2.3 Image enhancement

Image enhancement is the process of adjusting a digital image so the resultant one is more suitable for further image analysis (feature extraction, segmentation, etc.), in other words, **its goal is to improve the contrast and brightness of the image**.

There are three typical operations for enhancing images. We have already explored one of them in notebook *2.1 IP tools*: (linear) Look-Up Tables (LUTs). In this notebook we will play with other two:

- Non-linear look-up tables (Section 2.3.1).
- Histogram equalization (Section 2.3.2).
- Histogram specification (Section 2.3.3).

Also, some color-space conversions are going to be needed. If you are not familiar with the YCrCb color space, **Appendix 2: Color spaces** contains the information you need to know about it.

Problem context - Implementing enhancement techniques for an image editor tool

We have all tried an image editor tool, sometimes without even knowing it! For example, modern smartphones already include an application for applying filters to images, cut them, modify their contrast, brightness, color temperature, etc.

One example of open source tool is the GNU Image Manipulation Program (GIMP). Quoting some words from its website:

> GIMP is a cross-platform image editor available for GNU/Linux, OS X, Windows and more operating systems. It is free software, you can change its source code and distribute your changes. Whether you are a graphic designer, photographer, illustrator, or scientist, GIMP provides you with sophisticated tools to get your job done. You can further enhance your productivity with GIMP thanks to many customization options and 3rd party plugins.

In this case we were contacted by UMA for implementing two techniques to be included in their own image editor tool! Concretely, we were asked to develop and test two methods that are also part of GIMP: **gamma correction** and **equalize**

In [1]:
```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib
from ipywidgets import interactive, fixed, widgets
matplotlib.rcParams['figure.figsize'] = (20.0, 20.0)

images_path = './images/'

#To suppress MatplotlibDeprecationWarning when setting x/y axis limit to plot
import warnings
import matplotlib.cbook
warnings.filterwarnings("ignore",category=matplotlib.cbook.mplDeprecation)
```

2.3.1 Non-linear look-up tables

Gamma correction, or often simply **gamma**, is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems. In other words, it is the result of applying an (already defined) **non-linear LUT** in order to stretch or shrink image intensities.

In this way, the gamma LUT definition for grayscale images, where each pixel i takes values in the range $[0 \ldots 255]$, is:

$$LUT(i) = (\frac{i}{255})^{\gamma} * 255, \; \gamma > 0$$

The following images illustrate the application of gamma correction for different values of γ.
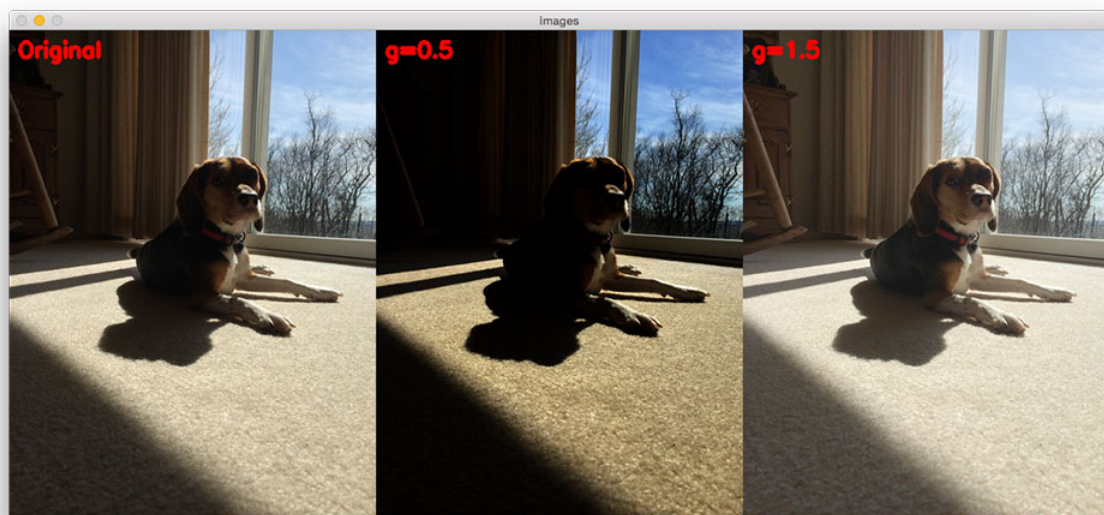
ASSIGNMENT 1: Applying non-linear LUTs

Your task is to develop the `lut_chart()` function, which takes as arguments the image to be enhanced and a gamma value for building the non-linear LUT. It will also display a chart containing the original image, the gamma-corrected one, the used LUT and the histogram of the resulting image.

