Introduction

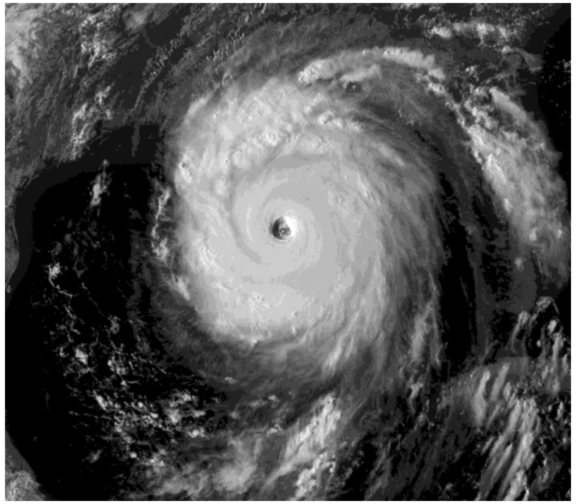
What is image processing

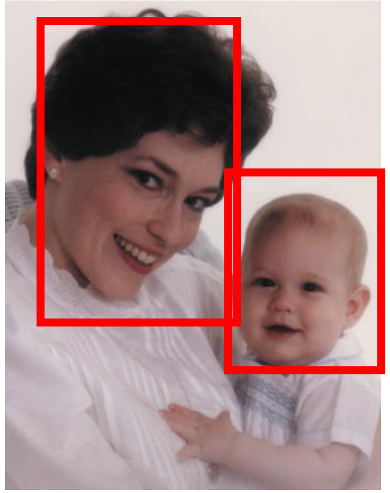
Operations on images to:

- Enhance an image
- Extract useful information
- Analyze image and make decisions

Applications

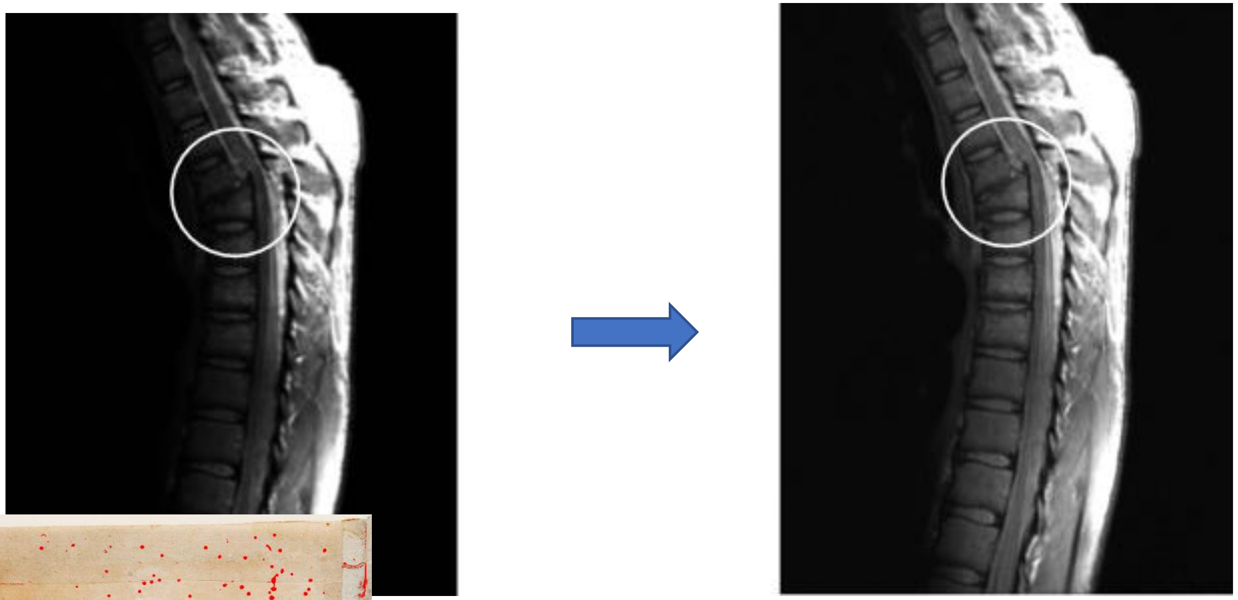
- Medical image analysis
- Artificial intelligence
- Augmented reality
- Surveillance
- Robotics
- …

