

## 0.1 Warmup

The input images for this problem are 256 x 256 grey scale images and they are loaded in the matlab environment by using the function `imread()`. Two matrices `imgx` and `imgy` are used to store the value of individual pixels from the two images respectively. The visible edges in both the images are technically high-frequency signals that could be visualized as edges. Hence in order to blur the images, we would need to eliminate the high frequency signals from them. The means through which we achieve these are called low-pass filters and their objective is to pass the low frequency inputs from the image blocking the high frequencies. For the Einstein's image we use a mean filter, which is a 3x3 matrix that has the arithmetic mean of all 9 elements with 1 as pixel value. This matrix is denoted by `avgflt`. Another special low pass filter called the *Gaussian* filter is created with the help the Matlab function `fspecial()`. We chose 13 as the number of rows and columns in the filter matrix and  $\sigma = 2$ , in the Gaussian Normal equation for two variables :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (1)$$

Both these convolution kernels are applied to the respective image matrices using the matlab function `conv2()`. The last parameter of the function is required to control the behavior of the convolution operation. For 'same' it returns the central part of the convolution of the same size as that of the image matrix and it discards non-existing neighboring pixels from the image boundary that appeared as a result of the convolution operation. Since both the matrices have real numbers, they are converted to integral values using the `uint8()` function. Finally the contents of the matrices are displayed with the help of `imshow()`. The Matlab function `imwrite()` is used to write the contents of the matrices to different image files for future reference.

In order to determine the DFTs of the image matrices, we use the matlab function `fft2()`. The function transforms a given matrix to frequency domain. The resultant matrix has complex numbers in all elements, hence to find the magnitude of the DFTs, we need to obtain the sum of root mean square of all the elements in the matrix. This is achieved by using `abs()`. Finally the magnitude could be displayed with the same `imshow()` function.

