JHU 553.493/693, Spring '20 – Homework #2

Edge detection, sharpening, the Total Variation norm, and histogram equalization.

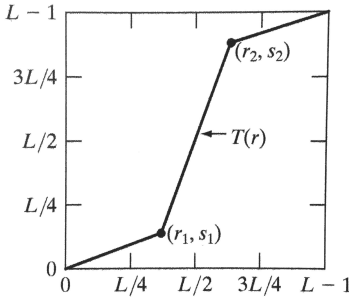
Due on Blackboard by the end of the day on Sunday, March 1st

Instructions: Write, on top of the first page of your assignment: Name (LAST, First), the HW# (in this case, "Homework #2"), and acknowledge others with whom you may have worked (just write "*Worked with ...*"). For the computational problems where you are asked to write computer code you may choose the programming language that you prefer, such as *Python* or *Matlab*. You do *not* have to type up your homework. If you handwrite it, please scan it making sure it is readable. You should submit *one* PDF file on Blackboard (or at most two, if you choose to submit your code separately). Feel free to use in-built functions for convolution (e.g. `conv2` in *Matlab*): you do *not* have to implement it from scratch using nested '`for`' loops.

In the problems that follow, we shall employ the following notation:

$b = \#$ of bits per pixel (typically $b = 8$), $\quad L = 2^b = \#$ of gray levels, $\quad \mathbb{P} = \{0, 1, 2, \ldots, L-1\}$,
$\mathbb{D} = \{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\}$, where: $\quad M = \#$ rows, $\quad N = \#$ columns.

Problem 2.1 (Contrast Enhancement)**.** A simple way to enhance contrast in an image is to apply a point operation $S : \mathbb{P} \to \mathbb{P}$ that further decreases luminance values that are already low, and on the other hand increases large luminance values. Implement, in *Matlab* or *Python* (or any programming language that you would rather use) the a point function $s = T(r)$ whose graph is given in the figure on the right. Note that the parameters (r_1, s_1) and (r_2, s_2) should be part of the input (i.e. the arguments) to your function. Now use your code to enhance contrast in the image `image-airport.tif`.



Problem 2.2 (More on isotropy)**.** In the last homework you showed that the Laplacian ∇^2 is *isotropic*. Isotropy is important if you want an operation to be rotation-invariant: that is, if you apply the same operation to an image that has been rotated by a certain angle, you want it to yield the same result as the operation applied to the non-rotated image. This is important, for example, for edge detection. Consider:

$$\|\nabla f(x,y)\| = \sqrt{\left(\tfrac{\partial f}{\partial x}\right)^2 + \left(\tfrac{\partial f}{\partial y}\right)^2}, \qquad \text{and} \qquad \|\nabla f(x,y)\| \simeq \left|\tfrac{\partial f}{\partial x}\right| + \left|\tfrac{\partial f}{\partial y}\right|.$$

(a) The first of the above equations is the formula for the magnitude of the gradient: show that it is an isotropic operation. **(b)** The second one is an approximation of $\|\nabla f\|$: show that this is *not* isotropic.

