Practical session 2: an image denoising energy

Optimization Techniques, UPF

April/May, 2023

In this assignment we will address the problem of image denoising. For that, we will minimize an energy which enforces close-by pixels to have a similar value. To minimize such an energy, we will use the gradient-descent method. Since this is the first assignment in which we work with energies defined over images, we will start by introducing some notation.

During the lab session, you will be given some incomplete Python funcions. Following the different parts of the practice you will complete them.

Contact and guidelines:

- Nneka Okolo: `nnekamaureen.okolo@upf.edu`
- Deadline **P101**: Friday, May 5th at 23:55.
- Deadline **P102**: Wednesday, May 10th at 23:55.
- Labs can be done in pairs.
- Grading: The evaluation is based on the report documenting your work (with figures), results, conclusions and the commented code. What I expect you to explain,
  - The goal of the lab.
  - Summary with your own words of the topic.
  - Conclusions for each exercise.

1 Discrete images

Before starting, we need to introduce some notation for two types of images. Scalar images and vector-valued images. A scalar image represents an image with one channel, typically a gray-scale image. Vector-valued images are images which for each pixel has a vector. It could be for example an RGB image: at each pixel, we have a 3-dimensional vector with the R, G and B components. As we are going to see next, although we will not work with color images, we still need vector valued images for gradient images.

1.1 Scalar images

We define a scalar discrete image as a real function $u : \Omega \to \mathbb{R}$ defined over the rectangular discrete lattice $\Omega = \{1, \dots, M\} \times \{1, \dots, N\}$ (N columns and M rows). We refer to the image value at location $(i, j) \in \Omega$ (row i, column j) as $u_{i,j}$ or u_{ij}. The following for example is an image defined on $\Omega = \{1, 2, 3, 4\} \times \{1, 2, 3, 4, 5\}$.

$$u = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} \\ u_{21} & u_{22} & u_{23} & u_{24} & u_{25} \\ u_{31} & u_{32} & u_{33} & u_{34} & u_{35} \\ u_{41} & u_{42} & u_{43} & u_{44} & u_{45} \end{bmatrix}$$

We will consider the image as a vector in $\mathbb{R}^{MN}$, by concatenating the columns in a huge (column) vector:

$$u = [\overbrace{u_{11}, \dots, u_{M1}}^{\text{column 1}}, \overbrace{u_{12}, \dots, u_{M2}}^{\text{column 2}}, \quad \cdots \quad , \overbrace{u_{1N}, \dots, u_{MN}}^{\text{column N}}]^T.$$

We define the notation $\mathcal{X} = \mathbb{R}^{MN}$. We can think of $\mathcal{X}$ is the space of all $M \times N$ images. Since we consider images as vectors in $\mathcal{X}$, we have a scalar product and a corresponding norm given as follows:

$$\langle u, v \rangle_{\mathcal{X}} = \sum_{i=1}^{M} \sum_{j=1}^{N} u_{ij} v_{ij} \quad \text{and} \quad \|u\|_{\mathcal{X}} = \sqrt{\langle u, u \rangle_{\mathcal{X}}} = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} u_{ij}^2},$$

where $u, v \in \mathcal{X}$ are two images. In some cases we will call these the $\mathcal{X}$-scalar product and the $\mathcal{X}$-norm.

Hint: *Notice that in Python the difference between matrix multiplication and element-wise multiplication.*

1.2 Vector images

We will also consider *vector-valued images* $g : \Omega \to \mathbb{R}^2$. The value of g at pixel i, j is a two component vector $g_{ij} = (g_{1,ij}, g_{2,ij}) \in \mathbb{R}^2$. The following for example is an image

defined on $\Omega = \{1,2,3,4\} \times \{1,2,3,4,5\}$:

$$g = \begin{bmatrix} (g_{1,11}, g_{2,11}) & (g_{1,12}, g_{2,12}) & (g_{1,13}, g_{2,13}) & (g_{1,14}, g_{2,14}) & (g_{1,15}, g_{2,15}) \\ (g_{1,21}, g_{2,21}) & (g_{1,22}, g_{2,22}) & (g_{1,23}, g_{2,23}) & (g_{1,24}, g_{2,24}) & (g_{1,25}, g_{2,25}) \\ (g_{1,31}, g_{2,31}) & (g_{1,32}, g_{2,32}) & (g_{1,33}, g_{2,33}) & (g_{1,34}, g_{2,34}) & (g_{1,35}, g_{2,35}) \\ (g_{1,41}, g_{2,41}) & (g_{1,42}, g_{2,42}) & (g_{1,43}, g_{2,43}) & (g_{1,44}, g_{2,44}) & (g_{1,45}, g_{2,45}) \end{bmatrix}.$$

