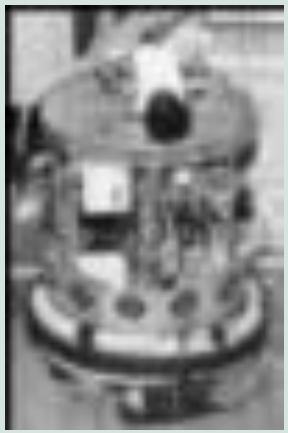
LP Filters: Gaussian

- Effects of sigma:

Original

sigma=1

5 x 5



sigma=2

9 x 9

sigma=4

15 x 15

Types of Noise

- Noise is often modeled as:
  - Gaussian (mean, sigma)
  - Salt-and pepper (0 or 255 with prob p)

Original

Gaussian
sigma=20
mean=0

Salt-and-pepper
p=0.3

Nonlinear Filters

- Linear Filters are good for removing Gaussian noise, but not good for salt-and-pepper (impulsive) noise
- Median filter, alpha-trimmed-mean filter better for impulsive noise

Median, Max, Mean Filters

123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Neighbourhood values:

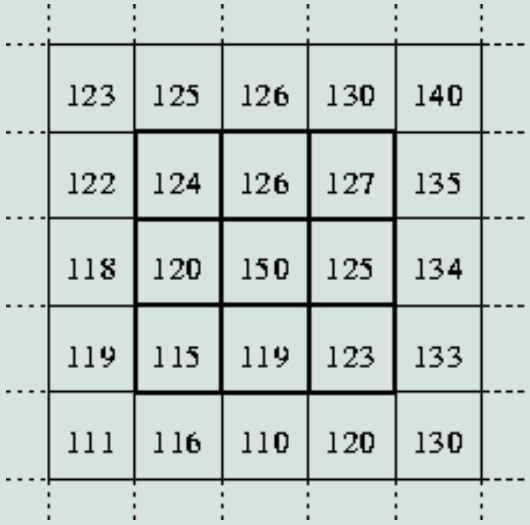
115, 119, 120, 123, 124,
125, 126, 127, 150

Median value: 124

original noisy 5 x 5 7 x 7

Alpha-trimmed-mean

- Remove the highest and lowest scores, then take average of the rest



Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Avg: 123.6

Median value: 124