Introduction to OpenCV

30263 – Computer Vision
Graduate in Computer Sciences
Escuela de Ingeniería y Arquitectura

Universidad Zaragoza
1542

Agenda

- What is OpenCV?
- Installation and configuration
- Main classes

- Suggested reading
  - http://opencv.org/
  - https://docs.opencv.org/4.x/d9/df8/tutorial_root.html
  - https://github.com/a-anjos/python-opencv/blob/master/cv2cheatsheet.pdf

Learning OpenCV 4 Computer Vision with Python 3: Get to grips with tools, techniques, and algorithms for computer vision and machine learning, J. Howse and J. Minichino, Pckt, 2020

OpenCV

- Open source library for computer vision http://opencv.org
  - Written in C++
  - Interfaces: C++, C, Java, Python, Ruby, Matlab,…
  - O.S.: Windows, Linux, Mac OS X, iOS, Android
  - Optimizations: SSE, MMX, AVX, NEON, OpenMP, TBB
  - Accelerated versions for some architectures
    - IPP: for Intel
    - CUDA for Nvidia GPUs
    - OpenCL for GPUs
- BSD license: it allows building commercial products using OpenCV

OpenCV modules

- Core: fundamental objects and operations
- Imgproc: image processing
- HighGUI: light library for managing GUI windows
- Video: read and write video sequences
- Calib3D: camera calibration (monocular and stereo)
- Features2D: feature detection and correspondences computation
- Objdetect: face recognition, pedestrian detection, etc
- ML, DNN: machine learning, Deep Neural Networks
- FLANN: fast library for approximate nearest neighbors
- Photo: for computational photography
- Stitching: building panoramas
- Cudaxxx: accelerated versions on Nvidia CUDA
- Xfeatures2D: features SIFT and SURF (under patent)
  - https://github.com/opencv/opencv_contrib

OpenCV installation

- We recommend the use of virtualenvs and pip
- For coding, your favorite text editor or jupyter notebooks are fine

- Installation should be easy
  https://pythonprogramming.net/loading-images-python-opencv-tutorial/

- Windows
  - Virtualenvs
    - https://www.youtube.com/watch?v=sG7Q-r_SZhA
  - pip install numpy
  - pip install matplotlib
  - Install Python OpenCV

- Linux and Mac OS
  - Virtualenvs
    https://www.lunium.com/blog/entornos-virtuales-en-python-3-con-virtualenv-en-linux/
  - pip install numpy
  - pip install matplotlib
  - pip install opencv-python

OpenCV in Google colab

- Google Colab

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

- Colab notebooks are very similar to jupyter notebooks

- Your codes should be stored in the drive associated with your gmail account

- Get started playing with

  https://colab.research.google.com/github/xn2333/OpenCV/blob/master/Image_Processing_in_Python_Final.ipynb

  in your Colab account

Example 1: read and show an image

```
import cv2 # Import python-supported OpenCV functions
import numpy as np # Import numpy and call it np
from matplotlib import pyplot as plt # Import pyplot and call
it plt

img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
cv2.namedWindow( 'Example1', cv2.WINDOW_AUTOSIZE )
cv2.imshow('Example1',img)
print(type(img))
print(img.shape)
cv2.waitKey(0)
cv2.destroyWindow( 'Example1' ) # cv2.destroyAllWindows()
```

- `imread` automatically finds out the file format (e.g. jpg)
- images are represented as numpy arrays
- Windows are associated with a name (error if a different window is closed)
- https://pythonprogramming.net/loading-images-python-opencv-tutorial/
- https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html

Matrices in OpenCV

https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html

Matrices to represent images
Several rows and columns
1 channel (grayscale) or 3 channels (color BGR, HSV, etc.)
With different numpy data types (uint8, int16, float32, float64 …)

https://docs.scipy.org/doc/numpy-1.13.0/user/basics.types.html

```
uint8  Unsigned integer (0 to 255)
uint16 Unsigned integer (0 to 65535)
int16  Integer (-32768 to 32767)
int32  Integer (-2147483648 to 2147483647)
float32       Single precision float: sign bit, 8 bits
exponent, 23 bits mantissa
float64       Double precision float: sign bit, 11 bits
exponent, 52 bits mantissa
```

