CV Practice Class
2. OpenCV Tutorial, Edge Detection, Image Pyramid, Template Matching

2017-07-04

FIRA 인공지능 에이전트 과정
SNUVL Lab

실습수업 구성

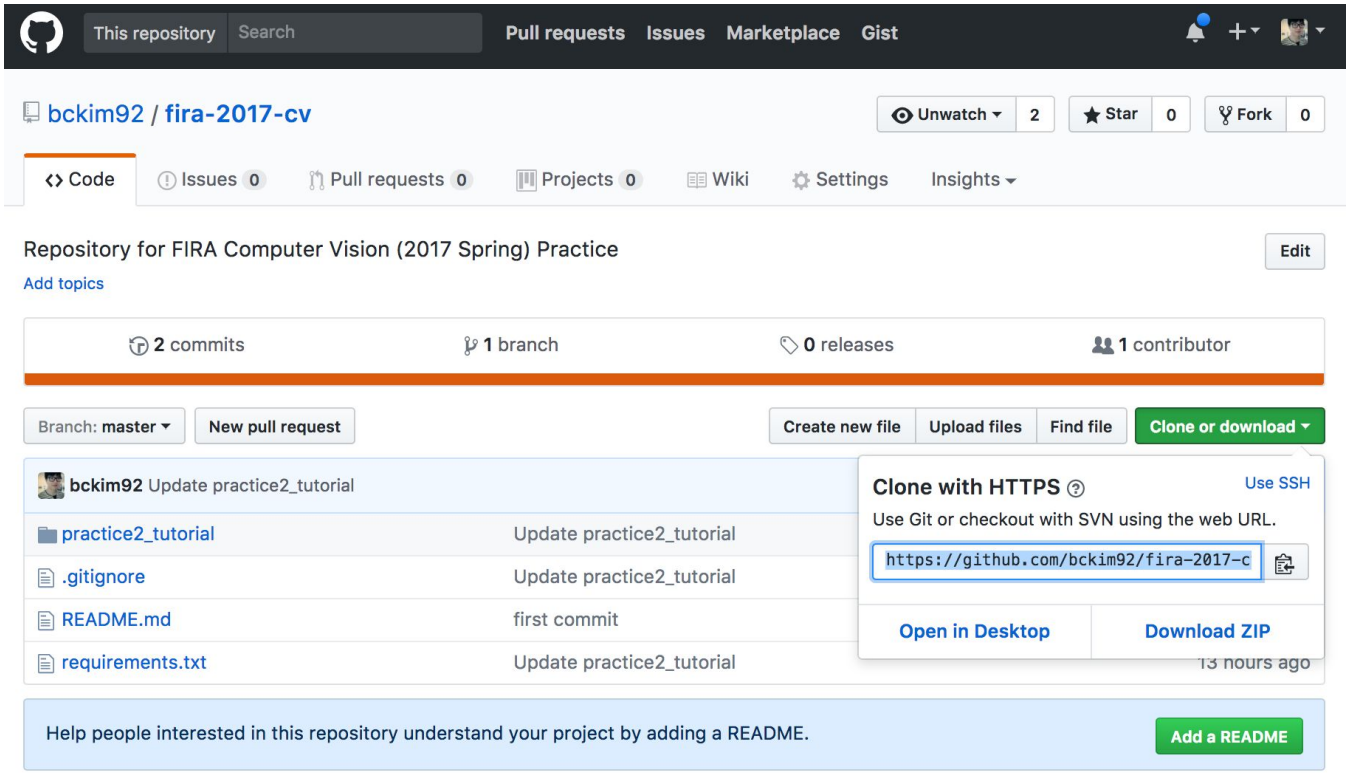
- 실습수업시간
  - 과제정답확인 및 질문답변
  - OpenCV API 소개 및 코드 작성
  - 과제 출제

- 과제
  - API로 구현된 함수를 numpy랑 python으로 직접 구현하기

실습코드 받는법

https://github.com/bckim92/fira-2017-cv

```
$ git clone https://github.com/bckim92/fira-2017-cv.git
$ cd fira-2017-cv/practice2_tutorial
$ jupyter notebook
```