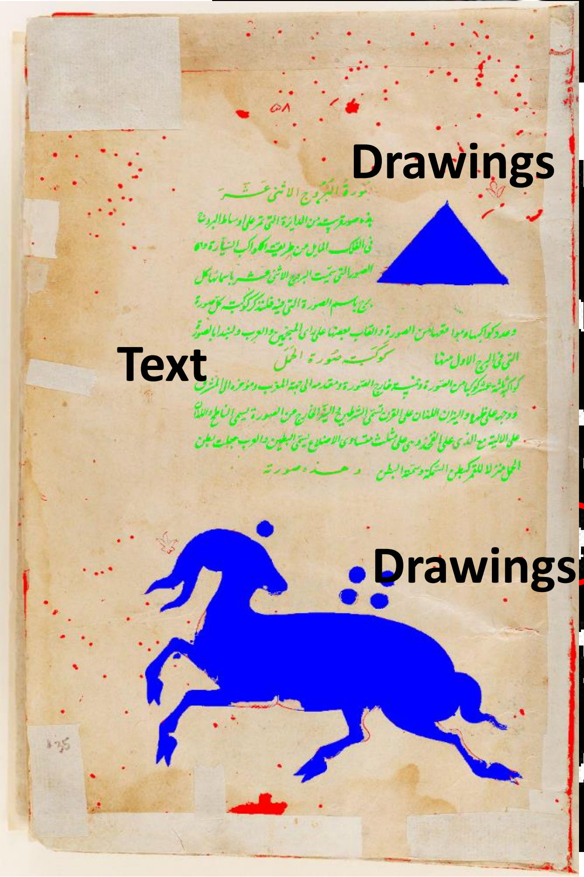


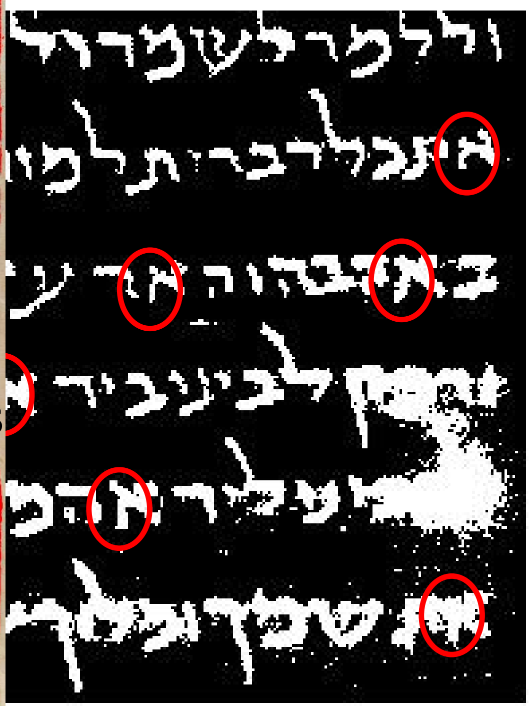


Purposes

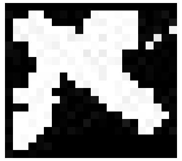
- Image enhancement
  - A better image
- Image retrieval
  - Seek for image of interest
- Image recognition
- Image segmentation







The query image (model)



Flower

What is a digital image?

- Two-dimensional function f(x,y)
  f(x,y) is the color or intensity in location (x,y)
- Image resolution, e.g. 400 dpi
- Depth – a number of bits to represent the brightness or color (e.g. 8 bit-image will have a range $[0, 255]$)

Y

177	180	175	169	179	179	166	111	160	180
140	148	165	163	179	186	153	63	105	179
84	57	104	154	170	174	154	64	84	174
140	96	60	72	87	101	141	77	83	173
182	172	97	42	49	54	74	71	72	164
181	156	63	52	111	142	103	58	61	156
185	140	52	115	171	177	171	113	55	137
179	129	49	132	180	176	168	154	83	132
166	132	53	130	182	179	169	166	112	92
173	170	129	154	183	177	169	168	141	125

