**Assignment 3 Solution  
Computer Vision (CS-559)**

**Name: Dhaval Harish Sharma  
Red ID: 824654344**

**(1) (a) Let f(x,y) be an image. Let h(x,y) be the image obtained by applying the a 3 by 3 spatial low pass mask (averaging filter) to f(x,y). Similarly let g(x,y) be the image obtained by applying a 3 by 3 spatial high pass mask to f(x,y). Prove that g(x,y) = f(x,y) - h(x,y)  
Note: one or more examples do not constitute a proof.  
(b) Is the high pass mask separable? What is the implication of separability on computations?**

**Solution:** (a) The averaging filter smoothes the image and tends to blend or average out the noise with the image. The mask for this filter is typically of the form,  
   
where, m is the size of the mask and all the entries of the mask are 1. Typically, the value of m is 3.

The convolution of this mask with the input image results in an output image h(x,y):  
h(x,y) = 1/m2 [f(x - m/2, y - m/2) +…+ f(x - 1, y) + f(x, y - 1) + f(x,y) + f(x + 1, y) + f(x, y + 1) +…+ f(x + m/2, y + m/2)]

Since the above operation is equivalent to finding the mean gray level values over all, the mask is also called a mean mask or mean filter.

Unsharp masking (edge crisping) is used to make edges in the image sharper which makes the image more pleasing to the eye. Unsharp making performs the following task:  
Original image -> blur with low pass filter -> scale with k<1 -> subtract from original image -> scale for display. Note that this filter emphasizes on the center pixel. The mask of this filter is of the form,  
   
where, ≥ 8 for the above mask, and in general so that 𝑐 ≥ 𝑚2 − 1. This implies that when the mask is applied to slowly varying gray level area, the result of the convolution is zero. However, when the gray level changes rapidly, the output image emphasizes these changes.

The proof for the statement that states sharpening is equivalent to subtracting the image from a blurred version of that image is as follows,  
g(x,y) = f(x,y) + h(x,y)  
R.H.S. = f(x,y) + h(x,y)  
 = f(x,y) – (1/9) [f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) + f(x,y-1) + f(x,y) + f(x,y+1) + f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)]  
 = f(x,y) – (1/9) f(x,y) – (1/9) [f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) + f(x,y-1) + f(x,y+1) + f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)]  
 = (8/9) f(x,y) - (1/9) [f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) + f(x,y-1) + f(x,y+1) + f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)]  
 = (1/9) [8f(x,y) + f(x-1,y-1) + f(x-1,y) + f(x-1,y+1) + f(x,y-1) + f(x,y+1) + f(x+1,y-1) + f(x+1,y) + f(x+1,y+1)] = R.H.S.  
Hence, it is proved that the effect of applying the high pass filter is equivalent to subtracting the image from the averaged image.

(b) A separable filter in image processing can be written as a product of two more simple filters. Typically, a 2 dimensional convolution operation is separated in to two 1 dimensional filters. This reduces the cost of computing the operator. The high pass mask is having -1 value in the whole mask except for the one in the centre. It cannot be separated in to two 1 dimensional filters because there are no two 1 dimensional filters that exists which satisfy the values of high pass mask after being multiplied. Also, the high pass mask is not having symmetrical values which means that it cannot be converted in to two 1 dimensional filters.

By separating the mask in to two 1 dimensional filters, we can reduce the number of operations performed on the image for convolving it with the respective filter. In the case of traditional filters, for a mask of size mxm and an image of size nxn, it requires **m2n2** multiplication + **(m2 – 1)n2** additions giving a complexity of **O(m2n2)** for the image or **O(m2)** per pixel. But, after separating the filters, each of the two convolutions requires **m** multiplication and **(m-1)** addition per pixel; giving a total of **2m** multiplication and **2(m-1)** additions per pixel. This proves that in the case of a separable mask, the complexity has been reduced from **O(m2)** to **O(m)** per pixel, and to **O(mn2)** for the image.

**2) Suppose that the image gray level values under a3 X 3 mask are   
 3 2 1  
 7 8 4  
 3 6 5**

**Determine the value of the corresponding pixel in the output image for:   
(a) Median, (b) Harmonic mean.  
In each case comment on the suitability of the filter for reducing Gaussian noise, and provide reasoning for your comments.**

**Solution:**In median filtering, the value of the output pixel g(x,y) is the median of the values under the mask with the center of the mask placed at (x,y). Note that the median is the middle value in the ordered set of values and is the |m2 / 2 | + 1 largest value in the set.

After sorting the image grey level values under the mask, we get  
1, 2, 3, 3, 4, 5, 6, 7, 8.

So, the median filter will select the 5th element from this sorted list which is “4”.

The formula for harmonic mean is g = m2 / ∑ (1 / f)   
where,  
m is the number of pixels in the window that is 9  
f is the input image grey level value.

Applying the harmonic mean on the given question, we get  
g = (3)2 / (1 / 1) + (1 / 2) + (1 / 3) + (1 / 3) + (1 / 4) + (1 / 5) + (1 / 6) + (1 / 7) + (1 / 8)   
g = 9 / 3.05  
g = 2.9508

But, the grey level can never be a decimal value, so we convert it to integer value which is “3”.

The Gaussian noise is very common and models natural noise processes such as electronic noise. Its normalized histogram is described by the Gaussian distribution,  
  
where l is the gray level (usually 0<= l <= 255 for an 8-bit gray level image), is the average input image gray level, and is the standard deviation of lave input gray levels. The histogram is bell shaped.

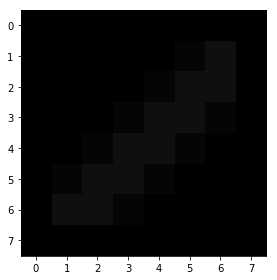
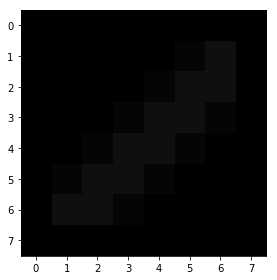
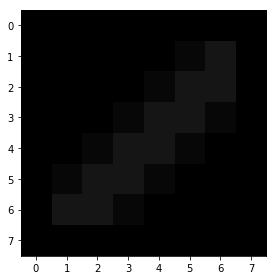
The median filter is not suitable for removing Gaussian noise because most of the grey levels affected by Gaussian noise lie in the middle of the histogram, so if we use median filter then there is a high probability to pick the pixel with noise. Whereas, the harmonic mean filter can be useful for processing Gaussian noise as it tends to average the pixels with a better immunity to noises than the average filter.

**(3) Find the output images if Sobel edge operators are applied to the following 8 by 8 input image. Note that you will have three gradient images, one in x-direction, one in y-direction and one gradient magnitude. Ignore the border effects, and produce only 6 by 6 output images.  
 2 2 2 2 2 2 2 2  
 2 2 2 2 2 2 2 7  
 2 2 2 2 2 2 7 7  
 2 2 2 2 2 7 7 7  
 2 2 2 2 7 7 7 7  
 2 2 2 7 7 7 7 7  
 2 2 7 7 7 7 7 7   
 2 7 7 7 7 7 7 7**

**Solution:**The gradient is calculated using values of two pixels, one just before the current pixel (x, y) and one immediately after the current pixel. If one of these pixels has an error due to noise, then the gradient will be in error. To avoid this problem, Prewitt suggested that the gradient be calculated based on six surrounding pixel values with equal emphasis of all the neighboring pixels. But, Sobel gave a similar set of masks giving more emphasis on the centre pixels. His masks can be given as follows:

**Mx =** -1 0 1, **My =** -1 -2 -1-2 0 2,0 0 0-1 0 11 2 1

There is a program attached to the folder of this document which consists of code to find the sobel edge filter on the input image given in the question. It convolves the image using different kernels in order to find gradient of X, gradient of Y and the magnitude.

The output images formed by applying sobel operator to the input image are as follows:  
    
 Gradient X Gradient Y Magnitude

**(4) Compute the Fourier transform of the one-dimensional image f(0) = 8, f(1) = 4, f(2) = 2, f(3) = 1. Find Fourier spectrum |F(u)|. Comment on your results.**

**Solution:**Fourier transform provides useful information about the frequency contents of an image, and is used to filter the noise, which is associated with the high frequency components in an image. The formula for the fourier transform in 1-D is given as,

We are given that f(0) = 8, f(1) = 4, f(2) = 2, f(3) = 1 which gives the input spatial image as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 4 | 2 | 1 |

Let us start by calculating F(0) as below,  
F(0) = ¼ [8e0 + 4e-j2ᴨ(0) + 2e0 + 1e0]  
 = ¼ [8 + 4 + 2 + 1]  
F(0) = 15/4  
|F(0)| = 15/4

F(1) = ¼ [8e0 + 4e-j2ᴨ/4 + 2e-j2ᴨ2/4 + e-j2ᴨ3/4]  
 = ¼ [8 + 4e-jᴨ/2 + 2e-jᴨ + e-j3ᴨ/2]  
where,  
e-jᴨ/2 = cos(ᴨ /2) –jsin(ᴨ /2) = -j  
e-jᴨ = cos(ᴨ) – jsin(ᴨ) = -1  
e-j3ᴨ/2 = cos(3ᴨ/2) –jsin(3ᴨ/2) = j  
 = ¼ [8 – 4j -2 + j]  
 = ¼ [6 – 3j]  
 F(1) = 3/2 – 3/4j  
|F(1)| = √(9/4 + 9/16) = √(45/16) = ¾ √5

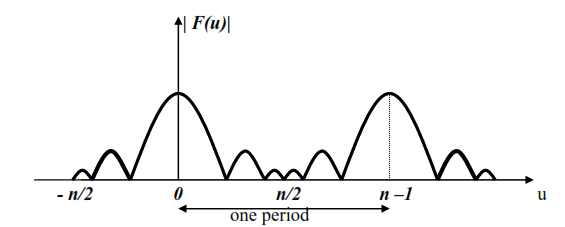
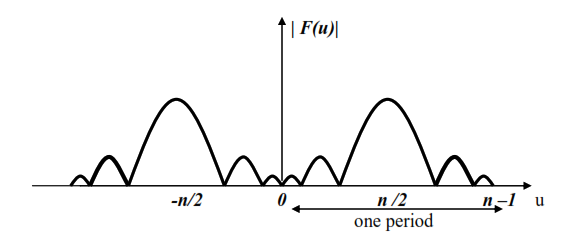
F(2) = ¼ [8e0 + 4e-j2ᴨ2/4 + 2e-j2ᴨ2.2/4 + e-j2ᴨ2.3/4]  
 = ¼ [8 + 4e-jᴨ + 2e-j2ᴨ + e-j3ᴨ]  
 = ¼ [8 – 4 + 2 – 1]  
F(2) = 5/4  
|F(2)| = 5/4

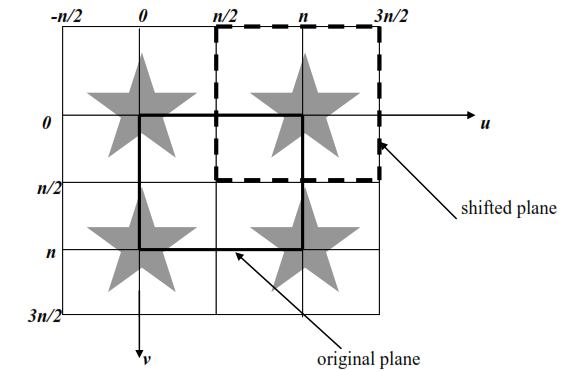
F(3) = ¼ [8e0 + 4e-j2ᴨ3/4 + 2e-j2ᴨ3.2/4 + e-j2ᴨ3.3/4]  
 = ¼ [8 + 4e-j3ᴨ/2 + 2e-j3ᴨ + e-j9ᴨ/4]  
 = ¼ [8 + 4j -2- j]  
 = ¼ [6 + 3j]  
F(3) = 3/2 + 3/4j  
|F(3)| = ¾ √5

The magnitude of the image is highest at 0th pixel. Also, the odd and even number of elements tend to behave in a similar way as in one contains imaginary parts and the other one only consists of real parts.