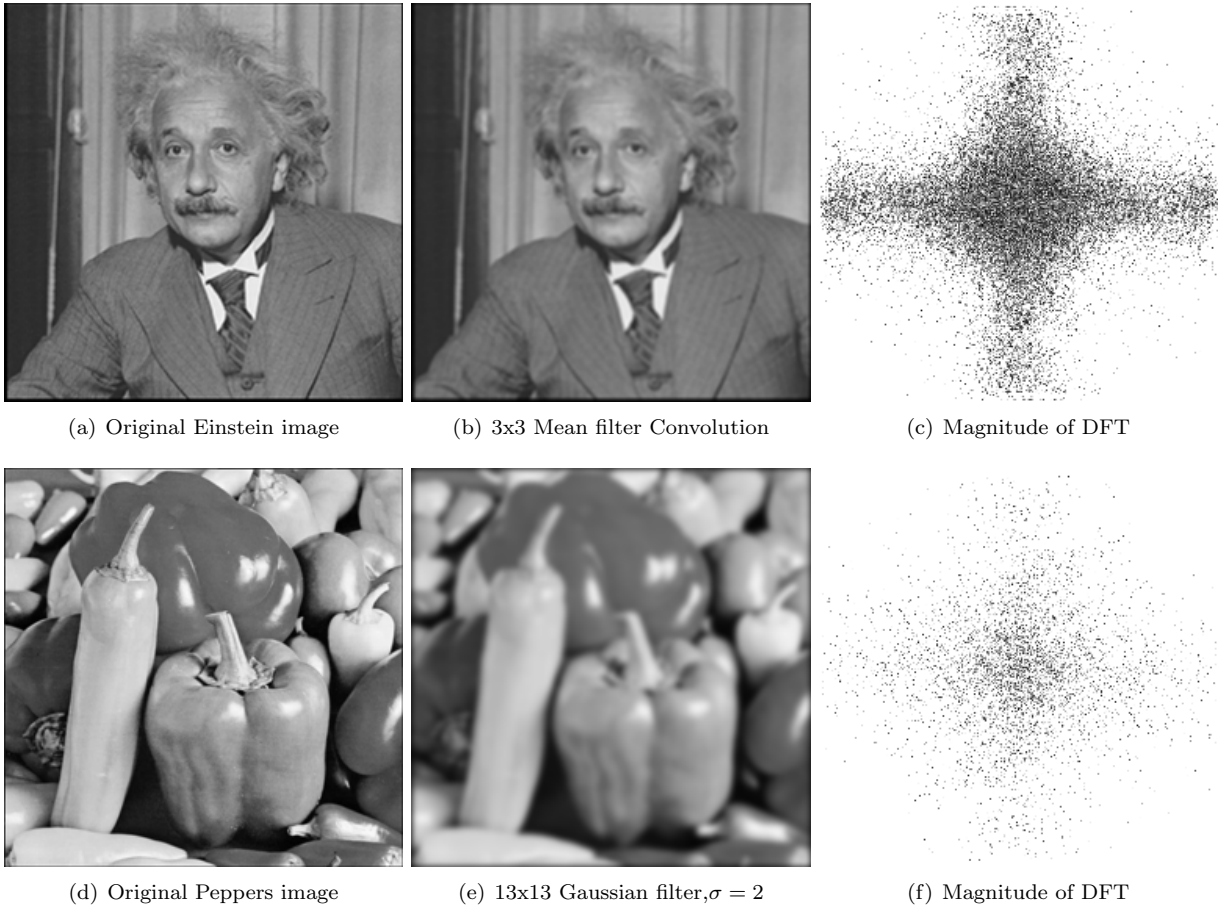


Figure 1: Image transformations

```

1 %Loading the images in matlab, storing in 2D matrices
2 imgx = imread('./einstein.png');
3 imgy = imread('./peppers.png');
4
5 %Defining Convolution filters
6 avgflt = 1/9*[1 1 1; 1 1 1; 1 1 1];
7 gausflt = fspecial('gaussian', 13, 2);
8
9 %Perform Convolution
10 blurx = conv2(double(imgx), double(avgflt), 'same');
11 blurry = conv2(double(imgy), double(gausflt), 'same');
12 %Show image from resultant matrices
13 imshow(uint8(blurx), []);
14 imshow(uint8(blurry), []);
15 %write images into files
16 imwrite(uint8(blurx), './einstein-blur.png');
17 imwrite(uint8(blurry), './peppers-blur.png');
18
19 % Transform image into frequency domain using FFT
20 dftx = fft2(single(imgx));
21 dfty = fft2(single(imgy));
22 imshow(uint8(abs(dftx)));
23 imshow(uint8(abs(dfty)));
24 imwrite(uint8(abs(dftx)), './einstein-mag.png');
25 imwrite(uint8(abs(dfty)), './peppers-mag.png');

```

Listing 1: Matlab script for the Problem 1

## 0.2 Fast Convolution

- Let us suppose matrix A of dimensions  $N \times N$  be the image and B be the convolution kernel with  $M \times M$  dimensions. In order to convolve A with B, each element in A needs to be operated  $M \times M$  times (By definition of the Convolution operation). Now, since this operation has to be carried out for all the elements in A, there would be a total of  $N \times N \times M \times M$  operations. Therefore the complexity of convolving an  $N \times N$  image with an  $M \times M$  kernel would be  $O(N^2 M^2)$ .
- Since convolution in Spatial Domain is same as multiplication in Fourier Domain, we could take DFTs of each of the matrices A and B and then multiply them.
- The complexity of computing convolution for a single variable using FFT is  $O(N \log N)$ . So for an image signal which is  $N \times N$  in size, the complexity is  $O(N^2 \log N)$ .
- In order to convolve an  $N \times N$  image with a  $M \times M$  kernel using FFT we would need to perform the following operations :
  - Convert both the image and the kernel to frequency domain by using FFT.  $M \times M$  kernel is being padded adequately with Zeros, so that both the transformed Image and the Kernel are of the same size. The complexity of this would be :  $O(N^2 \log N) + O(N^2 \log N) = O(2N^2 \log N)$
  - Multiplying each of the Frequency domain matrices, require  $N \times N$  operations, the complexity of the same would be :  $O(N^2)$
  - Finally we would need to transform back the convolved matrix to spatial domain, for which we would need to use the inverse FFT method. The complexity of this is same as that of FFT, so the total complexity of the operations performed :  $O(2N^2 \log N) + O(N^2) + O(N^2 \log N) \approx O(N^2(3 \log N))$  We already know that FFT based convolution methods are faster than the traditional convolution technique, so :