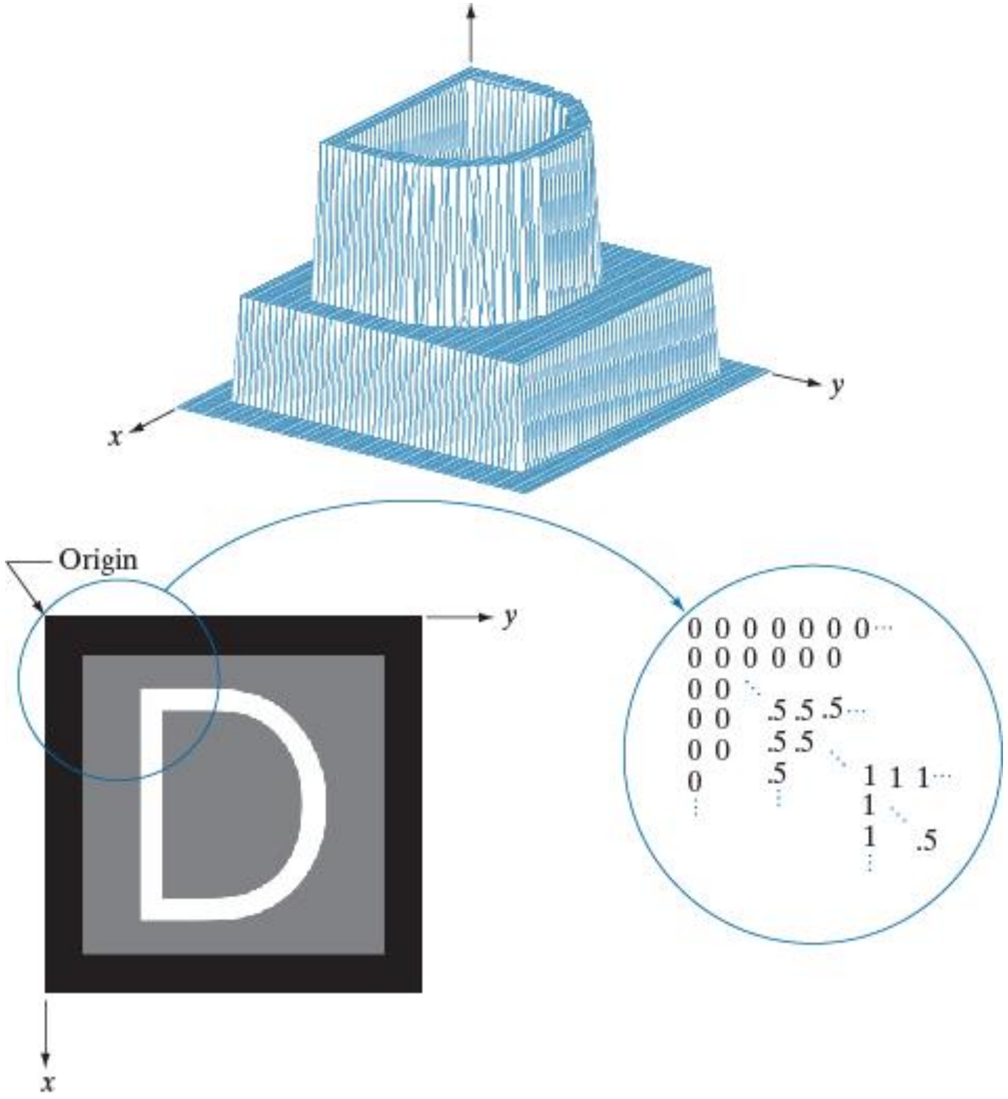
pixel

X

What is a digital image?