As users from UMA will use color images, you will have to **implement it for color images**. This can be done by:

1. **transforming** a image in the BGR color space **to the YCrCb one**,
2. then, **applying gamma LUT only to first band** of the YCrCb space (that's because it contains pixel intensities and you can handle it like a gray image), and
3. finally, as matplotlib displays RGB images (if verbose is True), it should be **converted back**. Also, return the resultant image.

Interesting functions:

- `np.copy()` : method that returns a copy of the array provided as input.
- `cv2.LUT()` : *function that performs a look-up table transform of an array of arbitrary dimensions.*
- `plt.hist()` *function that computes and draws the histogram of an array.* `numpy.ravel()` *is a good helper here, since it converts a n-dimensional array into a flattened 1D array.*

In [2]:
```python
# ASSIGNMENT 1
# Implement a function that:
# -- converts the input image from the BGR to the YCrCb color space
# -- creates the gamma LUT
# -- applies the LUT to the original image
# -- displays in a 2x2 plot: the input image, the gamma-corrected one, the app
def lut_chart(image, gamma, verbose=False):
    """ Applies gamma correction to an image and shows the result.

        Args:
            image: Input image
            gamma: Gamma parameter
            verbose: Only show images if this is True

        Returns:
            out_image: Gamma image
    """

    #Transform image to YCrCb color space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb) #image is YCrCb
    out_image = np.copy(image) #out_image is YCrCb

    # Define gamma correction LUT
    lut = np.array([((i / 255.0) ** gamma) * 255 for i in np.arange(0, 256)])

    # Apply LUT to first band of the YCrCb image
    out_image[:,:,0] = cv2.LUT(out_image[:,:,0], lut)

    if verbose:
        # Plot used LUT
        plt.subplot(2,2,3)
        plt.title('LUT')
        plt.plot(range(256), lut)
        plt.subplot(2,2,3).set_xlim([0-20, 255+20]) #Warning removed from here
        plt.subplot(2,2,3).set_ylim([0-20, 255+20]) #Warning removed from here

        # Plot histogram of gray image after applying the LUT
        plt.subplot(2,2,4)
        plt.hist(np.ravel(out_image[:,:,0]), 256, (0, 255)) #256 bins from 0
        plt.title('Histogram')

        # Reconvert image to RGB
        image = cv2.cvtColor(image, cv2.COLOR_YCrCb2RGB) #image is RGB
        out_image = cv2.cvtColor(out_image, cv2.COLOR_YCrCb2RGB) #image is RGB

        # Show the initial image
        plt.subplot(2,2,1)
        plt.imshow(image)
        plt.title('Original image')

        # Show the resultant one
        plt.subplot(2,2,2)
        plt.imshow(out_image)
        plt.title('LUT applied')

    return out_image
```

You can use the next code to **test if results are correct**:

In [3]:
```python
image = np.array([[[10,60,20],[60,22,74],[72,132,2]],[[11,63,42],[36,122,27],
gamma = 2
print(lut_chart(image, gamma))
```

```
[[[  6 112 110]
  [  6 151 138]
  [ 29  68 120]]

 [[ 10 122 105]
  [ 27  87 101]
  [ 25  92 104]]

 [[  0 127 126]
  [  1 122 122]
  [  0 122 127]]]
```

Expected output:

```
[[[  6 112 110]
  [  6 151 138]
  [ 29  68 120]]

 [[ 10 122 105]
  [ 27  87 101]
  [ 25  92 104]]

 [[  0 127 126]
  [  1 122 122]
  [  0 122 127]]]
```

Thinking about it (1)

In the interactive code cell below, **you are asked to** explore how your new `lut_chart()` function works with `gamma_1.jpg` (an underexposed image) and `gamma_2.jpeg` (an overexposed image). Then, **answer the following question** (you can take a look at the LUT and the resulting histogram):

- What is happening when the *gamma value* is modified?

