MUEIB/MUNR
ETSEIB

Course on Medical Imaging
2023-2024
Q1

Session 3
Raster images

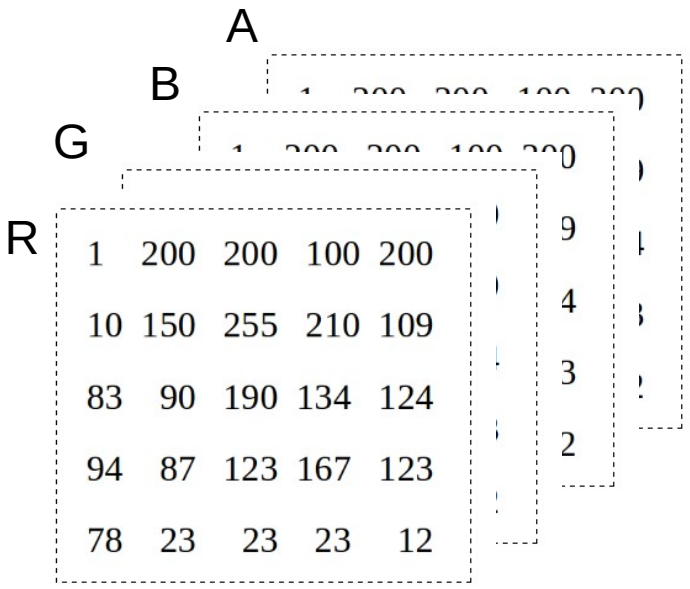
Dani Tost

Remember that last week we introduced the concept of raster images.
We learned how to create a raster image and render it with matplotlib.

Today we'll learn other ways to create images
We'll introduce the concept of memory requirements, file format and storage needs.

We'll introduce the cocept of histogram and cummulative distribution

Raster images



Matrix of data (1 to 4 channels)

Properties

- pixel resolution (number of pixels in width and height)
- spatial resolution (number of pixels per inch in width and height)
- number of channels (1= gray-level; 3 =RGB, 4 = RGBA)
- number of bytes per channel (in memory)

File storage

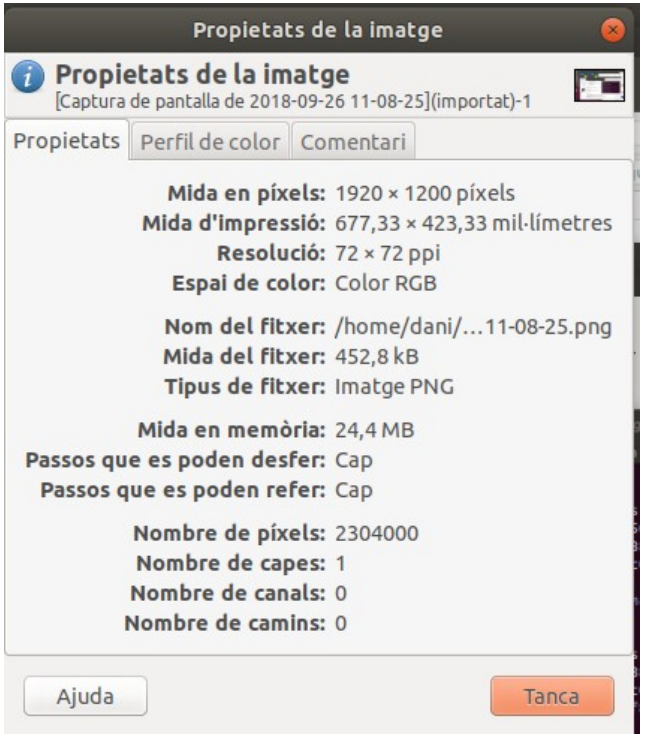
- Encoding: ascii/binary
- Format
- File size

Raster images resolution

- Pixel resolution: number of pixels in width and height

  Eg: an image of 2048 per 1536 pixels has a pixel resolution of 2048×1536 (3.1 megapixels)

