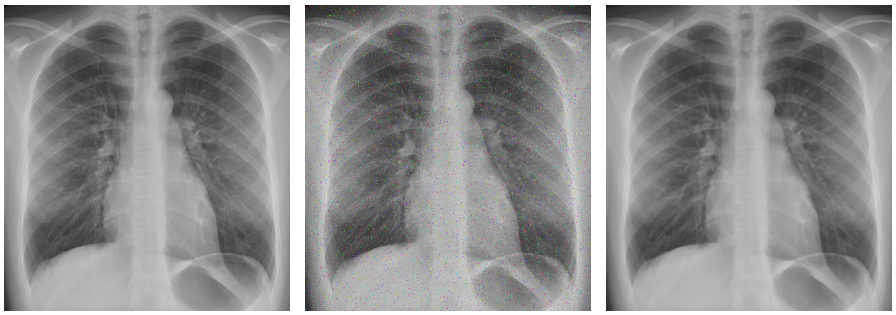
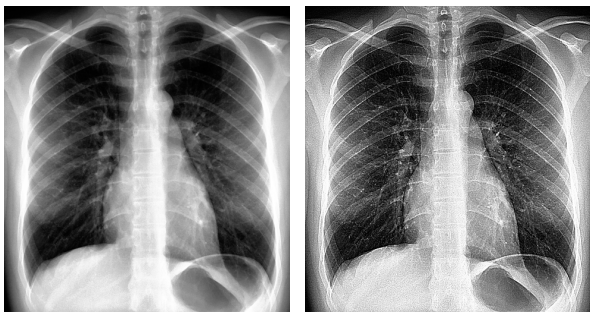
ECE 435, Medical Image Processing
Assignment 2: Image Enhancement

Spring 2020

Medical images can greatly benefit from image enhancement methods: noise reduction, histogram equalization and image sharpening are some examples of techniques that aid in the efficient interpretation of such data. In this assignment, you will study these techniques and design MATLAB scripts and functions that implement them. Figure 1 illustrates the diverse output expected throughout this exercise.



(a) Original image. (b) Salt & Pepper Noise. (c) Median filtered.



(d) Histogram equalized. (e) Sharpened.

Figure 1: Image enhancement in medical images.

1 Image Denoising

Salt & Pepper and Gaussian are two common noise types found in medical images. The Gaussian noise is interpreted as an extra value added to a pixel intensity and can be modeled based on its *mean* and *standard deviation*. The Salt & Pepper noise, on the other hand, presents itself as dark pixels in bright regions, or white pixels in dark regions (abrupt color changes in multi-channel images).

In this section we will investigate an approach suitable for the mitigation of the effects of the Salt & Pepper noise: median filtering. This filter substitutes each pixel intensity by the median of a neighborhood of pixels around it, as illustrated in Figure 2. The shape of the kernel defining this neighborhood (e.g., square, cross-like) influences in the filter's output.

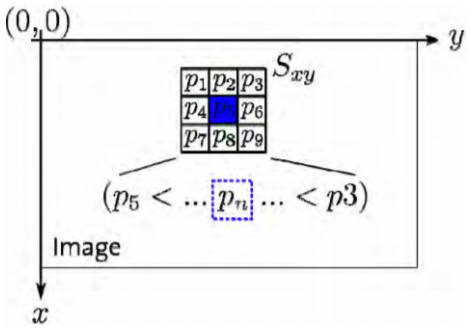


Figure 2: The intensity of pixel P_5 is substituted by the median P_n between the pixels included in a square kernel.

Problem 1: Median Filtering.

(a) (**0.5 point**) Load the 'ChestXRay.png' file (henceforth referred as 'original' image) and add a Salt & Pepper noise with density (i.e., percentage of affected pixels) of 2% to it. Present the noise-plagued version of the image.

Note. You might use MATLAB's *imnoise* function.

(b) (**1 point**) Implement the median non-linear filter. Write the function 'myMedian' to be called using the following form:

$$denoised = myMedian(img, wsize);$$

This function receives the image afflicted by Salt & Pepper noise (img) created in Problem 1(a), and also the size of the square kernel to be used ($wsize$). For example, $wsize = 3$ means that a 3×3 kernel should be created. In order to perform this filtering operation, the input image should be padded by repeating its border elements. Filter the image using $wsize = 3$, present both images

(noisy and denoised) and save them as '1-Noisy.png' and '2-Denoised.png'.

Note. You **should not** MATLAB's *medfilt2* function.