$$\begin{array}{ll}
 O(N^2 M^2) & \geq O(3N^2 \log N) \\
 \text{or, } N^2 M^2 & \geq 3N^2 \log N \\
 \text{or, } M^2 & \geq 3 \log N (N \neq 0) \\
 \text{or, } M & \geq \sqrt{(27)} (N = 512 = 2^9) \\
 \text{or, } M & \geq 5
 \end{array}$$

Hence for kernel sizes more than or equal to 5x5 the FFT based method will perform faster than the traditional convolution methods.

### 0.3 Deconvolution

For this, we chose a 13x13 Gaussian Filter with  $\sigma = 3$  in order to blur the image significantly. In order to obtain the Fourier Transform of this Gaussian Filter, we use the function `imfilter()`. The first two arguments to this function are the image matrix and the convolution kernel matrix. The next three arguments control the behavior of the function. The 'circular' value assumes that the image is a periodic function. Now our objective is to retrieve an image close to the original one from the blurred image. We also know the blurring function that actually blurred the image. We compute the DFT of the blurred image and the DFT of the blurring kernel. Both of these are Frequency domain transform of real images. Now we know that the blurred image is basically the result of convolution between the original image and the gaussian filter in Spatial domain.

$$f(x, y) * g(x, y) = h(x, y) \quad (2)$$

where f, g and h are image in Spatial Domain, filter and the blurred output respectively, \* is the convolution operator. Since this operation is equal to multiplication of their Fourier Transforms in Frequency Domain, we could write :

$$F(U, V).G(U, V) = H(U, V) \quad (3)$$

where F, G and H are the respective DFTs of the functions discussed in 3. Therefore, we could write:

$$F(U, V) = H(U, V)/G(U, V) \quad (4)$$

As both H and G are complex matrices, by division here we mean element by element division). By taking Inverse Fourier Transform on both sides we could obtain the original image, f(x, y)