- Spatial resolution: pixels per inch(ppi) 1 inch = 2.54 cm

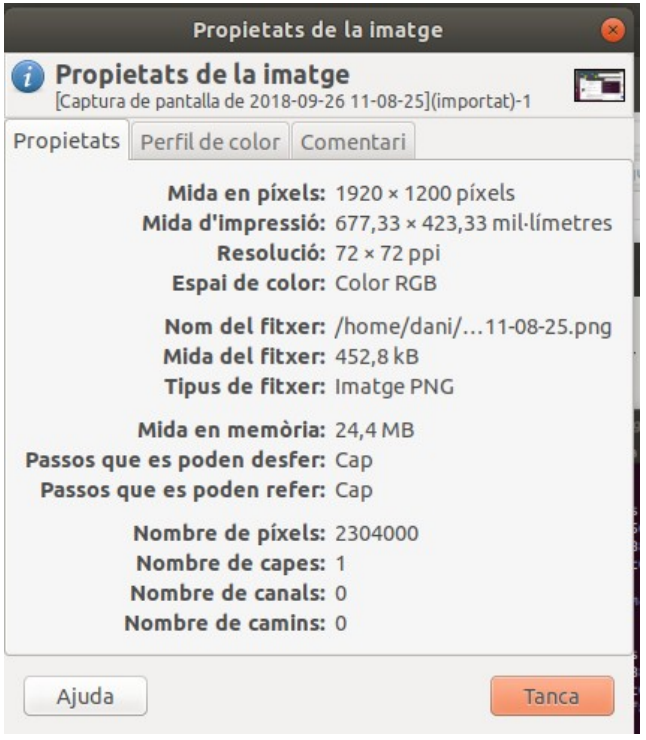


1920 = width
1200 = height

Raster images resolution

P1. What is the printing size in mm of the image?

P2. Is the number of pixels is coherent with the pixel resolution?



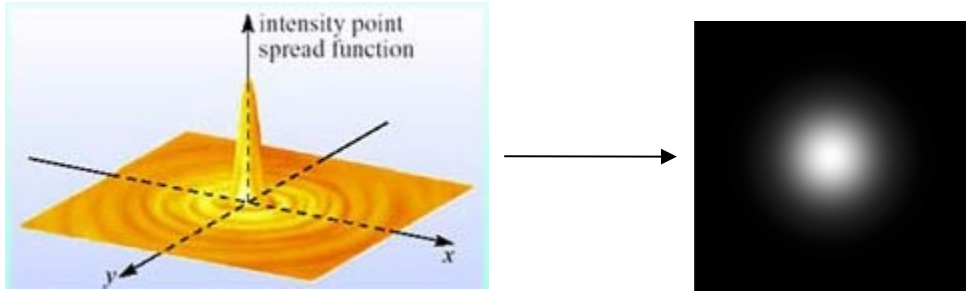
P3. Compute the resolution of an image printed at 90x90 ppi, knowing that its size in 254x254mm?

Raster images resolution

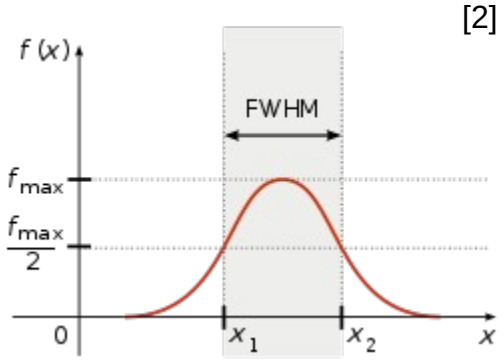
Other measures of the resolution take into account the imaging process and measure the accuracy of the digital representation of an object.

Point Spread Function (PDF) is the extended blob in an image corresponding to a bright spot. Its spread measures the amount of blurring of the image thus, the quality of the image (unit of measure Full Width at Half Maximum FWHM)

[1]



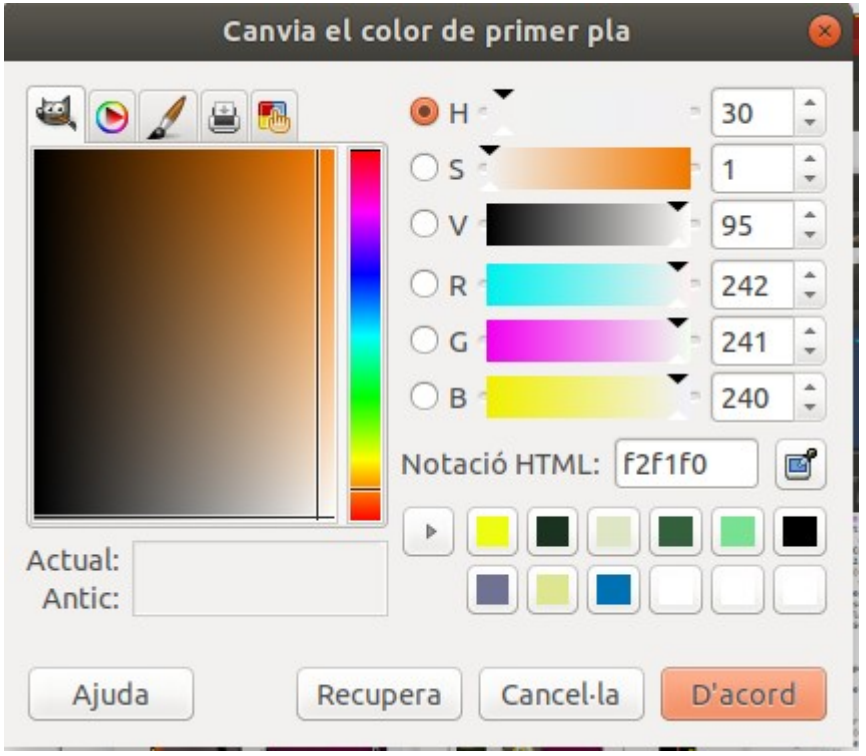
[2]



