LAB SESSION 1

AUDIO PROCESSING, VIDEO PROCESSING AND COMPUTER VISION

BACHELOR IN DATA SCIENCE AND ENGINEERING

1. READING AND DISPLAYING IMAGES IN PYTHON

READING IMAGES

To load an image, we will use the *io.imread* function of the Scikit-Image library:

```
from skimage import io
x = io.imread('peppers.png')
```

In the variable x we will have stored a 2 or 3 dimensional matrix, depending on whether it is a gray-scale or color image, respectively.

In the first case, the dimensions of the matrix are N x M, where M represents the horizontal dimension of the image (number of pixels of the x-axis) and N the vertical dimension (number of pixels of the y-axis), being (0,0) the coordinates of the upper left pixel. Each element contains the luminance (intensity) value of each image pixel.

In the second case (color image), the dimensions of the matrix are N x M x 3, as in the third dimension we have the intensities in the three color channels R, G and B (i.e., red, green and blue).

When reading the data of an image with the *io.imread* function the intensity values of each pixel will be given in 8-bit unsigned integer format (*uint8*). Thus, a variable in this format can only take integer values between 0 and 255. When you have to use certain mathematical functions (e.g. logarithm or exponential) you may encounter the problem that these functions operate only on floating-point numbers. In these cases, the commands will help you:

```
x_float = skimage.img_as_float(x)
x_uint = skimage.img_as_uint(x_float)
```

These functions transform a number, vector or matrix from its original format to *float* or *uint8* format, respectively.

DISPLAYING IMAGES

There are several useful functions for displaying an image; in particular, we will use *plt.imshow* from the Matplotlib library.

```
plt.imshow(x)
plt.show()
```

If you use the *plt.imshow* function, the image format must be *uint8* for correct display without setting additional parameters. If the data is in *float* format, Matplotlib "expects" that these data are between 0 to 1. If not, we must normalize the data to that range.

A map or color palette (*cmap* parameter *in plt.imshow*) is a $m \times 3$ array that establishes a correspondence between an index and a color specified by 3 components. In other words, it provides a list of colors, each identified by its index, so we can represent a color image by a two-dimensional array that stores the index (color) associated with each pixel. This is practical, for example, in the display of monochrome infrared images, where we can assign colors close to red (warm) to high intensity values and colors close to blue (cold) to low intensity values.

This link includes a list of color maps available in Matplotlib:

https://matplotlib.org/tutorials/colors/colormaps.html

If the color map is not specified, Matplotlib will display:

– A heat map for 2D images. If you want to display the image in gray-scale, you need to specify *cmap='gray'*.
– If you have 3 components, it will assume that it is an RGB image.

EXERCISE 1.1

– Read the image '*dreamers.png*' (use *io.imread*) and display it by using *plt.imshow*. What are the image dimensions? Is it a color image or gray-level image? What is the range of the pixel values?

EXERCISE 1.2

Read the image 'peppers.png' (check its dimensions: $M \times N \times 3$). Display the red component in the following ways:

– Extract the red component as a 2-dimensional matrix and display it as a gray-scale image. How would you visually check that it is correct?
– Display it again, this time in pseudocolor, by changing the color palette (use, for example, *jet*).
– Display a true color image with the G and B components set to zero.

2. COLOR REPRESENTATION: RGB AND HSV

EXERCISE 2.1

Let's start using RGB.

– Read the image 'peppers.png'. Display the original image and its R, G and B components separately in one figure, using *plt.subplot*.

To move from one color space to another, we can use the functions of the module 'color' from the Scikit-Image library. We list some of these functions below:

- RGB to HSV: x_hsv = skimage.color.rgb2hsv(x_rgb)
- Gray-scale to RGB: x_gray = skimage.color.rgb2gray(x_rgb)
- RGB to Lab: x_lab = skimage.color.rgb2lab(x_rgb)
- HSV to RGB: x_rgb = skimage.color.hsv2rgb(x_hsv)

