Lab2 Bildanalys

Oscar Jacobson

December 29, 2022

For future reference

I = imread('napoleon.png'), write figure as matrix **imagesc**, shows image with scaled colours
imshow, shows image,
imtool, is an image inspection tool
imwrite(I,''), save image I in ''
print(gcf, '-dpng', 'nap_fig.png'), write to figure
montage({fig1, fig2, fig...}) display multiple figures

Q1

Figure 1: Q1: Original, disk filter 63 x 63 and a laplace filter 3 x 3.

The original figure is to the left. The figure in the center has an average filter applied to it with a disk shaped filter-size of 63 x 63 the outermost edges of the filter is assigned zeros because values considered are all values within a circular radius of 31. The right figure has a laplacefilter applied to it with an alpha of 0.2 and a size of 3 x 3. The laplace filter yielded is shown in matrix form in eq 1. The averaging filter will be applied for every pixel in the original figure and set the value for that pixel in the new figure to be the average value of all the original pixels within a 31 pixel radius of the central pixel. The laplace filter is applied similarly to every pixel in the original figure but this filter is designed to highlight where the derivative of the pixel intensity is high, meaning quick variations in the intensity will be more visible.

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$
$$\nabla^2 = \frac{4}{(\alpha+1)} \begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix} \tag{1}$$

Q2

An averaging filter is a sort of low-pass filter only letting through very slow changing features of the figure.
The disk filter is a type of averaging filter, considering all pixels in a circular radius of the evaluated pixel compared to the averaging filter considering all pixels within a square centered at the evaluated pixel. The disk filter is therefore also a low-pass filter. Important to note is that if a averaging filter is for example [10, 1] only one axis will be averaged.
A gaussian filter can be tuned to be either a low-pass or high-pass filter meaning the frequency band can be tuned at will. Therefore the gaussian filter is a band-pass filter. The gaussian can also be tuned to work on a specific axis, either X (Up and down) or Y (Side to side).

Q3

Assuming a low-pass filter kernel of size n at $[x,\ y]$ is *lp(n)*, a high-pass filter kernel of size n can be synthesized by eq 2 and a band-pass filter kernel can be synthesized by eq 3

$$hp(n) = V(x,y) - lp(n) \tag{2}$$

$$bp(n1,n2) = \quad lp(n1) + hp(n2) = \quad lp(n1) + (V(x,y) - lp(n2)) \tag{3}$$

Where V(x, y) signifies the original value of pixel (x,y). An example of eq 2 in matrix form:

$$lp = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \tag{4}$$

$$hp = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{-1}{9} & \frac{-1}{9} & \frac{-1}{9} \\ \frac{-1}{9} & \frac{8}{9} & \frac{-1}{9} \\ \frac{-1}{9} & \frac{-1}{9} & \frac{-1}{9} \end{bmatrix} \tag{5}$$

Q4

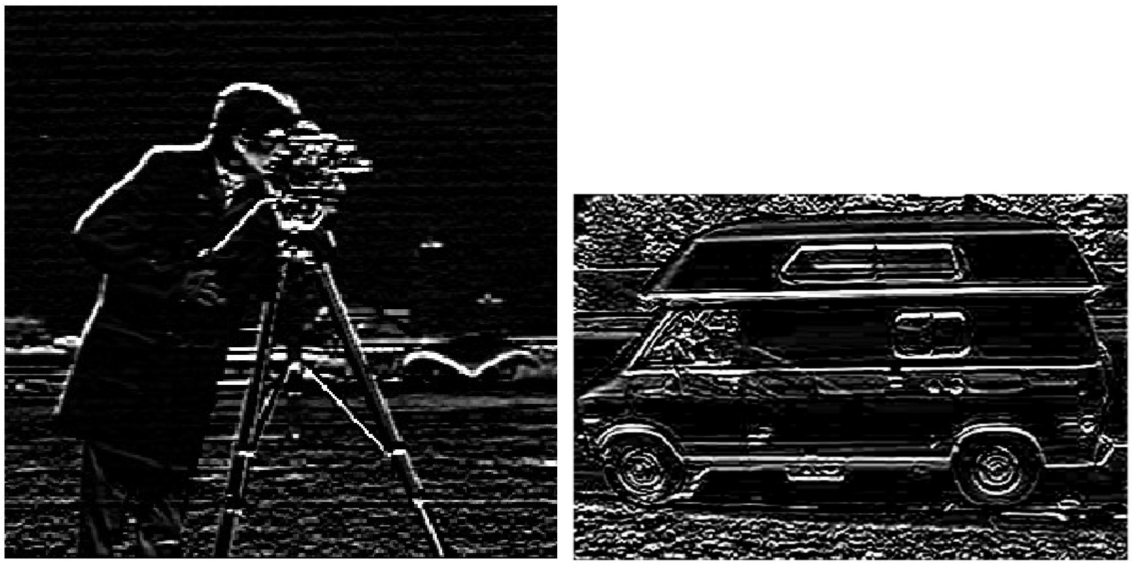

Figure 2: Sobel filtering

Sobel filter kernel used is shown in eq 6.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{6}$$