[1] Image from http://www.open.edu/openlearn/science-maths-technology/science/telescopes-and-spectrographs/content-section-1.5.5
[2] Image from wikipedia https://en.wikipedia.org/wiki/Full_width_at_half_maximum

Pixel value

- Value:
  - W/B: (binary images)
  - Gray scale (e.g. 0-255)
  - Color
    - Direct
    - An index to a color table
- Codification
    - RGB
    - YCM
    - RGBA
    - ....
- Channels RGBA (A = alpha channel).



Pixel value occupancy

Type of value and number of bytes needed to codify a channel in memory:

- Unsigned char: 1 byte (2^8) from 0 to 255 (uint8)
- Signed char: 1 byte (2^8) from -127 to 128 (int8)
- 2 bytes unsigned int: (2^{16}) from 0 to 65536 (uint16)
- 2 bytes signed int: (2^{16}) from -32767 to 32768 (int16)

Occupancy in memory: **npixels*nchannels*nbytes_per_channel**

e.g.: An RGB image of 512x340 pixels with 1 byte per pixel

512*340*3*1 = 522240 bytes = 522 KB (decimal) = 510 KB (binary)

Recall: 1 Byte = 8 bits
1 KB = 2^{10} Bytes = 1024 bytes in basis 2
= 10^3 Bytes in basis 10

Image file formats

ASCII versus BINARY encoding

When we store images in files ...

- ASCII files are text files. They are readable for humans
- Binary files instead are not readable, they codify the values as they are (compression mechanisms at a side)

E.g. a 100x100 RGBA image of 1 byte per pixel stored as binary file without compression (only the data) would occupy: 100*100*4 bytes = 40000 = 40 KB

The same file stored in ASCII would occupy between

100*100*4*2 and 100*100*4*4 bytes because a value between 0 and 255 occupies between 2 and 4 text characters

Image file formats

ASCII versus BINARY encoding

- Edit a file with a text editor (it will be **ascii**). Write inside the number 2. Check the size of the file
- Write the number 22. Check the size again. Do you understand it?
- What is the expected size of a binary file fully made of 100 times the number 23? What is the expected size of the file in ASCII supposing that there will be a white space between each number and a single line?

```
$ ls -ll number.txt
-rw-rw-r-- 1 dani dani 300 sep 30 16:04 number.txt

$ ls -ll number.bin
-rw-rw-r-- 1 dani dani 100 sep 30 16:17 number.bin
```

Image file formats

ASCII versus BINARY encoding

- In a python3 interpreter, try to create two binary files using the function `bytearray`. One file with the integer 2 and another with the integer 222.

```
>>> with open('ab.bin', 'wb') as f:
...         bytes = bytearray([222])
...         f.write(bytes)
...
```

- Check the size of the files. Do you understand why?

```
ls -ll *.bin
-rw-r--r-- 1 dani dani 1 d'oct  2 08:36 ab2.bin
-rw-r--r-- 1 dani dani 1 d'oct  2 08:31 ab.bin
```

ASCII versus binary encoding

Numpy provides to us the method `tofile` that saves into a file the contents of the array. Let's try to do it. We can save it in two ways: **Ascii or binary**. In the first case the image file will be readable by humans, but every single character will occupy one byte. In the second case we will use exactly one byte per channel but won't be able to read the file.

