Intensive Course on Image Processing

Python tools for your project

First run the following command in the terminal:

```
export PATH=/cal/softs/anaconda/anaconda3/bin:$PATH
```

You can then go in the directory where your python files are located using the `cd` command in the terminal :

```
cd /<path_of_directory>
```

When you are in this directory, you can launch the python editor and compiler Spyder in your terminal by typing :

```
spyder utils.py
```

You can either use the functions that exist in the Scikit-image library, or use some functions that have already been put on your account. However, make sure to import those functions at the beginning of each python script, by writing the following lines at the top of your .py file. Here is a suggested header for your file :

```
import numpy as np
import skimage
from skimage import io as skio
import matplotlib.pyplot as plt
from utils import *
```

In this document, you have some information on different functions that can be useful for your project. They are grouped according to the different subjects you have seen in the courses. Please refer to the utils.py file for already coded functions.

If needed, you may change your keyboard from azerty to qwerty typing the following command in a terminal: `setxkbmap us`

The following link gives you course slides or articles useful for the project:

`http://perso.telecom-paristech.fr/~tupin/ATHENS/athens.html`

1 Histogram, Fourier domain, linear filtering

1.1 Image visualization

1.1.1 Image visualization in color

To load the colored image as a matrix for Python, use the function `skio.imread` from the `skimage.io` library, with the synthax: `name=skio.imread('nameimage.tif');` For instance here, `ima=skio.imread('images/monkey.tif')`.
To visualize the image in GIMP, you may use the `viewimage_color` function defined

in the `utils.py` file : `viewimage_color(ima)`.
If you want to visualize your images directly in the Spyder window, you can also use the following commands of the `matplotlib.pyplot` library:

```
plt.imshow(ima)
plt.show()
```

1.1.2 Image visualization in greyscale

To load the image as a matrix for Python, use the function `skio.imread` from the `skimage` library, with the synthax:
`name=skio.imread('nameimage.tif', as_gray = True)`
For instance here, `ima=skio.imread('images/monkey.tif', as_gray = True)`.
To obtain a grey image with 256 levels in GIMP, you can use the function `viewimage` defined in the same `utils.py` file : `viewimage(ima)`.

If you want to visualize your images directly in the Spyder window, you can also use the following commands of the `matplotlib.pyplot` library:

```
plt.imshow(ima)
plt.show()
```

If you want to know the grey-level of a pixel, select the "Pointer" tool while visualizing the image with GIMP. This tool can be found in the image menu : Windows / Dockable Dialogs / Pointer. This cursor indicates the spatial coordinates and the grey-level are indicated.

1.2 Histogram visualization

The histogram visualization can be done using the command `plt.hist`. To consider the whole image and to have 256 bins for the histogram, you have to use the following command :

```
plt.hist(ima.reshape((-1,)),bins=255)
plt.show()
```

1.3 Fourier domain

To compute the 2D Fourier transform of an image you can use `np.fft.fft2`. To have display the Fourier transform centered on the (0,0) component, we recommend to use the `np.fft.fftshift` command applied to the Fourier transform. For instance, you can try :

```
fft = np.fft.fft2(ima)
shiffted_fft = np.fft.fftshift(fft)
```

Do not forget that the Fourier transform is a set of complex numbers and use `np.abs` to obtain the modulus and use `plt.imshow` to have a good display. Why is the log function also usefull for the display?

```
modfft=np.log(np.abs(np.fft.fftshift(np.fft.fft2(ima)))+0.1);
plt.imshow(modfft)
plt.show()
```

If you want to have a look to the phase of an image, you can use the following commands.

```
phifft=np.angle(fftshift(fft2(ima)));
plt.imshow(phifft)
plt.show()
```

1.4 Subsampling

To apply a direct sub-sampling on an image keeping one pixel of each 2x2 square in the original image, you can use the following commands:

```
nlines=np.shape(ima)[0]
ncolumns=np.shape(ima)[1]
imasubsampled=ima[0:nlines:2,0:ncolumns:2];
```

1.5 Linear filtering

1.5.1 Low-pass filtering

Let us build a mean filter on a $5\times$ 5 window.

```
hh = (1./25)*np.ones((5,5), dtype = np.float32)
plt.plot(np.log(np.abs(np.fft.fftshift(np.fft.fft2(hh)))))
plt.show()
```

You can use `scipy.signal.convolve2d` to do the filtering and apply this kernel on an image. You can also filter in the Fourier domain with a Gaussian filter using the function `skimage.filters.gaussian`. For instance let's filter the image `ima` with a gaussian filter of standard deviation $\sigma = 2$:

```
gauss_ima = skimage.filters.gaussian(ima,sigma = 2)
```

2 Radiometry and colorimetry

2.1 Image display and histogram manipulation

This part is dedicated to visualization and histogram manipulation. First, use the image `spot.tif`.