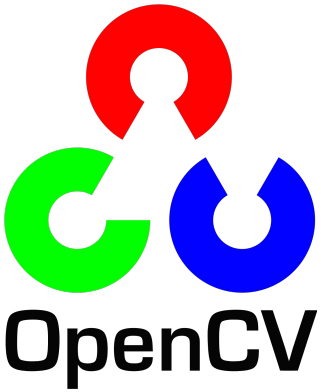
Contents

OpenCV-Python

Introduction
Image manipulation
Draw objects

OpenCV-Python

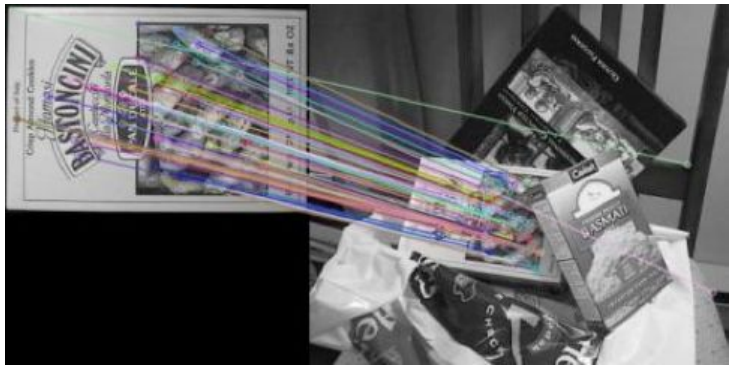
- OpenCV
  - **Computer vision library** started from 1995(Intel)
  - Now supports a multitude of algorithms related to CV and ML (a little of)
- OpenCV-Python
  - OpenCV is basically written in C++
  - OpenCV-Python is a **Python wrapper** of OpenCV
- Prior knowledge of **Python** and **Numpy** is needed
  - A Quick guide to Python - A Byte of Python
  - Numpy Quickstart Tutorial / Justin Johnson's Numpy Tutorial



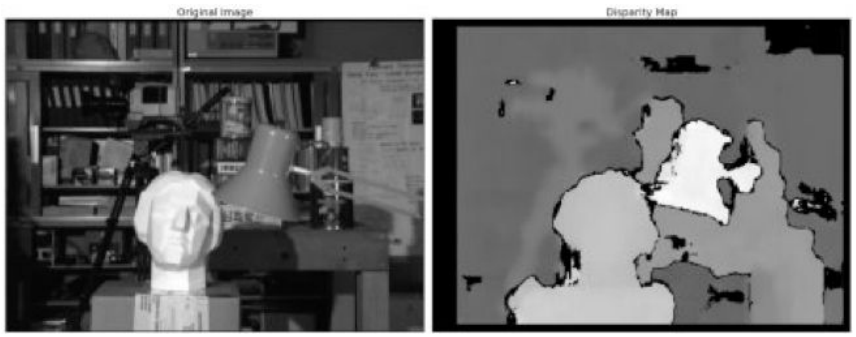
OpenCV-Python

- Examples of algorithms with OpenCV
  - Face Detection

  - Feature extraction/matching



  - Depth map for stereo images



  - Image inpainting

Using OpenCV-Python

- Import OpenCV-Python package “cv2”

```
import numpy as np
import cv2 # OpenCV-Python
%matplotlib inline
import matplotlib.pyplot as plt
```

- **Numpy** arrays are data structure used in cv2
  - Converted from/to CvMat(OpenCV in C++) by OpenCV-Python
  - Also used in many python packages

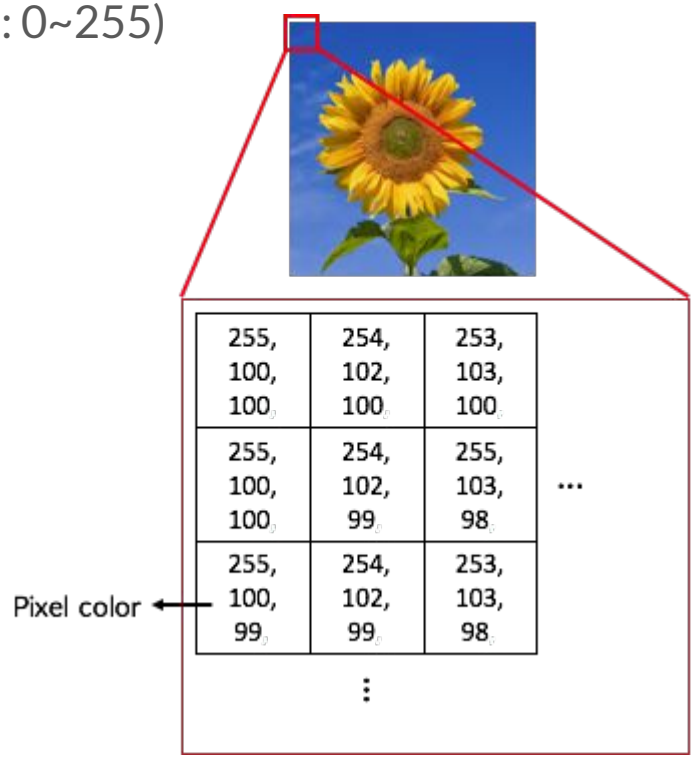
Open/Display an Image

- Open an image

```
img = cv2.imread('image.jpg', cv2.IMREAD_COLOR)
```

- The output is a Numpy array
  - 3D (H x W x C for color) / 2D (H x W for grayscale)
  - Top-left to bottom-right
  - Data type (dtype): `np.uint8` (1-byte unsigned: 0~255)