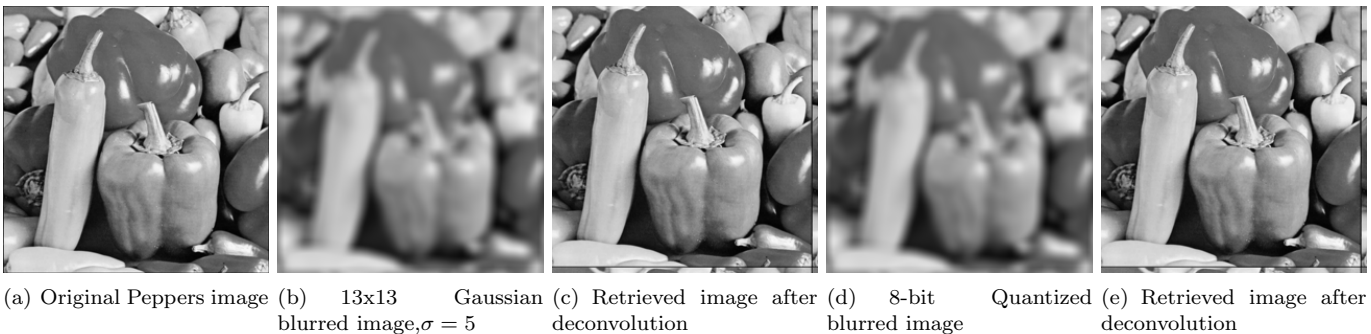


Figure 2: Deconvolving an image blurred by a Gaussian Filter

The results are likely to vary when deconvolution is done after the 8-bit Quantization step, because in the process we are losing information pertaining to the image.

```

1 imgx = double(imread('/home/subh/courses/cap5415/ps1/peppers.png'));
2 % Gaussian 13x13 filter with standard deviation = 3
3 fx = fspecial('gaussian',13, 5);
4 % Blur image by convolving it with gaussian filter, same keeps the blurred
5 % image as the same size of the original image
6 blrx = imfilter(imgx, fx, 'circular', 'same', 'conv');
7
8 % The following Three lines to be uncommented for problem 3b
9 % blrx = uint8(imfilter(imgx, fx, 'circular', 'same', 'conv'));
10 % blrx = imfilter(double(imgx), fx, 'circular', 'same', 'conv');
11 % imwrite(uint8(blrx), '/home/subh/courses/cap5415/ps1/peppers-p3b-blur.png');
12
13 imwrite(uint8(blrx), '/home/subh/courses/cap5415/ps1/peppers-p3a-blur.png');
14 % Computing FFT of the kernel - padding the kernel with zeroes to make the
15 % kernel same size of the image matrix
16 dftfx = fft2(fx, 255, 255);

```

```

17 % Computing the DFT of the blurred image
18 dftblrx = fft2(blrx);
19 % Dividing DFT matrix of image with the DFT matrix of the convolution blur
20 % kernel, element by element
21 res = dftblrx ./ dftfx;
22 % Inverse Fourier transform
23 orig = ifft2(res);
24 imwrite(uint8(orig), '/home/subh/courses/cap5415/ps1/peppers-p3a-deconv.png');
25 % The following line needs to be uncommented for Problem 3b
26 % imwrite(uint8(orig), '/home/subh/courses/cap5415/ps1/peppers-p3b-deconv.png');

```

Listing 2: Matlab script for the Problem 3

## 0.4 Deriving the convolution theorem

We are required to prove that the DFT of the convolution of two discrete functions is equal to the multiplication of the DFTs of the individual functions.

$$\begin{aligned}
 DFT_u(f[n] * h[n]) &= \sum_{n=0}^{N-1} f[n] * h[n] e^{(-j2\pi nu)/N} \\
 &= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f[m] h[n-m] e^{(-j2\pi nu)/N} \\
 &= \sum_{m=0}^{N-1} f[m] \sum_{n=0}^{N-1} h[n-m] e^{(-j2\pi nu)/N} \\
 &= \sum_{m=0}^{N-1} e^{(-j2\pi nu)/N} . H[u] \text{ By Shift Theorem} \\
 &= F[u] H[u]
 \end{aligned}$$

## 0.5 Median Filter

Applying mean and median filters blur the images but there is a difference between these two blurring function. The mean filter computes the arithmetic mean of pixels in the filter neighborhood and on being convolved with the image, it only allows frequencies less than a threshold (mean of neighborhood in this case) to pass through, those that are above the threshold are blocked. For a median filter, the threshold is the median frequency in the neighborhood. For real-life images like the einstein image, pixels do not vary much in a small neighborhood. So there is no apparent difference in the blurred images obtained through mean and median filters. However, as we increase the neighborhood size (size of the filter), pixels are more likely to vary. Selection of a proper threshold is important here. Since a median filter computes the statistical average of all frequencies presence in a neighborhood, it would surely perform a better blurring operation. In both cases, when the blurred images are required to be deblurred back to the original, the median filtered image would fair relatively better.

```

1 imgx = imread('./einstein.png');
2 % Creating mean filters of sizes 5x5, 7x7 and 9x9
3 af5 = 1/25 * ones(5,5);
4 af7 = 1/49 * ones(7,7);
5 af9 = 1/81 * ones(9,9);
6 % Convoluting image with filters and writing them back
7 imwrite(uint8(imfilter(imgx,af5, 'circular','same','conv')), './einstein-af5.png');
8 imwrite(uint8(imfilter(imgx,af7, 'circular','same','conv')), './einstein-af7.png');
9 imwrite(uint8(imfilter(imgx,af9, 'circular','same','conv')), './einstein-af9.png');
10 % Convoluting with median filters of similar dimensions
11 imwrite(uint8(medfilt2(imgx, [5 5])), './einstein-mf5.png');
12 imwrite(uint8(medfilt2(imgx, [7 7])), './einstein-mf7.png');
13 imwrite(uint8(medfilt2(imgx, [9 9])), './einstein-mf9.png');

