Image processing with Python

Part 2 - Advanced Image Processing

OpenCV is an open-source library of computer vision for real-time 2D/3D image processing. **OpenCV's** Image Processing module for Python provides many powerful functions for pixel-wised processing.

By finishing this section, you will be able to use the useful functions of **OpenCV** package to edit photos. The functions include blur, edge detection, and color mapping by a lookup table.

Installing OpenCV Package

Basically, **OpenCV** package for Anaconda is available in the lab PCs. But, if you want to set up the environment for your home PC, you may follow the procedure mentioned in Appendix 1 to install the **OpenCV** package.

Loading Image Pixel Data

Before we use OpenCV modules, we have to load the image's pixel data, RGB values) into an unsigned 8-bit integer array. The following is the code to load pixel data into an array:

```
import cv2
data = cv2.imread('images/oversharpened_car.jpg')
```

The *data* variable stores a *ndarray* that represents the pixel data of the image. The shape of the array is (height, width, 3), as you can find out by typing **data.shape** in IPython console. The following is the example layout of the array of the RGB system. You can see that the **cv2.imread()** loads the data in reversed order – BGR.

	column 1			column 2			…			column n		
row 1	0	0	0	0	255	128	…	…	…	25	43	110
row 2	0	0	0	0	220	101	…	…	…	100	125	250
…	…	…	…	…	…	…	…	…	…	…	…	…
row m	112	140	231	200	200	123	…	…	…	0	0	0

If you want to load an image as a grayscale image, you can use **cv2.imread()** function with **cv2.IMREAD_GRAYSCALE** as follows:

```
data = cv2.imread('images/oversharpened_car.jpg', cv2.IMREAD_GRAYSCALE)
```

The following is the example layout of the array of the grayscale system:

	column 1	column 2	…	column n
row 1	0	0	…	25
row 2	0	0	…	100
…	…	…	…	…
row m	112	200	…	0

Smoothing Image

Applying **blur effect** is able to smooth an image. Blurring can be achieved by applying different filters – mean filter, weighted filter, and Gaussian filter. The filter is usually a small matrix (a.k.a. kernel). We can perform a convolution operation between the filter matrix and the image matrix to get the result image. **OpenCV** package provides some functions that allow us to makes the blur effect directly.

Mean-blur (a.k.a box blur)

In mean-blur method, each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image. To apply the mean-blur effect, the **cv2.blur()** function can be used. The syntax is as follows:

```
result = cv2.blur(src, kernel_size)
```

The **cv2.blur()** function accepts two parameters – the image pixel data and the kernel size (2-tuple). The following is the example of applying the mean-blur effect to an image using 3 x 3 mean kernel:

```
mb = cv2.blur(data, (3, 3))
```

The 3 x 3 mean kernel is:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Type the following example and compare the original image with the resulted image.

```
#%%
import cv2
from PIL import Image

data = cv2.imread('images/oversharpened_car.jpg')
mb = cv2.blur(data, (3,3))

b = cv2.cvtColor(mb, cv2.COLOR_BGR2RGB)
```

```
img = Image.fromarray(b)
img.show()
```

The last three lines are used to convert the pixel data back to an image object. As the **cv2.imread()** function loads pixel data in BGR format (blue, green, red). To create an image object using the data, we have to use the **cv2.cvtColor()** function with **cv2.COLOR_BGR2RGB** option in order to change the data back to RGB format (red, green, blue). Next, the **Image.fromarray()** function is used to create an image object from the ndarray data. Finally, the **show()** function shows the image in the default image viewer.

You can also use **Image.fromarray(mb).show()** to see the image without calling cvtColor().

Median Blur

The **cv2.medianBlur()** function takes the median value of all pixels under the kernel, and apply the value to the central element. It accepts two parameters – the pixel data and the kernel size in integer format. The syntax is as follows:

```
result = cv2.medianBlur(src, kernel_size)
```

Let's modify the example code of the mean-blur with the following line and compare the resulted image with the result of the **cv2.blur()** function.

```
mb = cv2.medianBlur(data, 3)
```

Gaussian Blur

Gaussian blur replaces pixels with a weighted average of surrounding pixels. The weights come from the Gaussian probability distribution, and the nearest pixels are more influential. The cv2.GaussianBlur() function three parameters – the pixel data, the kernel size (2-tuple) and the sigma. The syntax is as follows:

```
gb = cv2.GaussianBlur(src, kernel_size, sigma)
```

The cv2.GaussianBlur() function generates a kernel based on the provided kernel size and sigma. The following is an example 3 x 3 kernel with a sigma of 1.

0.078	0.123	0.078
0.123	0.195	0.123
0.078	0.123	0.078

3 x 3 Gaussian kernel
with sigma = 1

Modify again the example code of the mean-blur with the following line and compare the resulted image with the results of other blur functions.

```
gb = cv2.GaussianBlur(data, (3, 3), 1)
```

Edge Detection

Edge detection is a method to find the points and edges in a digital image. OpenCV library includes **Canny Edge Detection** that is a very popular algorithm. In the next example, we will use the **cv2.Canny()** function to do the edge detection.