Q5

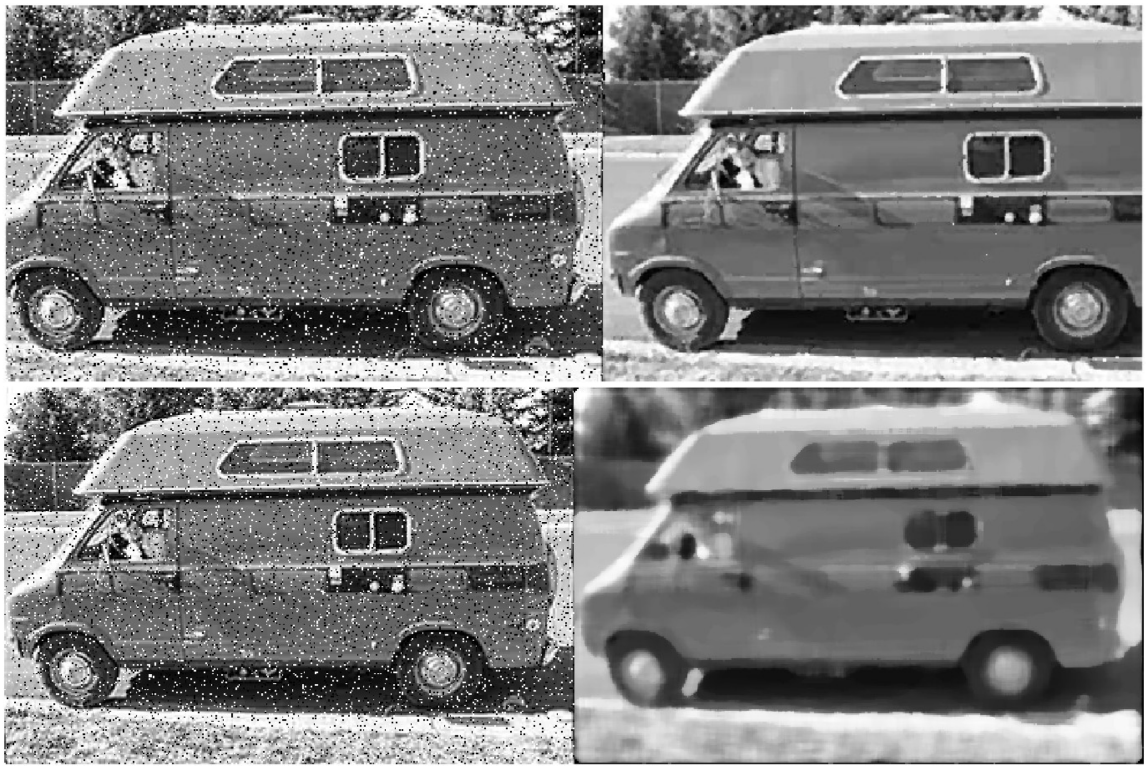

Figure 3: Original figure vs median filtering using 3 x 3 and 10 x 10 filter kernel

A larger filter kernel will look more and more like an averaging filter, loosing more information the larger the kernel becomes. A 3 x 3 filter kernel still allows for very sharp edges and a small loss of information.

