

Lab Instructions - session 2

Image's processing and manipulation

Reading and displaying a grayscale image:

```
>>> import numpy as np
>>> import cv2
>>> I = cv2.imread('lake.jpg', cv2.IMREAD_GRAYSCALE)
>>> I.shape
>>> I.dtype
>>> I[10,20]
>>> cv2.imshow('Lake', I)
>>> cv2.waitKey(5000)
>>> cv2.destroyAllWindows()
>>> color_image = cv2.imread('lake.jpg')
>>> gray_I = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
>>> gray_I.shape
>>> gray_I.dtype
>>> cv2.waitKey(5000)
>>> cv2.destroyAllWindows()
>>> cv2.imshow('Grayscale (Direct)', I)
>>> cv2.imshow('Grayscale (Converted)', gray_I)
>>> cv2.waitKey(0)
>>> cv2.destroyAllWindows()
```

Reading a color image:

```
>>> I = cv2.imread('lake.jpg', cv2.IMREAD_UNCHANGED)
>>> I.shape
>>> I.dtype
>>> I = cv2.imread('lake.jpg')
>>> I.shape
>>> I.dtype
>>> I[10,20]
>>> cv2.imshow('lake.jpg', I)
>>> cv2.waitKey()
>>> cv2.destroyAllWindows()
```

Side note: if we use `cv2.waitKey()` without waiting time, it will wait for the user to press a key to continue.

Display the image using matplotlib:

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

- Are the image colors strange? Remember scipy/matplotlib use the RGB system, while opencv uses the BGR system. Reading an image with opencv and displaying with matplotlib makes the blue and red channels swapped.

```
>>> plt.imshow(I[:, :, ::-1])
>>> plt.show()
```

Remember, `I[:, :, ::-1]` reverses the order of the last dimension (the channels) and hence converts BGR (opencv) to RGB (for displaying with matplotlib). Could you suggest another way using cv2 for the above conversion?

Set the green channel equal to zero

remember openCV uses BGR: Blue: 0, Green: 1, Red: 2

```
>>> I[:, :, 1] = 0
>>> plt.imshow(I[:, :, ::-1])
>>> plt.show()
```

Saving an image

```
>>> cv2.imwrite('lake_nogreen.jpg', I)
```

Displaying 'red', 'green' and 'blue' channels

Open a **code editor** and write the following code (or open the file **brg1.py** in the instructions folder). It displays the original image, the blue channel, the green channel and the red channel of the image when the keys 'o', 'b', 'g' and 'r' are pressed respectively. Notice that the green channel is brighter in green areas. So are the red and blue channels.

```
import cv2
import numpy as np

I = cv2.imread('lake.jpg')
```

```
# notice that OpenCV uses BGR instead of RGB!
B = I[:, :, 0]
G = I[:, :, 1]
R = I[:, :, 2]

cv2.imshow('win1', I)

while 1:
    k = cv2.waitKey()

    if k == ord('o'):
        cv2.imshow('win1', I)
    elif k == ord('b'):
        cv2.imshow('win1', B)
    elif k == ord('g'):
        cv2.imshow('win1', G)
    elif k == ord('r'):
        cv2.imshow('win1', R)
    elif k == ord('q'):
        cv2.destroyAllWindows()
        break
```

Task 1:

The above code shows the B, G and R channels as **grayscale** images. Change the above code so that the blue channel is displayed in the blue, green channel in the green, and red channel in red. Notice that since these are color images they have to be **mxnx3** arrays. For a purely red image, the green and blue channels are entirely zero. The following code might help:

```
B = np.zeros(I.shape, dtype=np.uint8)
print(B.shape)
B[:, :, 0] = I[:, :, 0]
```

Alternatively, you can create a copy of an image and change it:

```
B = I.copy()
# now change B so that it only contains the blue channel
```

Blending two images

Method 1:

```
import cv2
import numpy as np

I = cv2.imread('damavand.jpg')
J = cv2.imread('eram.jpg')

print(I.shape)
print(J.shape)

K = I.copy()
K[:, ::2, ::2, :] = J[:, ::2, ::2, :]

cv2.imshow("Image 1", I)
cv2.imshow("Image 2", J)
cv2.imshow("Blending", K)

cv2.waitKey(3000)
cv2.destroyAllWindows()
```

- What does the above piece of code do?
- What does `K[:, ::2, ::2, :] = J[:, ::2, ::2, :]` do? (notice that `J[:, ::2, ::2, :]` represents every second row and every second column of J)
- Why have we written `K = I.copy()` instead of `K = I`?

Method 2

```
K = I//2+J//2
```

- What is the difference between methods 1 and 2?

numpy addition vs opencv addition

```
>>> a = np.array([1,2,3,4,5,6,7,8,9,10], dtype=np.uint8)
>>> a
>>> b = np.full((10,), 250, dtype=np.uint8)
>>> b
```

```
>>> a+b  
>>> cv2.add(a,b)
```

- What is the difference between **a+b** and **cv2.add(a,b)**? Which one do you think should be used for adding up images?

Blending with arbitrary ratios

0.8 of image **I** plus .2 of image **J**.

```
K = np.clip((0.8*I + 0.2*J), 0, 255).astype(np.uint8)
```

- Why have we used `.astype(np.uint8)`? `print((0.8*I).dtype)` to see why.

```
K = cv2.addWeighted(I,0.8,J,0.2, 0)  
K = cv2.addWeighted(I,0.1,J,0.9, 0)  
K = cv2.addWeighted(I,0.3,J,0.7, 0)
```

For floating point variables **alpha**, **beta** and **gamma**, running `cv2.addWeighted(I,alpha,J,beta,gamma)` is similar to running `np.clip((alpha*I + beta*J + gamma), 0, 255).astype(np.uint8)`

Task 2:

Make an animated smooth transition from the image **I** to the **J**. You can use “`cv2.waitKey(n)`” to delay for **n** milliseconds.

Task 3:

Create a smooth animation that changes image **I** from grayscale to color.

Hint: Grayscale images have equal values in all three color channels.

References

- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_core/py_table_of_contents_core/py_table_of_contents_core.html#py-table-of-content-core
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html