TP n°1 : Introducing Image Processing and scikit-image

Objectives:

- **Jump into digital image structures and learn to process them!**
- **Extract data, transform and analyze images using NumPy and Scikit-image.**
- **With just a few lines of code, you will convert RGB images to grayscale, get data from them,**
- **obtain histograms containing very useful information,**
- **and separate objects from the background!**

I. Image processing and scikit-image

1. Visualizing images : RGB vs Grayscale

These images have been preloaded as coffee_image and coins_image from the scikit-image data module using:

```
coffee_image = data.coffee()
coins_image = data.coins()
```

Choose the right answer that best describes the main difference related to color and dimensional structure.

In the console, use the function shape() from NumPy, to obtain the image shape (Height, Width, Dimensions) and find out. NumPy is already imported as np.

Instructions :

Possible Answers

☐ Both have 3 channels for RGB-3 color representation.

☐ `coffee_image` has a shape of (303, 384), grayscale. And `coins_image` (400, 600, 3), RGB-3.

☐ `coins_image` has a shape of (303, 384), grayscale. And `coffee_image` (400, 600, 3), RGB-3.

☐ Both are grayscale, with single color dimension

2. RGB to grayscale

In this exercise you will load an image from scikit-image module data and make it grayscale, then compare both of them in the output.

Instructions :

- Import the `data` and `color` modules from Scikit image. The first module provides example images, and the second, color transformation functions.
- Load the `rocket` image.
- Convert the RGB-3 rocket image to grayscale.

II. Numpy for image

1. Flipping out

As a prank, someone has turned an image from a photo album of a trip to Seville upside-down and back-to-front! Now, we need to straighten the image, by flipping it. Using the NumPy methods learned in the course, flip the image horizontally and vertically. Then display the corrected image using the show_image() function.

NumPy has to be imported as np.

Instructions

- Flip the image vertically.
- Now, flip the vertically-flipped image horizontally.
- Show the, now fixed, image.

2. Histograms

In this exercise, you will analyze the amount of red in the image. To do this, the histogram of the red channel will be computed for the image :
Extracting information from images is a fundamental part of image enhancement. This way you can balance the red and blue to make the image look colder or warmer.

You will use hist() to display the 256 different intensities of the red color. And ravel() to make these color values an array of one flat dimension.

Matplotlib is preloaded as plt and Numpy as np.

Remember that if we want to obtain the green color of an image we would do the following:

```
green = image[:, :, 1]
```

instructions :

- Obtain the red channel using slicing.
- Plot the histogram and bins in a range of 256. Don't forget `.ravel()` for the color channel.

III. Thresholding

1. Apply global thresholding

In this exercise, you'll transform a photograph to binary so you can separate the foreground from the background.
To do so, you need to import the required modules, load the image, obtain the optimal thresh value using `threshold_otsu()` and apply it to the image.
Remember we have to turn colored images to grayscale. For that we will use the `rgb2gray()` function.

Instructions

- Import the otsu threshold function.
- Turn the image to grayscale.
- Obtain the optimal threshold value of the image.
- Apply thresholding to the image.

2. When the background isn't that obvious

Sometimes, it isn't that obvious to identify the background. If the image background is relatively uniform, then you can use a global threshold value as we practiced before, using `threshold_otsu()`. However, if there's uneven background illumination, adaptive thresholding `threshold_local()` (a.k.a. local thresholding) may produce better results.

In this exercise, you will compare both types of thresholding methods (global and local), to find the optimal way to obtain the binary image we need.

Insructions

- Import the otsu threshold function, obtain the optimal global thresh value of the image, and apply global thresholding.
- Import the local threshold function, set block size to 35, obtain the local thresh value, and apply local thresholding.

3. Trying others methods

Not being sure about what thresholding method to use isn't a problem. In fact, scikit-image provides us with a function to check multiple methods and see for ourselves what the best option is. It returns a figure comparing the outputs of different **global** thresholding methods.

You will apply this function to this image, `matplotlib.pyplot` has been loaded as `plt`. Remember that you can use `try_all_threshold()` to try multiple global algorithms.

Intructions

- Import the try all function.
- Import the rgb to gray convertor function.
- Turn the fruits image to grayscale.
- Use the try all method on the resulting grayscale image.

4. Apply the appropriate thresholding

In this exercise, you will decide what type of thresholding is best used to binarize an image of knitting and craft tools. In doing so, you will be able to see the shapes of the objects, from paper hearts to scissors more clearly.

What type of thresholding would you use judging by the characteristics of the image? Is the background illumination and intensity even or uneven?

Instructions :

- Import the appropriate thresholding and `rgb2gray()` functions.
- Turn the image to grayscale.
- Obtain the optimal thresh.
- Obtain the binary image by applying thresholding.