  When the gamma value is modified we are changing a parameter of the function we use to generate the LUT, so we are changing the entire LUT.

  If the gamma value decreases below 1, the exponential function we are using to generate the LUT gets concave shape, this means that pixels with lower intensity value change their value to a higher one. In practice this change shifts the histogram to the right and our image gets more brightness. The increase/decrease of the contrast depends on the original histogram, if the histogram was oriented to the right and we use a gamma value under 1 to move the histogram to the right the contrast decreases, otherwise it increases.

  If the gamma value increases above 1, the exponential function gets convex shape, this mean that majority of pixels are going to decrease their value. In practice this change moves the

histogram to the left and the image decreases its brightness. The increase/decrease of the contrast depends on the original histogram, if the histogram was oriented to the left and we use a gamma value higher than 1 to move the histogram to the left the contrast decreases, otherwise it increases.

Cuando el valor de gamma se modifica estamos cambiando un parámetro de la función que utilizamos para generar la LUT, así que en realidad estamos modificando la LUT entera.

Si el valor de gamma se vuelve menor que 1, la función exponencial que usamos para generar la LUT obtiene forma cóncava, lo que significa que los pixeles con menor valor de intensidad aumentan su intensidad. En la práctica este cambio desplaza el histograma a la derecha y nuestra imagen se vuelve más brillante. El aumento/decremento del contrasete depende del histograma original, si el histograma original estaba orientado a la derecha y utilizamos un valor de gamma menor que 1 para mover el histograma a la derecha el contraste se decrementa, en otro caso aumenta.

Si el valor de gamma es mayor que uno, la función exponencial obtiene forma convexa, esto significa que la mayoría de pixeles van a decrementar su valor. En la práctica este cambio mueve el histograma a la izquierda y la imagen se vuelve menos brillante. El aumento/decremento del contrasete depende del histograma original, si el histograma original estaba orientado a la izquierda y utilizamos un valor de gamma mayor que 1 para mover el histograma a la izquierda el contraste se decrementa, en otro caso aumenta.

In [4]:
```python
# Create widget object
gamma_widget = widgets.FloatSlider(value=1, min=0.1, max=5, step=0.2, descript

#Read image
image = cv2.imread(images_path + 'gamma_1.jpg',-1)

#Interact with your code!
interactive(lut_chart, image=fixed(image), gamma=gamma_widget, verbose=fixed('
```

In [5]:
```python
gamma_widget = widgets.FloatSlider(value=1, min=0.1, max=5, step=0.2, descript
image = cv2.imread(images_path + 'gamma_2.jpeg',-1)
interactive(lut_chart, image=fixed(image), gamma=gamma_widget, verbose=fixed('
```

## 2.3.2 Histogram equalization

Histogram equalization is an image processing technique used to improve contrast in images. It operates by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image so each possible pixel intensity appears the same number of times as every other value. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.
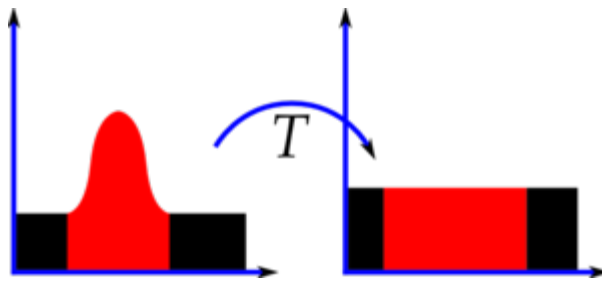
To put an example, the **equalize** command from GIMP applies histogram equalization. But... how is this equalization achieved?

- First it is calculated the PMF (probability mass function) of all the pixels in the image. Basically, this is a normalization of the histogram.

- Next step involves calculation of CDF (cumulative distributive function), producing the LUT for histogram equalization.

- Finally, the obtained LUT is applied.

The figure below shows an example of applying histogram equalization to an image.