The syntax of the **cv2.Canny()** function is as follows:

```
ed = cv2.Canny(src, threshold1, threshold2, apertureSize, L2gradient)
```

The meanings of the parameters are:

- data – the pixel data.
- threshold1 and threshold2 – thresholds used to determine the potential edges. In general, threshold1 should be less than threshold 2.
- apertureSize – aperture size for the Sobel operator: 3, 5, or 7. The default value is 3.
- L2gradient – a flag that indicates whether a more accurate calculation should be used or not. The default one (L2gradient = False) is good enough.

Since the edge detection is susceptible to the noise in the image, usually we smooth (i.e., blur) the image first. The typical procedure is as follows:

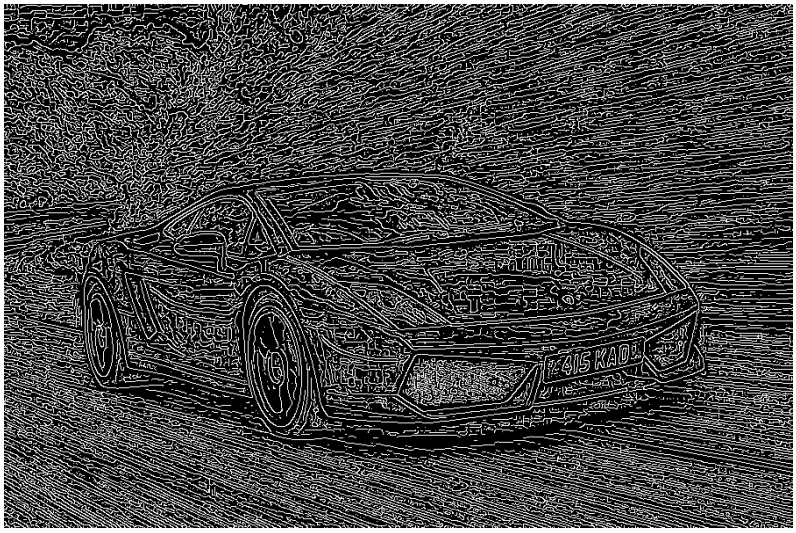
- Load the image content from an image file.
- Apply Gaussian blur to remove the noise.
- Run the edge detection.

1. Try the following example code and check the result:

```
#%%
import cv2
from PIL import Image

data = cv2.imread('images/oversharpened_car.jpg')
blur = cv2.GaussianBlur(data, (5, 5), 1)
edges = cv2.Canny(blur, 150, 200, apertureSize = 3, L2gradient = False)

img = Image.fromarray(edges)
img.show()
```

The following are the example outputs with different values of apertureSize parameter:

apertureSize = 3 (default)



apertureSize = 7

2. Add the following line before the statement of **Image.fromarray()** to invert the color.

```
edges = 255 - edges
```

3. Run the code cell again and check the result.

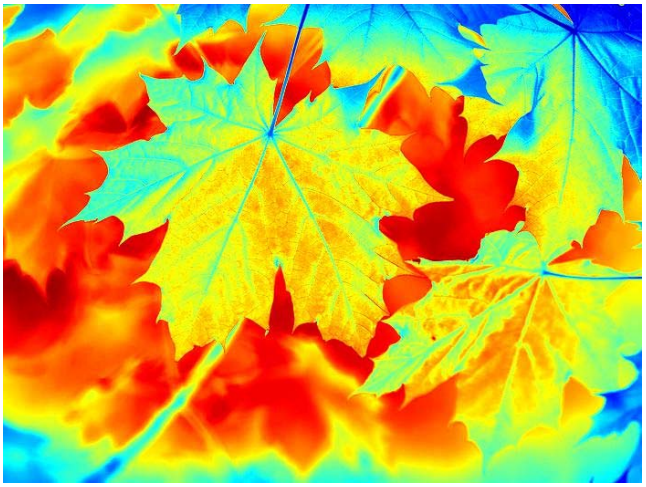
4. Adjust the size of the Gaussian kernel, run the code again and compare the results.

5. Adjust the thresholds of Canny function, run the code again and compare the results.

Color mapping

A colormap is a mapping from 0-255 values to 256 RGB colors. This can be helpful if you only have a grey picture but want to make it colorful. In OpenCV, we can apply a color map using a lookup table. For example, we apply a colormap “Jet” to a grayscale image.

