DIGITAL IMAGE PROCESSING COURSE - 505060
PRACTICE LABS

LAB 02. IMAGE MANIPULATION

Requirements

(1) Follow the instructions with the help from your instructor.
(2) Finish all the exercises in class and do the homework at home. You can update your solutions after class and re-submit all your work together with the homework.
(3) Grading
Progress Evaluation 2 score = 50% * Attendance + 50% * Exercises
Rules:
- If the number of finished exercises is less than **80% total number of excercises**, you will get **zero** for the lab.
- Add the text of your Student ID to each of the output image.
- Submit the source code and output image files directly to Elearning assignment, donot compress the files.

(4) Plagiarism check
If any 2 of the students have the same output images, then all will get zero for the corresponding exercises.

INTRODUCTION

In this Lab, you will learn how to
- Pixel, region manipulation
- Color, image manipulation
- Video processing
- Image brightness and contrast

INSTRUCTIONS

---

1. Pixel, region manipulation

1.1 Accessing and Modifying pixel values

Let's load a color image first:

```
>>> import numpy as np
>>> import cv2 as cv
>>> img = cv.imread('messi5.jpg')
```

You can access a pixel value by its row and column coordinates. For BGR image, it returns an array of Blue, Green, Red values. For grayscale image, just corresponding intensity is returned.

```
>>> px = img[100,100]
>>> print( px )
[157 166 200]
```

```
# accessing only blue pixel
>>> blue = img[100,100,0]
>>> print( blue )
157
```

You can modify the pixel values the same way.

```
>>> img[100,100] = [255,255,255]
>>> print( img[100,100] )
[255 255 255]
```

Warning: *Numpy is an optimized library for fast array calculations. So simply accessing each and every pixel value and modifying it will be very slow and it is discouraged.*

Better pixel accessing and editing method :

```
# accessing RED value
>>> img.item(10,10,2)
59
# modifying RED value
>>> img.itemset((10,10,2),100)
>>> img.item(10,10,2)
100
```

Image Addition (eg. in brightness enhancement)

You can add two images with the OpenCV function, **cv.add()**, or simply by the numpy operation res = img1 + img2. Both images should be of same depth and type, or the second image can just be a scalar value. ***Note:*** *There is a difference between OpenCV addition and Numpy addition. OpenCV addition is a saturated operation while Numpy addition is a modulo operation.*

For example, consider the below sample. We can add two images by using the function cv2.add().

```
>>> x = np.uint8([250])
>>> y = np.uint8([10])
>>> print( cv.add(x,y) ) # 250+10 = 260 => 255
>>> print( x+y )          # 250+10 = 260 % 256 = 4
```

This will be more visible when you add two images. Stick with OpenCV functions, because they will provide a better result, such as: *cv2.**addWeighted**(img1, wt1, img2, wt2, gammaValue)* or *cv2.**subtract**(src1, src2)*

```python
# Python programe to illustrate
# arithmetic operation of
# subtraction of pixels of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
image1 = cv2.imread('input1.jpg')
image2 = cv2.imread('input2.jpg')

# cv2.subtract is applied over the
# image inputs with applied parameters
sub = cv2.subtract(image1, image2)

# the window showing output image
# with the subtracted image
cv2.imshow('Subtracted Image', sub)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

```python
# Python programe to illustrate
# arithmetic operation of
# addition of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
image1 = cv2.imread('input1.jpg')
image2 = cv2.imread('input2.jpg')

# cv2.addWeighted is applied over the
# image inputs with applied parameters
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)

# the window showing output image
# with the weighted sum
cv2.imshow('Weighted Image', weightedSum)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Source: https://www.geeksforgeeks.org/arithmetic-operations-on-images-using-opencv-set-1-addition-and-subtraction/

1.2 Accessing Image Properties

Image properties include number of rows, columns, and channels; type of image data; number of pixels; etc. The shape of an image is accessed by img.shape. It returns a tuple of the number of rows, columns, and channels (if the image is color):

```
>>> print( img.shape )
(342, 548, 3)
```

Note: *If an image is grayscale, the tuple returned contains only the number of rows and columns, so it is a good method to check whether the loaded image is grayscale or color.*

Total number of pixels is accessed by `img.size`:

```
>>> print( img.size )
562248
```

Image datatype is obtained by `img.dtype`:

```
>>> print( img.dtype )
uint8
```

Note: *img.dtype is very important while debugging because a large number of errors in OpenCV-Python code are caused by invalid datatype.*

1.3 Image ROI

Sometimes, you will have to play with certain regions of images. For eye detection in images, first face detection is done over the entire image. When a face is obtained, we select the face region alone and search for eyes inside it instead of searching the whole image. It improves accuracy (because eyes are always on faces :D) and performance (because we search in a small area).