Original image

LUT

Equalized image

Histogram of the original image

Histogram of the equalized image
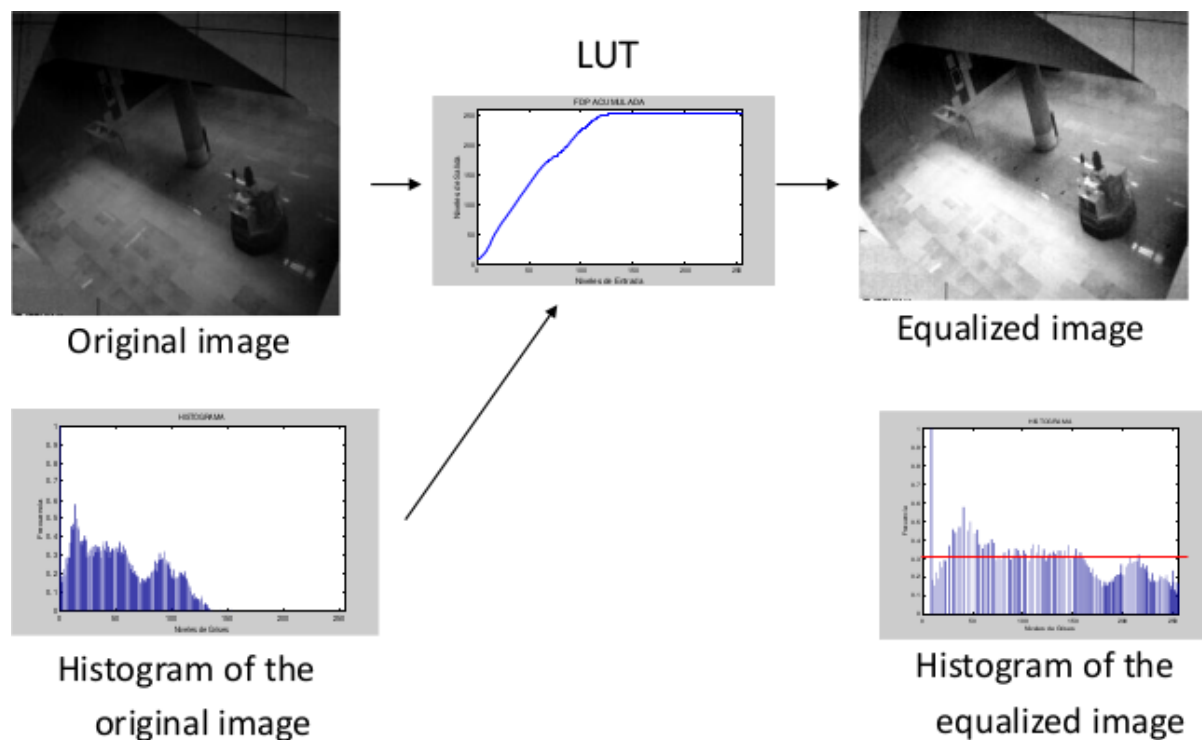
ASSIGNMENT 2: Equalizing the histogram!

Similarly to the previous exercise, **you are asked to** develop a function called `equalize_chart()` . This method takes a **color** image, and will display a plot containing:

- the original image,

- the equalized image,
- the original image histogram, and
- the equalized image histogram.

Tip: openCV implements histogram equalization in cv2.equalizeHist()

In [6]:

```python
# ASSIGNMENT 2
# Implement a function that:
# -- converts the input image from the BGR to the YCrCb color space
# -- applies the histogram equalization
# -- displays in a 2x2 plot: the input image, the equalized one, the original
def equalize_chart(image, verbose=False):
    """ Applies histogram equalization to an image and shows the result.

        Args:
            image: Input image
            verbose: Only show images if this is True

        Returns:
            out_image: Equalized histogram image
    """

    #Transform image to YCrCb color space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb) #image is YCrCb
    out_image = np.copy(image) #image is YCrCb

    # Apply histogram equalization to first band of the YCrCb image
    out_image[:,:,0] = cv2.equalizeHist(out_image[:,:,0])

    if verbose:

        # Plot histogram of gray image
        plt.subplot(2,2,3)
        plt.hist(np.ravel(image[:,:,0]), 256, (0, 255))
        plt.title('Original histogram')

        # Plot equalized histogram of the processed image
        plt.subplot(2,2,4)
        plt.hist(np.ravel(out_image[:,:,0]), 256, (0,255))
        plt.title('Equalized histogram')

        # Reconvert image to RGB
        image = cv2.cvtColor(image, cv2.COLOR_YCrCb2RGB) #image is RGB
        out_image = cv2.cvtColor(out_image, cv2.COLOR_YCrCb2RGB) #image is RGB

        # Show the initial image
        plt.subplot(2,2,1)
        plt.imshow(image)
        plt.title('Original image')

        # Show the resultant one
        plt.subplot(2,2,2)
        plt.imshow(out_image)
        plt.title('Equalized histogram image')

    return out_image
```