Q6

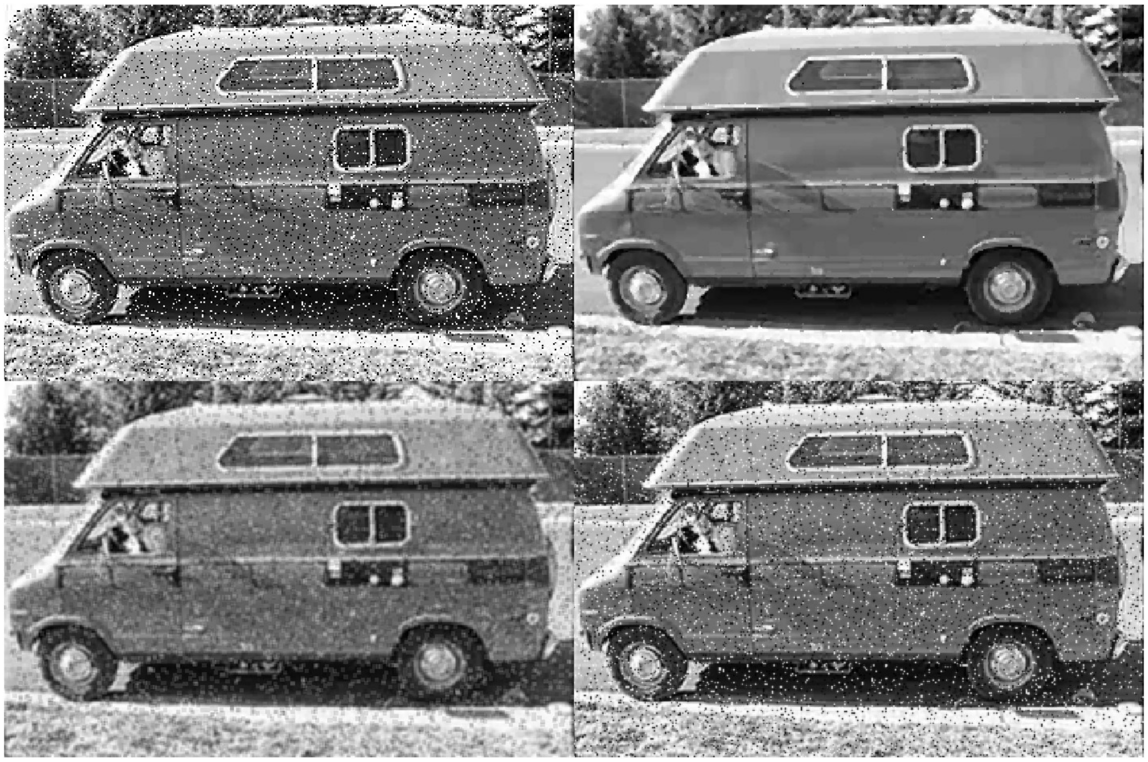

Figure 4: Top left: Original, Top right: median filter, Bottom left: average filter, Bottom right: gaussian filter

The top right image has a median filter applied to the original figure in the top left. A median filter simply takes the median value of every pixel within a 3 x 3 area from the original pixel. This allows to completely remove pixel values that could be considered outliers. The average filter (Bottom left) an the Gaussian filter (Bottom right) see significant noise even after the filtering because of the impact the outliers have on the average values (The Gaussian is also a sort of averaging filter, only rotationally symmetrical instead of homogeneous).

Q7

The median filter requires comparison operations to be performed while a mean requires arithmetic operations when performed. These are fundamentally different when considering computing time and arithmetic operations are going to be way simpler and faster.

Q8

```
clear all
I = imread("wagon_shot_noise.png");
new = [];
figsize = size(I);
sz = 3;
border = (sz-1)/2; % Assuming odd input
for i = 1+border:figsize(1)-border
    for j = 1+border:figsize(2)-border
        new(i,j) = median(I(i-border:i+border, j-border:j+border)
            ,"all");
    end
end

figure
montage({I,uint8(new)})
```

Slightly sloppy code as it will disregard values close to the borders, output below:

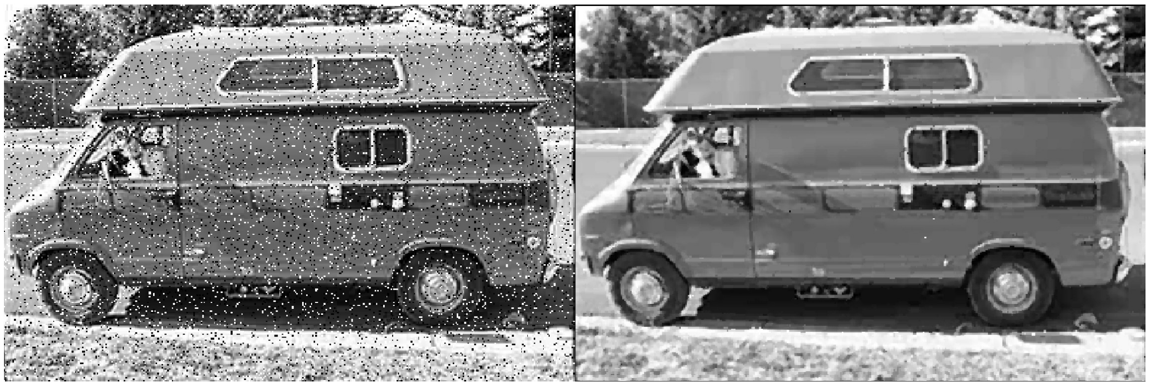
Figure 5: Left: Original, Right: My median filter