- Flag specifies the way image should be read
  - cv2.IMREAD_COLOR
  - cv2.IMREAD_GRAYSCALE
  - cv2.IMREAD_UNCHANGES

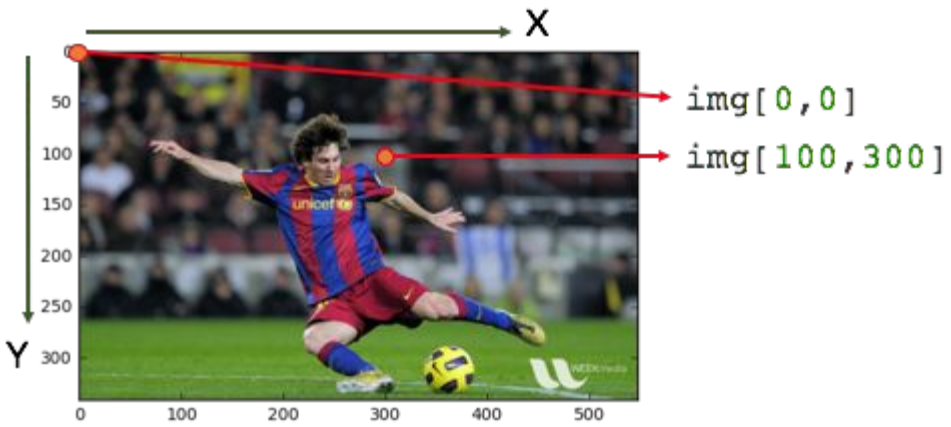


Open/Display an Image

- Display an image using Matplotlib

```
# display an image using matplotlib
plt.imshow(img)  # => The output in wrong color!!
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

  - `plt.imshow(img)` displays an (RGB, RGBA, grayscale) image
  - OpenCV represents RGB images as Numpy arrays in **REVERSE** order (**BGR** not RGB)
  - `cv2.cvtColor(img, conversion)` provides conversion among many colortypes



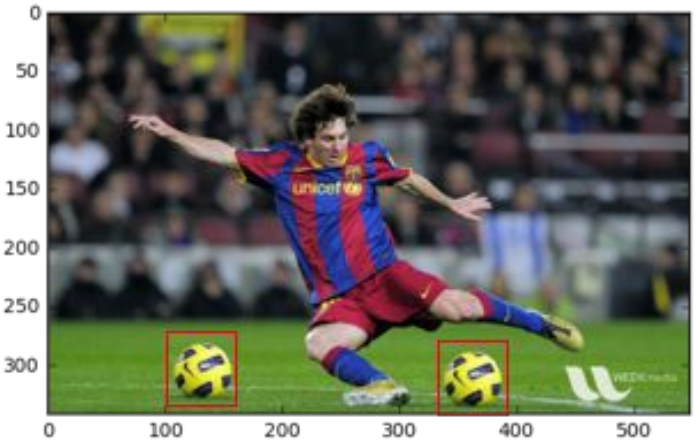
Modify Pixels & ROI

- Pixel and ROI(Region of Interest) can be accessed by Numpy indexing
  - [row, column] ordering - same as matrix indexing

```
# Access a pixel value (BGR order)
img[50, 235]
```

```
=> array([27, 25, 24], dtype=uint8)
```

```
# ROI is obtained using Numpy indexing
ball = img[280:340, 330:390]
img[273:333, 100:160] = ball
```



Draw Objects

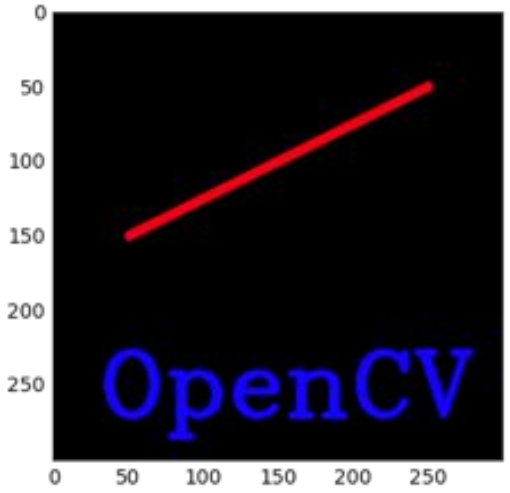
- Draw object (line, rectangle, circle, ellipse, polygon)
  - `cv2.line(), cv2.rectangle(), cv2.circle(), cv2.ellipse(), cv2.polyline()`
- Put some text
  - `cv2.putText()`
- Arguments
  - `cv2.function(image, {properties of object})`
  - RGB order in color (not BGR)
  - X, Y order in position (not row, column)

Draw Objects

- Example