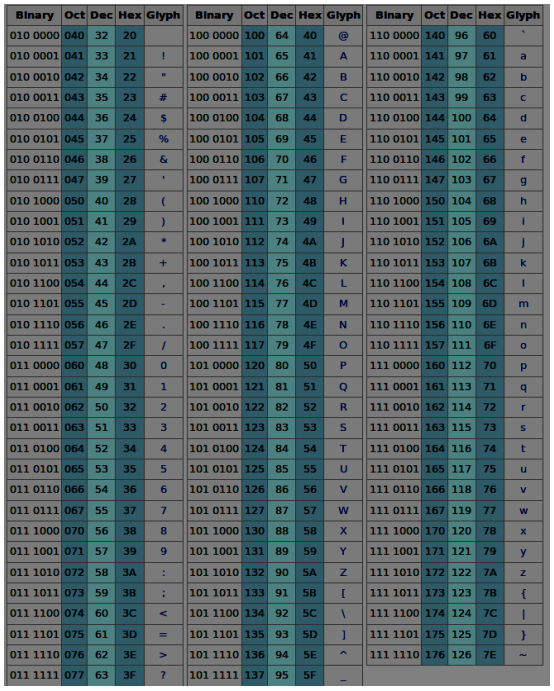
Try to save with the two formats. Try to read with gimp. The format is unknown. How would gimp know which is the width and which is the height? We need a **header**.

Finally, let us suppose that we have an image has a pixel resolution of 500x300 and it is uniformly filled with the RGB value (255, 255, 0). It seems rather insane to store 500x300x(255, 255, 0). Why can't we just store something like: "255, 255, 0, 150000"? This would be a compressed, lossless representation of the image. We can use **compression schemes.**

Image formats can be ascii or binary, compressed or not and if compressed lossless or lossy. They have a header that allows to interpret the array of data.

Image file formats

- Structure
  - Header + metadata + values
- Codification: ASCII /binary
  - ASCII: ascii chars
  - Binary: codified values
    - 8-bit unsigned integer
    - 16-bit signed integer (converted to unsigned by adding 32,768)
    - 16-bit unsigned integer
    - 32-bit signed integer (converted to float)
    - 32-bit floating-point
    - 24-bit RGB color (interleaved)
    - 24-bit RGB color (planar)
    - 32-bit ARGB color
    - 1-bit Bitmap (converted to 8-bit)



Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20		100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i
010 1010	052	42	2A	*	100 1010	112	74	4A	J	110 1010	152	106	6A	j
010 1011	053	43	2B	+	100 1011	113	75	4B	K	110 1011	153	107	6B	k
010 1100	054	44	2C	,	100 1100	114	76	4C	L	110 1100	154	108	6C	l
010 1101	055	45	2D	-	100 1101	115	77	4D	M	110 1101	155	109	6D	m
010 1110	056	46	2E	.	100 1110	116	78	4E	N	110 1110	156	110	6E	n
010 1111	057	47	2F	/	100 1111	117	79	4F	O	110 1111	157	111	6F	o
011 0000	060	48	30	0	101 0000	120	80	50	P	111 0000	160	112	70	p
011 0001	061	49	31	1	101 0001	121	81	51	Q	111 0001	161	113	71	q
011 0010	062	50	32	2	101 0010	122	82	52	R	111 0010	162	114	72	r
011 0011	063	51	33	3	101 0011	123	83	53	S	111 0011	163	115	73	s
011 0100	064	52	34	4	101 0100	124	84	54	T	111 0100	164	116	74	t
011 0101	065	53	35	5	101 0101	125	85	55	U	111 0101	165	117	75	u
011 0110	066	54	36	6	101 0110	126	86	56	V	111 0110	166	118	76	v
011 0111	067	55	37	7	101 0111	127	87	57	W	111 0111	167	119	77	w
011 1000	070	56	38	8	101 1000	130	88	58	X	111 1000	170	120	78	x
011 1001	071	57	39	9	101 1001	131	89	59	Y	111 1001	171	121	79	y
011 1010	072	58	3A	:	101 1010	132	90	5A	Z	111 1010	172	122	7A	z
011 1011	073	59	3B	;	101 1011	133	91	5B	[	111 1011	173	123	7B	{
011 1100	074	60	3C	<	101 1100	134	92	5C	\	111 1100	174	124	7C	|
011 1101	075	61	3D	=	101 1101	135	93	5D	]	111 1101	175	125	7D	}
011 1110	076	62	3E	>	101 1110	136	94	5E	^	111 1110	176	126	7E	~
011 1111	077	63	3F	?	101 1111	137	95	5F	_					

Raster image file formats

Diversity of formats

- **pnm** portable anymap format
  - ppm, pgm, pbm
  - (http://en.wikipedia.org/wiki/Netpbm_format)
- **jpg (jpeg)**: from Joint Photographic Experts Group), compressed lossy. For 8-bits gray level or 24 bits RGB images
- **tiff:** compressed, lossy or lossless. For 24 or 48 bits RGB images
- **giff:** compressed, lossless, supports animation. Only 1 8-bits channel (index to a color palette).
- **png:** compressed, lossless. 8 or 16 bits per channel + alpha channel)
- **svg** (Scalar Vector Graphics), eps, ai: for vectorial images