Q9

When using a large filter the padding of the values outside of the actual figure is very important, if the values outside of the borders are considered 0 (black) these are going to be averaged into the picture if using a large standard deviation. (The large standard deviation gives more importance to the outer edges of the filter kernel)

Q10

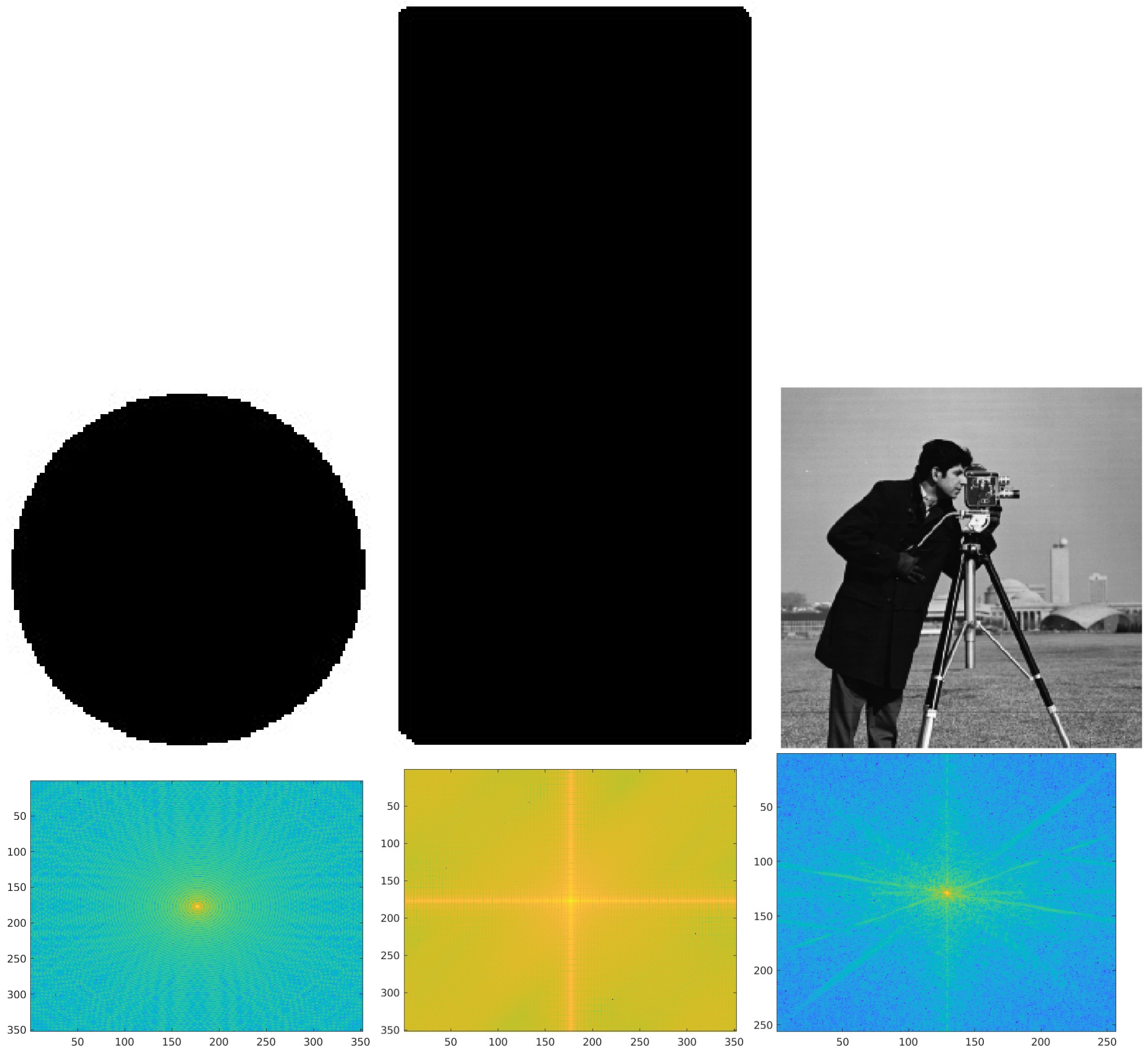

Figure 6: Top: original figure, Bottom: the Fast Fourier transformed version for every respective figure