You can use the next code to **test if your results are correct**:

In [7]:

```
image = np.array([[[10,60,20],[60,22,74],[72,132,2]],[[11,63,42],[36,122,27],
print(equalize_chart(image))
```

```
[[[128 112 110]
  [128 151 138]
  [255  68 120]]

 [[159 122 105]
  [223  87 101]
  [191  92 104]]

 [[  0 127 126]
  [ 64 122 122]
  [ 32 122 127]]]
```

Expected output:

```
[[[128 112 110]
  [128 151 138]
  [255  68 120]]

 [[159 122 105]
  [223  87 101]
  [191  92 104]]

 [[  0 127 126]
  [ 64 122 122]
  [ 32 122 127]]]
```

Thinking about it (2)

We have developed our second image enhancement technique! Now try `equalize_chart()` with the `park.png` image in the code cell below. Then, **answer following questions**:

- What is the difference between the original histogram and the equalized one?

    The equalized histogram is expanded along the X-axis of the histogram, what means a higher contrast. Actually, the shape of both histograms is the same, but the second one is a "stretched" version of the first.

    El histograma ecualizado está expandido a lo largo del eje X del histograma, lo que significa un contraste mayor. De hecho, la forma de ambos histogramas es la misma, pero el segundo es una versión "estirada" del primero.

- Is the final histogram uniform? why?

    We have changed it to the most uniform histogram we could obtain. The problem with getting a totally uniform histogram is that cutting the histogram bars into several pieces is not allowed; this is due to bars represent a count of the image pixels with certain value. If we cut bars of some value we couldn't decide which pixels are going to continue with the same intensity and which aren't. So the unique way we have to change the shape an histogram is

multiplying the value of each pixel (better said: of each group of pixels with the same value)
by a number. That number is obtained by the LUT.

Lo hemos cambiado al histograma más uniforme que podemos obtener. El problema con
conseguir un histograma totalmente uniforme es que cortar las barras del histograma en
varias piezas no está permitido; esto se debe a que las barras representan un recuento de
pixeles de la imagen con cierto valor de intensidad. Si cortamos barras del mismo valor no
podríamos decidir qué pixeles van a continuar con la misma intensidad y cuáles no. Por lo
tanto, la única forma que tenemos de cambiar la forma de un histograma es multiplicando
el valor de cada pixel (mejor dicho: de cada grupo de pixeles con el mismo valor) por un
número. Este número lo proporciona la LUT.

```
In [8]:   # Read image
          image = cv2.imread(images_path + 'park.png',-1)

          # Equalize its histogram
          interactive(equalize_chart, image=fixed(image), verbose=fixed(True))
```

2.3.3 Histogram specification

Histogram specification is the transformation of an image so that its histogram matches a
specified one. In fact, the histogram equalization method is a special case in which the specified
histogram is uniformly distributed.