Format

Format for medical images

- Proprietary formats

  http://www.dclunie.com/medical-image-faq/html/index.html

- Standard https://www.dicomstandard.org

  DICOM: Digital Imaging and Communications in Medicine

  (previously ACR-NEMA *)

**ACR: American College of Radiologists*
NEMA: National Electrical Manufacturers Association

Format

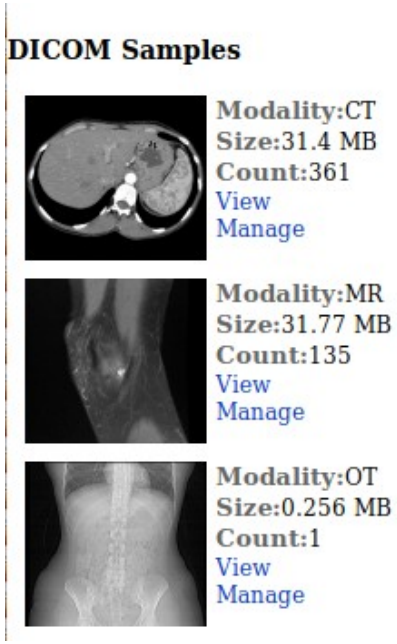
Digital Imaging and Communications in Medicine (DICOM)

http://dicom.nema.org/

A Protocol to archive, capture and exchange image data in picture Archiving and Communication Systems (PACS)

Storage of pixel data information about the imaging procedure and the patient: {<attribute, value>}

One image o a series of slices, even time-varying image series



Many viewers of dicom images exist:

Haak D, Page C-E, Deserno TM. A Survey of DICOM Viewer Software to Integrate Clinical Research and Medical Imaging. Journal of Digital Imaging. 2016;29(2):206-215. doi:10.1007/s10278-015-9833-1. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4788610/

From https://www.dicomlibrary.com/

https://www.dicomstandard.org/

scikit-image

A python library to process images with many modules and functionalities

https://scikit-image.org/

Scikit-image- submodule io

Download from Atenea the file `data.zip`. In a terminal, using skimage.o.imread, try to read all the images. Analyze the shape and dtype of the array.

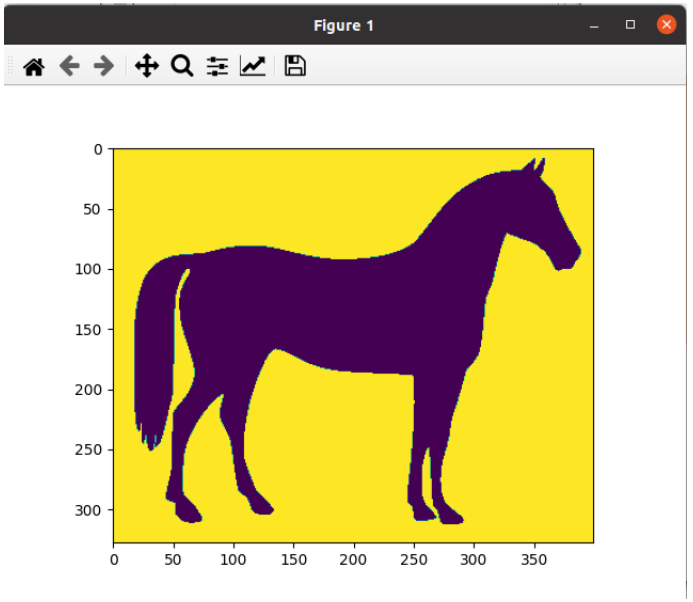
```
>>> import skimage
>>> from skimage import io
>>> a1 = io.imread('gray.ppm')
>>> a1.shape    #actually it is an RGB image with all 3
channels with the same value!
(400, 500, 3)
>>> a1.dtype
dtype('uint8'9
>>> a2 = io.imread('rgb.jpg')
>>> a2.shape
(400, 500, 3)
>>> a3 = io.imread('rgba.png')
>>> a3.shape
(400, 500, 4)
```

https://scikit-image.org/docs/dev/api/skimage.io.html

Scikit-image submodule data

Explore the module data and create procedural images.

```
>>> from skimage import data
>>> im = data.horse()
>>> import matplotlib.pyplot as plt
>>> fig, sfig = plt.subplots()
>>> sfig.imshow(im)
<matplotlib.image.AxesImage object at 0x7feba7445640>
>>> plt.show()
```