ROI is again obtained using Numpy indexing. Here I am selecting the ball and copying it to another region in the image:

```
>>> ball = img[280:340, 330:390]
>>> img[273:333, 100:160] = ball
```

Check the results below:

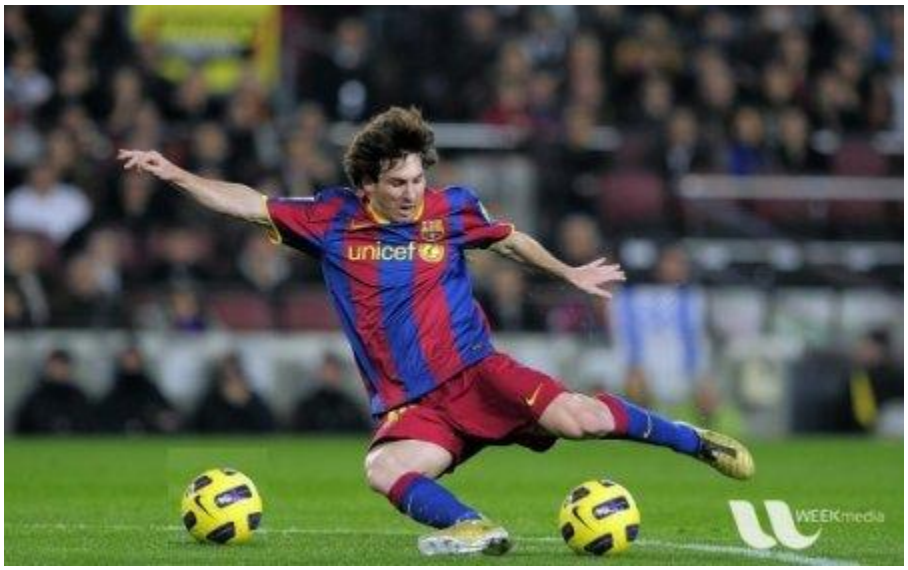


Image from link

1.4 Splitting and Merging Image Channels

Sometimes you will need to work separately on the B,G,R channels of an image. In this case, you need to split the BGR image into single channels. In other cases, you may need to join these individual channels to create a BGR image. You can do this simply by:

```
>>> b,g,r = cv.split(img)
>>> img = cv.merge((b,g,r))
```

Or

```
>>> b = img[:,:,0]
```

Suppose you want to set all the red pixels to zero - you do not need to split the channels first. Numpy indexing is faster:

```
>>> img[:,:,2] = 0
```

Warning: ***cv.split()*** *is a costly operation (in terms of time). So use it only if necessary. Otherwise go for Numpy indexing.*

2. Color, image manipulation

2.1. Convert color space with OpenCV

Syntax: cv2.**cvtColor**(src, code[, dst[, dstCn]])

Parameters:

src: It is the image whose color space is to be changed.
code: It is the color space conversion code. More details here
dst: It is the output image of the same size and depth as src image. It is an optional parameter.
dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.

Return Value: It returns an image.

HSV color space :

H : Hue represents dominant wavelength.

S : Saturation represents shades of color.

V : Value represents Intensity.

```
# Python program to read image
# as HSV color space

# Importing cv2 module
import cv2

# Reads the image
img = cv2.imread('g4g.png')

# Converts to HSV color space
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Shows the image
cv2.imshow('image', img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

LAB color space :

L – Represents Lightness.

A – Color component ranging from Green to Magenta.

B – Color component ranging from Blue to Yellow.

```
# Python program to read image
# as LAB color space

# Importing cv2 module
import cv2

# Reads the image
img = cv2.imread('g4g.png')

# Converts to LAB color space
img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

# Shows the image
cv2.imshow('image', img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Source: https://www.geeksforgeeks.org/python-visualizing-image-in-different-color-spaces/

2.2. Filter Color with OpenCV

For color segmentation, all we need is the threshold values or the knowledge of the lower bound and upper bound range of colors in one of the color spaces. It works best in the Hue-Saturation-Value color space. After specifying the range of color to be segmented, it is needed to create a mask accordingly and by using it, a particular region of interest can be separated out.

It is very simple and you can use the same function, **cv.cvtColor()**. Instead of passing an image, you just pass the BGR values you want. For example, to find the HSV value of Green, try the following commands in a Python terminal:

```
>>> green = np.uint8([[[0,255,0 ]]])
>>> hsv_green = cv.cvtColor(green,cv.COLOR_BGR2HSV)
>>> print( hsv_green )
[[[ 60 255 255]]]
```

Now you take [H-10, 100,100] and [H+10, 255, 255] as the lower bound and upper bound respectively. Apart from this method, you can use any image editing tools like GIMP or any online converters to find these values, but don't forget to adjust the HSV ranges.

```
# convert color to hsv because it is easy to track colors in this color model
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
lower_hsv = np.array([ilowH, ilowS, ilowV])
higher_hsv = np.array([ihighH, ihighS, ihighV])
# Apply the cv2.inrange method to create a mask
mask = cv2.inRange(hsv, lower_hsv, higher_hsv)
# Apply the mask on the image to extract the original color
frame = cv2.bitwise_and(frame, frame, mask=mask)
cv2.imshow('image', frame)
```

