COMPUTER VISION 22/23

LAB 1

TABLE OF CONTENTS

Course outline 2

Teaching Assistant Contacts 2

Installation 2

Lab Objective 2

Lab Content 2

i. Introduction: 2

ii. Loading images in opencv 3

- Read Image 3
- Display Image 3
- Write Image 4
- Resize Image 4

iii. Matplotlib 5

iv. Numpy 6

- Import the NumPy library 7
- Create array in specific range: 7
- Reshaping array: 7
- The slicing in Numpy array: 7
- The negative index: 7
- filtering 8
- The ellipsis [...] in Numpy array 8
- Copying in Numpy: 9

v. Draw shapes 9

vi. Broadcasting 10

vii. Image processing using OpenCV 10

viii. References 13

COURSE OUTLINE

- Classical feature extraction methods
  - Harris, SIFT, and HOG
- Outliers removal (RANSAC)
- Object Recognition
- Object Detection
- Image Segmentation (Semantic segmentation – Instance segmentation)
- Object Tracking
- Activity Recognition (Videos)
- Image Generation & Style transfer (GANs)

TEACHING ASSISTANT CONTACTS

- TA. Yomna Ahmed → yomna.ahmed@cis.asu.edu.eg
- TA. Mohamed Magdi → muhammedmagdi411@gmail.com

INSTALLATION

- Create conda environment (if you don't have one), and install Matplotlib and OpenCVpackages:
  - conda create -n vision
  - conda activate vision
  - conda install matplotlib
  - install pip : conda install -c anaconda pip
  - pip install opencv-contrib-python

LAB OBJECTIVE

- Review OpenCV basics.
- Review Numpy basics.
- Image Processing using OpenCV

LAB CONTENT

I. INTRODUCTION:

Computer Vision is about analyzing the scene.

It deals with how computers can be made for gaining high-level understanding from Images/Videos.

It is concerned with the theory behind artificial systems that extract information from images.

This field includes scene reconstruction, action or event detection, object tracking in videos, object recognition, 3D poses estimation, motion estimation, and image restoration.

II. LOADING IMAGES IN OPENCV

➢ READ IMAGE

Use the function **cv2.imread()** to read an image. The image should be in the working directory or a full path of image should be given as First argument. Second argument is a flag which specifies the way image should be read.

1. cv2.IMREAD_COLOR (1): Loads a color image. Any transparency of image will be neglected. It is the default flag.
2. cv2.IMREAD_GRAYSCALE (0): Loads image in grayscale mode
3. cv2.IMREAD_UNCHANGED (-1): Loads image as such including alpha channel

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread('messi5.jpg',0)
```

```
'''load the input image and show its dimensions, keeping in mind that
   images are represented as a multi-dimensional NumPy array with
   shape no. rows (height) x no. columns (width) x no. channels (depth)'''
image = cv2.imread("messi5.jpg")
(h, w, d) = image.shape
print("width={}, height={}, depth={}".format(w, h, d))
```

➢ DISPLAY IMAGE

Use the function **cv2.imshow()** to display an image in a window. First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

```
img = cv2.imread('messi5.jpg')
cv2.imshow('image',img)
cv2.waitKey(0)
```

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time,

the program continues. If 0 is passed, it waits indefinitely for a key stroke. It can also be set to detect specific keystrokes like, if key a is pressed.

If waitkey() is not called, the image will open and close too fast to be able to see it.

➢ WRITE IMAGE

Use the function **cv2.imwrite()** to save an image. First argument is the file name, second argument is the image you want to save.

```
import numpy as np
import cv2

img = cv2.imread('messi5.jpg',0)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

➢ RESIZE IMAGE

Resizing images is important for several reasons. First, you might want to resize a large image to fit on your screen. Image processing is also faster on smaller images because there are fewer pixels to process. In the case of deep learning, we often resize images, ignoring aspect ratio, so that the volume fits into a network which requires that an image be square and of a certain dimension.

```
# resize the image to 300x300px, ignoring aspect ratio
resized = cv2.resize(image, (300, 300))
cv2.imshow("Fixed Resizing", resized)
cv2.waitKey(0)
```

Sometimes resizing an image to a fixed width and height without considering the aspect ratio of the original image can lead to distorting the image. This can be avoided by resizing using the aspect ratio.

```
# fixed resizing can distort aspect ratio so let's resize the width
# to be 300px but compute the new height based on the aspect ratio
r = 300.0 / w
dim = (300, int(h * r))
resized = cv2.resize(image, dim)
cv2.imshow("Aspect Ratio Resize", resized)
cv2.waitKey(0)
```

Instead of calculating the aspect ratio manually in the previous example, we can use the resize function from the 'imutils' library:

```
resized = imutils.resize(image, width=300)
```

III. MATPLOTLIB

Matplotlib is a plotting library for Python which gives you wide variety of plotting methods.