**(5) Answer the following questions and support your answers with reasoning and analysis;  
 (a) Why is it necessary to move the origin of the Fourier transformed image to the center (i.e. to u = n/2, v = n/2). How is this shifting implemented?   
 (b) Why is bit reversal needed in FFT? Explain.  
 (c) The Fourier spectrum |F(u,v)| of an image f(x,y) is known, but f(x,y) is not known. Can f(x,y) be computed? Explain.  
 (d) Prove that the two-dimensional Fourier transform of an image f(x,y) can be achieved using two one-dimensional transforms. What is the significance of this?**

**Solution:**(a) The fourier transform and its inverse are periodic functions with the period n because F(u, v) = F(u + n, v) = F(u, v + n) = F(u + n, v + n). This property implies that only one period is needed to completely specify F(u, v). Furthermore, F(u, v) is symmetric since |F(u, v)| = |F(-u, -v)|. To see the implications of periodicity and symmetry, for simplicity consider the case of 1D transform in which F(u) = F(u + n) and |F(u)| = |F(-u)|.

Figure shows the typical plot of an image fourier spectrum spectrum,  
  
Since, fourier transform is formulated in the interval 0 to n-1, the result produces two “back to back” half periods. To display a full period with the maximum value at the center (rather than at either side of the display frame), we must move the origin to u = n/2 as shown in the figure below,  
  
Let us consider an example of the periodicity and symmetry in 2D as an image. When the origin is unchanged the fourier transform image appears in four corners of the u-v display plane. In order to bring the fourier transform image to the center of the display, F(u, v) must be shifted to u = n/2 and v = n/2.



To implement this shift, we must determine the value of F(u – n/2, v – n/2) as follows:  
F(u – n/2, v – n/2) = 1/n ∑ ∑ f(x, y) e–j2ᴨ((u – n/2)x + (v – n/2)y) / n   
 = 1/n ∑ ∑ f(x, y)(-1)x+y e–j2ᴨ(ux + vy) / n

The above equation implies that in order to shift the origin of the F(u, v), we must first multiply the image function f(x, y) by (-1) x+y and then perform the fourier transform. This simply means that when (x + y) is odd, we change f(x, y) to –f(x, y), otherwise we leave it unchanged.

(b) The computation of a 2D fourier transform requires two 1D transforms. A 1D transform involves n complex number multiplication of the exponential by f(x) plus n – 1 additions for each u, with the total complexity of O(n2). Note that e-j(2ᴨux)/n can be computed once and kept in an array. Using FFT, the number of multiplication and addition will reduce to O(nlgn).

In order to determine which image data f(x) is to be used for Fe, Fo, Fee, Feo, etc, we perform bit reversal, as the following table demonstrates,

|  |  |  |  |
| --- | --- | --- | --- |
| **Array index** | **Binary representation** | **Bit reversal** | **Resulting array index** |
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 1000 | 8 |
| 2 | 0010 | 0100 | 4 |
| 3 | 0011 | 1100 | 12 |
| 4 | 0100 | 0010 | 2 |
| 5 | 0101 | 1010 | 10 |
| 6 | 0110 | 0110 | 6 |