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):
    _, frame = cap.read()
    # It converts the BGR color space of image to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold of blue in HSV space
    lower_blue = np.array([60, 35, 140])
    upper_blue = np.array([180, 255, 255])

    # preparing the mask to overlay
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # The black region in the mask has the value of 0,
    # so when multiplied with original image removes all non-blue regions
    result = cv2.bitwise_and(frame, frame, mask = mask)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('result', result)

    cv2.waitKey(0)

cv2.destroyAllWindows()
cap.release()
```

Source: https://www.geeksforgeeks.org/filter-color-with-opencv

3. Video processing using the OpenCV Python

To capture a video, we need to create a VideoCapture object. VideoCapture have the device index or the name of a video file. Device index is just the number to specify which camera. If we pass 0 then it is for first camera, 1 for second camera so on. We capture the video frame by frame.

Syntax :

cv2.***VideoCapture***(**0**): Means first camera or webcam.

cv2.***VideoCapture***(**1**): Means second camera or webcam.
cv2.***VideoCapture***("***file name.mp4***"): Means video file
(Note : Video file should have in same directory where program is executed.)

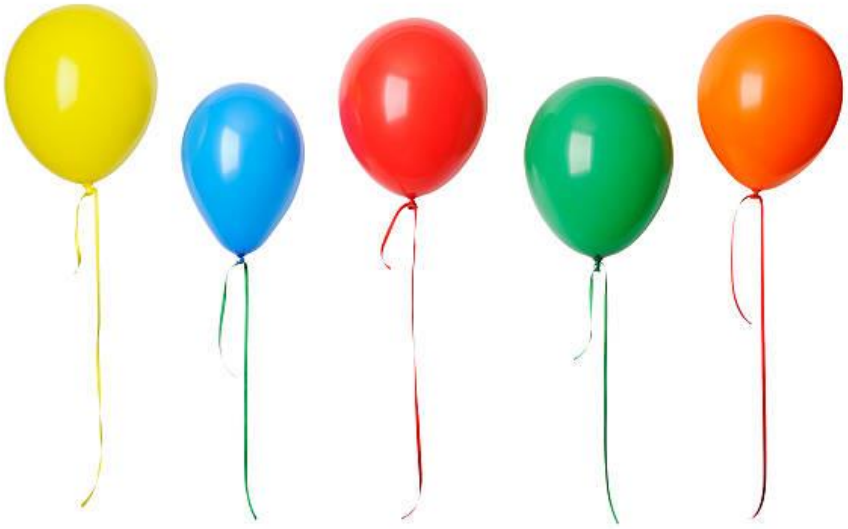
```
# importing libraries
import cv2
import numpy as np

# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture('tree.mp4')

# Check if camera opened successfully
if (cap.isOpened()== False):
  print("Error opening video  file")

# Read until video is completed
while(cap.isOpened()):

  # Capture frame-by-frame
  ret, frame = cap.read()
  if ret == True:

    # Display the resulting frame
    cv2.imshow('Frame', frame)

    # Press Q on keyboard to  exit
    if cv2.waitKey(25) & 0xFF == ord('q'):
      break

  # Break the loop
  else:
    break

# When everything done, release
# the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```

Source: https://www.geeksforgeeks.org/python-play-a-video-using-opencv/

EXERCISE

Ex2.1. The balloons

Given the input image as below (Image link here)



1. Split each color channel of the image

   ***Hint**: Split & merge (check instructions)*

2. Locate the position of each balloon by drawing a rectangle (bounding-box) surrounding each balloon

   ***Hint**: Drawing rectangle howto (review)*

3. Name each balloon by putting a text of color name right above the bounding boxes.

   ***Hint**: Drawing text howto (review)*

4. Extract the yellow balloon by creating a new image of only one balloon.

   ***Hint**: Extract a rectangle ROI using a predefined position (check instructions)*

5. Extract the yellow balloon automatically by using HSV color space to extract only pixels of yellow color.

   ***Hint**: Change into HSV color space, then filter the yellow pixels by keeping only pixels having a range of H-values of yellow (try to find the range by yourself)*
   https://docs.opencv.org/master/df/d9d/tutorial_py_colorspaces.html

6. Re-paint the yellow balloon by replacing the pixels of yellow by green.

   ***Hint**: Filter yellow pixels by HSV space and replace the value of the pixels by new values (green color)*

7. Rotate the first balloon an angle of 20 degree

   ***Hint**: Rotate image howto (review)*

Ex2.2. Image enhancement

Open your webcam to read an image of your face

8. Increase the brightness of the image
9. Enhance the image contrast by using global histogram equalization
10. Enhance the image contrast by using adaptive histogram equalization (CLAHE)

Additional exercises

Do exercises in pages 25, 32, 38, 46, 48 of the "Theory1" file in Elearning website.