Color image loaded by OpenCV is in BGR mode. But Matplotlib displays in RGB mode. Therefore, colored images will not be displayed correctly in Matplotlib if they are loaded with OpenCV without changing its default color mode.

To plot an image loaded using opencv in Matplotlib, we will need to convert it into RGB first then plot it using matplotlib as follow:

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# get pixel color values at row 52 and column 30
R,G,B = image[52,30]
print("R={}, G={}, B={} at pixel (52,30)".format(R, G, B))

# get R value only at pixel(52,30)
R = image[52,30,0]
# get G value only at pixel(52,30)
G = image[52,30,1]
# get B value only at pixel(52,30)
B = image[52,30,2]
```

To use plot using Matplotlib, we will use the sub-library PyPlot:

Note that `plt.show()` displays the figure (and enters the mainloop of whatever gui backend you're using). You shouldn't call it until you've plotted things and want to see them displayed.

`plt.imshow()` draws an image on the current figure (creating a figure is there isn't a current figure). Calling `plt.show()` before you've drawn anything doesn't make any sense. If you want to explicitly create a new figure, use `plt.figure()`.

```
from matplotlib import pyplot as plt
plt.imshow(img)
plt.show()
```

IV. NUMPY

NumPy deals with multidimensional arrays like Images. image is 3-dimensional array of numbers (height, width, channels).

In NumPy dimensions are called AXES. The number of axes is RANK.

In example below, the array has rank 2 (it is 2-dimensional). The first dimension (axis) has a length of 2 (rows), the second dimension has a length of 3 (columns).

```
[[ 1., 0., 0.],
[ 0., 1., 2.]]
```

OpenCV uses NumPy to represent image, this can be verified as follow:

```
print type(img)
```

Therefore, mastering the NumPy library will make your coding much fasters and easier.

➢ IMPORT THE NUMPY LIBRARY

Usually we use np as a shorthand for numpy

```
import numpy as np
```

➢ CREATE ARRAY IN SPECIFIC RANGE:

In python, you can use range(a, b) to create a list of numbers. Using NumPy, you can use np.arange(a, b) to create a NumPy array of numbers

```
print (type(range(1, 10)))
print (type(np.arange(1, 10)))
```

➢ RESHAPING ARRAY:

You can change the dimension of a numpy keeping the same size using the reshape method, you can also print test_array.shape to check/validate the shape of the array.

```
test_arr = np.array([1, 2, 3, 4, 5, 6])
print (test_arr.shape, test_arr)
test_arr = test_arr.reshape(2, 3)
print (test_arr.shape, test_arr)
```

➢ THE SLICING IN NUMPY ARRAY:

The colon is used to get a specific range of elements from the array.
You indicate the range of elements to return (start_index : end_index). Note that the returned elements are starting from the exact start index you provide but ends before the end_index that you provide.

```
test_arr = np.array([1, 2, 3, 4, 5])
print (test_arr[1 : 3]) # prints [2, 3]
```

You can also specify an increment step:

```
print (test_arr[0 : 5 : 2]) # prints [1, 3, 5]
```

You can leave it empty, meaning to take all the range:

```
print (test_arr[:]) # prints [1, 2, 3, 4, 5]
```

➢ THE NEGATIVE INDEX:

Negative index means reverse the counting to start from the end of the array:

```
print (test_arr[-1]) # prints 5
```

Reversing an array is therefore this easy:

```
print (test_arr[::-1]) # prints [5, 4, 3, 2, 1]
```

This means to take the whole range (the start and the end) with a step size of 1 but as there is a negative sign, that means to do that in reverse so start from the last till the start with a step size of 1

➢ FILTERING

You can filter an array by giving it an array of specific integers this will get these specific indices from the array

```
print (test_arr[[1, 2, 3]]) # prints [2, 3, 4]
```

You can also pass an array of Boolean to get specific elements from the array

```
print (test_arr[[True, True, True, False, False, False]])
# prints [1, 2, 3]
```

It is possible then to do something like this:

```
print (test_arr[test_arr >= 4]) # prints [4, 5, 6]
```

The operation `test_arr >= 4` is applied element wise, returning an array of boolean, which is then used to filter the test array

➢ THE ELLIPSIS […] IN NUMPY ARRAY

Ellipsis means all the dimension of the array.