a
b c

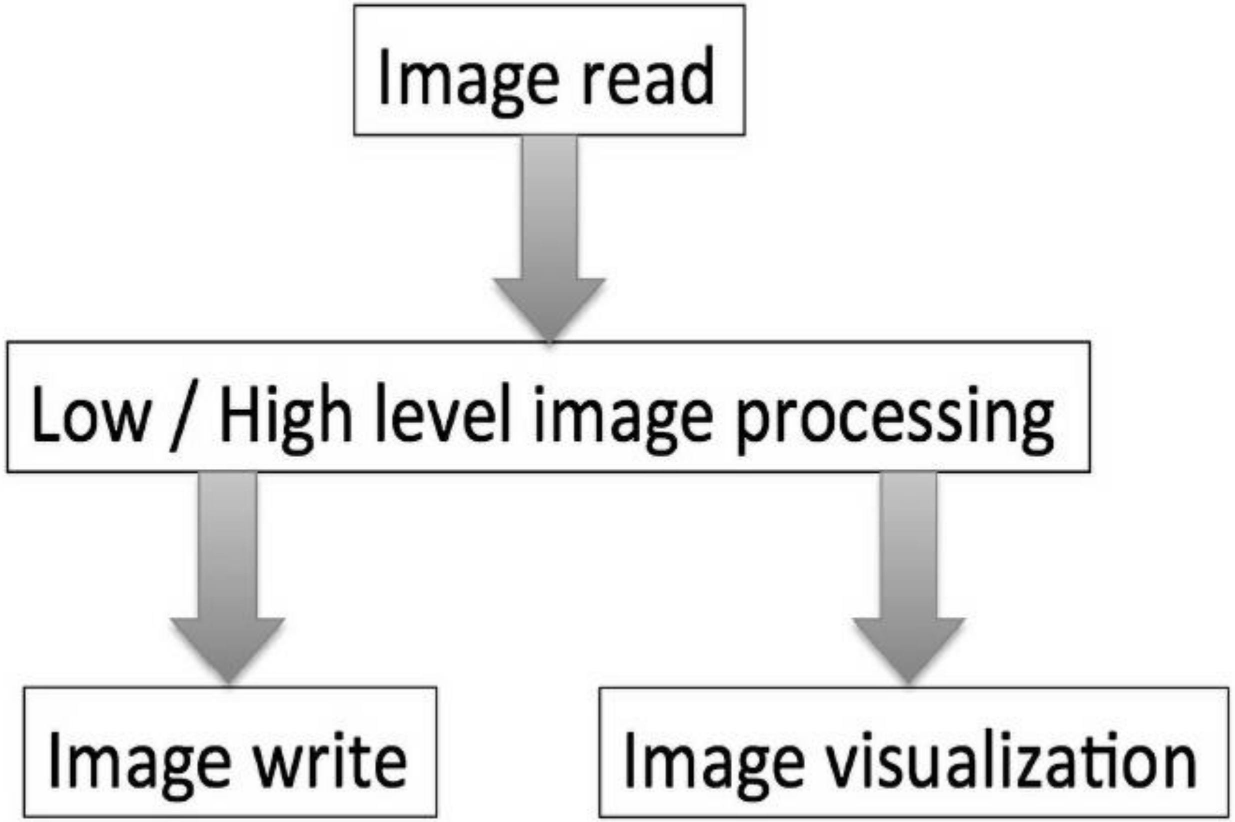
FIGURE 2.18
(a) Image plotted as a surface. (b) Image displayed as a visual intensity array. (c) Image shown as a 2-D numerical array. (The numbers 0, .5, and 1 represent black, gray, and white, respectively.)



Reading, writing and displaying images

- In this course we will use
  - OpenCV for reading and writing images (`import cv2`)
  - Matlpotlib pyplot module to display images (`import matplotlib.pyplot`)

Image processing workflow



Reading, writing and displaying images with OpenCV

```
import cv2
```

- `cv2.imread(filename, flag)` reads an image
  - Flags option:
  - `cv2.IMREAD_GRAYSCALE` or `0`
  - `cv2.IMREAD_COLOR` or 1
  - `cv2.IMREAD_UNCHANGED` or `-1`

- `cv2.imwrite(filename,image)` writes an image into the file directory
  - `cv2.imwrite('my_image.jpg', img)`

Reading, writing and displaying images with OpenCV

```
import matplotlib.pyplot as plt
import cv2

 #cv2 module's imread to read an image as an ndarray.
img = cv2.imread('filename', flag)

#display an image
plt.imshow(img)
plt.show()

# cv2.imwrite will take an ndarray and store it.
cv2.imwrite('filename', img)
```

!

always check the type, cv2.imread doesn't produce an error message when something is wrong in read