Original Image (leaves.jpg)



Applied with colormap “JET”

The followings are 12 color maps defined by **OpenCV**:

COLORMAP_AUTUMN	COLORMAP_JET	COLORMAP_RAINBOW
COLORMAP_BONE	COLORMAP_OCEAN	COLORMAP_SPRING
COLORMAP_HOT	COLORMAP_PARULA	COLORMAP_SUMMER
COLORMAP_HSV	COLORMAP_PINK	COLORMAP_WINTER

The **cv2.applyColorMap()** function is used to apply a color map to an image. It accepts two parameters – *src* (grayscale image data) and *colormap* (name of the color map). The syntax is as follows:

```
cm = cv2.applyColorMap(src, colormap)
```

1. Try the following example:

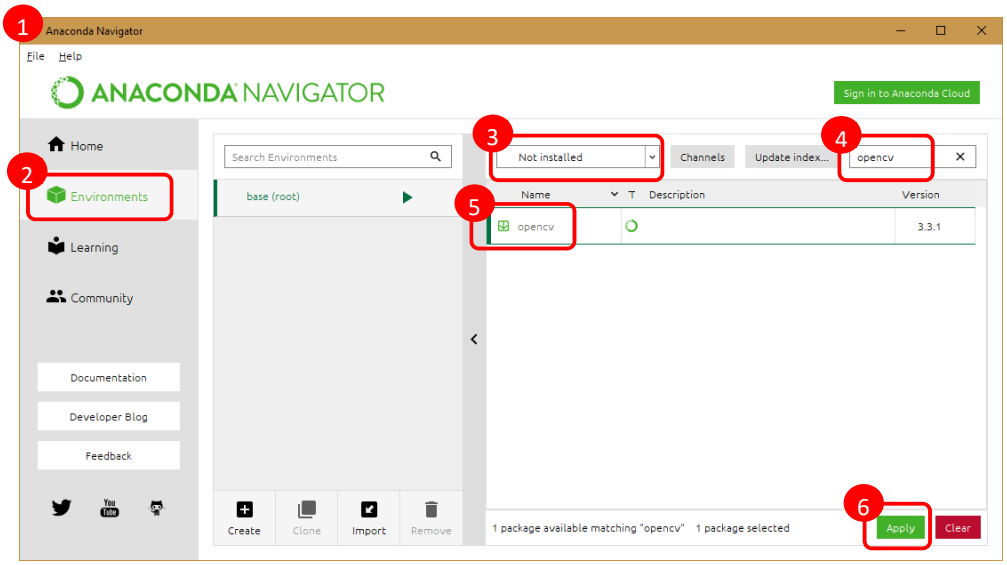
```
#%%
import cv2
from PIL import Image

data = cv2.imread("images/leaves.jpg", cv2.IMREAD_GRAYSCALE)
data = cv2.applyColorMap(data, cv2.COLORMAP_JET)
data = cv2.cvtColor(data, cv2.COLOR_BGR2RGB)
img = Image.fromarray(data)
img.show()
```

2. Try different color maps and check the results.

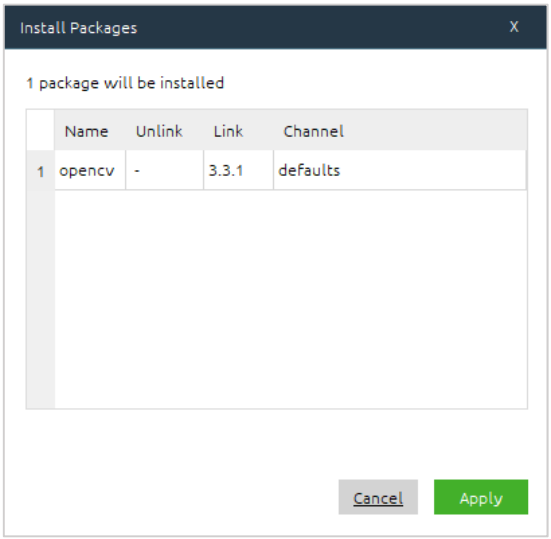
Appendix 1

Follow the procedure below to install the **OpenCV** package.

1. Launch Anaconda Navigator with Administrator permission.
2. Select Environments in the left panel.
3. At the top of the main panel, select “Not installed”.
4. Type “opencv” in the search box.
5. Check the item opencv in the package list.
6. Click “Apply” to confirm the selection.



7. In the “Install Packages” window, click “Apply” to start installation.



8. After the installation, you can see that the package list of Anaconda becomes empty. So, you can now use the OpenCV in Spyder. If Spyder is running already, you have to restart it.

Appendix 2

The sample code of the **kaleidoscope()** function:

```
def kaleidoscope(src_img):
    w, h = src_img.size;
    s = min(w, h)
    l = (w - s) // 2
    r = l + s
    t = (h - s) // 2
    b = t + s

    tmp1 = src_img.crop((l, t, r, b))
    tmp2 = tmp1.transpose(Image.FLIP_TOP_BOTTOM)
    tmp1 = Image.blend(tmp1, tmp2, 0.5)

    for i in range(0,5):
        tmp2 = tmp1.rotate(36)
        tmp1 = Image.blend(tmp1, tmp2, 0.5)

    tmp1 = ImageEnhance.Color(tmp1).enhance(2.0)
    tmp1 = ImageEnhance.Contrast(tmp1).enhance(2.0)
    return tmp1
```