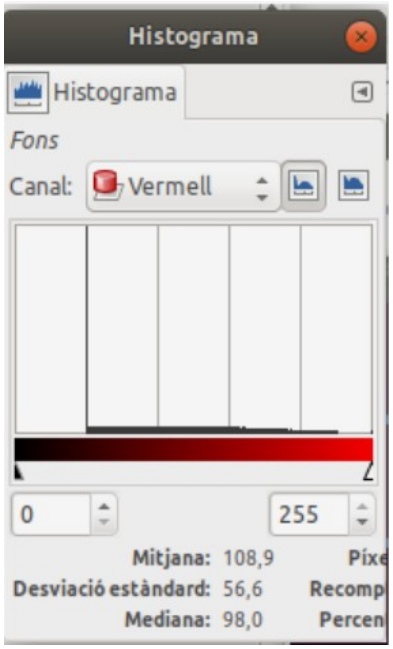


Explore others

Histogram and cumulative distribution function

Histogram

A histogram of an image is the number of counts (pixels) per each value (or group of values -*bin*-) of the image.



For each value v of a channel c

$$h(v) = \frac{Numberofpixelofchannelcwithvaluev}{Numberofpixelsoftheimage}$$

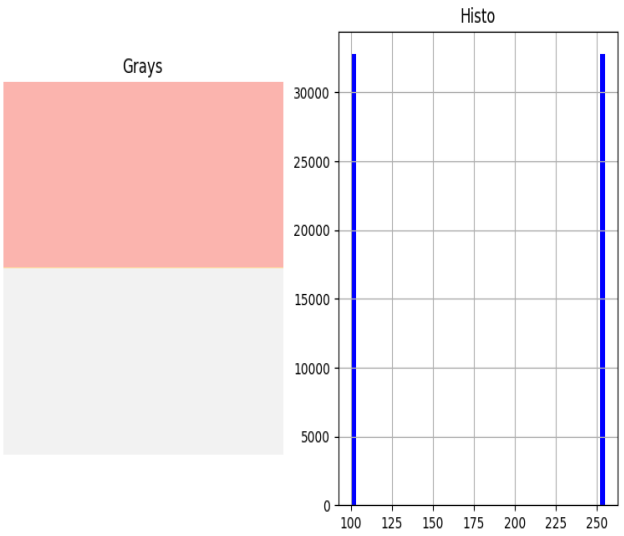
We can also compute the histogram of the gray level of an RGB image

Histogram

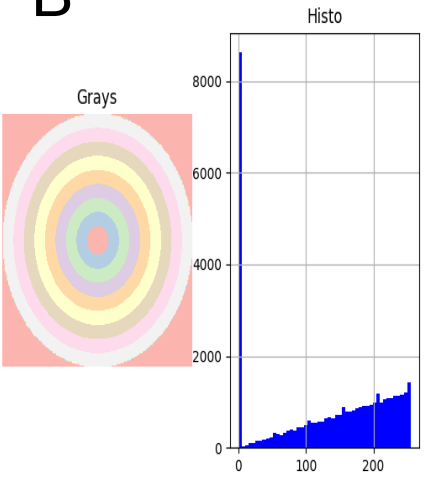
The histogram is important to understand which colors or gray level are present (and therefore which anaotomical or functional structures).

Example: Observe the figures below. Do you understand their histograms? Do you understand the colors? Could you create these images?

