Assignment 5: CS 663

Due: 31st October before 11:55 pm

Remember the honor code while submitting this (and every other) assignment. You may discuss broad ideas with other students or ask me for any difficulties, but the code you implement and the answers you write must be your own. We will adopt a zero-tolerance policy against any violation.

Submission instructions: Follow the instructions for the submission format and the naming convention of your files from `http://www.cse.iitb.ac.in/~suyash/cs663/submissionStyle.pdf`. However, please do not submit the face image databases in your zip file that you will upload on moodle. Please see `http://www.cse.iitb.ac.in/~ajitvr/CS663_Fall2018/HW5/assignment5_DFT.rar`. Upload the file on moodle before 11:55 pm on 31st October. Policy for late submissions will be the same as in the aforementioned guidelines document. Please preserve a copy of all your work until the end of the semester. Note: You stand to lose 50% of the marks if you submit the assignment within 24 hours after the deadline, after which you do not get any points.

1. Suppose you are standing in a well-illuminated room with a large window, and you take a picture of the scene outside. The window undesirably acts as a semi-reflecting surface, and hence the picture will contain a reflection of the scene inside the room, besides the scene outside. While solutions exist for separating the two components from a single picture, here you will look at a simpler-to-solve version of this problem where you would take two pictures. The first picture g_1 is taken by adjusting your camera lens so that the scene outside (f_1) is in focus (we will assume that the scene outside has negligible depth variation when compared to the distance from the camera, and so it makes sense to say that the entire scene outside is in focus), and the reflection off the window surface (f_2) will now be defocussed or blurred. This can be written as $g_1 = f_1 + h_2 * f_2$ where h_2 stands for the blur kernel that acted on f_2. The second picture g_2 is taken by focusing the camera onto the surface of the window, with the scene outside being defocussed. This can be written as $g_2 = h_1 * f_1 + f_2$ where h_1 is the blur kernel acting on f_1. Given g_1 and g_2, and assuming h_1 and h_2 are known, your task is to derive a formula to determine f_1 and f_2. Note that we are making the simplifying assumption that there was no relative motion between the camera and the scene outside while the two pictures were being acquired, and that there were no changes whatsoever to the scene outside or inside. Even with all these assumptions, you will notice something inherently problematic about the formula you will derive. What is it? [7+8 = 15 points]

   ANSWER:

   See model code ReflectionSeparation.m in the solutions folder, though no code was expected from you for this question. Taking Fourier Transforms, we have $G_1(\mu) = F_1(\mu) + H_2(\mu)F_2(\mu)$ and $G_2(\mu) = H_1(\mu)F_1(\mu) + F_2(\mu)$. Solving these equations simultaneously for $F_1(\mu)$ and $F_2(\mu)$, we get $F_2(\mu) = \frac{G_2(\mu) - H_1(\mu)G_1(\mu)}{1 - H_1(\mu)H_2(\mu)}$ and $F_1(\mu) = \frac{G_1(\mu) - H_2(\mu)G_2(\mu)}{1 - H_1(\mu)H_2(\mu)}$. This solution is well-defined for all values of μ, except if $H_1(\mu)H_2(\mu) = 1$. Now, remember that h_1 and h_2 are low-pass filter kernels, due to the defocussing of the image. These blur kernels will always integrate to 1, i.e. $\int_{-\infty}^{+\infty} h_1(x)dx = 1$ or $\int_a^b h_1(x)dx = 1$ for a blur kernel defined on the interval $[a, b]$. Therefore, we can conclude that $H_1(0) = H_2(0) = 1$ (why? Look at the Fourier transform formula and plug in $\mu = 0$). For all other values of μ, we will have $|H_1(\mu)| \leq 1$ and $|H_2(\mu)| \leq 1$ as this is a low pass filter. This therefore means, that our solution is undefined for those low frequencies, where $H_1(\mu)H_2(\mu) = 1$ (such as $\mu = 0$). Here is a (somewhat rare) situation where reconstruction of higher frequency components is fine, but where lower frequencies (especially the DC component) cannot be recovered robustly. In the case of image denoising, the situation is exactly opposite - the lower frequencies are easy to reconstruct, and hence smooth regions look fine. But the higher frequencies are difficult to reconstruct and hence the loss of finer edges and textures in image denoising. In practice, we would need to add in a small ϵ to the denominators, i.e. we would have $F_2(\mu) = \frac{G_2(\mu) - H_1(\mu)G_1(\mu)}{1 - H_1(\mu)H_2(\mu) + \epsilon}$ and $F_1(\mu) = \frac{G_1(\mu) - H_2(\mu)G_2(\mu)}{1 - H_1(\mu)H_2(\mu) + \epsilon}$. The resultant image would look somewhat artificial due to incorrect reconstruction of lower frequencies, such as what you see below in Figure 1.

Figure 1: Left to right, top to bottom: barbara, mandrill, mixture 1, mixture 2, reconstructed barbara and reconstructed mandrill. Notice that the features of barbara and mandrill are correctly reconstructed. But the image looks as if it were sent through a high pass filter - which is due to error in the reconstruction of the DC component.

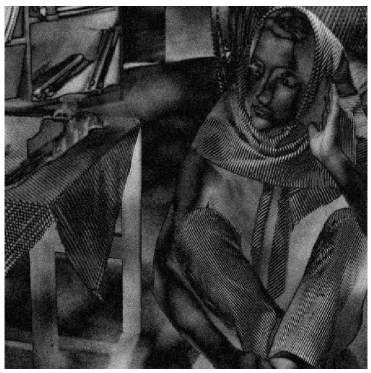
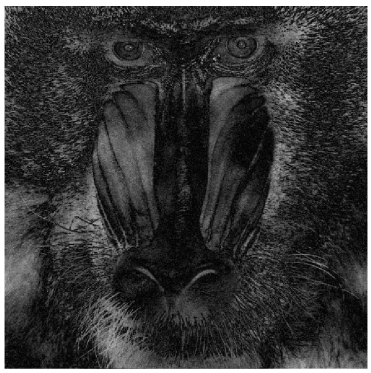

Figure 2: Left to right: reconstructed barbara and reconstructed mandrill when the mixtures had noise. Notice that the features of barbara and mandrill are correctly reconstructed. The reconstructions are not totally different from the earlier results despite the noise. The errors in the DC component, however, remain as before.

Now, if there is image noise, i.e. $g_1 = f_1 + h_2 * f_2 + \eta_1$ and $g_2 = h_1 * f_1 + f_2 + \eta_2$, we incur errors proportional to $\frac{N_1(\mu) - H2(\mu)N_2(\mu)}{1 - H_1(\mu)H_2(\mu)}$ and $\frac{N_2(\mu) - H_1(\mu)N_1(\mu)}{1 - H_1(\mu)H_2(\mu)}$. For higher frequencies, the denominator is large (i.e. close to 1) and hence there is no amplification of the noise, unlike the case with the inverse filter under noise. For lower frequencies, the noise will get amplified especially as $H_1(\mu)H_2(\mu)$ gets closer to 1, but the signal strength is also very high in these frequencies, so the relative error will not totally blow off, unlike the case with the inverse low-pass filter under noise. See below a reconstruction result under noise in Figure 2.

Also take note that the blurring of the images was actually useful over here (somewhat counter-intuitively). Without the blur, we would simply have two identical images and the separation would have been impossible.

MARKING SCHEME: 7 points for correct derivation of formula. While describing the problem with this approach, you must argue that $H_1(0) = H_2(0) = 1$ and hence the lower frequency components (particularly the DC component) are not properly reconstructed. Merely saying that there is a division by 0 when $H_1(\mu) = H_2(\mu) = 1$ will fetch you only 6 points out of 8.

2. Consider a 1D image (for example, a single row from a 2D image). You know that given such an image, computing its gradients is trivial. An inquisitive student frames this as a convolution problem to yield $g = h * f$ where g is the gradient image (in 1D), h is the convolution kernel to represent the gradient operation, and f is the original 1D image. The student tries to develop a method to determine f given g and h. What are the fundamental difficulties he/she will face in this task? Justify your answer. You may assume appropriate boundary conditions. Now consider that you are given the gradients of a 2D image in the X and Y directions, and you wish to determine the original image. What are the difficulties you will face in this task? Justify your answer. Again, you may assume appropriate boundary conditions. [5+5 = 10 points]

ANSWER:

The convolution kernel can be typically given as $[-1, 1]$, yielding $g(x) = f(x+1) - f(x)$ for $1 \leq x \leq N$. We know that the DFT of g is given by $G(u) = (e^{\frac{j2\pi u}{N}} - 1)F(u)$, which yields $F(u) = \frac{G(u)}{e^{\frac{j2\pi u}{N}} - 1}$. The problem with using this formula to estimate $F(u)$ is that for $u = 0$ we have zero in the denominator. This leads to problems in estimating the DC component of F. So you have to make some assumption about what you expect the DC component to be. This is equivalent to assuming a boundary condition for the gradients, i.e. if f has N elements, we will assume $f(N+1) = 0$. An alternative method is to assume a boundary condition and simply integrate the image from the rightmost pixel leftwards.

Given a 2D image, it is trivial to compute the gradients. Given the gradients in the X and Y directions, it is not easy to get back the image. The DFT of I_x is $\hat{F_x}(u,v) = (e^{\frac{j2\pi u}{N}} - 1)\hat{F}(u,v)$ where $F(u,v)$ is the FT of image f. To estimate $F(u,v)$ from $\hat{F_x}(u,v)$, the problem again occurs for the frequency components with $u = 0$. Similarly, the DFT of f_y is $\hat{F_y}(u,v) = (e^{\frac{j2\pi v}{N}} - 1)F(u,v)$. To estimate $F(u,v)$ from $\hat{F_y}(u,v)$, the problem occurs for the frequency components with $v = 0$. If you knew both $\hat{F_y}(u,v)$ and $\hat{F_x}(u,v)$, the problem still occurs for the component $u = v = 0$, which is the DC component. Again you need to make some assumptions

on what this DC component is. If you use the integration based method using boundary conditions, there may be inconsistent answers when you use f_x and f_y.

(Note: Here we have assumed that we had accurate estimates of the image gradients f_x and f_y. But sometimes, you may get noisy estimates of the image gradients either as the output of some computational procedure, or else (rarely) using gradient cameras such as those in `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.415.6700&rep=rep1&type=pdf` (the title of the article is 'Why I want a gradient camera?'.) This makes this problem much more complicated than what we have seen so far. Want to figure out what that is?).

Marking scheme: 5 points for the first part with the 1D image for either the Fourier based solution or the simple computational solution using digital integration (i.e. addition). In each case, the problem faced must be properly outlined. 5 points for the second part - where the analysis must show that the problem now exists for $u = v = 0$ only. For the second part, if you write a solution using integration across rows or columns, you lose 3 points if you don't mention that the solution using integration across rows will be inconsistent with the solution using integration across columns.

3. Consider the image with the low frequency noise pattern shared in the homework folder in the form of a .mat file. Your task is to (a) write MATLAB code to display the log magnitude of its Fourier transform, (b) to determine the frequency of the noise pattern by observing the log magnitude of the Fourier transform and guessing the interfering frequencies, and (c) to design and implement (in MATLAB) an ideal notch filter to remove the interference(s) and display the restored image. To this end, you may use the fft2, ifft2, fftshift and ifftshift routines in MATLAB. [10 points]
ANSWER: The code is in the file notchfilter.m in the homework folder. The interfering pattern has the equation $Z = 50\sin(X/8+2*Y/8)$, leading to some unnatural peaks in the frequency domain at two frequencies. These peaks can be determined by inspection and removed using an ideal band-reject filter. (A Gaussian band-reject filter could also have been used).
MARKING SCHEME: 4 points for identifying the peaks and 6 points for implementing the notch filter.

4. Consider the barbara256.png image from the homework folder. Implement the following in MATLAB: (a) an ideal low pass filter with cutoff frequency $D \in \{40, 80\}$, (b) a Gaussian low pass filter with $\sigma \in \{40, 80\}$. Show the effect of these on the image, and display all images in your report. Display the frequency response (in log Fourier format) of all filters in your report as well. Comment on the differences in the outputs. Make sure you perform appropriate zero-padding! [15 points]
ANSWER: The code for the ILPF and GLPF is lpf.m (with appropriate zero-padding) and lpf2.m (without appropriate zero padding).
MARKING SCHEME: 7 points for ILPF and 8 points for GLPF. Deduct 3 points for ILPF and 3 points for GLPF if appropriate zero-padding was not performed. To convolve two signals of size $N_1 \times N_2$ with each other, both need to be zero-padded to form signals of size $(2N_1 - 1) \times (2N_2 - 1)$. Hence the frequency domain filters must be of size $(2N_1 - 1) \times (2N_2 - 1)$ to avoid aliasing.

5. In this part, we will apply the PCA technique for the task of image denoising. Take the barbara256.png image present in the corresponding data/ subfolder - this image has gray-levels in the range from 0 to 255. Add zero mean Gaussian noise of $\sigma = 20$ to it using the MATLAB code 'im1 = im + randn(size(im))*20'. (Do not clamp the values in im1 to the [0,255] range as that alters the noise statistics). Note that this noise is image-independent. If during the course of your implementation, your program takes too long, you can instead work with the file barbara256-part.png which has size 128 by 128 instead of 256 by 256.

   (a) In the first part, you will divide the entire noisy image im1 into overlapping patches of size 7 by 7, and create a matrix $\mathbf{P}$ of size $49 \times N$ where N is the total number of image patches. Each column of $\mathbf{P}$ is a single patch reshaped to form a vector. Compute eigenvectors of the matrix $\mathbf{P}\mathbf{P}^T$, and the eigen-coefficients of each noisy patch. Let us denote the j^{th} eigen-coefficient of the i^{th} (noisy) patch (i.e. $\mathbf{P}_i$) by α_{ij}. Define $\bar{\alpha}_j^2 = \max(0, \frac{1}{N}[\sum_{i=1}^{N} \alpha_{ij}^2] - \sigma^2)$, which is basically an estimate of the average squared eigen-coefficients of the 'original (clean) patches'. Now, your task is to manipulate the noisy coefficients $\{\alpha_{ij}\}$ using the following rule, which is along the lines of the Wiener filter update that we studied in class: $\alpha_{ij}^{\text{denoised}} = \dfrac{\alpha_{ij}}{1 + \frac{\sigma^2}{\bar{\alpha}_j^2}}$. Here, $\alpha_{ij}^{\text{denoised}}$ stands for the j^{th} eigencoefficient of the i^{th} denoised patch. Note that

$\frac{\sigma^2}{\bar{\alpha}_j^2}$ is an estimate of the ISNR, which we absolutely need for any practical implementation of a Wiener filter update. After updating the coefficients by the Wiener filter rule, you should reconstruct the denoised patches and re-assemble them to produce the final denoised image which we will call 'im2'. Since you chose overlapping patches, there will be multiple values that appear at any pixel. You take care of this situation using simple averaging. Write a function `myPCADenoising1.m` to implement this. Display the final image 'im2' in your report and state its RMSE computed as $\frac{\|im2_{denoised} - im2_{orig}\|_2}{\|im2_{orig}\|_2}$.

(b) In the second part, you will modify this technique. Given any patch $\mathbf{P}_i$ in the noisy image, you should collect $K = 200$ most similar patches (in a mean-squared error sense) from within a 31×31 neighborhood centered at the top left corner of $\mathbf{P}_i$. We will call this set of similar patches as Q_i (this set will of course include $\mathbf{P}_i$). Build an eigen-space given Q_i and denoise the eigen-coefficients corresponding to **only** P_i using the Wiener update mentioned earlier. The only change will be that $\bar{\alpha}_j^2$ will now be defined using only the patches from Q_i (as opposed to patches from all over the image). Reconstruct the denoised version of P_i. Repeat this for every patch from the noisy image (i.e. create a fresh eigen-space each time). At any pixel, there will be multiple values due to overlapping patches - simply average them. Write a function `myPCADenoising2.m` to implement this. Reconstruct the final denoised image, display it in your report and state the RMSE value.

(c) Now run your bilateral filter code `myBilateralFiltering.m` from Homework 2 on the noisy version of the barbara image. Compare the denoised result with the result of the previous two steps. What differences do you observe? What are the differences between this PCA based approach and the bilateral filter?

(d) Now consider that the noise in the image was Poisson distributed. In fact, the Poisson noise model is known to be the dominant noise model in photographic as well as X-ray imaging. In this problem, we use the MATLAB command 'im1 = poissrnd(im)', i.e. every pixel of 'im1' is a Poisson-corrupted version of the corresponding pixel in'im'. There exists the well-known Anscombe transform for Poisson noise, which states that if $J \sim \text{Poisson}(I)$, then $\sqrt{J} \sim \mathcal{N}(\sqrt{I}, 1/4)$, that is $\sqrt{J}$ is approximately Gaussian distributed with variance 1/4 and mean $\sqrt{I}$. This approximation gets more and more accurate as $I \to \infty$, but it is often used as is for smaller values of I. You should use the routine for Gaussian denoising developed in part (b) above for denoising $\sqrt{J}$. Display the final image obtained after inverting the square root, in your report. Also report the RMSE. Repeat the exercise if we have 'im1 = poissrnd(im/20)'. This actually represents image acquisition with a lower exposure time. How does the result of this compare with the earlier case (when im was not divided by 20)? Why? Explain. [10 + 10 + 5 + 10 = 35 points]

ANSWER: See model code in the homework solutions folder. The reconstruction results are given below in Figure 3. For part (a), the MSE is around 94, and the image looks extremely blurred even though the noise is removed. For part (b), the results are given below for two case $K = 25$ and $K = 200$. The respective MSEs are 190 and 60, and the result with $K = 200$ is far superior. This PCA-based method in part 3(b) is one of the much better denoising methods available today. Compare the results with the output of the bilateral filter to see how much better the texture preservation is.

So what is going on? When you apply a filter, there is an implicit assumption you make about the image. This assumption is often ignored in the literature! If you did a simple mean filter, the assumption is that the underlying image had a constant intensity. If that is violated, i.e. there are edges separating distinct regions, the edges will get blurred away too! The next best is to do a piecewise mean filter, or a weighted piecewise mean filter, which is what the bilteral filter is. The underlying assumption is that the image is piecewise constant. The bilateral filter preserves significant edges well, but obliterates subtle textures. In fact, if you had an image that was piecewise linear instead of piecewise constant (i.e. within a single region, the image intensity was of the form $I(x, y) = ax + by + c$ for some constants a, b and c - this looks like a shading with constant slope), you would lose them depending on the parameters of the bilateral filter.

The PCA filter makes no such assumption - be it piecewise linear or piecewise constant. Instead it assumes that given any patch P_r of small size such as 8×8, there exist patches elsewhere in the image which are structurally very similar to it. This is true for many natural images (see barbara for instance, there is so much repetition of patterns on her hair, her jeans, the tablecloth, the bookshelf and the background on the top right corner!). For noise levels whose standard deviation is much less than the average image intensity, this sort of self-similarity is still preserved even in the noisy image. So the method in part (b) says - why not collect these similar patches,

Figure 3: Left to right, top to bottom:Original image; noisy image (under zero mean, iid Gaussian noise with $\sigma = 20$); image reconstructed usign global PCA, i.e. part (a); image reconstructed using spatially vaying PCA, i.e. part (b), with $K = 25$; image reconstructed using spatially vaying PCA, i.e. part (b), with $K = 200$; result with bilateral filtering for 20 iterations with $\sigma_{spatial} = 8$ and $\sigma_{intensity} = 8$; result with bilateral filtering for 20 iterations with $\sigma_{spatial} = 8$ and $\sigma_{intensity} = 5$. Second last row: left - Poisson noisy image when intensities were divided by 20, right: denoised image using the PCA based method (RMSE: 0.12). Last row: left - Poisson noisy image when intensities were divided by 1, right: denoised image using the PCA based method (RMSE: 0.038).

denote them as a group $\{P_i\}_{i=1}^{K}$, and build an eigenspace from them?
For part (d), we have Poisson noise which is converted to approximately gaussian via the square root transform. The denoising routine will perform much better when the intensity is higher. This is because for a Poisson random variable with mean λ, the variance is also λ and hence the standard deviation is $\sqrt{\lambda}$. Hence the noise to signal ratio is $1/\sqrt{\lambda}$, i.e. higher the mean value, lower is the noise to signal ratio, and vice versa.

MARKING SCHEME: 10 points for correct implementation of part (a) and 10 points for part (b). part (c): 5 points for a sensible explanation of the difference between PCA and bilateral filtering. In particular, you should state that bilateral filtering performs local averaging with parameters not immediately tied to noise level whereas PCA is specifically designed for a particular noise model. Part (d): 6 points for proper implementation and 4 points for the NSR argument. Deduct 2 points if the image was not squared (inverse of square-root transform) at the end after Gaussian denoising with variance 0.25. Also, for all four parts: For every RMSE value not mentioned in the report and not directly printed by the code, deduct 1 point.

6. Read Section 1 of the paper 'An FFT-Based Technique for Translation, Rotation, and Scale-Invariant Image Registration' published in the IEEE Transactions on Image Processing in August 1996. A copy of this paper is available in the homework folder. Implement the technique in Equation 3 of the paper to align two images which are related to each other by a 2D in-plane translation. Test your implementation on images I and J as follows. I is a 300×300 image containing a 50×70 white rectangle (intensity 255) whose top-left corner lies at pixel $(50, 50)$. All other pixels of I have intensity 0. The image J is obtained from a translation of I by values $(t_x = -30, t_y = 70)$. Verify carefully that the predicted translation agrees with the ground truth translation values. Repeat the exercise if I and J were treated with iid Gaussian noise with mean 0 and standard deviation 20. In both cases, display the logarithm of the Fourier magnitude of the cross-power spectrum in Equation 3 of the paper. What is the time complexity of this procedure to predict translation if the images were of size $N \times N$? How does it compare with a pixel-wise image comparison procedure for predicting the translation? [15 points]
ANSWER: The inverse Fourier transform of the cross-spectrum should peak at the correct value of the translation, which in this case is $(270, 70)$, since $300 - 30 = 270$. In MATLAB, the peak will be at $(271, 71)$. The peak is preserved even under addition of noise, though the inverse Fourier transform now contains a great deal of noisy artifacts.
The time complexity of this procedure is $O(N^2 \log N)$ for an $N \times N$ image. A pixel-wise translation prediction will have time complexity $O(N^2 W^2)$ wheer $W \times W$ is the window size for the range of translations.
Marking scheme: 5 points for time complexity, 6 points for code implementation and 4 points for correct prediction of the translation (with justification - you need to check that the translation is indeed predicted by the code).