ECE 5273
Homework 5

Spring 2023 Dr. Havlicek

1. Obtain the image `salesman.bin` from the course web site. This is a 256×256 grayscale image with 8-bit pixels. In this problem you will use three different methods to apply a 7×7 linear low-pass average filter to the image. The resulting filtered images should be exactly the same in all three cases.

   The filter impulse response is a 7×7 square where each pixel is equal to 1/49. In terms of the notation on page 2.54 of the course notes, the filter window is SQUARE(49). The reference point is located in the center, so that the output image is not shifted relative to the input image.

   Note: this is a linear translation invariant (LTI) filter. The output is linear convolution, *not* circular convolution.

   (a) Implement the 7×7 linear average filter using image domain convolution. Set each pixel in the output image equal to the average of the pixels in a 7×7 neighborhood about the corresponding pixel in the input image. Handle edge effects by zero padding (i.e., use a value of zero for pixels where the window "hangs over" the edges of the input image). Show the input and output images with full scale contrast.

   Note: to obtain agreement with the DFT filtering methods that you will implement in parts (b) and (c), you need to apply the filter at *every* pixel of the input image. At all four edges of the image, this creates an issue about how to implement the zero padding. One way to handle the issue is to treat the first three rows, the last three rows, the first three columns, and the last three columns as special cases. An alternative way that is probably easier is the following: add three rows of all zero pixels to the top and bottom of the image and add three columns of all zero pixels to the left and right sides of the image. Then you will have a new image of size 262×262. You can then apply the filter to this new enlarged image starting at pixel (4,4) (for Matlab indexing; for C indexing it is pixel (3,3)) and ending at pixel (259,259) (for Matlab indexing; for C indexing it is pixel (258,258)).

   (b) Implement the same filter by pointwise multiplication of DFT's using the method of Example 3 on pages 5.61-5.62 of the course notes. In this case, your impulse response image **H** will be of size 128×128 and will have all pixels equal to zero except for a 7×7 square of pixels in the center that will all be equal to 1/49. As in Example 3, we have $(p_0, q_0) = (64, 64)$, which is Matlab $(\text{row}, \text{col}) = (r, s) = (65, 65)$. This means that the *center* of the square (the reference point shown in green on page 2.54 of the course notes) will be for $(m, n) = (0, 0)$ and should be located at $(p, q) = (64, 64)$ in the $\mathbf{H}$ image (which is again $(r, s) = (65, 65)$ in Matlab).

Note: according to Example 3 on pages 5.61-5.62 of the course notes, the zero-padded size should be 383×383 (since $256 + 128 - 1 = 383$). If you are using C and the FFT routine `fft2d()` from the course web site, this will cause a problem because `fft2d()` assumes that the size is *even*. To solve this problem, increase the zero-padded size by one to 384×384.

Show the following images with full scale contrast:

- The original input image.
- The zero padded original image.
- The zero padded impulse response image.
- The centered DFT log-magnitude spectrum of the zero padded input image.
- The centered DFT log-magnitude spectrum of the zero padded impulse response image.
- The centered DFT log-magnitude spectrum of the zero padded output image.
- The zero padded output image.
- The final 256×256 output image.

Verify that the output image (obtained after performing a full-scale contrast stretch and rounding each pixel to the nearest integer) is the same as the one in part (a). In Matlab, if the output image from part (a) is in the array `Y1a` and the output image from this part is in the array `Y1b`, then you can do this using the Matlab statement

```
disp(['(b): max difference from part (a): ' num2str(max(max(abs(Y1b-Y1a))))])
```

(c) Implement the same filter again using the zero-phase impulse response and DFT method of Example 5 on page 5.76 of the course notes. This method is explained on pages 5.66-5.75. As in Example 5, use a size of 256×256 for the impulse response image $\mathbf{H}$. Your solution should be almost the same as the Matlab code in Example 5, except that you will replace the Gaussian impulse response in lines 3 and 4 with a 7×7 square of pixels that are equal to $1/49$.

To do this in Matlab, initialize the original non-zero-phase impulse response (`h` in Example 5) to all zeros and then set elements $(126{:}132, 126{:}132)$ equal to $1/49$. This will give you a 7×7 square of pixels with value $1/49$ centered at $(p_0, q_0) = (128, 128)$, which is Matlab $(\text{row}, \text{col}) = (r, s) = (129, 129)$. Then use the Matlab `fftshift` function to obtain the zero-phase impulse response image exactly as in Example 5. If you are using C or some other language then you will need to program the equivalent operations yourself, but you can still use the example Matlab code on page 5.76 as a guideline.

Show the following images with full scale contrast:

- The original input image.
- The 256×256 zero-phase impulse response image (this is `h2` in Example 5).
- The 512×512 zero padded zero-phase impulse response image (this is `h2ZP` in Example 5).
- The final 256×256 output image.

Verify that the output image (obtained after performing a full-scale contrast stretch and rounding each pixel to the nearest integer) is the same as the one in part (a).

2. Obtain the images `girl2.bin`, `girl2Noise32Hi.bin`, and `girl2Noise32.bin` from the course web site. All three are 256×256 grayscale images with 8-bit pixels.

   - `girl2.bin` is the original image, also known as *Tiffany*.
   - `girl2Noise32Hi.bin` has hi-pass Gaussian white noise added. The DFT of the noise is nonzero for $\sqrt{u^2 + v^2} > 64$ cpi and zero for $\sqrt{u^2 + v^2} \leq 64$.
   - `girl2Noise32.bin` has broadband Gaussian white noise added.