2 Contrast augmentation

Important features of an image might be difficult to identify if its pixel intensities are not well distributed (i.e., it has low contrast). These images are characterized by the concentration of most of their information in one end of a histogram (excessively dark or bright images). By applying a transformation function that generates a more uniform intensity distribution, one is actually enhancing the visibility of the image. A typical global transformation that aims to do that is called *histogram equalization*. This transformation is defined in Equation 1.

$$T_k = \frac{L-1}{NM} H_c(k) \tag{1}$$

Where T_k is the intensity of a pixel in the output image whose original intensity was k, k ranges in the integer interval $[0, L-1]$, N and M are the dimensions of the input image, and H_c is the cumulative histogram of input image. This histogram approximates the discrete distribution function and its calculation is detailed in Problem 2(a).

Problem 2 (1 point): Histogram equalization.

Implement the histogram equalization algorithm. Write the function 'myHistEq' to be called using the following form:

$$[imgHE, orgHist, heHist] = myHistEq(img);$$

Where img represents the input and $imgHE$, $orgHist$ and $heHist$ provide, respectively, the histogram-equalized image, the original image's histogram and the histogram of the equalized output. The algorithm to be implemented by this function is summarized as follows.

1. **Check for image properties**. For simplicity, make sure that the input image is converted to a grayscale, 'uint8' image of 256 intensity levels.
2. **Histogram generation**. Create a graphical representation of the pixel's intensity distribution on the image. Each bin k represents the number of image pixel's with intensity level k.
3. **Probability array**. Divide each bin of the histogram by the total number of image's pixels to obtain an array (P_a) that approximates the probability distribution of each intensity level.

4. **Cumulative histogram (H_c) generation**. Each bin of this histogram (which approximates the discrete distribution function) is represented by the sum of the values of the previous bins in P_a.

5. **Transformation function**. Apply the transformation function described in Equation 1.

Present both images (original and contrast-augmented) and save them as '3-LowContrast.png' and '4-HistogramEqualized.png'. Present the histograms of the original and output images and save them as 'OriginalHistogram.png' and 'EqualizedHistogram.png' (hint: use MATLAB's *bar* function).

Note. You **should not** use MATLAB's *histogram* or *histeq* functions.

3 Image smoothing and sharpening

Images with enhanced edges (i.e., sharpened) better highlight features that can guarantee an effective medical interpretation. In this section we will investigate two sharpening strategies: 1) using a classical sharpening operator, and 2) using a Laplacian of Gaussian (LoG)-based operator. The first approach is sometimes referred to as an *unsharp* filter: a technique that use an unsharp (or smoothed) version of the image in the enhancement process. A regular unsharp filter is illustrated in Figure 3.

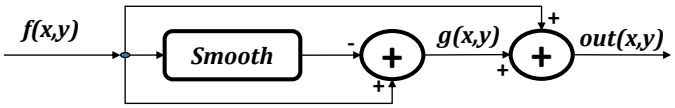


Figure 3: Regular unsharp filter using a smoothing function.

In Figure 3, $f(x, y)$ represents the original image, *smooth* is a function (e.g., Gaussian) that removes the image's high spacial frequency components, $g(x, y)$ offers an 'edge' version of the image, while $out(x, y)$ presents the edge-enhanced (i.e., sharpened) image.

Given that the LoG of an image identify its edges (i.e., $g(x, y)$ in Figure 3), the unsharp filter can alternatively be implemented as in Figure 4.

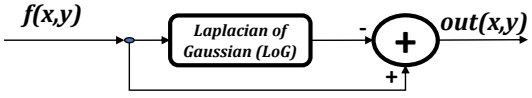


Figure 4: Sharpening with a Laplacian of Gaussian.

Problem 3: Image smoothing.

(a) **(1 point)** Implement the Gaussian smoothing operator. As a first step, you need to calculate the Gaussian kernel G_k described in Equation 2.

$$G_k(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

Where σ is the standard deviation of the Gaussian function and (x, y) are the coordinates inside of the kernel. The $\frac{1}{2\pi\sigma^2}$ term is responsible for the normalization of the kernel, ensuring that its sum is unity for any value of σ. Note that since the Gaussian kernel needs to be isotropic (i.e., circularly symmetric), it should be centered in the $(0, 0)$ coordinate. In other words, for a 3×3 kernel, the coordinates used in Equation 2 should range from $(-1, -1)$ to $(1, 1)$. As a reference, a 3×3 Gaussian kernel with $\sigma = 0.55$ would be $\begin{pmatrix} 0.019 & 0.100 & 0.019 \\ 0.100 & 0.526 & 0.100 \\ 0.019 & 0.100 & 0.019 \end{pmatrix}$.

