

Image processing

-

[Exercise 11 \(to grayscale\)](#)

-

[Exercise 12 \(radial fade\)](#)

Image processing

An image is a collection of *pixels*, which is abbreviation for picture elements. A grayscale image can be represented as a two dimensional array, whose first axis corresponds to the x coordinate of the image and the second axis corresponds to the y coordinate. The array contains at each coordinate pair (x,y) a value, which is typically a float between 0.0 and 1.0, or an integer between 0 and 255. This specifies the level of grayness. For example, if the array contains value 255 at coordinates (0,0), then in the image the pixel at top left is white.

In color images there is third axis is for the red, green, and blue components of each pixel. For each of these color components there is a value between 0.0 and 1.0 (or between 0 and 255). The combinations of different values for the three components red, green, and blue can result in at least 16.7 million colors.

Since images can be represented as multidimensional arrays, we can easily process images using NumPy functions. Let's see examples of these.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
painting=plt.imread("painting.png")
print(painting.shape)
print(f"The image consists of {painting.shape[0] * painting.shape[1]} pixels")
plt.imshow(painting);
```

```
(368, 640, 3)
```

```
The image consists of 235520 pixels
```



Because the image is now a NumPy array, we can easily perform some operations:

```
plt.imshow(painting[:,::-1]); # mirror the image in x direction
```



In the following we set the pixels on the first 30 rows white:

```
painting2 = painting.copy() # don't mess the original painting!  
painting2[0:30, :, :] = 1.0 # max value for all three components produces white  
plt.imshow(painting2);
```



For a bit more complicated operation we can create a function that returns a copy of the image so that it fades to black as we move to left.

```
def fadex(image):  
    height, width = image.shape[:2]  
    m=np.linspace(0,1, width).reshape(1,width,1)  
    result = image*m # note that we rely on broadcasting here  
    return result
```

```
modified=fadex(painting)  
print(modified.shape)  
plt.imshow(modified);
```

```
(368, 640, 3)
```



Exercise 11 (to grayscale)

This exercise can give two points at maximum!

Part 1.

Write a function `to_grayscale` that takes an RGB image (three dimensional array) and returns a two dimensional gray-scale image. The conversion to gray-scale should take a weighted sum of the red, green, and blue values, and use that as the value of gray. The first axis is the x, the second is y, and the third is the color components (red, green, blue). Use the weights 0.2126, 0.7152, and 0.0722 for red, green, and blue, respectively. These weights are so because the human eye is most sensitive to green color and least sensitive to blue color.

In the main function you can, for example, use the provided image `src/painting.png`. Display the grayscale image with the `plt.imshow` function. You may have to call the function `plt.gray` to set the color palette (colormap) to gray. (See `help(plt.colormaps)` for more information about colormaps.)

Part 2.

Write functions `to_red`, `to_green`, and `to_blue` that get a three dimensional array as a parameter and return a three dimensional arrays. For instance, the function `to_red` should zero out the green and blue color components and return the result. In the `main` function create a figure with three subfigures: the top one should be the red image, the middle one the green image, and the bottom one the blue image.

Exercise 12 (radial fade)

Make program that does fading of an image as earlier, except now not in horizontal direction but in radial direction. As we move away from the centre of the image, the pixels fade to black.

Part1.

Write function `center` that returns coordinate pair (`center_y`, `center_x`) of the image center. Note that these coordinates might not be integers.

Example of usage:

```
print(center(np.zeros((10, 11, 3))))  
(4.5, 5)
```

The function should work both for two and three dimensional images, that is grayscale and color images.

Write also function `radial_distance` that returns for image with width `w` and height `h` an array with shape `(h,w)`, where the number at index `(i,j)` gives the euclidean distance from the point `(i,j)` to the center of the image.

Part 2.

Create function `scale(a, tmin=0.0, tmax=1.0)` that returns a copy of the array `a` with its elements scaled to be in the range `[tmin, tmax]`.

Using the functions `radial_distance` and `scale` write function `radial_mask` that takes an image as a parameter and returns an array

with same height and width filled with values between 0.0 and 1.0. Do this using the `scale` function. To make the resulting array values near the center of array to be close to 1 and closer to the edges of the array are values closer to be 0, subtract the previous array from 1.

Write also function `radial_fade` that returns the image multiplied by its radial mask.

Test your functions in the `main` function, which should create, using `matplotlib`, a figure that has three subfigures stacked vertically. On top the original `painting.png`, in the middle the mask, and on the bottom the faded image.

Finding clusters in an image

Let's first generate some data:

```
n=5
l=256
im = np.zeros((l,l))
np.random.seed(0)
points = np.random.randint(0, l, (2, n**2)) # sample n*n pixels from the array im
im[points[0], points[1]] = 1
plt.imshow(im);
```



```
from scipy import ndimage
im2 = ndimage.gaussian_filter(im, sigma=1/(8.*n)) # blur the image a bit
plt.imshow(im2);
```



Let's try to find clusters from the above image:

```
mask = im2 > im2.mean() # mask those pixels whose intensity is above mean
label_im, nb_labels = ndimage.label(mask) # connected components form clusters
print(f"Number of clusters is {nb_labels}")
plt.imshow(label_im);
```

Number of clusters is 12



Although this method we used was very simple, it could still be used for example to automatically count number of birds or stars in an image. Of course, humans can do this easily, but when there are hundreds or

thousands of images, then it is better to use machines to do this mechanical work.

There is large number of applications of image processing of which we list only a few here:

- denoising
- deblurring
- image segmentation
- feature extraction
- zooming, rotating
- filtering

Additional libraries

There are several libraries written in Python that allow easy processing of images. Few examples of these:

- [pillow](#)
- [scikit-image](#)
- In Scipy there is the subpackage [ndimage](#) that also contains routines for processing images

You have reached the end of this section! Continue to the next section:

➔ 2. Matplotlib

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

1. Image processing

2. Matplotlib

3. NumPy (continues)

4. Pandas

This course is created by the Agile Education Research -research group [of the University of Helsinki](#).

[Credits and about the material.](#)



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI



MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI