TÉCNICO LISBOA

Learning based Multimedia Processing

2023/2024

Lab classes (2)

1 Basic image processing with OpenCV

1.1 Open and visualize image

From Anaconda:

- Select **LBMP environment**
- Launch **Jupyter Notebook**

Create a new notebook: `python3 (ipkernel)`

Rename the file: `image_basics.ipynb`

Read and display an image

An image can be read using `imread()` from **OpenCV** (`cv2` module), specifying the path and the image name:

```
import cv2
import matplotlib.pyplot as plt


img = cv2.imread("images/i01.jpg")
print("image shape (lines, columns, channels) = ", img.shape)
print("\nimage type: ", type(img))
```

The image is now treated as a matrix with rows and columns, and the pixel values stored in `img`.

```
image shape (lines, columns, channels) = (400, 600, 3)
```

The image is stored in a NumPy array:

```
<class 'numpy.ndarray'>
```

The image can be displayed using **`matplotlib pyplot`** function `imshow`:

```
#cv2.imshow() doesn't work well on Jupyter notebooks - use matplotlib
#matplotlib considers RGB images, but OpenCV uses the BGR format
#convert from BGR to RGB:
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # BGR to RGB
plt.imshow(rgb_img)
plt.axis('off')
plt.show()
```

Example 1 code:

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("images/i01.jpg")  # read image
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # BGR to RGB
plt.imshow(rgb_img)
plt.axis('off')
plt.show()
```

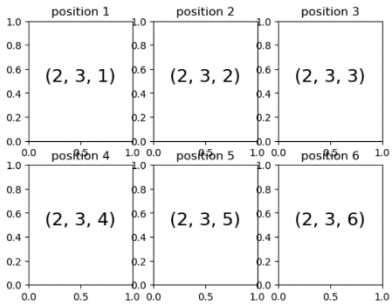
Check image properties:

```
print("image shape (lines, columns, channels) = ", img.shape)
print("Total number of pixels: ", img.size)
print("Image datatype", img.dtype)
```

```
image shape (lines, columns, channels) =  (400, 600, 3)
Total number of pixels:  720000
Image datatype uint8
```

Using **`subplot`** to display several images *(lines, cols, pos)*:

```
for i in range(1, 7):      # 1 <= i < 7
    plt.subplot(2, 3, i)
    plt.text(0.5, 0.5, str((2, 3, i)),
             fontsize=18, ha='center')
    plt.title('position {}'.format(i))
```



To display the first 6 images in the *images* directory:

```
import cv2
import matplotlib.pyplot as plt

for i in range(1, 7):
    plt.subplot(2, 3, i)
    name = f"images/i0{i}.jpg"
    img = cv2.imread(name)    # read image
    rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # convert to RGB
    plt.imshow(rgb_img)
    plt.title('i0{}.jpg'.format(i))  # title of each plot
    plt.axis('off')
```

1.2 Image manipulation

Crop image

To crop an image just consider a selected area of the original matrix

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("images/i01.jpg")
print("image shape (lines, columns, channels) = ", img.shape)

rows, cols = img.shape[0:2]
startRow = int(rows*.15)
startCol = int(cols*.15)
endRow = int(rows*.85)
endCol = int(cols*.85)

cropped_img = img[startRow:endRow, startCol:endCol]

print("cropped image shape (lines, columns, channels) = ",
cropped_img.shape)
```

image shape (lines, columns, channels) = (400, 600, 3)
cropped image shape (lines, columns, channels) = (280, 420, 3)

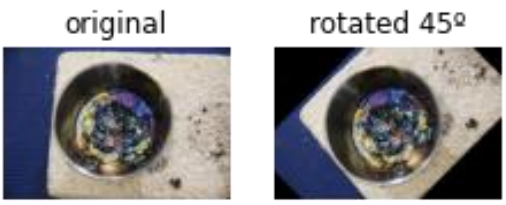
Rotate image

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("images/i01.jpg")
height, width = img.shape[0:2]

# get rotation matrix: cv2.getRotationMatrix2D(center, angle, scale)
angle = 45
rot_matrix = cv2.getRotationMatrix2D((width/2, height/2), angle, 1)

# rot = cv2.warpAffine(input_img, rot_matrix, (width, height) output)
rot1 = cv2.warpAffine(img, rot_matrix, (width, height))
```



Resize image

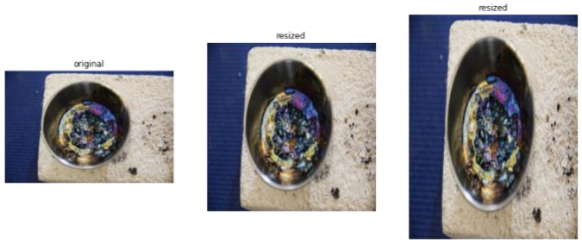
```
import cv2

img = cv2.imread("images/i01.jpg")
new_height = 300
new_width = 300
new_size = (new_width, new_height)

# cv2.resize(src, dest_size, fx, fy, interpolation)
r1 = cv2.resize(img, new_size)
print("resized image shape (lines, columns, channels) = ", r1.shape)

r2 = cv2.resize(img, None, fx=1, fy=2)
print("resized image shape (lines, columns, channels) = ", r2.shape)
```