Write the function 'myGaussian' to be called using the following form:

$$kernel = myGaussian(wsize, sigma)$$

Where *sigma* is the standard deviation of the Gaussian function and *wsize* defines the size of the kernel (similarly to Problem 1(b)).

Using the *myGaussian* function, create and report a 3D representation of a Gaussian kernel with $wsize = 11$ and $\sigma = 1.5$ (hint: use MATLAB's *surf* function), as well as the sum of its values. Apply that kernel in an image using convolution. Present both images (original and smoothed) and save the smoothed version as '5-SmoothedwithGaussian.png'.

Note. You **should not** use MATLAB's *fspecial* function.
Extra (1 point). Implement *myGaussian* without the usage of any *loop*.

(b) **(0.5 point)** Use the *myGaussian* function to calculate and report the sum of values from a Gaussian kernel with $wsize = 5$ and $\sigma = 2$. Is this sum unity? If not, detail why and what could be done to prevent that from happening.

Problem 4 (0.5 point): Gaussian-based image sharpening.

Implement the unsharp filter described in Figure 3 by using an image smoothed with a kernel created with the *myGaussian* function. The 11×11 Gaussian kernel created must have $\sigma = 1.5$. Use the *myHistEq* function to enhance both the sharpened and original images. Present both outputs (i.e., histogram-equalized original and sharpened images) and save the sharpened one as '6-SharpenedwithGaussian.png'.

Problem 5: Laplacian of Gaussian calculation and usage.

(a) **(1 point)** Implement the LoG edge detector. This detector, which combines Gaussian and Laplacian functions, both smooths and detects the edges in an image. An approximation of its effect can be obtained by subtracting two images smoothed with Gaussian kernels using different σ ('Difference of Gaussian'). As a first step, you need to calculate the LoG kernel L_k described in

Equation 3.

$$L_k(x,y) = \frac{x^2+y^2-2\sigma^2}{\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3}$$

Note that the $e^{-\frac{x^2+y^2}{2\sigma^2}}$ term represents a Gaussian function. Therefore, **use your myGaussian function** to calculate this term. Similarly to G_k, L_k should be centered in $(0,0)$. In order to make sure that the convolution between this kernel and an homogeneous region of an image is always zero, **force the sum of the elements from L_k to zero**.

Write the function 'myLoG' to be called using the following form:

$$kernel = myLoG(wsize, sigma)$$

Where *sigma* is the standard deviation of the Gaussian function and *wsize* defines the size of the LoG kernel (similarly to Problem 1(b)). Using *myLoG*, create and report a 3D representation of a LoG kernel with $wsize = 19$ and $\sigma = 3$ (hint: use MATLAB's *surf* function), as well as the sum of its values.

Note. You **should not** use MATLAB's *fspecial* function.
Extra (1 point). Implement *myLoG* without the usage of *loop*.

(b) **(0.5 point)** Implement the LoG-based sharpening operation described in Figure 4. In order to do so, first create an LoG kernel with the *myLoG* function, $wsize = 9$ and $\sigma = 0.6$. Then, use convolution to apply this kernel in the original image and calculate the edge image. Use the *myHistEq* function to enhance both the LoG-sharpened and original images. Present both outputs (i.e., histogram-equalized original and LoG-sharpened images) and save the sharpened one as '7-SharpenedwithLoG.png'.

Deliverables: When submitting your assignment, use the following format for the name of the folder possessing all the required files:
'FirstName_VNumber_Assignment2.zip'. For example,
'Claire_V00888888_Assignment2.zip'.
The folder should include:

- A '.pdf' file with your written report, which includes answers to all the questions and a description of your results. Whenever you are asked to present an image or histogram, please capture a screenshot of them and add to this report.
- The following MATLAB scripts and functions:

  solution.m (executes all the required tasks in the order they are presented in this document)

  myGaussian.m

  myHistEq.m

myLoG.m

myMedian.m

- The following output images:

  '1-Noisy.png'

  '2-Denoised.png'

  '3-LowContrast.png'

  '4-HistogramEqualized.png'

  '5-SmoothedwithGaussian.png'

  '6-SharpenedwithGaussian.png'

  '7-SharpenedwithLoG.png'

  'OriginalHistogram.png'

  'EqualizedHistogram.png'