Assignment : Histograms and filtering

January 24, 2019

1. Pixel-wise contrast enhancement

1.1. Write a Python function that implements the gamma transform of a gray scale image. Illustrate the function by applying it to an image of your choice. Try different values of gamma and notice how the image details become more or less visible. Include your code and a screenshot of the figure in your report.

1.2. Apply the grayscale gamma-correction function from the previous question to the color image 'autumn.tif' by doing so on each of the RGB component separately. Plot the original image and the result of this procedure side-by-side.

1.3. Apply the grayscale gamma-correction function from the previous question to the color image 'autumn.tif' by first converting to HSV color representation (see `skimage.color.rgb2hsv`) and apply the gamma correction to the v-channel, and finally convert back to RGB representation (see `skimage.color.hsv2rgb`). Plot the original image and the result of this procedure side-by-side. Which of the two approaches provide the best result?

2. Histogram-based processing

2.1. In mathematical terms, what is the cumulative distribution function (CDF) with respect to the probability density function (PDF) for continuous variables ? What can you infer about the intensity distribution from the variations of the CDF ?

2.2. What are the PDF and CDF of a constant image (e.g. an image with $I(x, y) = a$ for some constant intensity value a) ? What is the CDF corresponding to a constant PDF (a uniform distribution) ?

2.3. In the discrete setting, implement a Python function that, for a given histogram of a grayscale image, computes its cumulative histogram (you may use the *numpy.cumsum* function and normalize the expression). Display it for the image *pout.tif*. What do the regions of fast increase of the function correspond to ? And what about its flat regions ?

2.4. Given an image I and its CDF C, write a Python function that computes the floating-point image $C(I)$ such that the intensity at each pixel (x, y) is $C(I(x, y))$. Show the result of applying this routine to the *pout.tif* image.

2.5. The CDF is in general not invertible – why ? To overcome the problem of non-invertibility for histogram matching, we can consider instead a pseudo-inverse. For any CDF function f defined on the integer set $\{0, .., 255\}$, we define its pseudo-inverse $f^{(-1)}$ as follows :

$$f^{(-1)}(l) = \min\{s \mid f(s) \geq l\} \ ,$$

where $l \in [0, 1]$. Write a Python function that computes the pseudo-inverse of any given CDF (you may use *numpy.min* function).

2.6. Assume we have two gray scale images I_1 and I_2 and the corresponding CDFs C_1 and C_2, then histogram matching can be described by the mapping

$$f(x, y) = C_2^{-1}\left(C_1(I(x, y))\right) \ .$$

Based on this equation and on the previous questions, implement your own Python function to perform histogram matching between two images and show results. Plot and compare the cumulative histograms of the original image, the target and the one obtained by histogram matching.

2.7. Apply it in the particular case where the target CDF is the CDF corresponding to a constant histogram (a uniform distribution). Compare the result to the *skimage.exposure.equalize_hist* function.

Remark. *Instead of matching the histogram of one image on the other, it is often more robust and adequate to match both of them toward a 'midway' histogram of the two images. This is the idea of* ***midway equalization*** *method introduced below, which is notably used for the correction of* ***flicker*** *effects in old movies.*

2.8. Let I_1 and I_2 be two images with cumulative histograms C_1 and C_2. Let's introduce :

$$\phi = \frac{1}{2}\left(C_1^{(-1)} + C_2^{(-1)}\right)$$

and the images $\tilde{I}_1 = \phi(C_1(I_1))$ and $\tilde{I}_2 = \phi(C_2(I_2))$ called the *midway specifications* of I_1 and I_2. If I_1 and I_2 are two constant images with respective values a and b, prove that the midway specifications are both equal to the constant image $\frac{a+b}{2}$. Is the cumulative histogram of the midway specifications equal to the average of the cumulative histograms ?

2.9. Write a Python function that computes the midway specifications of two images. Show the result on the images *movie_flicker1.tif* and *movie_flicker2.tif* located in the Test Images folder and corresponding to two successive frames of the 1948 movie 'Les aventures des Pieds-Nickelés' (copyright Marc Sandberg). How would such a method generalize to an arbitrary number of images ?

3. Image filtering and enhancement

3.1. Consider the following finite difference approximations of the image derivatives given by :

$$\frac{\partial I}{\partial x}(x, y) \approx \frac{I(x+1, y) - I(x-1, y)}{2}$$
$$\frac{\partial I}{\partial y}(x, y) \approx \frac{I(x, y+1) - I(x, y-1)}{2}$$

Assuming that we filter an image using correlation, what are the kernels implementing these approximate derivatives ? And what if you consider instead that the image is convolved by the kernel ? How does Python deal with those two possible conventions ? Does it matter for filters like Gaussian or mean ?

3.2. Based upon the linear separablility of the Prewitt and Sobel filters (see lecture slides), reinterpret the effect of the Prewitt and Sobel filters on images : explain in particular why they are more likely to be robust to noise than the simple finite difference approximation of the previous question.

3.3. Compare the effect of mean and median filtering on a noisy version of the image *eight.tif* for salt and pepper as well as gaussian noise. What is the effect of increasing the kernel size N ? Store and plot the different computational times obtained for N=1 to N=25 and each time for 100 executions (use Pythons *timeit* package). Comment on your results.

3.4. Consider a Gaussian filter of fixed standard deviation $\sigma = 5$ and filter the image with increasing value of kernel size N. What do you observe when N is big enough and how can it be explained ?

3.5. Experiment with filters of increasing σ's, choosing at each iteration $N > 3\sigma$ for the kernel size and comment on the effect of denoising vs edge accuracy.

4. **Bonus questions**

In order to better preserve edges when filtering noise, an alternative popular technique is called **bilateral filtering**. The principle is to average both in the spatial and photometric domains. Given a Gaussian function $f_\sigma(x, y) = \exp(-\frac{x^2+y^2}{2\sigma^2})$ on the 2D spatial domain of the image and a Gaussian function $g_\tau(u) = \exp(-\frac{u^2}{2\tau^2})$ on the 1D photometric domain, the filtered image $\tilde{I}$ writes :

$$\tilde{I}(x, y) = \frac{\sum_{i=-l/2}^{l/2} \sum_{j=-k/2}^{k/2} \omega(x, y, i, j) I(x + i, y + j)}{\sum_{i=-1/2}^{l/2} \sum_{j=-k/2}^{k/2} \omega(x, y, i, j)}$$

with $\omega(x, y, i, j) = f_\sigma(i, j).g_\tau(I(x + i, y + j) - I(x, y))$, and l and k being the width and height of a discrete filter kernel representing f_σ.

4.1. Is it a linear filter ? What term differs from a usual Gaussian filter and what interpretation do you give to parameter τ ? In what limit case does bilateral filtering become usual Gaussian filtering ?

4.2. Implement a Python function that takes as input a grayscale image, a window size N, standard deviation parameters σ and τ and returns the filtered image (beware to restrict the neighborhoods for boundary pixels) : add the code to your report.

4.3. Apply it on the image *eight.tif* corrupted by Gaussian noise, for various σ and τ. Compare in addition the output of edge detection's methods after the image is filtered by a Gaussian vs a bilateral filter.