Original Image   Specified histogram   Enhanced Image

Histogram of the original   Histogram of the enhaced
image   image
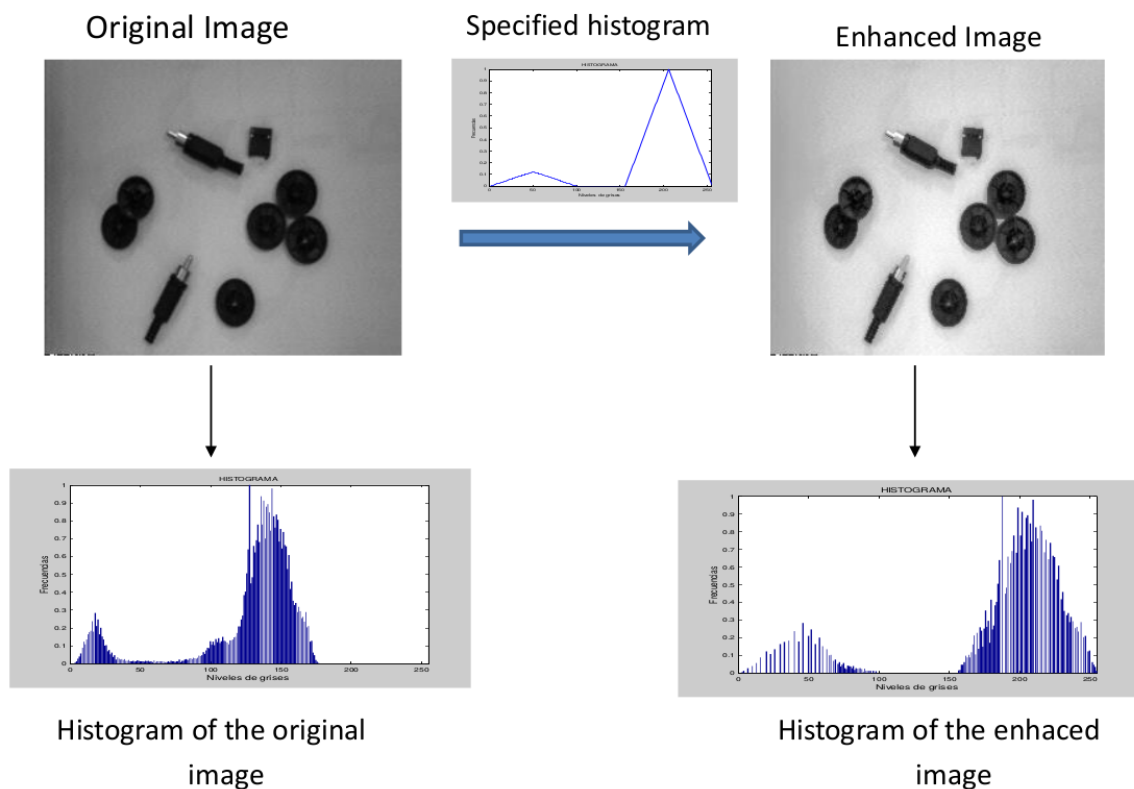
It's implementation is very similar to histogram equalization:

- First it is calculated the PMF (probability mass function) of all the pixels in both (source and reference) images.

- Next step involves calculation of CDF (cumulative distributive function) for both histograms (F_1 for source histogram and F_2 for reference histogram).

- Then for each gray level $G_1 \in [0, 255]$, we find the gray level G_2, for which $F_1(G_1) = F_2(G_2)$, producing the LUT for histogram equalization.

- Finally, the obtained LUT is applied.

ASSIGNMENT 3: Let's specify the histogram

Apply histogram specification using the `ramos.jpg` and `illumination.png` gray images. Then, show the resultant image along with input images (show their histograms as well).

Unfortunately, histogram specification is not implemented in our loved OpenCV. In this case you have to rely on the skimage.exposure.match_histograms() function from, the also popular scikit-image library.