```
# cv2.line(image, startPoint, endPoint, rgb, thinkness)
cv2.line(img, (50,150), (250,50), (255,0,0), 5)

# cv2.putText(image, text, bottomLeft, fontType, fontScale,
rgb, thinkness, lineType)
font = cv2.FONT_HERSHEY_COMPLEX
cv2.putText(img, 'OpenCV', (30,270), font, 2, (0,0,255), 3,
cv2.LINE_AA)
```



Let’s Check the Code

1_getting_started.ipynb

Edge Detection

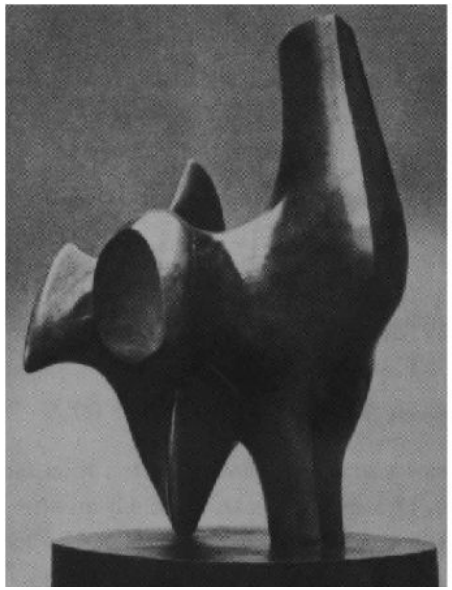
Image gradient

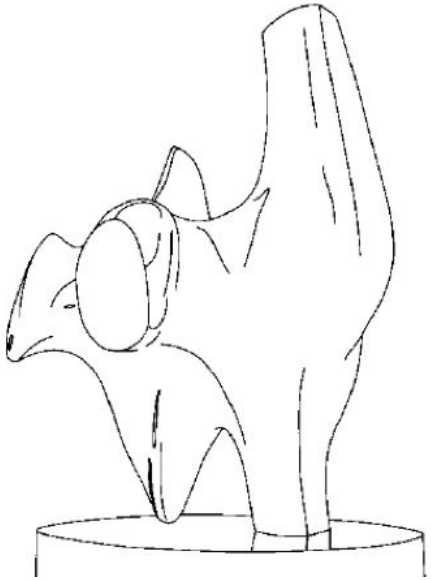
- Sobel operator

Canny edge detection

Edge Detection

- Finding discontinuity of the intensity in images
  - A salient feature of image
  - Much more compact representation than raw pixels
- HOW?
  - Image gradient: DoG, LoG, Sobel operator
  - Dealing with real, noisy images: Canny edge detector





(Image credit: Gunhee Kim's CV lecture slide)

Image Gradient

- The image gradient is approximated by filtering

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i,j+1}\right) + \left(I_{i+1,j} - I_{i,j}\right)\right)$$
$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i+1,j}\right) + \left(I_{i,j+1} - I_{i,j}\right)\right)$$

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \qquad \frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

- Sobel operator
  - A better approximation filter of size 3, 5, 7, …
  - OpenCV provides Sobel filters of high-dimensional derivatives.

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

Sobel 3x3 filter

-1	-4	-6	-4	-1
-2	-8	-12	-8	-2
0	0	0	0	0
2	8	12	8	2
1	4	6	4	1

Sobel 5x5 filter

Image Gradient

- Sobel operator in OpenCV