Note that vector-valued images can be separated into two components or channels g_1 and g_2. Each component is a scalar image: $g_1, g_2 \in \mathcal{X}$.

$$g_1 = \begin{bmatrix} g_{1,11} & g_{1,12} & g_{1,13} & g_{1,14} & g_{1,15} \\ g_{1,21} & g_{1,22} & g_{1,23} & g_{1,24} & g_{1,25} \\ g_{1,31} & g_{1,32} & g_{1,33} & g_{1,34} & g_{1,35} \\ g_{1,41} & g_{1,42} & g_{1,43} & g_{1,44} & g_{1,45} \end{bmatrix}, \quad g_2 = \begin{bmatrix} g_{2,11} & g_{2,12} & g_{2,13} & g_{2,14} & g_{2,15} \\ g_{2,21} & g_{2,22} & g_{2,23} & g_{2,24} & g_{2,25} \\ g_{2,31} & g_{2,32} & g_{2,33} & g_{2,34} & g_{2,35} \\ g_{2,41} & g_{2,42} & g_{2,43} & g_{2,44} & g_{2,45} \end{bmatrix}.$$

We will arrange vector-valued images into a vector in $\mathbb{R}^{2MN}$ by "vectorizing" first the first component followed by the second:

$$g = [\overbrace{g_{1,11}, \ldots, g_{1,N1}}^{\text{col. 1 of comp. 1}}, \quad \cdots \quad \overbrace{g_{1,1M}, \ldots, g_{1,NM}}^{\text{col. M of comp. 1}}, \\ \overbrace{g_{2,11}, \ldots, g_{2,N1}}^{\text{col. 1 of comp. 2}}, \quad \cdots \quad \overbrace{g_{2,1M}, \ldots, g_{2,NM}}^{\text{col. M of comp. 2}}]^T \in \mathcal{Y} = \mathbb{R}^{2MN}.$$

The space of all $M \times N$ vector-valued images (with two-component vectors) will be denoted by $\mathcal{Y} = \mathbb{R}^{2MN}$. Note that we have to be careful to avoid confusions: the value of a vector-valued image g at pixel (i,j), g_{ij}, is a vector in $\mathbb{R}^2$. On the other hand g itself is a vector in $\mathcal{Y} = \mathbb{R}^{2MN}$. Therefore we will use different notations for the scalar product and the norm in $\mathbb{R}^2$ and in $\mathcal{Y}$.

For vectors in $a = (a_1, a_2), b = (b_1, b_2) \in \mathbb{R}^2$ we use the following notations:

$$a \cdot b = a_1 b_1 + a_2 b_2 \quad \text{and} \quad |a| = \sqrt{a \cdot a} = \sqrt{a_1^2 + a_2^2}.$$

On the other hand, for two vector-valued images g, h in $\mathcal{Y}$, we will use the following $\mathcal{Y}$-scalar product and $\mathcal{Y}$-norm, which is defined based the scalar product in $\mathbb{R}^2$:

$$\langle g, h \rangle_{\mathcal{Y}} = \sum_{i=1}^{M} \sum_{j=1}^{N} g_{ij} \cdot h_{ij}.$$

The $\mathcal{Y}$-scalar product between two vector-valued images can be computed as the sums of the $\mathcal{X}$-scalar products of their channels as follows

$$\langle g, h \rangle_{\mathcal{Y}} = \sum_{i=1}^{M} \sum_{j=1}^{N} g_{ij} \cdot h_{ij} = \sum_{i=1}^{M} \sum_{j=1}^{N} (g_{1,ij} h_{1,ij} + g_{2,ij} h_{2,ij}) \\ = \sum_{i=1}^{M} \sum_{j=1}^{N} g_{1,ij} h_{1,ij} + \sum_{i=1}^{M} \sum_{j=1}^{N} g_{2,ij} h_{2,ij} = \langle g_1, h_1 \rangle_{\mathcal{X}} + \langle g_2, h_2 \rangle_{\mathcal{X}},$$