Color Image Processing

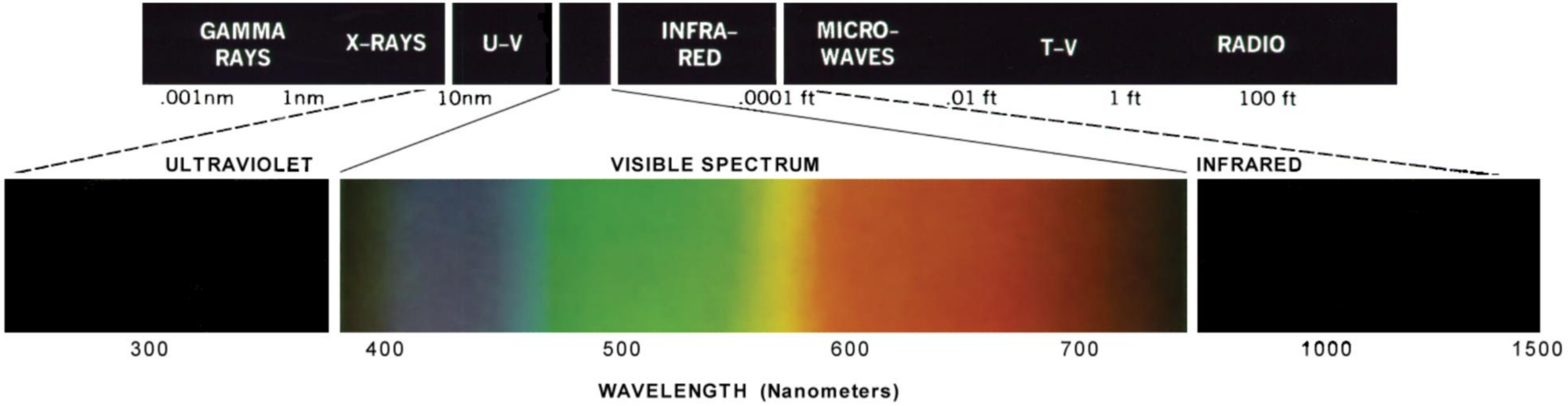
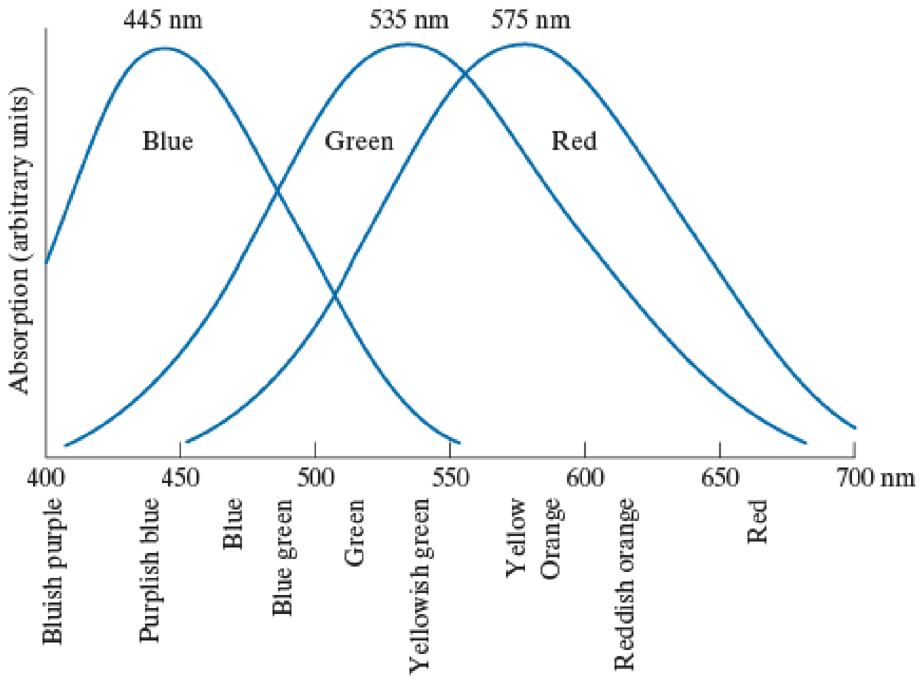


FIGURE 6.2
Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lighting Division.)