```
dst = cv2.Sobel(src, ddepth, dx, dy, ksize=3, scale=1.0)
```

  - `src`: input image(grayscale, 2-dim array [HxW])
  - ddepth: output image depth
  - dx: order of the derivative x
  - dy: order of the derivative y
  - `ksize`: size of the extended Sobel kernel; it must be 1, 3, 5 or 7
  - `scale`: (optional) scale factor for the computed derivative values

$$\text{dst} \approx \frac{\partial^{\text{dx+dy}}}{\partial^{\text{dx}}x \partial^{\text{dy}}y} \text{src}$$

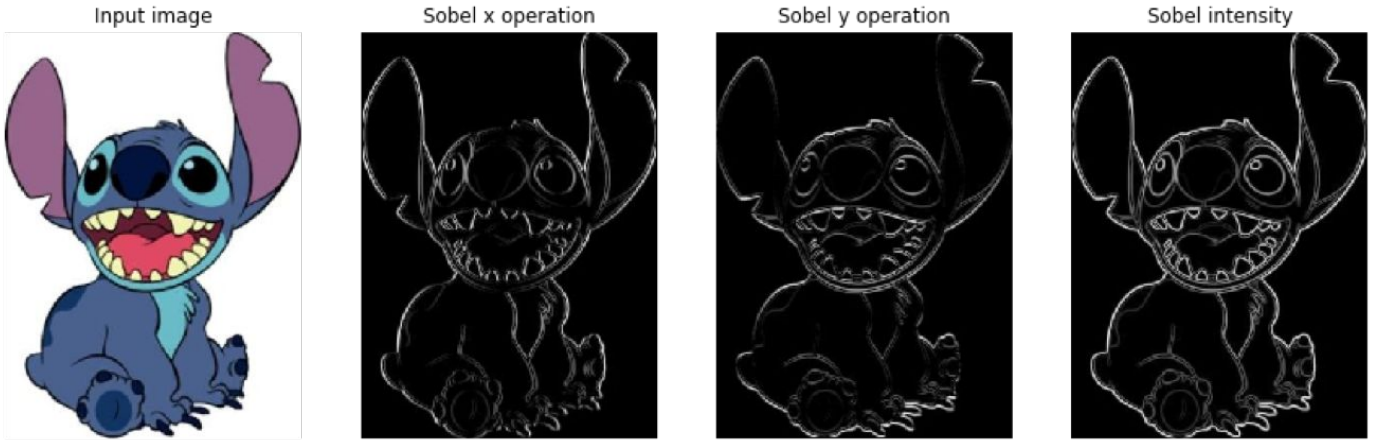


Image Gradient

- Sobel operator in OpenCV (example)

```
img_color = cv2.imread('images/stitch.jpg', cv2.IMREAD_COLOR)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

sobelx = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0)
```

Grayscale input image

64-bit float output

1st order derivative in x-direction

Input image

Sobel x operation

cv2.CV_64F
(Gray when 0)

Sobel x operation

cv2.CV_8U
(Black when 0)

Canny Edge Detector

- A popular edge detection algorithm
- The detection consists of the following procedures (multi-stage algorithm)
  - Noise reduction (5x5 Gaussian filter)
  - Finding image gradient from Sobel-x/y operator
  - Non-maximum suppression (edge-thining)
  - Hysteresis thresholding

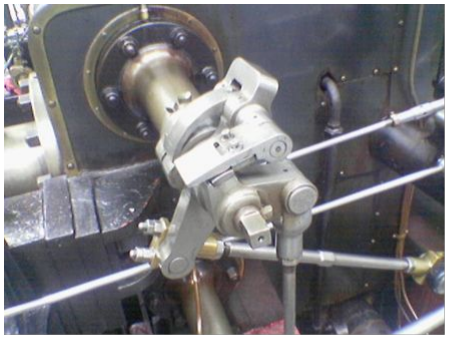
(Image credit: Gunhee Kim's CV lecture slide)

Canny Edge Detector

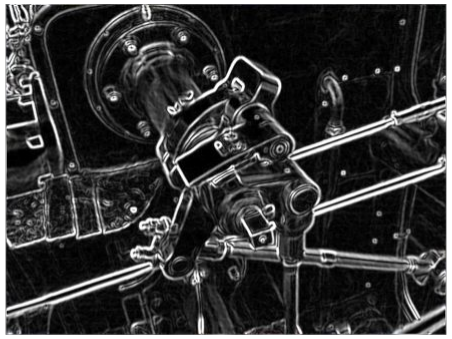
- Sobel operator in OpenCV

