Prof. Dr. Daniel Tenbrinck, Ehsan Waiezi SS 24

Mathematical Image Processing

Exercise sheet 1 - due on Monday, May 13th 2024, 2pm

Exercise 1: Contrast enhancement 5 P.

The goal of this exercise is to manipulate image histograms by performing image contrast enhancement.

1. Write your own function `H = my_hist(I,nbins)` that computes the **histogram** of an image `I` with number of bins `nbins`. Compare this function to the Matlab function `hist`.

   Note:

   - The Matlab function `hist` works for one-dimensional signals, so you need to call the function for the vectorized image `I(:)`.
   - Refrain from using the Matlab function `imhist` that works for images with values in $[0, 1]$ rather than in $[0, 255]$.
   - When displaying images to illustrate contrast enhancement, remember to indicate the range you wish to display, e.g. `imagesc(I,[0 255])` for all images. Otherwise, the image value range will be chosen adaptively by Matlab.

2. **Linear transformation**
   Write a function `I_transformed = hist_linear(I,range_min,range_max)` that linearly transforms the histogram of a given image `I` to the image value range specified by `range_min,range_max` (typically $[0, 1]$ or $[0, 255]$).

3. **Nonlinear Gamma transformation**
   Write a function `I_gamma = hist_gamma(I,gamma)` that enhances the conrast of a given image `I` with a nonlinear Gamma transformation. Apply it to both too dark or too bright images, and try different values of `gamma`.

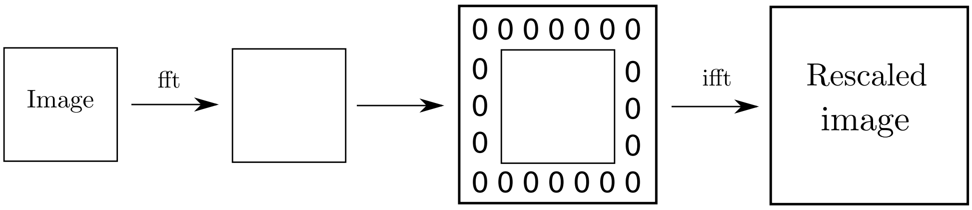
4. **Histogram equalization**
   Write a function `I_equalized = hist_eq(I)` that performs a histogram equalization for a given image `I`.
   Note: Remember that you are supposed to handle integer valued images in $[0, 255]$.

Exercise 2: Image upscaling 6 P.

The goal of this exercise is to manipulate image grids and pixel coordinates by upscaling a given image.

1. **Rescaling by duplication**
   The most naive method consists in copying the pixels of an image to make it bigger. Write a function `I_resize = resize_copy(I,factor)` that rescales image `I` by an integer factor `factor` by copying the pixel values accordingly.

2. **Rescaling by zero padding**
   It is also possible to interpolate grids in the frequency domain (i.e., the Fourier space) with a method known as *zero-padding*. It consists in upscaling the Fourier transform of the image by adding extra zeros to the high frequencies. Write a function `I_zeropad`



   `= zero_padding(I,factor)` that implements the zero-padding method.
   Note:

   - The Fourier transform for images is performed using the `fft2` function, the inverse Fourier transform with `ifft2`.
   - Please note that Matlab's Fourier transform puts the high frequencies at the middle, so if you want the low frequencies to be centered use the `fftshift` command.
   - The Fourier transform is complex, so remember to use the real part to get a real image at the end.
   - To keep the gray levels in the same range through the transform, it is better to multiply the transformed image by `factor`^{2}.

3. **Rescaling with interpolation filters**
   Rather than copying the pixels values as in the `resize_copy` function, it is more sophisticated to interpolate the pixel values to have a smoother image via *spatial convolution* with (for example) the following filters:

$$\textbf{tent:} \quad h(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1, \\ 0 & \text{else.} \end{cases} \tag{1}$$

$$\textbf{bell:} \quad h(x) = \begin{cases} -x^2 + \frac{3}{4} & \text{if } |x| \leq \frac{1}{2}, \\ \frac{1}{2}\left(|x| - \frac{3}{2}\right)^2 & \text{if } \frac{1}{2} < |x| < \frac{3}{2}, \\ 0 & \text{else.} \end{cases} \tag{2}$$

$$\textbf{Mitchell-Netravali:} \quad h(x) = \begin{cases} \frac{7}{6}|x|^3 - 2|x|^2 + \frac{8}{9} & \text{if } |x| < 1, \\ -\frac{7}{18}|x|^3 + 2|x|^2 - \frac{10}{3}x + \frac{16}{9} & \text{if } 1 \leq |x| < 2, \\ 0 & \text{else.} \end{cases} \tag{3}$$

   Write a function `I_filter = resize_filter(I,factor,filter)` that performs the upscaling by interpolation with the given filter.

   Note:

- For each pixel you have to perform the spatial convolution with the given filter in both directions, i.e. on the row and the columns of the image to be upscaled.
- Mind the image borders when performing the convolution!

Exercise 3: Image filtering **5 P.**

The goal of this exercise is to manipulate basic image filtering using spatial convolution filters and the median filter.

1. Write a function `I_filter = image_filter(I,filter)` that performs the spatial convolution of a (possibly noisy) image `I` with one of the following convolution filters:

$$m = \frac{1}{25}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad g = \frac{1}{81}\begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \nabla = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

   Here, m is the *averaging filter* and g is an approximation of the *Gaussian filter*, while Γ and ∇ are approximations of the amplitude of *gradient* and *Laplacian* of the image.

2. Write a function `I_median = median_filter(I,size)` that performs median filtering of the image `I` by taking for each pixel the median value over a surrounding square $N(i)$ of size `size` of pixel i as follows:

$$I_{\text{median}}(i) = \text{median}\{I_k \in N(i)\}$$

   Try different window sizes and observe the immediate effect.

   Note: The median filter is especially well suited to denoise *salt and pepper* or *impulsive noise* that you can generate with the Matlab function `imnoise`.