   (a) Read and display all three images. Compute the MSE of `girl2Noise32Hi` and `girl2Noise32` relative to `girl2` using the formula given on page 5.114 of the course notes.

   (b) Apply an isotropic ideal low-pass filter as on page 5.111 of the course notes with $U_{\text{cutoff}} = 64$ cpi to all three images. Since the cutoff frequency of the hi-pass noise is known exactly in this case, it is best to implement the filter using circular convolution in order to avoid the edge effects associated with zero padding and linear convolution. Therefore, for all three images, use the formula on page 5.111 of the notes to make a 256×256 $\widetilde{\mathbf{H}}$ DFT array. In Matlab, you can do it like this:

```
U_cutoff = 64;
[U,V] = meshgrid(-128:127,-128:127);
HLtildeCenter = double(sqrt(U.^2 + V.^2) <= U_cutoff);
HLtilde = fftshift(HLtildeCenter);
```

   Then, simply pointwise multiply this $\widetilde{\mathbf{H}}$ array times the 256×256 DFT array of each image to obtain the three filtered images.

   Display each filtered image and compute the MSE (relative to the original `girl2` image) using the formula on page 5.114 of the course notes. For the two noisy images, also compute the ISNR using the formula on page 5.115 of the course notes. You should display the images with full scale contrast. However, for computing the MSE and ISNR, it is **important** to use the floating point images obtained directly from the inverse DFT without rounding and without applying any full scale contrast stretch.

You should find that this ideal LPF is very effective for removing the hi-pass noise in `girl2Noise32Hi`. However, it will introduce ringing artifacts in the original image `girl2` and it will not work as well on the broadband noise in `girl2Noise32`.

Note: *if* you are using C, and *if* you are using the routine `fft2d()` from the course web site, then you may run into an issue here in Problem 2(b) (if you are using Matlab, then you can ignore this paragraph). The reason you may run into an issue is that `fft2d()` is designed to input and output *centered* DFT's. On a forward transform, it performs an fftshift operation after calling the 2D FFT. On a reverse transform, it performs an fftshift before calling the 2D FFT. Now, for doing DFT domain filtering, it does not really matter if you are using centered DFT's or un-centered DFT's, *as long as you use* `fft2d()` *to compute both transforms: the transform of the input image and the transform of the filter.* However, if, as we are doing here in Problem 2(b), you are only going to use `fft2d()` for *one* of the transforms, then you have to be careful! Here in Problem 2(b), you are going to compute the $\widetilde{\mathbf{H}}$ image yourself *without* calling `fft2d()`. So, if you *are* going to call `fft2d()` for the DFT of the input images, then realize that they will be *centered* DFT's. This means that you must also compute the $\widetilde{\mathbf{H}}$ image as a *centered* DFT. Otherwise, you will get the wrong answer when you multiply the two DFT's pointwise. That is a difference between using C and `fft2d()` as compared to the Matlab code shown above.

(c) Apply the Gaussian low-pass filter on page 5.113 of the course notes to all three images with $U_{\text{cutoff}} = 64$. Use the bottom formula with $N = 256$ and $U_{\text{cutoff}} = 64$ to design the space constant σ. Then use the top formula to make a 256×256 $\widetilde{\mathbf{H}}$ DFT array. Apply `fftshift` to "un-center" this DFT array and invert to obtain the 256×256 zero phase impulse response (as explained in the note for part (b), if you are using C and `fft2d()` then you do *not* have to un-center the DFT array before inverting). Apply `fftshift` again to the zero phase impulse response to center it. Then use the method of Example 4 on pages 5.64 and 5.65 of the course notes to obtain the three filtered images.

Here is an example of how to use Matlab to compute the 512×512 zero padded impulse response image:

```
U_cutoff_H = 64;
SigmaH = 0.19 * 256 / U_cutoff_H;
[U,V] = meshgrid(-128:127,-128:127);
HtildeCenter = exp((-2*pi^2*SigmaH^2)/(256.^2)*(U.^2 + V.^2));
Htilde = fftshift(HtildeCenter);
H  = ifft2(Htilde);
H2 = fftshift(H);
ZPH2 = zeros(512,512);
ZPH2(1:256,1:256) = H2;
```

You can then take the 512×512 DFT of `ZPH2` and multiply it pointwise with the DFT's of the 512×512 zero padded images to obtain the DFT's of the three zero padded filtered images.

Display each 256×256 filtered image and compute the MSE using the formula on page 5.114 of the course notes. For the two noisy images, also compute the ISNR using the formula on page 5.115 of the course notes. As in part (b), for computing the MSE and ISNR it is **important** to use the floating point images obtained directly from the inverse DFT without rounding and without applying any full scale contrast stretch.

Compared to the ideal LPF, you should find that this filter produces a much better MSE on the original image since it does not suffer from ringing artifacts. However, this filter will not perform as well as the ideal LPF on the noisy images.

(d) Use the method of Example 4 on pages 5.64 and 5.65 of the course notes to apply the Gaussian low-pass filter on page 5.113 of the course notes to all three images again, this time with $U_{\text{cutoff}} = 77.5$.

Display each 256×256 filtered image and compute the MSE using the formula on page 5.114 of the course notes. For the two noisy images, also compute the ISNR using the formula on page 5.115 of the course notes. As in part (b), for computing the MSE and ISNR it is **important** to use the floating point images obtained directly from the inverse DFT without rounding and without applying any full scale contrast stretch.

You should find that this filter distorts the original image much less than the other two filters. Overall, this filter will perform better than the other two against the broadband noise in `girl2Noise32`.

DUE: 4/12/2023, 11:59 PM