```
original image shape (lines, columns, channels) = (400, 600, 3)
r1 image shape (lines, columns, channels) = (300, 300, 3)
r2 image shape (lines, columns, channels) = (800, 600, 3)
```



(note: all images represented with same width due to usage of subplot)

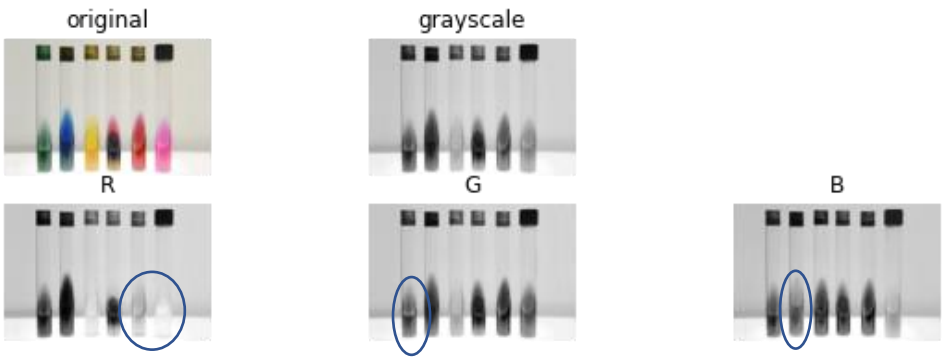
Grayscale and R, G, B image components

Get image components and conversion to grayscale:

```
import cv2

img = cv2.imread("images/i02.jpg")
b, g, r = cv2.split(img)
# alternative: b = img[:,:,0]; g = img[:,:,1];  r = img[:,:,2]

# image in gray scale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```



Overlap text, lines and shapes on images

```
import cv2

img = cv2.imread("images/i04.jpg")

# draw rectangle
rect = img.copy()
cv2.rectangle(rect, (380, 70), (470, 150), (255, 0, 0), 3)

# draw arrow + insert text
out = img.copy()
cv2.arrowedLine(out, (200, 60), (380, 110), (255, 0, 0), 3)
cv2.putText(out, "North Tower", (20, 40), cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 0, 0), 2)
```

2 Handling video with OpenCV

Get video from file

```
import cv2

# read video from file
capture = cv2.VideoCapture("images/short_IST.mp4")

# display the video
while capture.isOpened():
    ret, frame = capture.read()
    if not ret:
        break

    cv2.imshow("Video Window", frame)
    cv2.waitKey(25)

capture.release()
cv2.destroyAllWindows()
```

Capture video from camera

The file name in VideoCapture is replaced with 0 (zero)

```
import cv2

capture = cv2.VideoCapture(0)

# display the video
while capture.isOpened():
    ret, frame = capture.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    cv2.imshow("Video Window", frame)

    # enable keyboard shortcut to stop capturing - press "q" to stop
    if cv2.waitKey(1) == ord('q'):
        break

capture.release()
cv2.destroyAllWindows()
```

Capture and store video from camera

```
import cv2
capture = cv2.VideoCapture(0)

# Default resolutions of the frame are obtaine
frame_width = int(capture.get(3))
frame_height = int(capture.get(4))

# Define the codec and create VideoWriter object.
#   The output is stored in a 'mp4' file.
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('video\ex_output_video.mp4', fourcc, 10,
(frame_width,frame_height))

# start capture
while capture.isOpened():
    ret, frame = capture.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    # include any desired image processing here
    # write the captured frame
    out.write(frame)
    # show the captured frame
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) == ord('q'):
        break
# Release everything if job is finished
capture.release()
out.release()
cv2.destroyAllWindows()
```

3 Steganography example

```
import cv2
import matplotlib.pyplot as plt

def embed(cover, secret, k):
    mask = 256 - 2**k
    stego = (cover & mask) | (secret >> (8 - k))
#     cv2.imwrite('stego.png', stego)
    return stego

def extract(stego, k):
    mask = 2**k - 1
    output = (stego & mask) << (8 - k)
#     cv2.imwrite('extracted.png', output)
    return output

img1 = cv2.imread("images/i04.jpg") # load cover image
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB) # convert to RGB

img2 = cv2.imread("images/i02.jpg") # load secret image
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB) # convert to RGB

# INPUT PARAMETER
k = 3   # number of bits used for image embeding

print (f"k={k}\noriginal mask (256 - 2^k) =", 256 - 2**k)
print ("secret mask (2^k - 1) =", 2**k - 1)
print ("8-k =", 8-k, "bits shift to the right for embeding, and to
the left for secret recovery")
print (8-k, "bits shift to the rigth: 128 becomes ", 128 >> (8 - k))

img_stego = embed(img1, img2, k)

img_recover = extract (img_stego, k)

plt.subplot(2, 2, 1)
plt.imshow(img1, 'gray')
plt.title('original (cover)')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(img2, 'gray')
plt.title('secret')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(img_stego, 'gray')
plt.title('orig + secret')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(img_recover, 'gray')
plt.title('recovered')
plt.axis('off')
```