OpenCV with Python

OpenCV: Introduction

- the Open Source Computer Vision Library includes state of the art computer vision and machine learning algorithms (including running deep networks) and apps
- Originally developed by Intel's research center in Nizhny Novgorod (Russia), it was later supported by Willow Garage and is now maintained by Itseez.
- professionally coded and optimized. It can be used in C++, Python, Cuda, OpenCL and Matlab.
- It runs on: Android, iOS, Windows, Linux and MacOS and many embedded implementations.

Important Links

- The user site http://opencv.org/
- The developer site http://code.opencv.org/projects/opencv/wiki/WikiStart
- Code
- OpenCV (the core data structures, optimized algorithms, sample and tutorial code): https://github.com/Itseez/opencv
- opencv_contrib (new algorithms, applications and GSoC contributions and related tutorial and sample code): https://github.com/Itseez/opencv_contrib.git
- opencv_extra (extra data and code samples): https://github.com/Itseez/opencv_extra
- downloads for various OS and mobile devices: http://opencv.org/downloads.htm
- GSoC 2016 projects

https://summerofcode.withgoogle.com/archive/2016/organizations/6474535423442944/#projects

Loading

- Print vs print()
- To maintain consistency with python 3, we import it from __future__
- argparse – Module quite useful during parsing of parameters
- Cv2.imread() -> reads image as numpy array, hence we can get array dimension
- Sample qn:-

1. What does import cv2 do?

- Imports our OpenCV Python bindings

2. Given the following NumPy array shape, how would we interpret the width, height, and number of channels in the image: (400, 600, 3)?

Basics of Image

- Pixel :- These are raw building blocks of image

  For example, let's pretend we have an image with a resolution of 500×300. This means that our image is represented as a grid of pixels, with 500 rows and 300 columns. Overall, there are 500×300 = 150,000 pixels in our image

  Pixels are represented in two ways: grayscale and color

  Color pixels are represented in RGB color space. one value for the Red component, one for Green, and one for Blue

  Other color space like YCbCr, HSV also exist

Since image is represented as numpy 2 d array, to access pixel we have to give (x, y) coordinate

Coordinate System:-

Drawing Image

- Given that OpenCV interprets an image as a NumPy array, we can also define our images manually using NumPy arrays
- Ex:- canvas = np.zeros((300, 300, 3), dtype = "uint8")
- Here, canvas is drawn image buffer

Image Processing

- Image Transformations

1. Translation :-

Using translation, we can shift an image up, down, left, or right, along with any combination of the above

Ex:-

```
M = np.float32([[1, 0, 20], [0, 1, 30]])
shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape [0]))
```

Explanation :- M is translation Matrix which decide how we wish to translate image. It is always floating point matrix.

[1, 0, tx] is first row of matrix, which ensures we shift left (if tx < 0) or right (if tx > 0)

[0,1,ty] which ensures we shift up (if ty < 0) or down (if ty > 0)

Function warpAffine() does actual translation.

Image Processing

- Rotation:-

```
(h, w) = image.shape[:2]
center = (w // 2, h // 2) [Note // integer division]
M = cv2.getRotationMatrix2D(center, 45, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
```

Note:- In most cases, you will want to rotate around the center of an image; however, OpenCV allows you to specify any arbitrary point you want to rotate around

Image Processing

- Image Resizing

```
r = 150.0 / image.shape[1]
dim = (150, int(image.shape[0] * r))
 resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
```

Q)

Here we resized width (from image.shape[1] to 150

How could we do same with height?

Image Processing

- Flipping

```
flipped = cv2.flip(image, 1)
cv2.imshow("Flipped Horizontally", flipped)
```

In flip function, second parameter 1 =flip horizontally

0 =flip vertically

-1 = flip horizontally as well as vertically

Image Processing

- Cropping

```
cropped = image[30:120 , 240:335]  // assume name of input image is “image”
cv2.imshow(“Cropped image", cropped)
```

Note:- order in which we specify the coordinates is: [startY:endY, startX:endX]

Image Processing

- Image Arithmetic

What happens when we add 10 into 250?

What happens if we are examining pixel with intensity 250 and add into it 10??

Results can be surprising

Image Processing

- OpenCV and numpy are not same and so are results if you do arithmetic operations using them

- Numpy will perform modulo arithmetic and “wrap around”
- OpenCV will perform clipping and ensure pixel value never fall outside [0,255]

  250 + 10 ==➔ 4

  250 + 10 ==➔ 255

Ex :- print("max of 255: {}".format(cv2.add(np.uint8([200]), np.uint8 ([100]))))

print("wrap around: {}".format(np.uint8([200]) + np.uint8([100])))

Image Processing

- Bitwise

  ---- program

- Masking

  Masking allows us to focus only on the portion of image that interests us