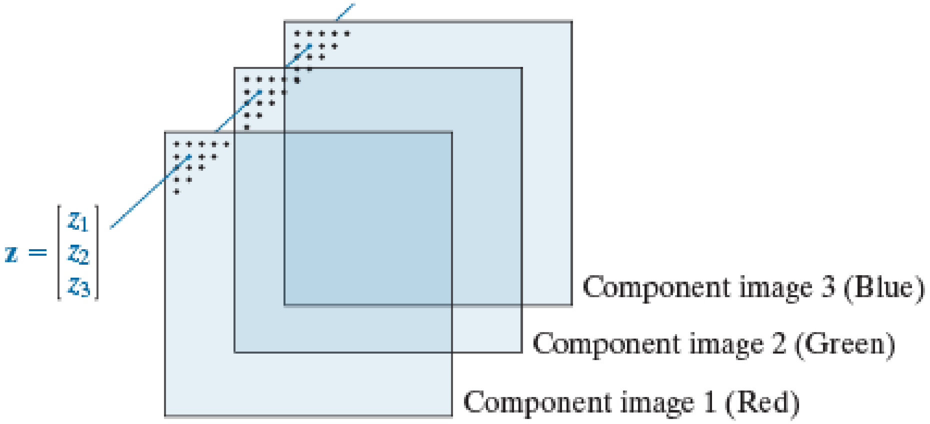
Color Image Processing

FIGURE 6.3 Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.



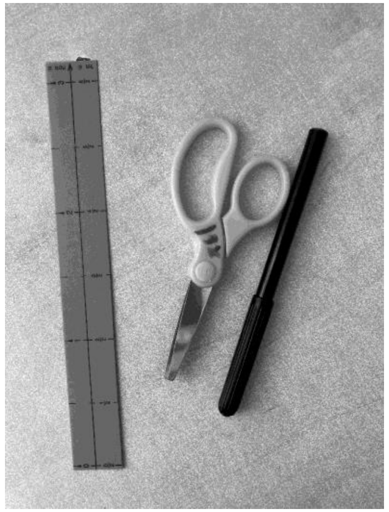
The RGB model

FIGURE 2.43 Forming a vector from corresponding pixel values in three RGB component images.



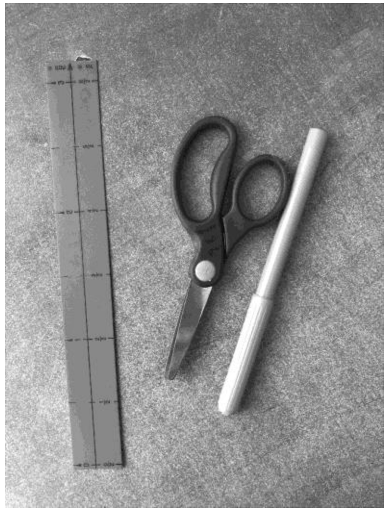
Color image

Red



Green

Blue



Load images in color or grayscale

- OpenCV

imgColor = cv2.imread(‘filename’, cv2.IMREAD_COLOR)
imgGrayscale = cv2.imread(‘filename’, cv2. GRAYSCALE)

or

imgGrayscale = cv2.cvtColor(imgColor, cv2.COLOR_BGR2GRAY)

!

Color image loaded by OpenCV is in BGR mode. But Matplotlib displays in RGB mode. So color images will not be displayed correctly in Matplotlib if image is read with OpenCV.

```
imgColor = cv2.imread('RGBimage.jpg', cv2.IMREAD_COLOR)
imgColor_fixed = cv2.cvtColor(imgColor, cv2.COLOR_BGR2RGB)
```

imgColor

imgColor_fixed

Splitting and Merging Color Channels

- `cv2.split()` - divides a multi-channel array into several single-channel arrays.
- `cv2.merge()` merges several arrays to make a single multi-channel array. All the input matrices must have the same size.