```
edges = cv2.Canny(image, threshold1, threshold2, apatureSize=3,
L2gradient=False)
```

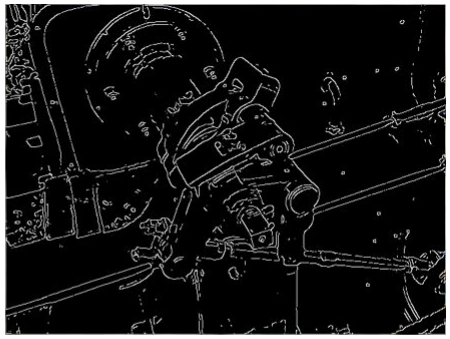
  - `image`: 8-bit grayscale input image
  - `threshold1/threshold2`: thresholds for the hysteresis procedure
  - `apertureSize`: aperture size for the Sobel() operator
  - `L2gradient`: A flag. `True` to use L_2-norm of gradients. `False` for L_1-norm



Input image



Sobel operator



Canny edge detector

(Image credit: Wikipedia)

Let’s Check the Code

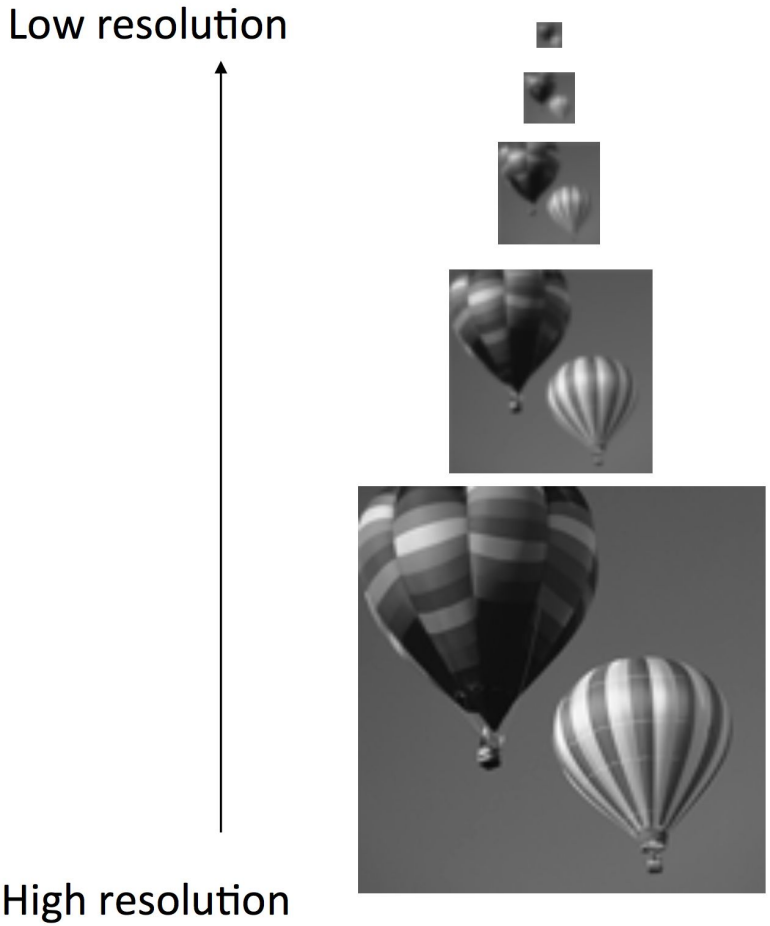
2_edge_detection.ipynb

Image Pyramid

Multi-resolution image pyramids
Image blending
Template matching

Multi-resolution Image Pyramids

- Inspired by human visual encoding
- Computationally efficient



(Image credit: Gunhee Kim's CV lecture slide)

Multi-resolution Image Pyramids

- (Gaussian) Pyramid operator in OpenCV