Problem 2.3 (Edge detection)**. (a)** The simplest way of detecting edges is precisely to compute the magnitude of the gradient of an image and display it as if it were an image. Implement code that detects the edge of `image-contactlens.tif` (it is a contact lens illuminated by a lighting arrangement designed to highlight imperfections): that is, define $J(i,j) = \|\nabla I(i,j)\|$, where partial derivatives are computed using Sobel filters. Also, enhance your image with a threshold point function, i.e. define $K(i,j) = (L-1)\,H(J(i,j) - t)$, where $H(\cdot)$ is the Heaviside function ($H(x) = 1$ for $x \geq 0$, and $H(x) = 0$ for $x < 0$) and t is a treshold selected at 33% of the highest value in the image J. **(b)** Another way to detect the edges of an image I is to use the *Laplacian* of the image $\nabla^2 I$. Remember that derivatives of an image (unlike the *magnitude* of its gradient) may have *negative* values, therefore we should use either the absolute value $|\nabla^2 I|$ or the *positive values* of the Laplacian, namely $J(i,j) = \max(\nabla^2 I(i,j), 0)$. Why would this work? (Think about the cross section of an edge.) Implement code that computes and displays the positive values of the Laplacian, and apply it to `image-circuit2.tif` (which shows a *binary* portion of a wire-bond mask for an electronic circuit). **(c)** Consider now `image-building.tif`, which exhibits some pretty regular geometry. First compute and display $\|\nabla I\|$; you will note that all edges, even the minor ones (corresponding to bricks on the walls), are detected and displayed. Edge detection can be made more selective by smoothing the image prior to computing the gradient. So, perform the following operations,

in this order: (i) smooth the original image with a 5×5 mean filter (each entry is equal to $1/25$); (ii) detect the edges by computing the gradient, using either of the two methods from **Problem 2.2**; and finally (iii) treshold the resulting image, with the technique described in part (a), with a threshold equal to 33% of the highest value in the image (you may try different thresholds). Submit your code and your results.

> **Norms.** Let V be a vector space (think of $\mathbb{R}^n$). Remember that a *norm* on V is a function $\|\cdot\| : V \to \mathbb{R}$ with the properties: **(N1)** $\|a\mathbf{v}\| = |a|\,\|\mathbf{v}\|$ for all $a \in \mathbb{R}$ and $\mathbf{v} \in V$; **(N2)** $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$ for all $\mathbf{v}, \mathbf{w} \in V$ (*triangle inequality*); **(N3)** $\|\mathbf{v}\| = 0 \Rightarrow \mathbf{v} = \mathbf{0}$ (the zero of V). Similarly, a map $\|\cdot\| : V \to \mathbb{R}$ with properties **(N1)** and **(N2)**, and *not* **(N3)**, is called a *seminorm* on V.

Problem 2.4 (The *Total Variation* Norm)**. (a)** Show, using the definition of norm above, that $\|\mathbf{0}\| = 0$ and $\|\mathbf{v}\| \geq 0$ for all $\mathbf{v} \in V$ (positivity). **(b)** We can think of images as elements of a linear space (this is technically correct as long as we admit images with negative values). In fact, let's first consider *continuous images*, i.e. defined on the continuous domain $D = [0, M] \times [0, N]$. A *measure of sharpness* of the image is the so-called *Total Variation* (TV) norm:

$$\|I\|_{\text{TV}} = \frac{1}{\text{Area}(D)} \iint_D \|\nabla I(x, y)\|\, dA,$$

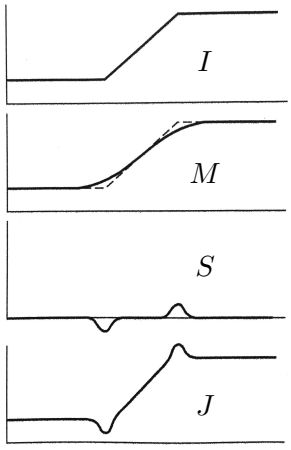
which is nothing but the mean value of the magnitude of the gradient of the image. Note that the norm in the integral is the usual Euclidean norm in $\mathbb{R}^2$: namely if $\mathbf{v} = (a, b)$ then $\|\mathbf{v}\| = \sqrt{a^2 + b^2}$. Show that it is, in fact, a *seminorm* on the set of $C^1(D)$ functions (functions that are continuously differentiable in D). Why is it a seminorm and not a norm? **(c)** In the *discrete* setting, i.e. for real images $I : \mathbb{D} \to \mathbb{P}$, we compute the TV norm numerically as follows:

$$\|I\|_{\text{TV}} = \frac{1}{M \cdot N} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \|\nabla I(x, y)\|.$$

Write computer code (in *Python*, or *Matlab*, or anything else) that computes the Total Variation norm of an image I, where the partial derivatives in the gradient are computed using Sobel filters.