```

Listing 3: Matlab script for the Problem 5

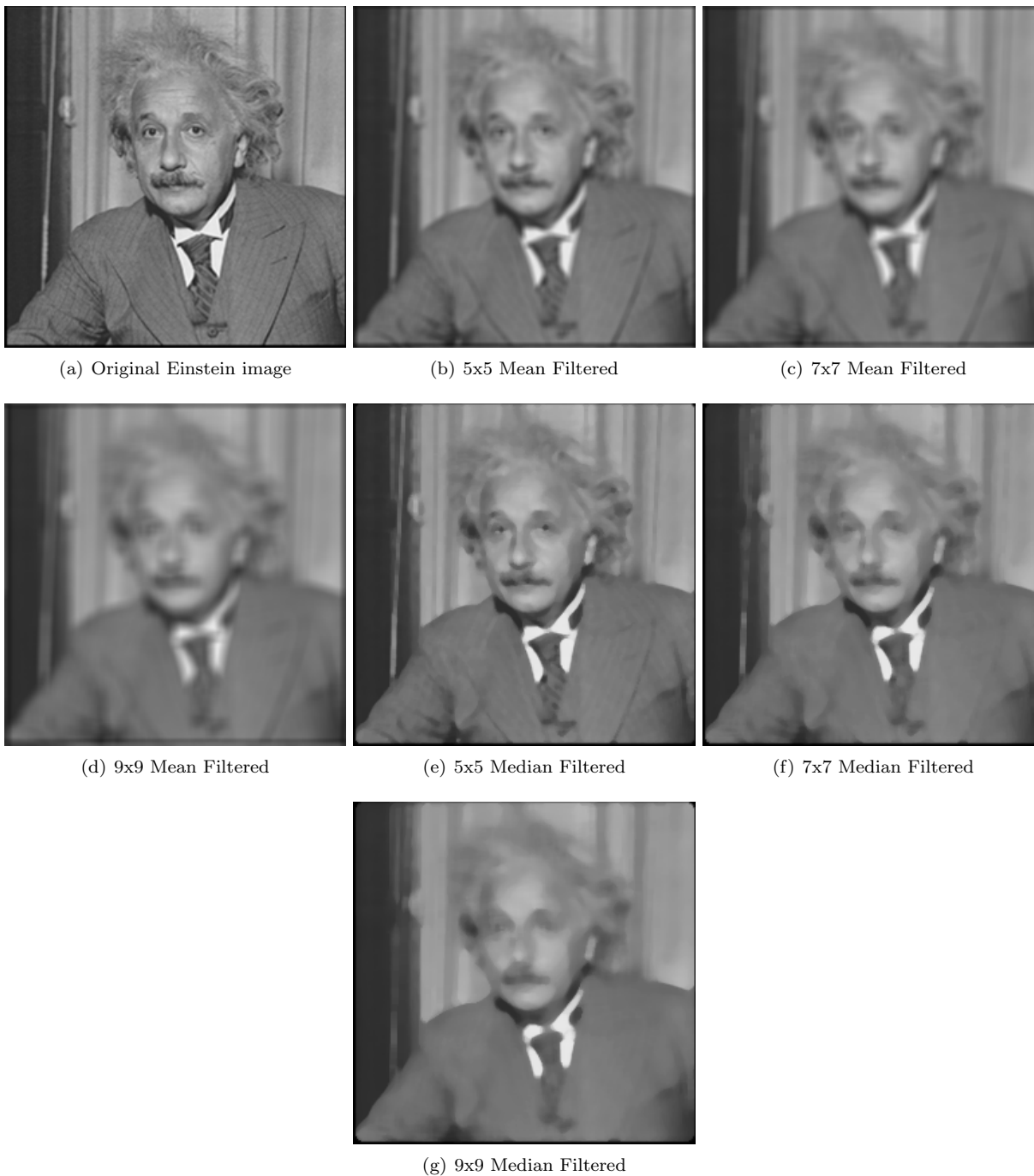


Figure 3: Difference between Mean and Median Filter Operations