The $\mathcal{Y}$-norm is defined similarly, and as we show next, it can also be computed in terms of the $\mathcal{X}$-norm of the channels:

$$\|g\|_{\mathcal{Y}} = \sqrt{\langle g, g\rangle_{\mathcal{Y}}} = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} |g_{ij}|^2} = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} (g_{1,ij}^2 + g_{2,ij}^2)} = \sqrt{\|g_1\|_{\mathcal{X}}^2 + \|g_2\|_{\mathcal{X}}^2}.$$

2 The discrete gradient as a matrix

Since we are working with discrete images, we will consider a discrete approximation of the gradient, using *forward differences*. We will use the notation $\nabla^+ u$ to refer to the image forward gradient. The discrete gradient of u is a vector-valued image, $\nabla^+ u : \Omega \to \mathbb{R}^2$. It has two components for the horizontal and vertical partial derivatives. Thus we have that

$$\nabla^+ u_{ij} = \left(\nabla_i^+ u_{ij}, \nabla_j^+ u_{ij}\right) \in \mathbb{R}^2.$$

Here ∇_i^+ and ∇_j^+ refer to the forward differences partial derivatives in the direction of i (rows) and j (columns). These are defined as follows:

$$\nabla_i^+ u_{ij} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } i < M \\ 0 & \text{if } i = M \end{cases} \qquad \nabla_j^+ u_{ij} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } j < N \\ 0 & \text{if } j = M \end{cases}$$

The i derivative corresponds to the vertical derivative, whereas the j derivative to the horizontal derivative.

We can consider ∇_i^+ and ∇_j^+ as square matrices of MN rows and columns. We can "derivate" the image by computing the product of ∇_i^+ by our vector representation of the image. The result is a vector representation of the i partial derivative. For a 4×5 image they would be as follows:

$$\nabla_i^+ u = \left[\begin{array}{cccccccccccccccccccc}
-1 & 1 & & & & & & & & & & & & & & & & & & \\
 & -1 & 1 & & & & & & & & & & & & & & & & & \\
 & & -1 & 1 & & & & & & & & & & & & & & & & \\
 & & & 0 & & & & & & & & & & & & & & & & \\
\hdashline
 & & & & -1 & 1 & & & & & & & & & & & & & & \\
 & & & & & -1 & 1 & & & & & & & & & & & & & \\
 & & & & & & -1 & 1 & & & & & & & & & & & & \\
 & & & & & & & 0 & & & & & & & & & & & & \\
\hdashline
 & & & & & & & & -1 & 1 & & & & & & & & & & \\
 & & & & & & & & & -1 & 1 & & & & & & & & & \\
 & & & & & & & & & & -1 & 1 & & & & & & & & \\
 & & & & & & & & & & & 0 & & & & & & & & \\
\hdashline
 & & & & & & & & & & & & -1 & 1 & & & & & & \\
 & & & & & & & & & & & & & -1 & 1 & & & & & \\
 & & & & & & & & & & & & & & -1 & 1 & & & & \\
 & & & & & & & & & & & & & & & 0 & & & & \\
\hdashline
 & & & & & & & & & & & & & & & & -1 & 1 & & \\
 & & & & & & & & & & & & & & & & & -1 & 1 & \\
 & & & & & & & & & & & & & & & & & & -1 & 1 \\
 & & & & & & & & & & & & & & & & & & & 0
\end{array}\right] \cdot \left[\begin{array}{c} u_{11} \\ u_{21} \\ u_{31} \\ u_{41} \\ \hdashline u_{12} \\ u_{22} \\ u_{32} \\ u_{42} \\ \hdashline u_{13} \\ u_{23} \\ u_{33} \\ u_{43} \\ \hdashline u_{14} \\ u_{24} \\ u_{34} \\ u_{44} \\ \hdashline u_{15} \\ u_{25} \\ u_{35} \\ u_{45} \end{array}\right] = \left[\begin{array}{c} u_{21} - u_{11} \\ u_{31} - u_{21} \\ u_{41} - u_{31} \\ 0 \\ \hdashline u_{22} - u_{12} \\ u_{32} - u_{22} \\ u_{42} - u_{32} \\ 0 \\ \hdashline u_{23} - u_{13} \\ u_{33} - u_{23} \\ u_{43} - u_{33} \\ 0 \\ \hdashline u_{24} - u_{14} \\ u_{34} - u_{24} \\ u_{44} - u_{34} \\ 0 \\ \hdashline u_{25} - u_{15} \\ u_{35} - u_{25} \\ u_{45} - u_{35} \\ 0 \end{array}\right]$$