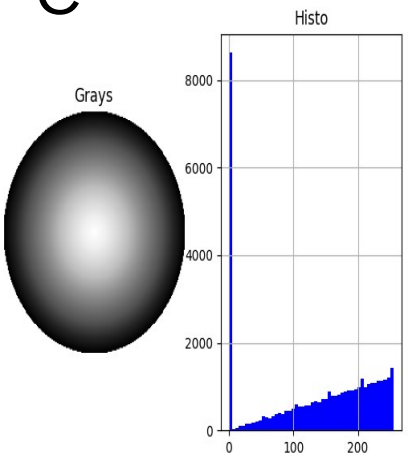
A



B

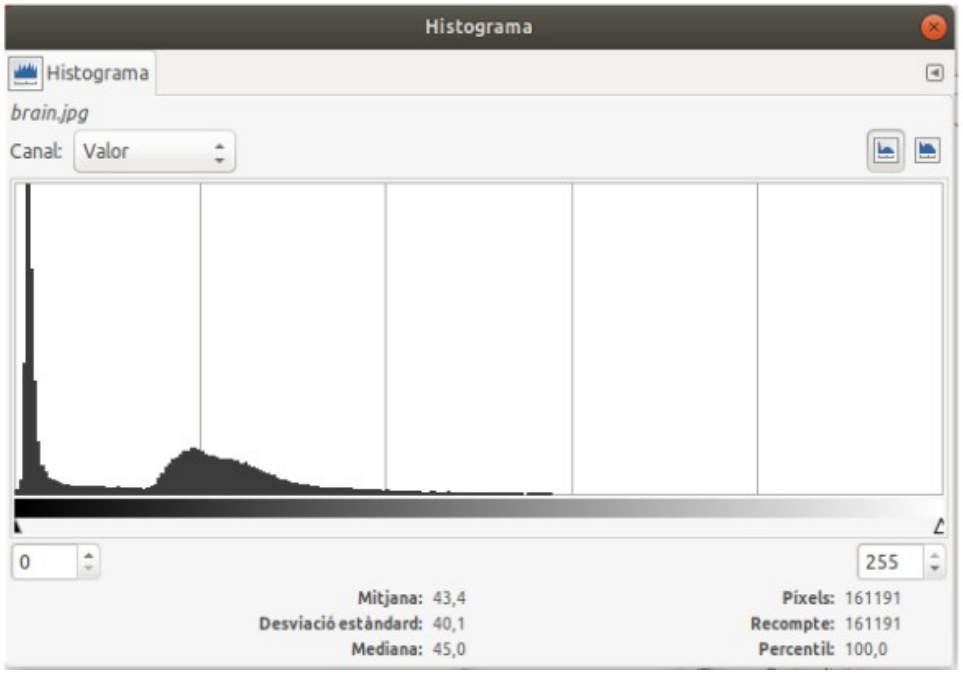


C

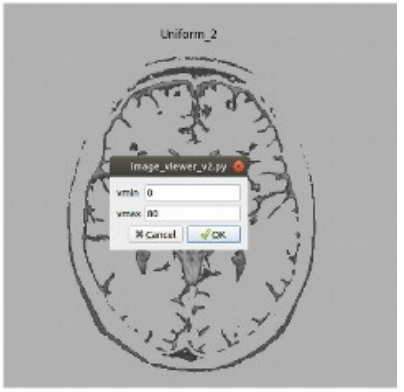


Histogram

The histogram is important to understand which colors or gray level are present (and therefore which structures)



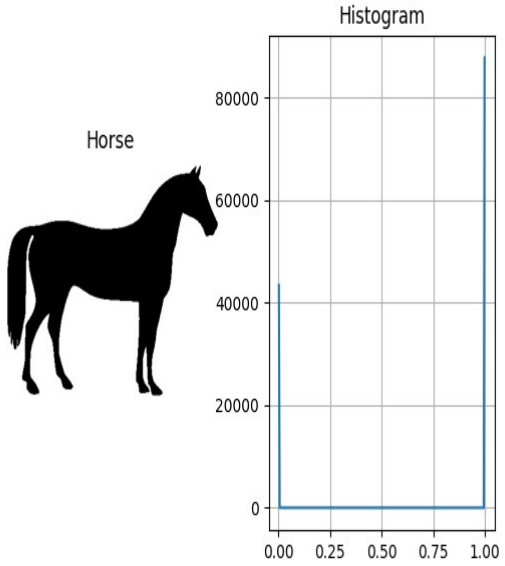
We can see in the histogram an important number of black pixels. They correspond to the background. It's a hint to remove the "external" part of the brain if we wish.



Histogram

There are different ways of computing histograms. Matplotlib provides one. We'll use instead the functions of scikit image in the module exposure.

Let's try with the image horse. It is a binary image (boolean values). Observe that histogram converts it automatically. The range of the histogram value is from 0.0 (value=0) to 1.0 (value=255). Do you understand its 2 peaks?



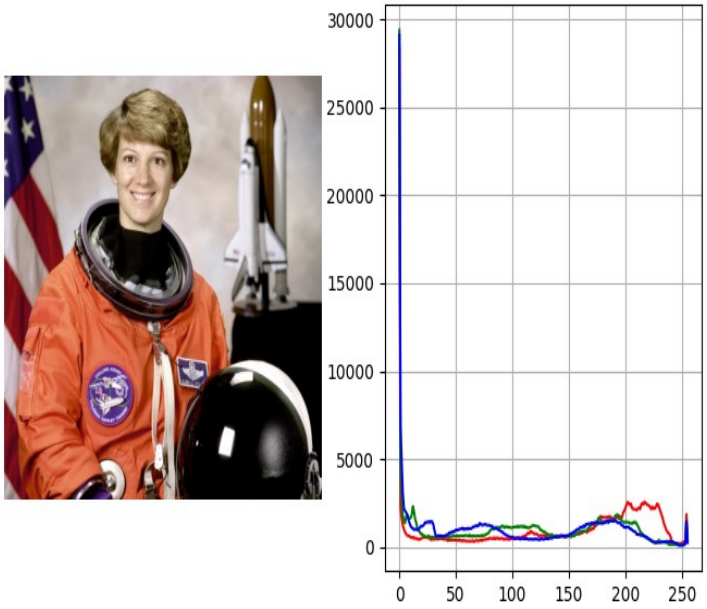

```
>>> from skimage import data, exposure
>>> im = data.horse()
>>> y, x = exposure.histogram(im)
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots(1, 2)
>>> ax[0].imshow(im, cmap=gray')
>>> ax[0].axis('off')
>>> ax[1].plot(x, y)
>>> ax[1].grid(True)
>>> plt.show()
```