Splitting and Merging Color Channels

```
# split color channels
colorImg = cv2.imread('RGBimage.jpg', cv2.IMREAD_COLOR)
b,g,r=cv2.split(colorImg)


plt.figure()
plt.subplot(141); plt.imshow(r, cmap = 'gray'); plt.title('Red Channel')
plt.subplot(142); plt.imshow(g, cmap = 'gray'); plt.title('Green Channel')
plt.subplot(143); plt.imshow(b, cmap = 'gray'); plt.title('Blue Channel')

# Merge the individual channels into a RGB image.
imgMerged = cv2.merge((r, g, b))

# Display the merged output.
plt.subplot(144)
plt.imshow(imgMerged)
plt.title('Merged Output')
```

Vertical and Horizontal flip

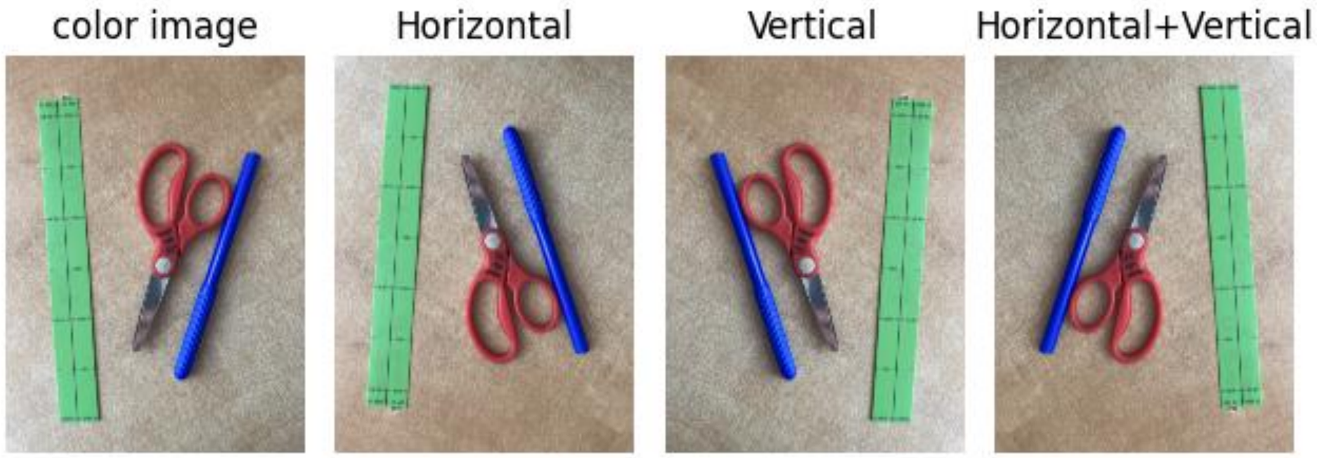
dst = cv2.flip(src, flipCode)

Parameters:
src: Input array.
dst: Output array of the same size and type as src.
flip code: A flag to specify how to flip the array; 0 means flipping around the x-axis and positive value (for example, 1) means flipping around y-axis. Negative value (for example, -1) means flipping around both axes.

Vertical and Horizontal flip

dst = cv2.flip(src, flipCode)

Parameters:
src: Input array.
dst: Output array of the same size and type as src.
flip code: A flag to specify how to flip the array; 0 means flipping around the x-axis and positive value (for example, 1) means flipping around y-axis. Negative value (for example, -1) means flipping around both axes.



Shape and resize

```
#get the image shape
img.shape
```

```
#resize
cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

Parameter	Description
src	[required] source/input image
dsize	[required] desired size for the output image
fx	[optional] scale factor along the horizontal axis
fy	[optional] scale factor along the vertical axis
interpolation	INTER_NEAREST INTER_LINEAR (used by default) INTER_CUBIC

Shape and resize

```python
# this will resize the image to have 100 cols (width) and 50 rows (height):
resized_image = cv2.resize(imgColor, (100, 50))
# this will resize both axes by half:
small = cv2.resize(imgColor, (0,0), fx=0.5, fy=0.5)