Matrices in OpenCV: properties

Try the following code:

```
import cv2 # Import python-supported OpenCV functions
import numpy as np # Import numpy and call it np
from matplotlib import pyplot as plt # Import pyplot and call it plt

def show_image_properties(my_img):
    cv2.namedWindow( 'Example1', cv2.WINDOW_AUTOSIZE )
    cv2.imshow('Example1',my_img)
    print("Properties of the matrix used to store the image")
    print("They are numpy arrays: type(my_img)= ", type(my_img))
    print("Rows, columns and channels: my_img.shape= ", my_img.shape)
    print("Total number of pixels: my_img.size= ", my_img.size)
    print("Image datatype: my_img.dtype = ", my_img.dtype )
    cv2.waitKey(1000) #cv2.waitKey(0)
    cv2.destroyWindow( 'Example1' ) # cv2.destroyAllWindows()
    return(0)

img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
img_bgr = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
show_image_properties(img)
show_image_properties(img_bgr)
```

Matrices in OpenCV

class numpy.ndarray

1D vector, 2D matrix, or multi-dimensional matrices.

Represents images, histograms, descriptors,…

It can contain any primitive data type.

Common initializations: np.zeros, np.ones, np.eye

Examples:

```
channels = 1
cvImgGray = np.zeros((rows, columns, channels), dtype = "uint8")
channels = 3
cvImgRGB = np.zeros((rows, columns, channels), dtype = np.uint8)
cvMatrix = np.zeros( (10, 10, 1), dtype = np.float32 )
```

https://numpy.org/doc/stable/user/basics.types.html

```
cvImgGray[i,j] = [3, 3, 3] # Value error
cvImgGray[i,j] = 3

cvImgRGB[i,j] = [3, 3, 3]
cvImgRGB[i,j] = 3          # Be careful!!
```

Matrices in OpenCV: elements

Try the following code:

```
(…)
# Create a black image (BGR: uint8, 3 channels)
img_uint8 = np.zeros((512,512,3), np.uint8)
# Draw a diagonal BLUE line with thickness of 5 px
cv2.line(img_uint8,(0,0),(511,511),(255,0,0),5)
show_image_properties(img_uint8)

# Create a black image (BGR: uint8, 3 channels)
img_uint8b = np.zeros((512,512,3), np.uint8)
# Access to elements and set them the value BLUE
img_uint8b[0:100,0:25]=[250,0,0]
# Alternatively..
#img_uint8b[0:100,0:25,0]=250
#img_uint8b[0:100,0:25]=np.array([250,0,0], np.uint8)
show_image_properties(img_uint8b)

# Try and observe the following mistakes (common mistakes; note that often
there is not even a warning..)
img_uint8d = np.zeros((512,512,3), np.uint8)
#img_uint8d[0:100,0:25,0]= -100 #in some versions, no warning!!
#img_uint8d[0:100,0:25,0]= -500 #in some versions, no warning!!
show_image_properties(img_uint8d)
```

Operations with matrices

```
m2 = m1  # new reference to m1, data is not copied
m3 = m1.copy()        # complete copy of m1
m3 += 5.0
m3 = 0.5 * m1 + 0.3 * m2

A = np.eye(3, dtype = np.float32)
Y = A * X
Y = np.dot(A, X) # observe the difference
Y = A.T * X          # transpose
X = np.linalg.inv(A) * Y;  # inverse
cv2.solve(A, Y ,X); # solves the system A*X = Y

v3 = np.cross(v1, v2)  # vector product
z  = np.dot(v1, v2)        # scalar product

m3 = m1 > m2         # returns a matrix of numpy.bool_

dst = cv2.add(img1, img2)  # sums all channels
masked_img = cv2.bitwise_and(img,img,mask = mask) # mask
…
```