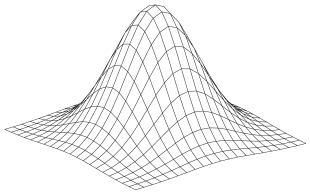
Problem 2.5. (More on sharpening) **(a)** In Problem 1.10 of the previous homework assignment we introduced the so-called Laplace Filter for sharpening: $J = I - \nabla^2 I$. Now consider the modified Laplace Filter $J = I - k\nabla^2 I$, where k is a positive constant. Modify the computer program that you wrote last time to incorporate this constant, and apply it to `image-moon.jpg` for different values of k varying from 0.1 to 4. For both the original image and the outputs compute the TV norm using the code from the previous problem. *Hint:* To avoid *saturation artifacts* in the image J (due to negative values or above 255) you should modify your result J defining $J_2(i, j) = \min(\max(J(i, j), 0), 255)$, and use it as your final result.

(b) As we will see in Fourier anaysis, the Laplace Filter has the problem that it may boost the higher frequencies "too much". A way around it is to use the so-called *Unsharp Masking* (USM), which was explained in class. We first compute a smooth version of the image I, namely $M = I * h_M$, where h_M is an averaging filter or a Gaussian mask. We then compute $S = I - M$, which is the "sharp component" of I (also known as the *unsharp mask*). Finally, we add back to M a "boosted" version of the sharp component, i.e. $J = M + kS$, with some $k > 1$ (note that for $k = 1$ we get back the original image I). We can also write $k = 1 + a$, with $a > 0$, and write $J = M + (1 + a)S = M + S + aS = I + aS$. The number $a > 0$ is sometimes called the *sharpening strength*. Note that a is not the only parameter, as another item that we can choose is the smoothing convolution kernel h_M: a common choice is a $n \times n$ Gaussian mask (n is odd). I.e, we fix a standard deviation σ and sample the Gaussian bell-shaped function:



$$G(x, y) = \exp\Big(-\frac{1}{2}\frac{x^2 + y^2}{\sigma^2}\Big)$$

(shown on the right) at integer coordinates, and divide the resulting mask by the sum of its elements (so to obtain a mask whose elements sum to one); σ is called the *spatial extent* of the filter. For example, for $n = 3$ and $\sigma^2 = .5$ we get:



$$h_M = \frac{1}{\sum_{i,j=-1}^{1} G(i,j)} \begin{bmatrix} G(-1,-1) & G(-1,0) & G(-1,1) \\ G(0,-1) & G(0,0) & G(0,1) \\ G(1,-1) & G(1,0) & G(1,1) \end{bmatrix} = \begin{bmatrix} 0.0449 & 0.1221 & 0.0449 \\ 0.1221 & 0.3319 & 0.1221 \\ 0.0449 & 0.1221 & 0.0449 \end{bmatrix}.$$

For larger values of σ, it is comvenient to use a larger convolution kernel size n. Write code that implements the Unsharp Masking method. Ideally your function should have, as inputs: I (the image to be sharpened), a (the sharpening strength), σ^2 (or just σ, which is the spatial extent of the Gaussian mask), n (its size); and as output: J (the sharpened image). Apply it to the image `image-irish.tif` with $a = 1$ and different values of σ^2, e.g. $\sigma^2 = 2.5$, $\sigma^2 = 5$, and $\sigma^2 = 10$. For both `image-irish.tif` and each resulting image, also compute the TV norm. Please turn in both your code and your results.