print(imgColor.shape)
print(small.shape)
print(resized_image.shape)
```

!

`(width, height)`

Output:
(4032, 3024, 3)
(2016, 1512, 3)
(50, 100, 3)

Image interpolation

8	10	200
120	5	60
34	108	75

Original image

8	?	10	?	200	?
?	?	?	?	?	?
120	?	5	?	60	?
?	?	?	?	?	?
34	?	108	?	75	?
?	?	?	?	?	?

Image enlarged 2 times

Nearest neighbor interpolation

8	10	200
120	5	60
34	108	75

Original image

8	8	10	10	200	200
8	8	10	10	200	200
120	120	5	5	60	60
120	120	5	5	60	60
34	34	108	108	75	75
34	34	108	108	75	75

Image enlarged 2 times

Bilinear and bicubic interpolations

8	10	200
120	5	60
34	108	75

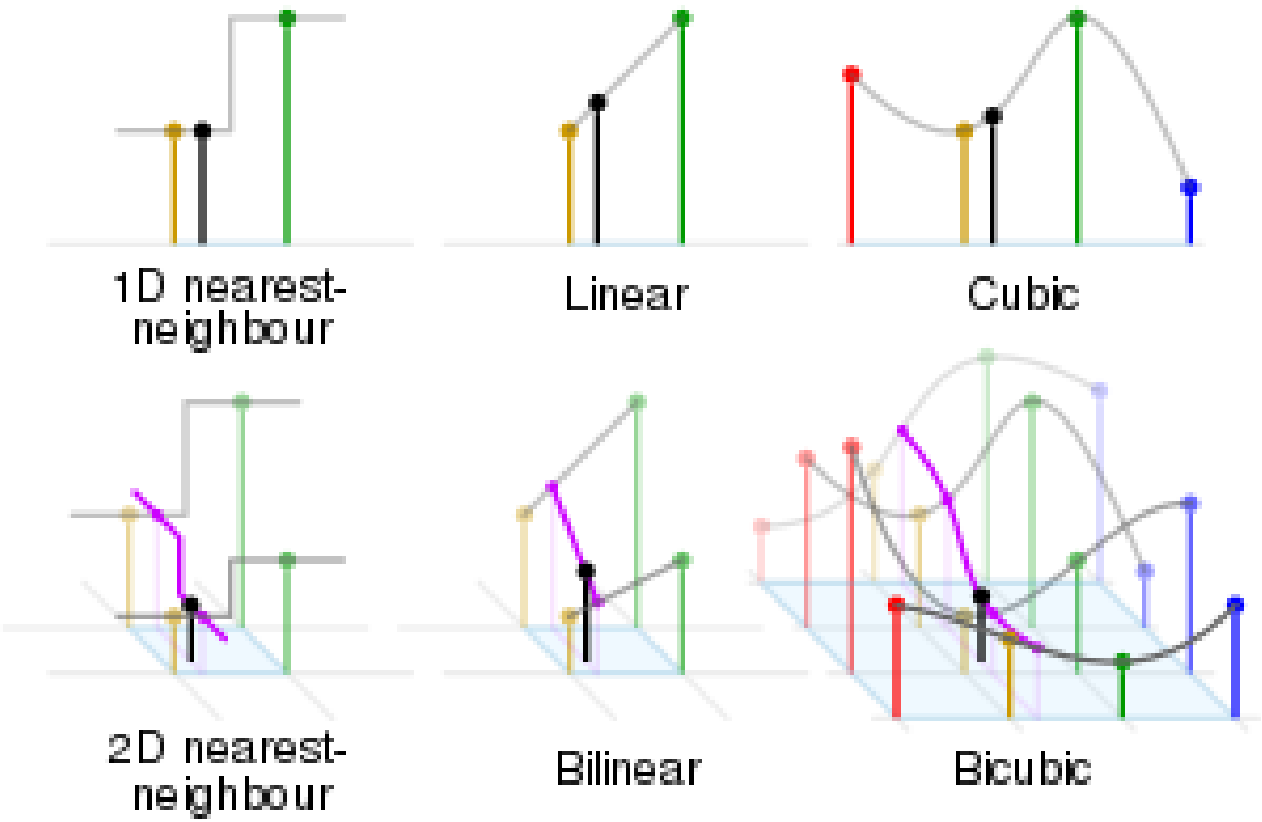
Original image

8		10		200	
120		5		60	
			?		
34		108		75	

bilinear

Image enlarged 2 times

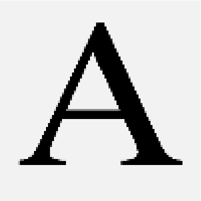
Bilinear - weighted sum of four nearest neighbors
Bicubic - weighted sum of 16 nearest neighbors



Comparison of some 1- and 2-dimensional interpolations. Black and red/yellow/green/blue dots correspond to the interpolated point and neighbouring samples, respectively. Their heights above the ground correspond to their values.

https://en.wikipedia.org/wiki/Multivariate_interpolation

Resize



Nearest neighbor **Bilinear** **Bicubic**

Lab session