https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.histogram

Histogram

There are different ways of computing histograms. Matplotlib provides one. We'll use instead the functions of scikit image in the module exposure.

Let's try with a color image.




```
>>> from skimage import data, exposure
>>> im = data.astronaut()
>>> yr, xr = exposure.histogram(im[:, :, 0])
>>> yg, xg = exposure.histogram(im[:, :, 1])
>>> yb, xb = exposure.histogram(im[:, :, 2])
>>> ax[1].plot(xr, yr, color='r')
>>> ax[1].plot(xg, yg, color='g')
>>> ax[1].plot(xb, yb, color='b')
>>> ax[1].grid(True)
>>> ax[0].imshow(im)
>>> ax[0].axis('off')
>>> plt.show()
```

https://scikit-image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.histogram

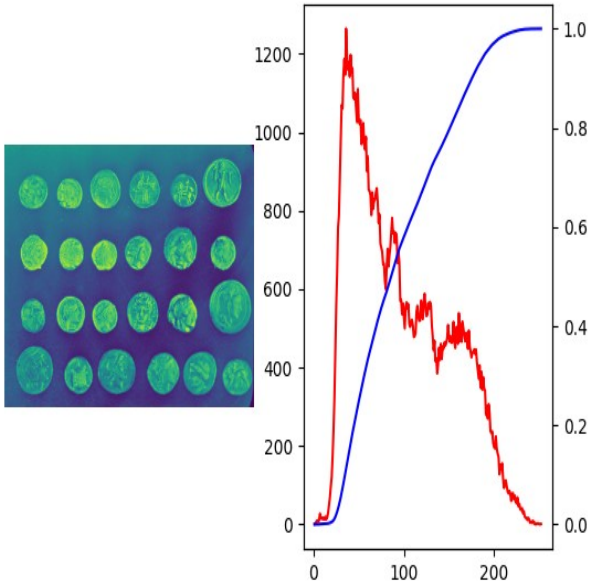
Cumulative distribution function

Cumulative distribution function

$$cdf(i) = \sum_{k=0}^{i} \left(\frac{n_i}{n}\right)$$

n = number of pixels of an image
i = an intensity level of the image 0<=i<=255
n_i = number of pixels with intensity i in the image

Equivalent to the cumulated sum of the normalized histogram




```
>>> from skimage import color, exposure, data
>>> import matplotlib.pyplot as plt
>>> coins = data.coins()
>>> histo = exposure.histogram(coins)
>>> cdf = exposure.cumulative_distribution(coins, 256)
>>> fig, ax = plt.subplots(1, 2)
>>> ax[0].imshow(im)
>>> ax[1].plot(histo[1], histo[0], 'red')
>>> cdf_plot = ax[1].twinx()
>>> cdf_plot.plot(cdf[1], cdf[0], 'blue')
>>> plt.show()
```

Homework

- Add a new class method to `RasImage` to read an image from a file and create an RBGA uint8 instance of the class. Convert to uint8 and RGBA if needed.

- Add a new class method to `RasImage` to create procedural images

- Investigate how to save an image in a file with a given extension. Add a new class method to `RasImage` to save it into a file of a given name

- Create a method to `RasImage` to compute its histogram in 1 channel (R,G,B or A) or in the gray version of the image

- Create a method to `RasImage` to compute its cummulative distribution functionin 1 channel (R,G,B or A) or in the gray version of the image

Homework

Class methods

read(filename)

Returns an instance to RasImage with the data_array read from the file, converted to RBGA uint 8. The name of the image is the basename of the file without extension

procedural

Returns an instance to RasImage with the data_array created with the astronaut, coins and horse skimages at least. The name of the image is the name of the procedure (‘horse’, coins, ‘astronaut’ ..)

Methods

save(extension):

Saves the RasImage in a file with its name and the given extension. It supports jpg and png at least

histo(channel=’G’)

Returns the 256-bins histogram of the given channel (RGBA or G)

cdf(channel=’G’)

Returns the cummulative distribution function of the given channel (RGBA or G)