Display it with the `viewimage` function to visualize it in greyscale.

Have a look to the histogram of the image to explain the dark appearance.

We have seen during the course how it is possible to give to an image u the histogram of an image v:

```
ushape=u.shape
uligne=u.reshape((-1,))
vligne=v.reshape((-1,))
ind=np.argsort(uligne)
unew=np.zeros(uligne.shape,uligne.dtype)
unew[ind]=np.sort(vligne)
unew=unew.reshape(ushape)
```

Take images `vue1.tif` and `vue2.tif` and apply this histogram transfert. Have a look to the absolute value of the difference between the two images.

We would like to use this transfert fonction to do an histogram egalization. This means that we would like to force the histogram of the `spot.tif` image to be uniform. How can you create an image with a uniform histogram? Apply the previous commands to do the transfert of the histogram. Compare histogram egalization to histogram stretching.

2.2 Color spaces

You can use the image `fleur.tif`, which is coded with 3 channels: red, green and blue.

The conversion to the HSV (Hue Saturation Value) space is given by the command `skimage.color.rgb2hsv` (inverse operation being `skimage.color.hsv2rgb`). It is then possible to display each channel separately: `imahsv[:,:,0]` selects the first channel of the HSV image `imahsv`, which corresponds to the hue value.

You can compare histogram stretching in the different spaces for color image visualization.

Other possible color space convertions are provided by the `skimage.color.rgb2lab` and `skimage.color.rgb2ycbcr` functions.

3 Mathematical morphology

The aim of this part is to get acquainted with mathematical morphology transformations. Applying different operations with several structuring elements on simple images will allow you understanding the actions, effects and properties of the operations, the role of the structuring element, as well as the need for appropriate preprocessing depending on the application at hand. First off, import the following library :

```
import skimage.morphology as morpho
```

3.1 Structuring element

The structuring element is defined by the function `strel` defined in the `utils.py` file. You can choose the shape: `'diamond', 'square', 'disk'`, and the radius. For instance `se = strell('square',1);`
creates a 3×3 square structuring element.

3.2 Binary morphology

For this part, you have to use a binary image (you can use `cafe.tif` or `cell.tif`). The image can be opened and visualized using:

```
cell=skio.imread('images/cell.tif')
viewimage(cell)
```

Apply the four basic operations:

```
ero=morpho.erosion(cell,se)
dil=morpho.dilation(cell,se)
open=morpho.opening(cell,se)
close=morpho.closing(cell,se)
```

with structuring elements of various shape and size. You can test the influence of the shape and size of the structuring element.

3.3 Grey-level morphology

In this part, the following images can be used:
— `bat512.tif`
— `anevrism.tif`
You can apply the four basic operations on a grey level image and study the influence of the size and shape of the structuring element. Here are some questions that can help you understand the course:
— How can you illustrate the iterativity property of the dilation ? What is the result of a dilation by a structuring element of size 2 followed by a dilation by a structuring element of size 3 (and same shape)?
— What is the result of an opening by a structuring element of size 2 followed by an opening by a structuring element of size 3 (and same shape)?
— How can you illustrate the idempotence of closing?
— Perform a top-hat transform (difference between the image and its opening). Comment the result depending on the choice of the structuring element.

3.4 Alternate sequential filters

You can perform alternate sequential filters, for instance by using:

```
se1=strell('disk',1)
```

```
se2=strell('disk',2)
se3=strell('disk',3)
se4=strell('disk',4)
se5=strell('disk',5)
fas1=morpho.closing(morpho.opening(im,se1),se1)
fas2=morpho.closing(morpho.opening(fas1,se2),se2)
fas3=morpho.closing(morpho.opening(fas2,se3),se3)
fas4=morpho.closing(morpho.opening(fas3,se4),se4)
fas5=morpho.closing(morpho.opening(fas4,se5),se5)
```

You can analyze the results according to the shape of the structuring element and the maximal size used in the filter. What kind of filtering can be expected from such operations?