```
a = np.arange(1, 17).reshape(4,2,2)
print (a[:, :, 1])
print (a[..., 1])
```

Only 1 ellipsis can be used in array index, however, you can also specify a certain index in the beginning or the end of the array:

```
a = np.arange(1, 17).reshape(4,2,2)
print (a[:, :, 1])
print (a[..., 1])
print (a[..., :])
print (a[1:3, ...])
```

➢ COPYING IN NUMPY:

When you are re-assigning an array, you usually change the reference variable to another object, however, using the colon operator or the ellipsis, you can change the value that the reference is point to

```
a = np.arange(1, 17).reshape(4,4)
print ('initial a=', a)
b = a
b = np.arange(11, 27).reshape(4,4)

print ('a=', a)
print ('b=', b)
```

You will notice that a and b are different objects, however, the following code shows that they are referencing (pointing to) the same object.

```
a = np.arange(1, 17).reshape(4,4)
print 'initial a=', a
b = a
b[:] = np.arange(11, 27).reshape(4,4)
# Also possible:
# b[...] =  np.arange(11, 27).reshape(4,4)
print ('a=', a)
print ('b=', b)
```

V. DRAW SHAPES

Draw different geometric shapes with OpenCV: cv2.line(), cv2.circle(), cv2.rectangle(), cv2.ellipse(), cv2.putText() etc.

1. Line: Pass starting and ending coordinates of line, color, line thickness.

```
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)
```

2. Rectangle: Pass top-left corner and bottom-right corner of rectangle, color, line thickness.

```
img = cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
```

3. Circle: Pass center coordinates and radius, color.

```
img = cv2.circle(img,(447,63), 63, (0,0,255), -1)
```

4. Add Text: pass bottom-left corner where data starts, Font type, Font Scale(size of font), color, thickness, lineType

```
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,50),font,1,(255,255,255),2,cv2.LINE_AA)
```

VI. BROADCASTING

A set of arrays is said to be broadcastable if the above rules produce a valid result and one of the following is true

- Arrays have exactly the same shape.
- Arrays have the same number of dimensions and the length of each dimension is either a common length or 1
- Array having too few dimensions can have its shape prepended with a dimension of length 1, so that the above stated property is true.

```
a = np.array([[0.0, 0.0, 0.0], [10.0, 10.0, 10.0], [20.0, 20.0, 20.0],
[30.0, 30.0, 30.0]])
b = np.array([1.0, 2.0, 3.0])

print('First array:')
print(a)
print('\n')

print('Second array:')
print(b)
print('\n')

print('First + Second array:')
print(a+b)
```

VII. IMAGE PROCESSING USING OPENCV

OpenCV has a wide range of image processing functions and filters built-in.

An example on image thresholding using OpenCV:

```
image = cv2.imread("messi5.jpg",0)
# threshold parameters: image,threshold,maxValue,threshold type
# all values above 127 will be set to 255
# returns two outputs: the first is the threhsold value used and
# the second is the thresholded image
ret, thresholdedImage = cv2.threshold(image,127,255,cv2.THRESH_BINARY)
print(ret)
cv2.imshow("Thresholded Image", thresholdedImage)
cv2.waitKey(0)
```

Some common filters used for image denoising and smoothing:

Median Filter: commonly used for removing salt & pepper noise

Image with salt & pepper noise

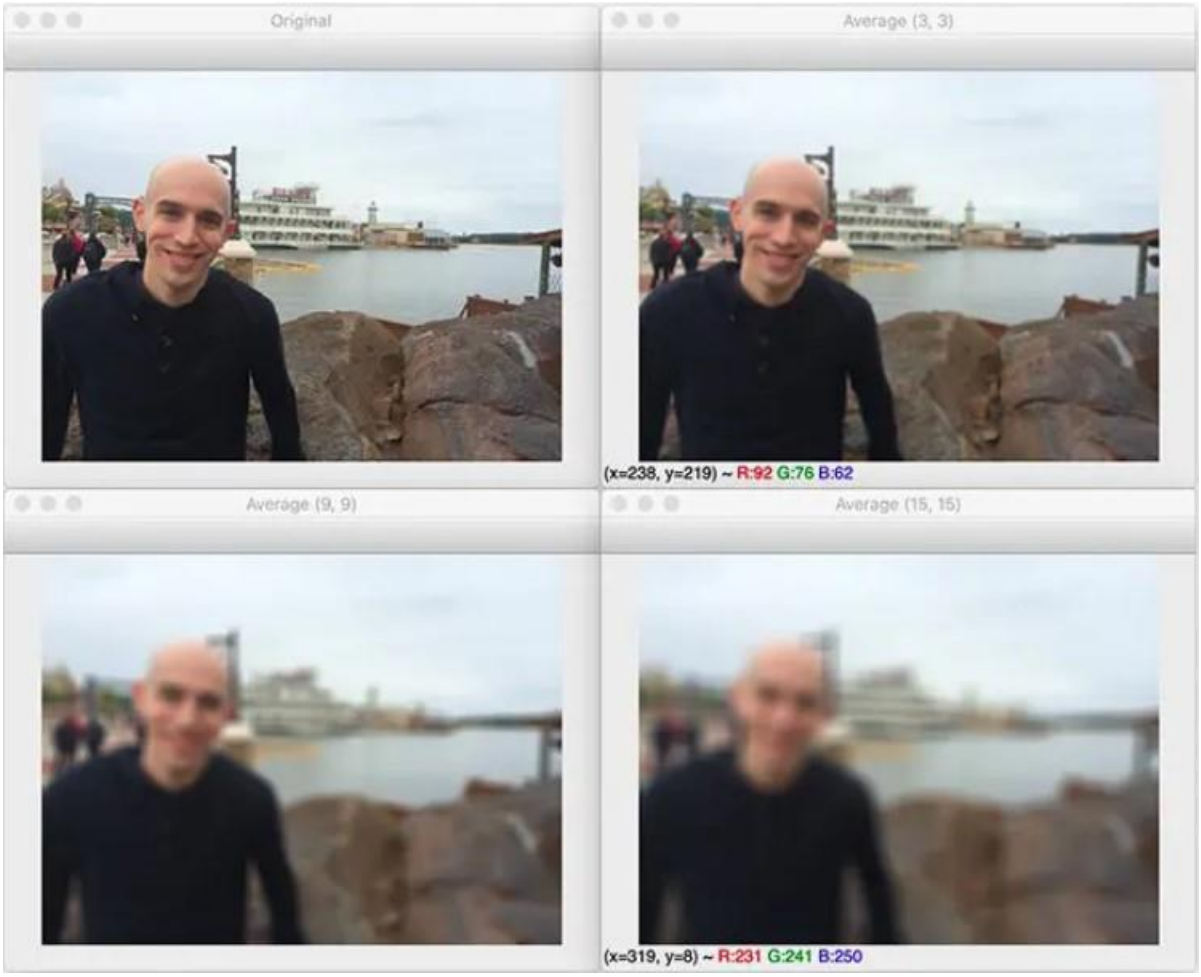
Median filter with window 3×3

Median filter with window 5×5

Median filter with window 7×7

```
image = cv2.imread("messi5.jpg",1)
# median blur parameters: image, kernel size
img = cv2.medianBlur(image,5)
cv2.imshow("Blurred Image", img)
cv2.waitKey(0)
```

Gaussian Filter: commonly used smoothing images and reducing noise




```
image = cv2.imread("person.jpg")
cv2.imshow("Original", image)
kernelSizes = [(3, 3), (9, 9), (15, 15)]
# close all windows to cleanup the screen
cv2.destroyAllWindows()
cv2.imshow("Original", image)
# loop over the kernel sizes again
for (kX, kY) in kernelSizes:
    # apply a "Gaussian" blur to the image
    blurred = cv2.GaussianBlur(image, (kX, kY), 0)
    cv2.imshow("Gaussian ({}, {})".format(kX, kY), blurred)
    cv2.waitKey(0)
```

There are many other functions available as well such as: cv2.boxfilter , cv2.erode(), cv2.Laplacian() and so on.

VIII. REFERENCES

[1] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html