In figure 8 three original figures are shown with their respective Fast Fourier Transforms. The FFT for the *cameraman* is very similar to some type of convolution of the first two FFTs for the circle and rectangle respectively. There is a clear pattern of a central point "radiating" outwards very much like the FFT for the circle. There are also some lines in the cameraman FFT which probably originate from the fast changing lines, appearing from the legs of the camera and the sharp edge produced by the cameraman in contrast to the background. These lines are slightly rotated and displaced because of the different placement in the original picture.

Q11

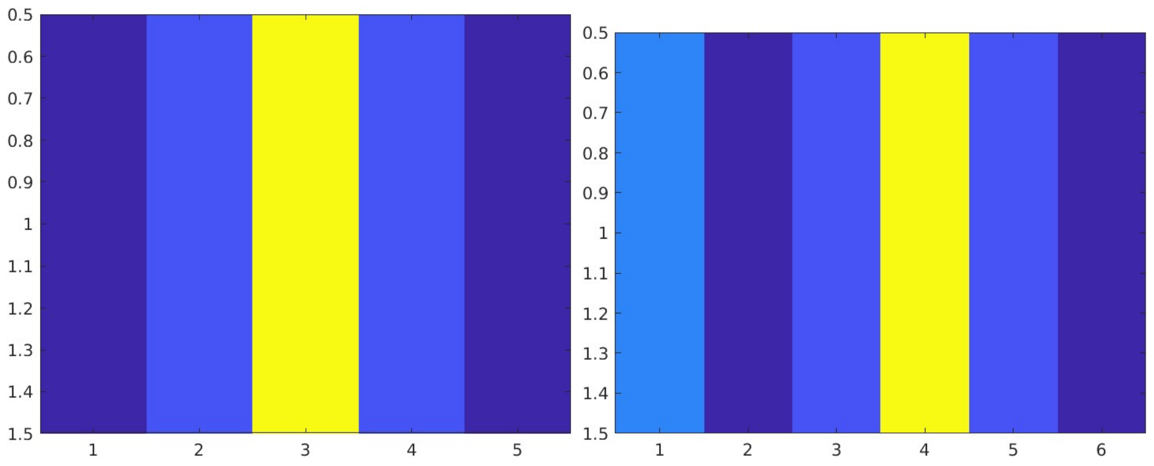

Figure 7: Left: Odd FFT, Right: Even FFT

When performing the FFT on a small vector the odd length array produces symmetry over the central value. The same symmetry seems to be observed in the even array but with one value missing on the right, the same behaviour can be observed in all cases, the rightmost lines are symmetrical around the central line when excluding the leftmost line.

When assigning *f(1,2) = 0* the output for the IFFT produces complex values for the figure, this is avoided if a *0* is also assigned to the respective symmetric position around the central value. In this case *f(1,4) = 0* produces real values when doing the IFFT.

Q12

A lowpass filter was constructed by setting values of the borders on the original FFT to 0 and doing the inverse FFT. The wavy patterns appearing on the filtered IFFT are probably caused by the square nature of the filter applied to the FFT and would be minimal if the border was circular. To keep all values real, the first row and column of the original FFT was discarded to keep the matrix size uneven (255 x 255). All the discarded values are discarded symmetrically over the X and Y axis.

(Taking the FFT and doing the IFFT on the same matrix yields a identical image to the original as far as I can tell. Might be some rounding/arithmetic errors that distort the image slightly but is in any case not noticeable.)

```
clear all
im = double(imread("cameraman.png"));
figure; imshow(uint8(im))
sz = size(im);
border = 105;
f = fftshift(fft2(im));  % This produces a complex FFT output
figure; imagesc(log(abs(f)));
title("FFT");
h = f(2:sz(1), 2:sz(2));

h(1:border,:) = 0;
h(:,1:border) = 0;
h(sz(1)-border:sz(1)-1,:) = 0;
h(:,sz(2)-border:sz(2)-1) = 0;

figure; imagesc(log(abs(h)));
title("FFT - border")

im = ifft2(ifftshift(f));
% figure; imagesc(abs(im));
figure; imshow(uint8(im))
title("No filter IFFT")

im2 = double(ifft2(ifftshift(h)));
% all(im2 >= 0)
% im22 = log(im2);
% figure; imagesc(im2);
figure; imshow(uint8(im2))
title("Filtered FFT")
```