In [9]:

```python
# ASSIGNMENT 3
# Write your code here!
from skimage.exposure import match_histograms

matplotlib.rcParams['figure.figsize'] = (15.0, 10.0)

image = cv2.imread(images_path + 'ramos.jpg', cv2.IMREAD_GRAYSCALE)
reference = cv2.imread(images_path + 'illumination.png', cv2.IMREAD_GRAYSCALE)

matched = match_histograms(image, reference)

# Plot results
plt.subplot(231)
plt.imshow(image, cmap='gray')
plt.title('Source')

plt.subplot(232)
plt.imshow(reference, cmap='gray')
plt.title('Reference')

plt.subplot(233)
plt.imshow(matched, cmap='gray')
plt.title('Matched')

plt.subplot(234)
plt.hist(np.ravel(image), 256, (0, 255))
plt.title('Source histogram')

plt.subplot(235)
plt.hist(np.ravel(reference), 256, (0, 255))
plt.title('Reference histogram')

plt.subplot(236)
plt.hist(np.ravel(matched), 256, (0, 255))
plt.title('Matched histogram');
```
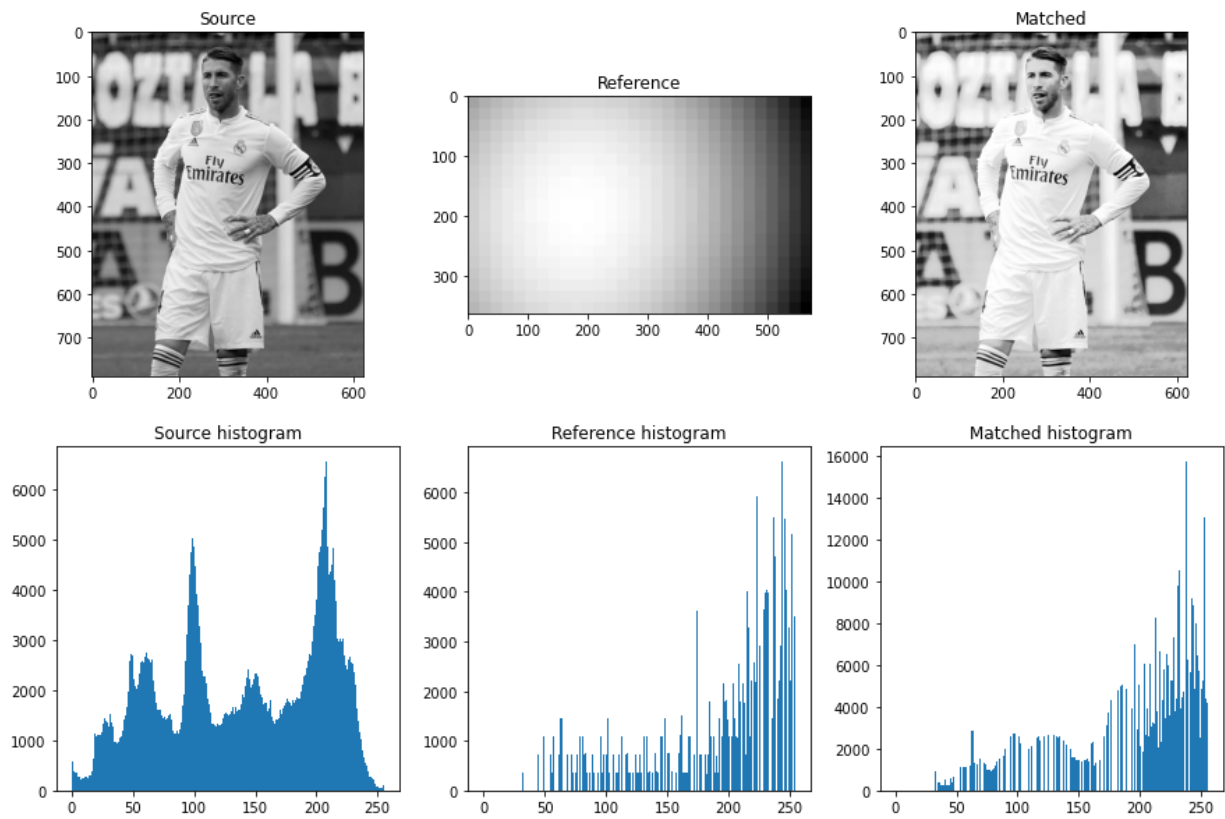
Conclusion

Great! We are sure that UMA users are going to appreciate your efforts. Also, next time you use an image editor tool you are going to have another point of view of how things work.

In conclusion, in this notebook you have learned:

- How to define a **gamma correction (non-linear) LUT** and to how to apply it to an image.
- How **histogram specification** works and its applications. When the specified histogram is uniformly distributed, we call it **histogram equalization**.

Extra

But this doesn't have to be the end, open GIMP and look through others implemented methods.

As you are learning about image processing, **comment how you think they are implemented from scratch.**