```
lower_reso = cv2.pyrDown(higher_reso)
```

  - `higher_reso`: [M x N] input image
  - `lower_reso`: [M/2 x N/2] output image

Multi-resolution Image Pyramids

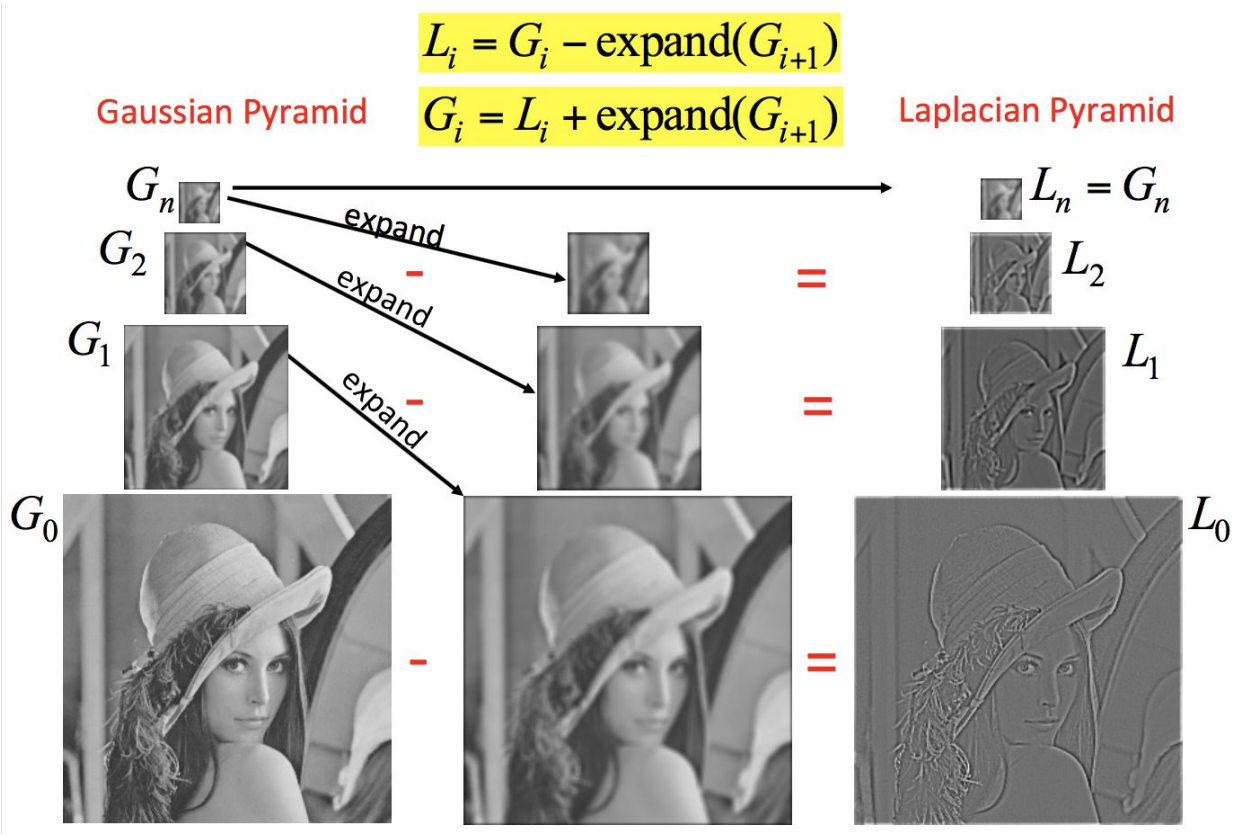
- (Gaussian) Pyramid operator in OpenCV

```
higher_reso2 = cv2.pyrDown(lower_reso)
```

  - `lower_reso`: [M x N] input image
  - `higher_reso2`: [2M x 2N] output image

Laplacian Image Pyramids