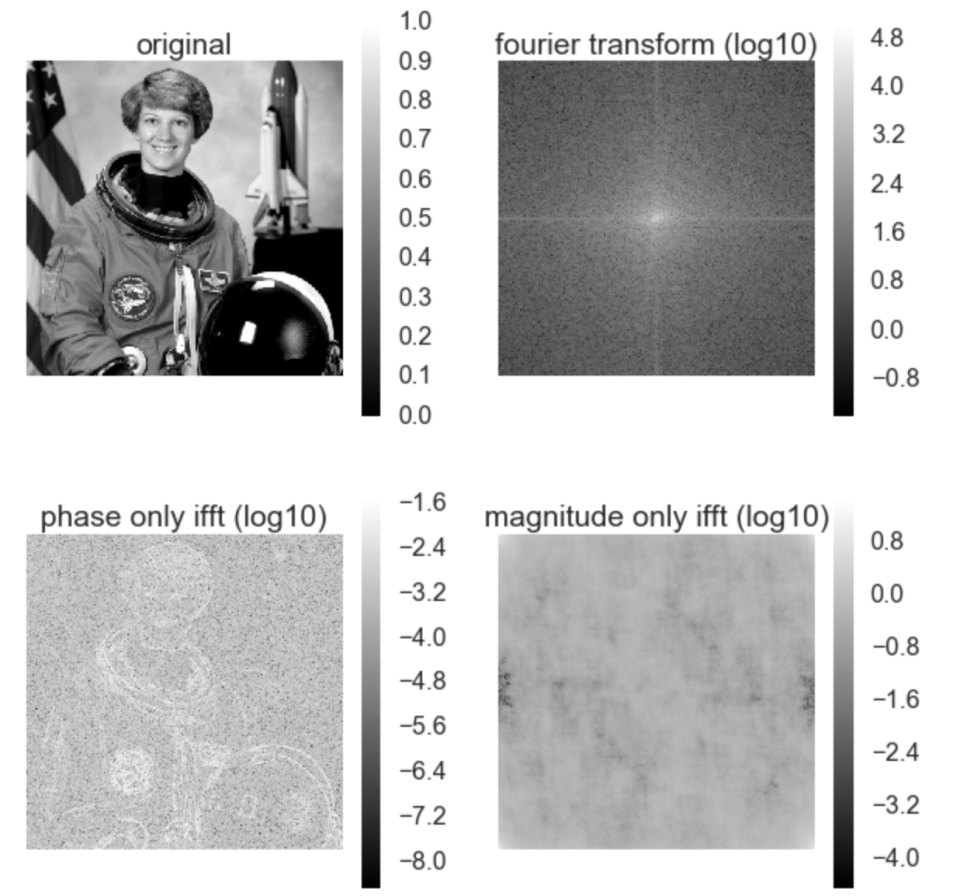
Note that the bit reversal allows to combine correct sequences for performing FFT. Thus, before carrying out the FFT, the image function f(x) must be reordered so that the resulting FFT is correct.

Fourier transform and its inverse are different only in the sign of exponential, and therefore the same method can be used to perform both forward and inverse transform by supplying an argument -1 to the former and +1 for the latter.

(c) If the fourier spectrum of an image f(x, y) is known, but f(x, y) is not known then it is not possible to get f(x,y) back because we do not know the phase of the image that is the direction of the amplitude. The important information in the images tends to edges -- edges that contain most of the details in the image. This information is carried in the phase information -- when decomposed into sinusoidal signals, a square wave can have arbitrary location only if the waves contain phase information.

Additionally, real world images tend to look roughly the same in the frequency spectrum when only plotting magnitude. These images tend to have large (in magnitude) low frequency components with (much) smaller high frequency components. It's reasonable to say that a significant portion of the entropy (or information) is carried in the phase information.

With a naïve inverse Fourier transform on the values obtained from the image, it is not possible (at least by experiment) to recover the original signal. However, the phase only inverse Fourier transform (where we divide by the magnitude to only preserve the phase).



(d) Fourier transform provides useful information about the frequency contents of an image, and is used to filter the noise, which is associated with the high frequency components in an image. The two dimensional fourier transform of an image can be given as,

There are several properties of the fourier transform such as seperability, periodicity, symmetry, etc. The seperability says that a two-dimensional (2-D) Fourier transform can be computed as two 1-D transforms. This property can be proved as follows:

which can also be written as,

Instead of calculating each pixel twice, the fourier seperability decreases the computational costs associated with the 2-D fourier transform. We can see that the separated 2-D fourier transform is another 1-D fourier transform. Therefore instead of writing a program to compute a 2-D Fourier transform, we write a simpler program for a 1-D transform and run it twice with different arguments. The same principles of separability apply to the inverse Fourier transform.