Remark: Here is some food for thought. Different regions of an image may need different sharpening strenghts—which leads to the idea of *adaptive sharpening*. That is, one can define: $J(i,j) = I(i,j) + a(i,j)S(i,j)$, where the sharpening strength a depends on the location (i,j). The simplest way to implement this is to set $a(i,j) = H(\|\nabla I(i,j)\| - t)$, where H is the Heaviside function ($H(x) = 1$ for $x \geq 0$, and $H(x) = 0$ for $x < 0$) and t is a user-defined threshold: that is, we have $J = I + S$ where $\|\nabla I\|$ is larger than t, otherwise it is left unchanged ($J = I$). Can you think of other ways to implement adaptive sharpening? (You don't have to answer this question, but it may be material for a future project.)

Problem 2.6 (Detecting diagonal edges)**.** The techniques that we have seen so far can be used to detect edges that are parallel to a prescribed direction. For example, if $\mathbf{n} = (a,b)$ is a unit vector (that is, $\sqrt{a^2+b^2} = 1$) that is *orthogonal* to a particular direction that we are interested in, we can find the points (x,y) where the *directional derivative* of I in the direction $\mathbf{n}$, defined as the number

$$\frac{\partial I}{\partial \mathbf{n}}(x,y) = \lim_{h \to 0} \frac{I(x+ah, y+bh) - I(x,y)}{h},$$

$$h_{\mathbf{n}} = \tfrac{1}{8} \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

is large (in absolute value). The above formula (which should be familiar from multivariable calculus) is valid in a continous setting; for digital images it is approximated with discrete filters, as it is the case for the partial derivatives $\partial I/\partial x$ and $\partial I/\partial y$. For example, for $\mathbf{n} = (1,1)/\sqrt{2}$ (remember that we always consider x pointing downwards and y poiting right!) the corresponding Sobel filter $h_{\mathbf{n}}$ is shown above. Consider again `image-building.tif`, used in **Problem 2.3**. Smooth it with a 5×5 mean filter, filter it with the convolution kernel $h_{\mathbf{n}}$, compute the absolute value of the result and display it (you may also treshold the resulting image with a threshold equal to 33% of its highest value). What is highlighted?

Problem 2.7. Suppose that g and h are two convolution kernels, of sizes $(2K_1+1) \times (2K_1+1)$ and $(2K_2+1) \times (2K_2+1)$. What is the size of the convolution kernel $f = g * h$? Justify your answer.

Problem 2.8. Given a function $f : \mathbb{Z}^2 \to \mathbb{R}$, we define its *volume* as $\mathrm{Vol}(f) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)$. Volume is a measure of the "total brightness" of an image. Show that $\mathrm{Vol}(f * g) = \mathrm{Vol}(f)\mathrm{Vol}(g)$.

Problem 2.9 (Image Equalization)**.** We introduced a technique for *equalizing* an image. Namely, we defined the *histogram* and the cumulative distribution function (CDF) of an image as follows:

$$h_I(\ell) = \#\big\{(i,j) \big| I(i,j) = \ell\big\}, \qquad \text{and} \qquad F_I(\ell) = \frac{L-1}{MN} \sum_{k=0}^{\ell} h_I(k), \qquad \text{both defined for } \ell \in \mathbb{P}.$$

We showed that, in the discrete setting, defining $J(i,j) = F_I(I(i,j))$, $(i,j) \in D$, yields a new emage whose histogram is *equalized*, in the sense that while it is not quite true that h_J is a constant, it is the case that $F_J(\ell) \simeq \ell$, for $\ell \in \mathbb{P}$. (In the continuous setting, however, we would obtain a constant density for J.)

(a) Would a second pass of histogram equalization produce a different result? **(b)** Write code to implement the above histogram equalization technique (ideally, write a function that has an image as an input and the equalized image as an output). **(c)** Download the image `image-fruits.tif`, and print it. Compute and print its histogram as well as the graph of its cumulative distribution function (CDF, as a continuous curve). Now perform histogram equalization. Display the new image, compute and display its histogram and the graph of its cumulative distribution function. Repeat for `image-airport.tif` and `image-spine.tif`.