3.5 Segmentation

You can apply a morphological gradient to the image `bat200.ima` (difference between dilation and erosion with a structuring element of radius1). What do you observe? You can try to threshold the gradient, for instance:

```
grad=morpho.dilation(im,se)-morpho.erosion(im,se)
temp = np.copy(grad)
temp[temp<100]=0
```

Why is it difficult to find an appropriate threshold value? You can apply the watershed algorithm to the gradient image:

```
local_mini = skimage.feature.peak_local_max(255-grad, indices=False)
markers = ndi.label(local_mini)[0]
wat=morpho.watershed(grad,markers,watershed_line=True)
```

What do you observe? You can select the watershed lines as the points with grey value 0 in the result and superimpose them to the original image for visualization purpose. You can also apply first a closing on the gradient image and then the watershed. Is the result better?

4 Useful Python commands

4.1 Matrix generation

4.1.1 Vector handling

To enter a **1-D vector** `v` type

```
v = np.array([7,2,5,6])
```

`v[0]` is the first element of vector `v`, `v[1]` is the second element and so forth. To access groups of elements use colon notation. For example, to access the three first elemenst of `v` type

```
>> v[0:2]
array([7, 2, 5])
```

4.1.2 Matrix handling

To enter a matrix `M` (or a 2-D vector) such as

$$\mathbf{M} = \begin{pmatrix} 2 & 5 & 8 \\ 23 & 12 & 7 \\ 1 & 8 & 15 \end{pmatrix}$$

type

```
M = np.array([[2, 5, 8], [23, 12, 7], [1, 8, 15]])
```

Use for example `M[2,1)` to access the element of the third row and second column, it's value is 8.

```
>> print(M[2,1])
8
```

Use for example `M[:,2]` to access all rows of the third column or `M(2,:)` to access all columns of the second row.

```
>> M[:,2]
    array([8,7,15])
>> M[1,:]
    array([23,12,7])
```

Use for example `M[0:1,1:2]` to access the second and third columns of the first and second row.

```
>> M[0:1,1:2]
    array([[5,8],[12,7]])
```

To access the **shape** of a matrix or vector use the command `np.shape(M)`. Not to be confused with `np.size(M)`, which returns the number of elements in the matrix.

To get the transposed matrix, you can either use `np.transpose(M)` or type `M.T` .

4.1.3 Matrix arithmetic

`M + N`	Matrix addition	`M*N`	Elementwise multiplication
`M - N`	Matrix subtraction	`M**n`	Elementwise exponentiation
`M.dot(N)`	Matrix multiplication	`M/N`	Elementwise division
`np.linalg.inv(M)`	Matrix inversion		

4.2 Control flow commands

Matlab supports the following control flow statements: `if, for, while` .

Type:

```
>> help()
>> help> if
```

to check the syntax of the `if` command. Idem for the other commands.

4.3 Open and visualize an image

`I = skio.imread('filename'))` reads a grayscale or color image from the file specified by the string 'filename'. If the file is not in the current directory specify the full pathname.

`plt.imshow(I)`

`plt.show()` Displays matrix I as an image.

4.4 Save an image

`skio.imsave('filename',I);` Writes the image I to the file specified by 'filename' in the format specified by the extension in the filename, e.g. `skio.imsave('test.png',I)`. The image I can be an M-by-N (grayscale image) or M-by-N-by-3 (color image) array.

4.5 Open data vector

`A = np.load('filename');` reads binary data from the specified .npy file and writes it into matrix A.

4.6 Save data vector

`np.save('filename',A);` Writes the elements of matrix A to the specified .npy file. The data are written in column order.

4.7 Labelling of connected components

`L = skimage.measure.label(BW)` Label connected components in 2-D binary image. Returns a matrix L, of the same size as BW, containing labels for the connected components in BW.

`STATS= skimage.measure.regionprops(BW);` Measure properties of image regions. Measures a set of properties for each connected component in the binary image BW. Properties examples are: area, bounding box, centroid. The complete list of properties is to be found on https://scikit-image.org/docs/dev/api/skimage.measure.html.

4.8 Logical operators

`C = np.logical_and(A,B)` Performs a logical AND between two binary images A and B.

`C = np.logical_or(A,B); C = A|B;` Performs a logical OR between two binary images A and B.

`B = np.logical_not(A);` Returns the complement image B of a binary image A.