Information and Communication Technologies — S6 — Work n°16

Digital images — Part 1

1 What is a digital image?

Let's take as example the following image (source: "Université d'Été Espace Éducation 2014")...

... let's zoom on the rocket...

... and let's zoom on the flags:

After two zooms, we arrived at the atomic level of the image: the pixels (big squares of uniform color). All digital images ("raster" images, not "vector" images) are made from pixels.

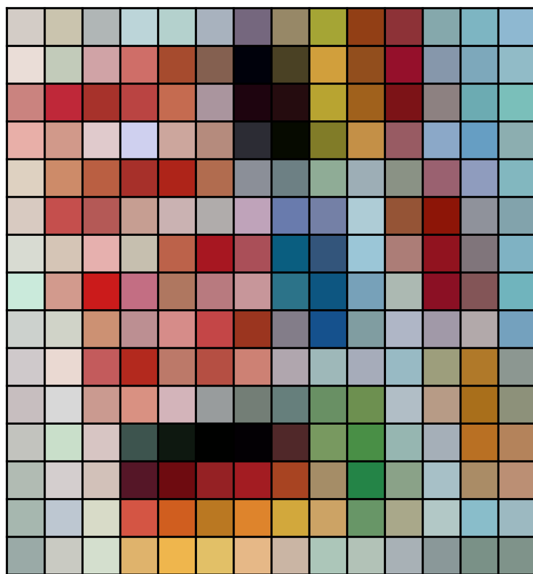
With our naked eye when we look at our screen, it's impossible to distinguish the different pixels of an image that fits our screen. Nevertheless, we can see them when we zoom inside the image: these little uniform squares that, together, form digital images.

We wrote that a pixel is the atomic level of the image. Indeed, as an atom is made of quarks, a pixel is made of smaller units: the colors. We will not go through all possible ways to encode a pixel, but we will study a basic encoding, thanks to the three primary colors red / green / blue (RGB encoding). We will thus work with additive colors, as opposed to the classical printers that work with subtractive colors (e.g. cyan / magenta / yellow / key (black)).

Each color is encoded with the quantity of red, green and blue it contains. If we put neither red, nor green, nor blue, we obtain black (with additive colors, if we put no color, it's black), and if we put a maximum of red, of green and of blue, we obtain white.

For "usual" images, the quantities of red, green and blue are integral numbers between 0 and 255. Those numbers can be put on 8 bits ($255 = 1111\ 1111_{(2)}$), or a byte. Thus, each pixel uses 3 bytes, which results in a "reasonable" weight for a digital image. If we use a digital camera with 20 millions pixels, one picture thus weights 60 megabytes to convey all the information. Luckily, compression algorithms exist (the .jpg format for instance, for *Joint Photographic Experts Group*, allows to compress image, with or without losing information): we thus get digital images that weight about 5 to 10 megabytes.

To get an idea of what this encoding is all about, here is the image of the flags on the rocket (width: 14 pixels, height: 15 pixels), with the associated red / green / blue encoding:



Full image

211	202	176	188	180	168	117	151	165	146	141	133	125	142
234	194	208	207	166	132	0	74	209	146	149	134	125	145
202	191	167	186	197	170	30	37	184	160	124	141	108	122
232	209	224	207	204	181	44	6	129	196	152	139	102	140
222	205	186	167	174	177	139	109	143	157	138	154	143	130
216	197	180	198	202	176	191	105	116	174	149	141	143	130
216	213	230	198	188	167	170	10	51	155	172	145	128	127
202	210	203	195	175	184	199	44	13	119	172	139	131	112
204	208	204	188	214	196	155	131	21	128	175	161	178	116
207	234	195	179	188	181	205	176	158	166	152	157	176	140
199	216	202	217	211	152	115	102	105	109	177	183	169	141
194	201	215	61	14	0	3	80	120	73	150	165	185	180
177	212	210	85	110	149	163	168	165	36	138	167	170	187
166	189	216	212	208	186	222	210	204	104	169	178	137	156
154	201	212	223	239	226	230	202	172	178	168	138	122	127

Values of the "Red" layer

204	196	182	213	209	178	103	136	165	63	50	168	181	184
221	203	163	110	75	96	1	65	159	78	16	151	168	187
131	40	50	68	107	149	4	12	164	97	19	129	171	191
175	153	202	208	166	139	44	10	124	144	91	168	158	174
209	139	95	48	36	108	143	128	172	174	146	97	156	183
202	79	89	158	178	172	163	123	128	204	84	21	146	163
219	197	176	191	98	23	79	94	85	198	125	19	117	178
234	154	27	110	119	122	150	115	86	161	185	16	85	180
209	211	145	143	140	70	53	125	81	157	182	153	169	161
201	217	91	41	121	79	129	166	184	172	186	158	121	151
190	216	154	145	180	156	126	127	144	144	190	155	111	145
195	223	197	84	24	1	0	40	153	143	182	175	112	131
187	206	193	22	11	33	28	68	141	132	162	192	140	143
183	199	219	85	94	120	132	168	163	150	168	200	189	185
170	202	223	179	182	192	184	181	198	194	177	152	145	147