$$\nabla_j^+ u = \left[\begin{array}{cccccccccccccccc}
-1 & & & 1 & & & & & & & & & & & & \\
 & -1 & & & 1 & & & & & & & & & & & \\
 & & -1 & & & 1 & & & & & & & & & & \\
 & & & -1 & & & 1 & & & & & & & & & \\
\hdashline
 & & & & -1 & & & 1 & & & & & & & & \\
 & & & & & -1 & & & 1 & & & & & & & \\
 & & & & & & -1 & & & 1 & & & & & & \\
 & & & & & & & -1 & & & 1 & & & & & \\
\hdashline
 & & & & & & & & -1 & & & 1 & & & & \\
 & & & & & & & & & -1 & & & 1 & & & \\
 & & & & & & & & & & -1 & & & 1 & & \\
 & & & & & & & & & & & -1 & & & 1 & \\
\hdashline
 & & & & & & & & & & & & 0 & & & \\
 & & & & & & & & & & & & & 0 & & \\
 & & & & & & & & & & & & & & 0 & \\
 & & & & & & & & & & & & & & & 0
\end{array}\right] \cdot \left[\begin{array}{c} u_{11} \\ u_{21} \\ u_{31} \\ u_{41} \\ \hdashline u_{12} \\ u_{22} \\ u_{32} \\ u_{42} \\ \hdashline u_{13} \\ u_{23} \\ u_{33} \\ u_{43} \\ \hdashline u_{14} \\ u_{24} \\ u_{34} \\ u_{44} \\ \hdashline u_{15} \\ u_{25} \\ u_{35} \\ u_{45} \end{array}\right] = \left[\begin{array}{c} u_{12} - u_{11} \\ u_{22} - u_{21} \\ u_{32} - u_{31} \\ u_{42} - u_{41} \\ \hdashline u_{13} - u_{12} \\ u_{23} - u_{22} \\ u_{33} - u_{32} \\ u_{43} - u_{42} \\ \hdashline u_{14} - u_{13} \\ u_{24} - u_{23} \\ u_{34} - u_{33} \\ u_{44} - u_{43} \\ \hdashline u_{15} - u_{14} \\ u_{25} - u_{24} \\ u_{35} - u_{34} \\ u_{45} - u_{44} \\ \hdashline 0 \\ 0 \\ 0 \\ 0 \end{array}\right]$$

The discrete gradient is a vector-valued image with two components. We construct a matrix for the whole discrete gradient by concatenating the matrices ∇_i^+ and ∇_j^+ in a block matrix:

$$\nabla^+ u = \left[\begin{array}{c} \nabla_i^+ \\ \hdashline \nabla_j^+ \end{array}\right] \cdot u = \left[\begin{array}{c} \nabla_i^+ u \\ \hdashline \nabla_j^+ u \end{array}\right] \Bigg\} \text{(vector with } 2MN \text{ components).}$$

3 An energy for image denoising

We have all the ingredients to formulate our energy. Let us consider a noisy image $f : \Omega \to \mathbb{R}$. We assume that the image f is the result of contaminating a clean image u^* by white Gaussian noise:

$$f_{ij} = u^*_{ij} + n_{ij}, \quad \text{for } 1 \leqslant i \leqslant M, 1 \leqslant j \leqslant N,$$

where $n_{ij} \sim \mathcal{N}(0, \sigma)$.