```
high_res_img2 = cv2.pyrUp(low_res_img)
laplacian = cv2.subtract(high_res_img, high_res_img2)
```



(Image credit: Gunhee Kim's CV lecture slide)

Image Blending using Pyramids

- One application of pyramids is image blending
- Seamless blending without leaving much data in the images

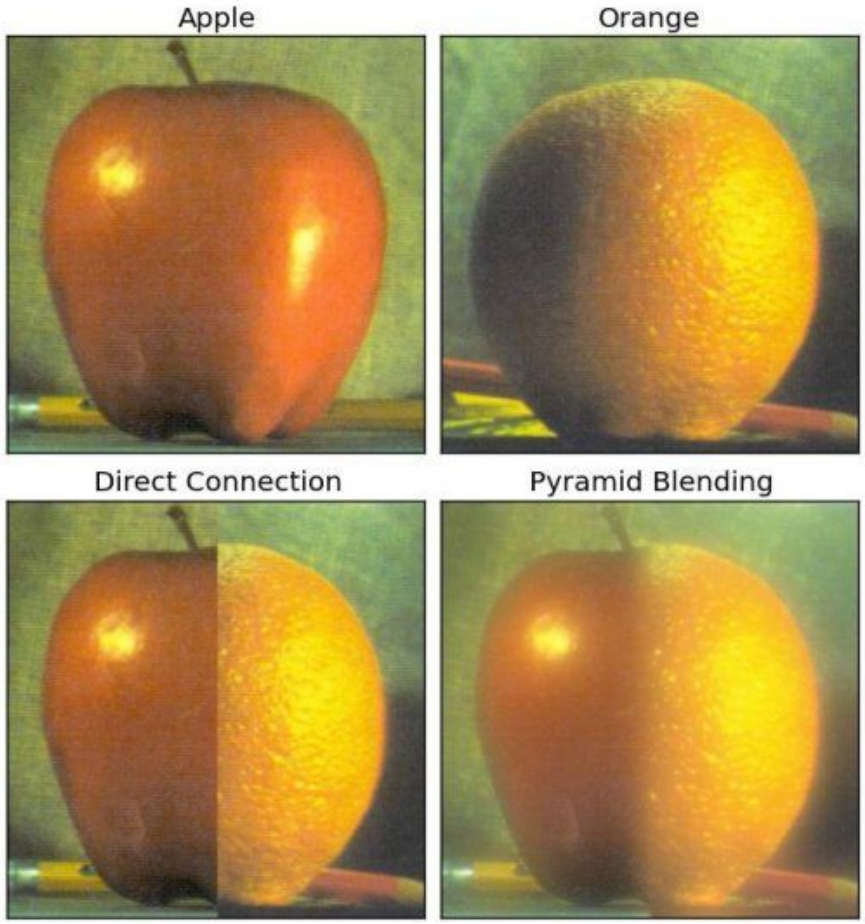


Image Blending using Pyramids

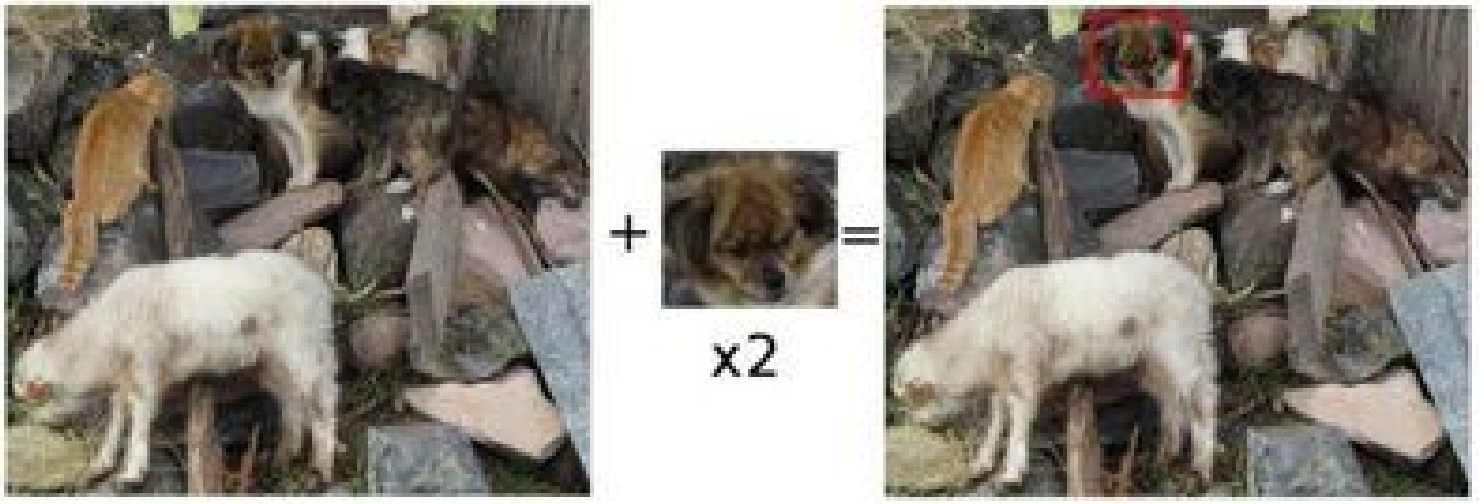
1. Load the two images of apple and orange
2. Find the Gaussian Pyramids for apple and orange
3. From Gaussian Pyramids, find their Laplacian Pyramids
4. Now join the left half of apple and right half of orange in each levels of Laplacian Pyramids
5. Finally from this joint image pyramids, reconstruct the original image

Let’s Check the Code

3_image_blending.ipynb

Template Matching

- Technique for finding areas of an image that match (are similar) to a template image
- Two primary components
  - **Source image (I)**: image to match the template image
  - **Template image (T)**: patch image to compare
- **Goal**: detect the highest matching area



Source image Template image

Template Matching

- Template matching operator in OpenCV

```
matching_result = cv2.matchTemplate(img, template, match_method)
```

  - `img`: image to match the template image
  - `template`: patch image to compare
  - `match_method`: comparison method
    - cv2.TM_CCOEFF
    - cv2.TM_CCOEFF_NORMED
    - cv2.TM_CCORR
    - cv2.TM_CCORR_NORMED
    - cv2.TM_SQDIFF
    - cv2.TM_SQDIFF_NORMED
  - `matching_result`: grayscale image; each pixel denotes how much does the template match with image

Template Matching

- Template matching operator in OpenCV

```
min_val, max_val, min_loc, max_loc =
cv2.minMaxLoc(matching_result)
```

  - `matching_result`: input single-channel array
  - `min_val`: minimum value
  - `max_val`: maximum value
  - `min_loc`: minimum location
  - `max_loc`: maximum location

Fast Template Matching

Template

Search Region

Original Image

(Image credit: Gunhee Kim's CV lecture slide)

Let’s Check the Code

4_template_matching.ipynb

Homework

See **5_homework.ipynb**

Reference

- OpenCV-Python Tutorials
  http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

Q&A

Any Question?