Values of the "Green" layer

198	174	182	217	205	190	126	103	53	21	55	172	194	209
215	186	166	104	46	80	11	36	60	29	43	171	187	199
127	57	43	66	80	158	15	15	49	28	23	129	178	186
168	138	204	239	157	125	52	0	40	71	99	200	195	176
193	105	66	42	25	79	152	132	150	182	133	112	190	191
193	77	86	146	178	171	186	173	166	214	54	7	155	172
210	182	174	175	74	33	88	128	123	215	119	31	123	195
219	141	27	131	96	127	154	137	129	185	178	36	87	189
205	200	115	146	137	71	31	137	141	161	198	168	170	190
203	210	92	30	105	67	116	174	185	186	196	124	41	145
191	216	144	130	186	157	118	124	100	80	198	134	27	122
190	202	195	78	16	0	4	41	96	70	177	184	35	91
179	206	185	39	16	36	34	34	103	71	136	199	102	116
175	209	200	68	31	34	44	60	101	103	138	198	202	193
167	194	206	108	77	103	135	164	185	183	182	153	135	138

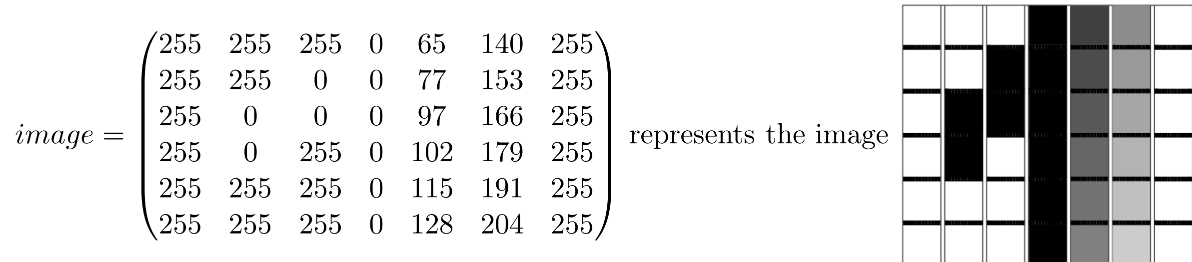
Values of the "Blue" layer

2 Image processing: grayscale

A black and white image (grayscale) is made of gray pixels, more or less dark. Each pixel has an associated number. As in the previous part, we will here use 0 for black and 255 for white. The portable graymap format (PGM) file format allows us to handle those kind of images, see for example https://en.wikipedia.org/wiki/Netpbm#File_formats. Thus, as the associated number is bigger, the pixel is lighter.

As an example, let's take the following two-dimensional array:

$$image = \begin{pmatrix} 255 & 255 & 255 & 0 & 65 & 140 & 255 \\ 255 & 255 & 0 & 0 & 77 & 153 & 255 \\ 255 & 0 & 0 & 0 & 97 & 166 & 255 \\ 255 & 0 & 255 & 0 & 102 & 179 & 255 \\ 255 & 255 & 255 & 0 & 115 & 191 & 255 \\ 255 & 255 & 255 & 0 & 128 & 204 & 255 \end{pmatrix}$$ represents the image



In this work we will manipulate black and white images, thus two-dimensional arrays. We will only focus on images of size 6×7 pixels, modeled by two-dimensional arrays of integers (indexes in $[0..5] \times [0..6]$), each integer representing the grayscale (between 0 and 255).

Please start by downloading the following file, that contains the array above, and a simple function to print two-dimensional arrays:

http://www.barsamian.am/2022-2023/S6ICTA/TP16_Images.py

1. To binarize an image, we must choose a threshold. For example if we choose the threshold 128: we will replace each pixel with value less than 128 with 0, and the others with 255. Write the algorithm of the function `binarize(array)` that takes an image as input, and returns another image as output, which is the binarized version of the input.

2. The purpose of the function given in Figure 1 is unknown. What value is returned by this function if we call it with the input *image* given as example before? Can you explain the role of this function?

Input:
array is an array of array of integers.

Variables:
i and j are two integers.
result is a real number.

Instructions of the function:

```
1   result ← 0
2   For i From 0 to 5
3       For j From 0 to 6
4           result ← result + array[i][j]
5       End For
6   End For
7   Return result/42
```

Figure 1: Function "mystery".

3. We now want to add more contrast to the image: a dark gray will be darker, a light gray will be lighter. For a given image, we determine a value g between 0 and 255 (the "grayscale" of this image). Then, for each pixel, if we note v the value:

   - if $v \leq g$, the new value is $v/2$
   - if $v > g$, the new value is $2v$ (except if this would exceed 255, in this case, the new value is 255)

   For example if an image has $g = 100$:

   - a pixel with value $v = 93$ ($93 < g$) is replaced by $93/2 = 46$
   - a pixel with value $v = 120$ ($120 > g$) is replaced by $2 \times 120 = 240$
   - a pixel with value $v = 200$ ($200 > g$) is replaced by 255, because $2 \times 200 = 400 > 255$

   Write the algorithm of a function `addContrast(image,g)` that takes as inputs an image and a grayscale, and outputs the new image with added contrast.

4. We now want to blur a little bit the image: in order to do this, we can for instance replace each pixel with the average (rounded down) of 3 pixels (when possible): the left neighbor, the right neighbor, and the current pixel.

   Write the algorithm of a function `blur(image)` that takes as input an image and outputs the new blurred image.

BONUS Modify the function `blur(image)` so that it computes the average of 5 pixels (when possible): in addition to the 3 pixels already taken in consideration, we also consider the upper neighbor and the lower neighbor.