How to traverse a matrix

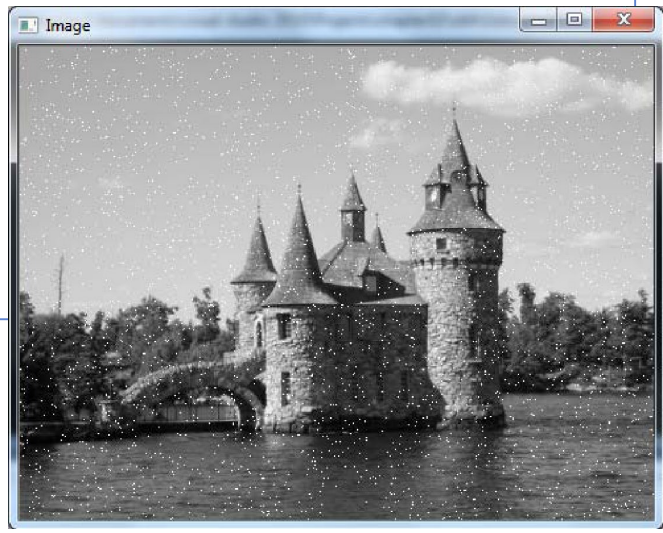
- With index

Attention! Assuming images of unsigned bytes, 1 or 3 channels

```
import random as r
def salt(image, n):
  # Adds salt noise to n pixels randomly selected in the image
  for k in range(n):
    i= r.randrange(image.shape[0]) # range of image rows
    j= r.randrange(image.shape[1]) # range of image cols

    if len(image.shape) == 2:             # Gray image
      image[i,j]= 255
    else:                                                  #
RGB image
      image[i,j,0]= 255
      image[i,j,1]= 255
      image[i,j,2]= 255

   return image
```



How to traverse a matrix

- Creating / not creating copies:

```
import cv2 # Import python-supported OpenCV functions
import numpy as np # Import numpy and call it np
from matplotlib import pyplot as plt # Import pyplot and call it plt
import random as r

def salt(image_in, n):
  # Adds salt noise to n pixels randomly selected in the image
    image=image_in.copy()
    for k in range(n):
        i= r.randrange(image.shape[0]) # range of image rows
        j= r.randrange(image.shape[1]) # range of image cols
        if len(image.shape) == 2:                # Gray image
            image[i,j]= 255
        else: # RGB image
            image[i,j,0]= 255
            image[i,j,1]= 255
            image[i,j,2]= 255
    return image

img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
#img = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
img2=salt(img, 10000)

cv2.imshow('Original', img)
cv2.imshow('Salt', img2)
cv2.waitKey(0)
cv2.destroyAllWindows
```

Attention! Assuming images of unsigned bytes, 1 or 3 channels

Are there any differences in the behavior?

How to traverse a matrix

- Rows and columns

Attention! It assumes images of bytes, 1 channel

```
import cv2 # Import python-supported OpenCV functions
import numpy as np # Import numpy and call it np

def colorReduce(img_in, div=np.uint8(32)):
    rows,cols = img_in.shape # grayscale images
    print(rows,cols)
    img2 = img_in.copy()
    for i in range(rows):
        for j in range(cols):
            img2[i,j] = np.uint8(np.uint8(img_in[i,j] /div)*div)
    return(img2)

img_bgr = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
img2 = colorReduce(img_bgr)
cv2.imshow('A', img_bgr)
cv2.imshow('Reduced',img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

How to traverse a matrix

- Iterating Arrays Using ndenumerate()
  - Images are stored by rows, then columns, …

```
img = np.array([[1, 2, 3], [4, 5, 6]])

print("Initial matrix:")
print(img)

img2 = img.copy()
for idx, x in np.ndenumerate(img):
  print("idx=", idx)
  print("x=", x)
```

Initial matrix:
[[1 2 3]
 [4 5 6]]

idx= (0, 0)
x= 1
idx= (0, 1)
x= 2
idx= (0, 2)
x= 3
idx= (1, 0)
x= 4
idx= (1, 1)
x= 5
idx= (1, 2)
x= 6

How to traverse a matrix

- Iterating Arrays Using ndenumerate()

Better suited for 1 or 3 channels

```
import cv2 # Import python-supported OpenCV functions
import numpy as np # Import numpy and call it np

def colorReduce(img_in, div=np.uint8(64)):
    img2 = img_in.copy()
    for idx, x in np.ndenumerate(img_in): # one or three channels
            img2[idx] = np.uint8(np.uint8(  x /div)*div)
    return(img2)

#img_bgr = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
img_bgr = cv2.imread('lena.jpg', cv2.IMREAD_COLOR)
img2 = colorReduce(img_bgr)
cv2.imshow('A', img_bgr)
cv2.imshow('Reduced',img2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Efficiency traversing a matrix

- We advise reducing the number of image traverses: memory access is usually a bottleneck
- We advise writing legible code: optimize only the critical sections

https://docs.opencv.org/4.x/dc/d71/tutorial_py_optimization.html

```
num_runs = 10
e1 = cv2.getTickCount()
for k in range(num_runs):
    img2 = colorReduce1(img_bgr)
e2 = cv2.getTickCount()
for k in range(num_runs):
    img3 = colorReduce2(img_bgr)
e3 = cv2.getTickCount()
t1 = (e2 - e1)/cv2.getTickFrequency()
t2 = (e3 - e2)/cv2.getTickFrequency()
print( num_runs, " executions of color reduce. Time t1=", t1,
"seconds and time t2=", t2, "seconds." )
```