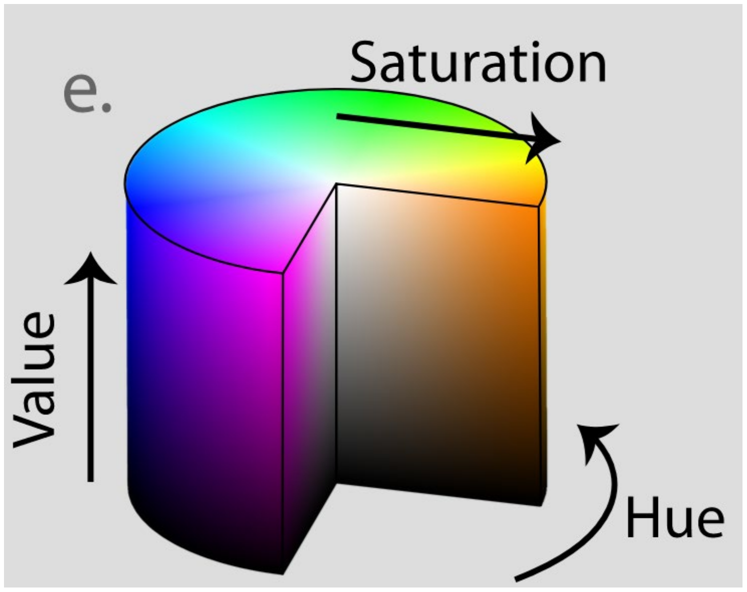


Figure 1. HSV color model (form Wikipedia)

- Repeat the previous exercise, now displaying the H, S and V components instead of R, G, and B. What do you see when displaying the HSV image? What is the range of the pixel values of the H component? Which H value represents red? Check with the R component. How does HSV encodes white?

EXERCISE 2.2

[OPTIONAL] Generate the following image of size 192 × 192. Use RGB. You can use *np.ones* or *np.zeros*

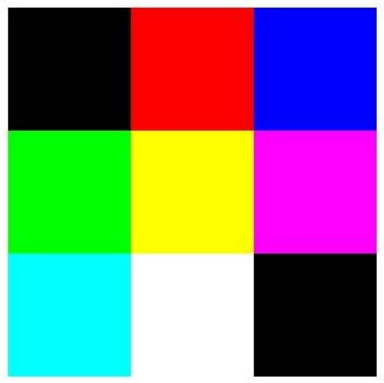


Figure 2. Color mosaic

3. HISTOGRAMS

HISTOGRAMS: GRAY-SCALE IMAGES

histogram (from *skimage.exposure* module) allows us to compute and display histograms:

```
from skimage import exposure
# x: image
h, bins = exposure.histogram(x)
plt.figure()
plt.plot(bins,h)
plt.show()
```

EXERCISE 3.1

- Read the grayscale image '*dreamers.png*' and display it.
- Use *exposure.histogram* to calculate and draw a histogram of 256 bins (0, 1, 2, ..., 255).
- Now compute a histogram with 32 bins and draw it using the plt.bar function, which is one of the usual ways of representing histograms.
- [OPTIONAL] Write your own function to calculate a histogram. The goal is to understand the concept and be able to implement it. Use the following syntax:

```
# x: image
def histogram(x):
# h: vector containing the histogram
# b: vector containing the corresponding gray levels
# - - - TYPE YOUR CODE HERE - - -
# ...
return h, b
```

– [OPTIONAL] Show the results of the previous sections in the same figure to check that they are identical.
– Calculate and draw a normalize version of the previous histogram so we can interpret it as a probability density function.

HISTOGRAMS: COLOR IMAGES

- Read the image '*187_0070.jpg*' and display it.
- Extract each one of the color components and display the 4 images (original and 3 color components) in the same figure (*subplot*).
- Divide a new figure into 6 parts and display each one of the components together with its histogram.

4. HISTOGRAM EQUALIZATION

HISTOGRAM EQUALIZATION: GRAY-SCALE IMAGES

The *exposure.equalize_hist* function of the Scikit-Image library allows you to perform the histogram equalization operation.

```
from skimage import exposure
img_eq = exposure.equalize_hist(img)
```

