**image\_gray = color.rgb2gray(img)->** to convert the image to grayscale with rgb2gray

It is a three-channel image (RGB). We need to convert it into grayscale so that we only have a single channel.

**img\_as\_ubyte->**Convert an image to unsigned byte format, with values in [0, 255].

**Reason**->RGB image contains lots of data which may not be required for your processing. When you convert a RGB image into Gray scale you discard lots of information which are not required for processing.

----------

**Entropy**

In information theory, information entropy is the log-base-2 of the number of possible outcomes for a message.

For an image, local entropy is related to the complexity contained in a given neighborhood, typically defined by a structuring element. The entropy filter can detect subtle variations in the local gray level distribution.

The example shows how to detect texture in the image using a smaller structuring element.

----------

**fig, (ax0,ax1) = plt.subplots(ncols=2, figsize=(10, 4))**=> plt.subplots() is a function that returns a tuple containing a figure and axes object(s). Thus when using fig, ax = plt.subplots() you unpack this tuple into the variables fig and ax. Having fig is useful if you want to change figure-level attributes or save the figure as an image file later (e.g. with fig.savefig('yourfilename.png'))

The matplotlib. pyplot. subplots method provides **a way to plot multiple plots on a single figure**. Given the number of rows and columns , it returns a tuple ( fig , ax ), giving a single figure fig with an array of axes ax .

----------

**cmap=plt.cm.gray->**The gray() function in pyplot module of matplotlib library is **used to set the colormap to “gray”**.

----------

A [histogram](https://homepages.inf.ed.ac.uk/rbf/HIPR2/histgram.htm" \t "_blank) is a graph showing the number of [pixels](https://homepages.inf.ed.ac.uk/rbf/HIPR2/pixel.htm" \t "_blank) in an image at different intensity values found in that image. Simply put, a histogram is a graph wherein the x-axis shows all the values that are in the image while the y-axis shows the frequency of those values.

## **Unsupervised thresholding**

Scikit-image has a number of automatic thresholding methods, which require no input in choosing an optimal threshold. Some of the methods are

1) **Otsu’s thresholding**

Otsu’s method 2 calculates an “optimal” threshold (marked by a red line in the histogram below) by maximizing the variance between two classes of pixels, which are separated by the threshold. Equivalently, this threshold minimizes the intra-class variance.

The multi-Otsu threshold [1](https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_multiotsu.html?highlight=thresholding" \l "id2) is a thresholding algorithm that is used to separate the pixels of an input image into several different classes, each one obtained according to the intensity of the gray levels within the image.

Multi-Otsu calculates several thresholds, determined by the number of desired classes.

## 2) **Local thresholding**

If the image background is relatively uniform, then you can use a global threshold value as presented above. However, if there is large variation in the background intensity, adaptive thresholding (a.k.a. local or dynamic thresholding) may produce better results. Note that local is much slower than global thresholding.

Here, we binarize an image using the threshold\_local function, which calculates thresholds in regions with a characteristic size block\_size surrounding each pixel (i.e. local neighborhoods). Each threshold value is the weighted mean of the local neighborhood minus an offset value.

3) **Minimum thresholding**

For instance, the minimum algorithm takes a histogram of the image and smooths it repeatedly until there are only two peaks in the histogram.

A computationally efficient solution to the problem of minimum error thresholding is derived under the assumption of object and pixel grey level values being normally distributed. The method is applicable in multithreshold selection.

# 4) Li thresholding

In 1993, Li and Lee proposed a new criterion for finding the “optimal” threshold to distinguish between the background and foreground of an image [1](https://scikit-image.org/docs/stable/auto_examples/developers/plot_threshold_li.html?highlight=thresholding#id3). They proposed that minimizing the cross-entropy between the foreground and the foreground mean, and the background and the background mean, would give the best threshold in most situations.

Until 1998, though, the way to find this threshold was by trying all possible thresholds and then choosing the one with the smallest cross-entropy. At that point, Li and Tam implemented a new, iterative method to more quickly find the optimum point by using the slope of the cross-entropy [2](https://scikit-image.org/docs/stable/auto_examples/developers/plot_threshold_li.html?highlight=thresholding#id4). This is the method implemented in scikit-image’s [skimage.filters.threshold\_li()](https://scikit-image.org/docs/stable/api/skimage.filters.html#skimage.filters.threshold_li).

# Supervised segmentation

rgb2gray.-> Before doing any segmentation on an image, it is a good idea to de-noise it using some filters.

**Active Contour segmentation**also called **snakes**andis initialized using a user-defined contour or line, around the area of interest, and this contour then slowly contracts and is attracted or repelled from light and edges.

def circle\_points(resolution, center, radius):

"""

**Generate points which define a circle on an image.Centre refers to the centre of the circle**

"""

radians = np.linspace(0, 2\*np.pi, resolution)

c = center[1] + radius\*np.cos(radians**)#polar co-ordinates**

r = center[0] + radius\*np.sin(radians)

return np.array([c, r]).T

*The above calculations calculate x and y co-ordinates of the points on the periphery of the circle. Since we have given the resolution to be 200, it will calculate 200 such points.*

# Exclude last point because a closed path should not have duplicate points

points = circle\_points(200, [70, 180], 100)[:-1]

fig, ax = image\_show(img)

ax.plot(points[:, 0], points[:, 1], '--r', lw=3)

The algorithm then segments the face of a person from the rest of an image by fitting a closed curve to the edges of the face.

snake = seg.active\_contour(image\_gray1, points,alpha=0.06,beta=0.3)

fig, ax = image\_show(img)

ax.plot(points[:, 0], points[:, 1], '--r', lw=3)

ax.plot(snake[:, 0], snake[:, 1], '-b', lw=3);

We can tweak the parameters called alpha and beta. Higher values of alpha will make this snake contract faster while beta makes the snake smoother.( Мы можем настроить параметры, называемые альфа и бета. Более высокие значения альфа заставят эту змею сжиматься быстрее, в то время как бета делает змею более гладкой.)