To denoise the image, we consider a model for noiseless images (the image *prior*). There are many possibilities. In this practice we will assume that noiseless images have gradients with low norm. This is one of the simplest models. The image prior represents what we know about the image, *before knowing the actual image*.

We will estimate u^* by computing the minimum of the following energy $E : \mathbb{R}^{MN} \to \mathbb{R}$:

$$E(u) = \overbrace{\sum_{i=1}^{M}\sum_{j=1}^{N} c_{ij}|\nabla^+ u_{ij}|^2}^{\text{regularization}} + \beta \overbrace{\sum_{i=1}^{M}\sum_{j=1}^{N}(u_{ij} - f_{ij})^2}^{\text{data attachment}}, \tag{1}$$

where $c : \Omega \to [0, 1]$ is a coefficients image which controls the regularization. Recall that $|\cdot|$ denotes the 2-norm in $\mathbb{R}^2$:

$$|\nabla^+ u_{ij}|^2 = \left(\nabla_i^+ u_{ij}\right)^2 + \left(\nabla_j^+ u_{ij}\right)^2.$$

The energy has a regularization term and a data attachment term. Let us explain them:

Regularization term. The regularization term penalizes high gradients. This corresponds to our model for clean images with low discrete gradients: the value at a pixel should be similar to those of its neighbors. This is an oversimplistic model: clean images may have discontinuities and high gradients at the edges of the objects in the image. As a result, our model for denoising will blur the image edges. This is why we add the c_{ij} coefficients, to control the penalization of high gradients. The idea is to use a lower c_{ij} around image edges, and a higher c_{ij} on regions far from the edges. We will consider $c_{ij} \in [0, 1]$.

Data attachment term. The data attachment penalizes high differences with respect to the noisy data. The parameter $\beta > 0$ sets the strength of the data attachment.

We can express the energy in matrix notation, using our vector representation of images and discrete gradients. For that, let us define some useful notation for diagonal matrices. Let us consider a given scalar image $c \in \mathcal{X}$. We then define diag(c) as the diagonal $MN \times MN$ matrix which has the vectorized image c in the diagonal. That is:

$$\operatorname{diag}(c) = \begin{bmatrix} c_{11} & & & & & & & \\ & \ddots & & & & & & \\ & & c_{M1} & & & & & \\ & & & c_{12} & & & & \\ & & & & \ddots & & & \\ & & & & & c_{M2} & & \\ & & & & & & \ddots & \\ & & & & & & & c_{1N} \\ & & & & & & & & \ddots \\ & & & & & & & & & c_{MN} \end{bmatrix},$$

where we have only shown the non-zero entries. Observe that the multiplication of a vectorized image u times a diagonal matrix $\operatorname{diag}(c)$, $v = \operatorname{diag}(c)u$, then v corresponds to the element-wise multiplication of c and u: thus $v_{ij} = c_{ij}u_{ij}$. In fact, this is the main utility of these diagonal matrices: to be able to write the element-wise multiplication between images with the matrix multiplication.

Now we can define our energy for image denoising as follows:

$$E(u) = \langle C\nabla^+ u, \nabla^+ u\rangle_{\mathcal{Y}} + \beta\|u - f\|_{\mathcal{X}}^2. \tag{2}$$

Here C is a $2MN \times 2MN$ diagonal matrix defined as follows:

$$C = \left[\begin{array}{c:c} \operatorname{diag}(c) & \\ \hdashline & \operatorname{diag}(c) \end{array}\right],$$

which in its diagonal has two copies of the vectorized image c. One copy performs the element-wise multiplication of c with the i partial derivative and the other with the j partial derivative.

Assigments to deliver

1. *Complete the Python functions* `im_fwd_gradient` *and* `im_bwd_divergence`. *These functions compute the forward gradient* ∇^+ *and the backwards divergence* div^-. *Follow the comments provided in the code.*

2. *Complete the Python functions* `denoise_energy_gradient` *following the comments provided in the code.*

3. *Run the function* `denoise_main` *with different parameters (for instance, changing the maximum number of iterations,* c, β,*etc. Can you explain what you observe? What are the effects of changing* c*? Try to find the parameters gives you a smaller Mean Square Error. You can use the Python function mse.*