EXERCISE 4.1

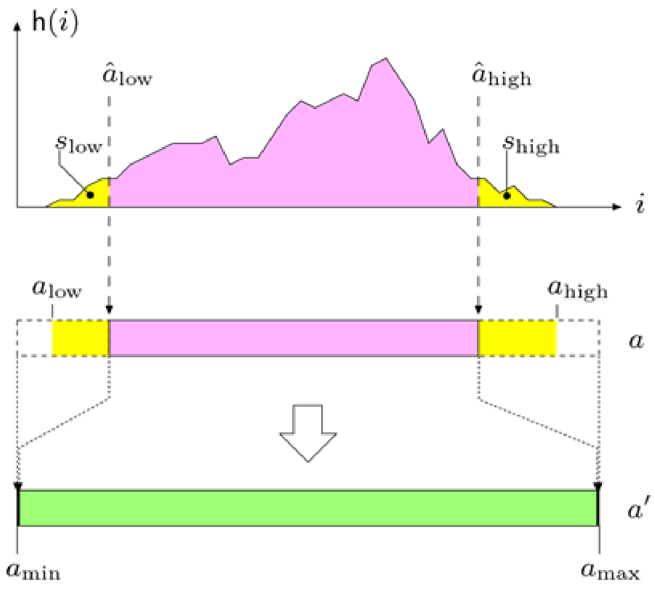
– Read the grayscale image '*pout.tif*'. Display the image and its histogram using *plt.subplot*.
– Use *exposure.equalize_hist* to perform a histogram equalization. Draw the new image and its histogram.
– Display both images next to each other.
– Calculate the cumulative histogram of the original image and the equalized image (use a standard histogram and *np.cumsum* function) and display them.

HISTOGRAM EQUALIZATION: COLOR IMAGES

EXERCISE 4.2

– Read the image '*187_0070.jpg*' and display it.

- Perform a histogram equalization of each RGB component and display the resulting image. Explain why the colors change.
- Convert the image into HSV and perform a histogram equalization of the V component. Are the colors now preserved?
- [OPTIONAL] As an alternative to histogram equalization, you can perform a linear contrast adjustment, as illustrated in the figure below. Use the Scikit-Image exposure.rescale_intensity function, and *alow* and *ahigh* values 15% above or below the real values, respectively.
- [OPTIONAL] Discuss the differences between histogram equalization and linear contrast adjustment.



5. FILTERS

For filtering we will rely mainly on the library *skimage.filters*. Alternatively, we could use *scipy.ndimage*.

5.1 LOW-PASS FILTERS

EXERCISE 5.1

- Perform a low-Gaussian filtering of the grayscale image '*dreamers.png*'. To do this, use the *skimage.filters.gaussian* function.
- Display the original image and the filtered version.
- Filter the original image using Gaussian filters with different standard deviations (sigma).

- [OPTIONAL] Now you have to contaminate the image with Gaussian noise. To do this, use the Scikit-Image *util.random_noise* function:

```
from skimage import util
dreamers_g = util.random_noise(dreamers, mode='gaussian')
```

- [OPTIONAL] Display the original and contaminated images. Repeat the Gaussian filtering operation, but this time on the contaminated image. Choose a suitable standard deviation to mitigate the noise and represent the original, contaminated and filtered images on the same figure.

5.2 HIGH-PASS FILTERS

EXERCISE 5.2.1: HORIZONTAL EDGES

- Detect the horizontal edges of the image 'dreamers.png'; you can use the following mask (*filters.sobel_h*):

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Display the filtered image.
- Compute the absolute value and diplay it.
- Detect edges on the filtered image using a threshold. Display the resulting binary edge image.

EXERCISE 5.2.2: VERTICAL EDGES

[OPTIONAL] Repeat the previous exercise for the case of vertical edges, you can use the following mask (*filters.sobel_v*):

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

EXERCISE 5.2.3: GRADIENT MAGNITUDE

- Use the function *filters.sobel* to compute the gradient magnitude, which combines the horizontal and vertical edges as follows:

$$\nabla I = \sqrt[2]{s_h^2 + s_v^2}$$

Where s_h and s_v are outputs of the horizontal and vertical sobel filters, respectively.

5.3 MEDIAN FILTER

EXERCISE 5.3

- Perform two 3x3 and 11x11 median filtering operations of the 'dreamers.png' image. Use the funcion *filters.median* from *Scikit-Image*:

```
# img: image, k: kernel size
from skimage import filters
img_filtered = filters.rank.median(img, np.ones((k,k))
```

- Display the original image and its filtered versions.
- Repeat the process for an image contaminated with salt & pepper noise *(noisy_image=util.random_noise(image, mode='s&p'))*. Which of the two filters studied so far (Gaussian or median) do you consider most useful for reducing impulsive noise?