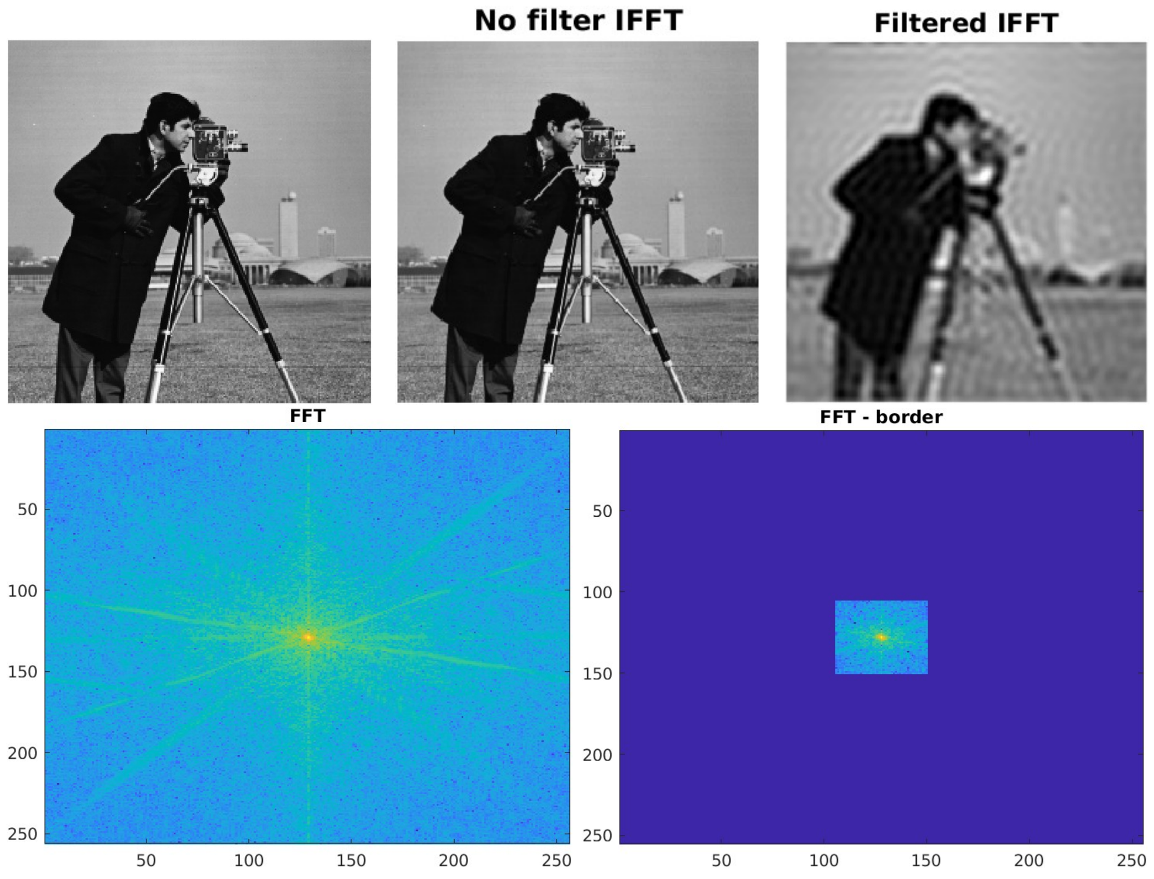

Figure 8: Top: original figure, Bottom: the Fast Fourier transformed version for every respective figure

Q13

When visualizing the FFT in the frequency domain on the left of figure 9 it is clear that 4 points in the frequency domain are disturbing the underlying image, to suppress the wavy pattern these points together with all the points on their respective X and Y axis are assigned 0 values. These points are symmetrically located around the central point [129, 129] and the resulting filter will therefore not result in complex numbers when taking the inverse FFT. The code can also be modified to have a wider filter-border, this will cause some loss of information in the underlying figure at the benefit of further reducing the wavy pattern. See the difference in the central figure compared to the right figure below.

```
clear all
I = imread("freqdist.png");
```

```
figure; imshow(I);
f = fftshift(fft2(I));
figure; imagesc(log(abs(f)));

border = 3;
sz = size(f);
% f = f(2:sz(1),2,sz(2));
% sz = size(f);

point1 = [100,101, 92; 157,158, 149];
point2 = [100,101, 109; 157,158, 166];


f(point1(1,1)-border:point1(1,1)+border,:) = 0;
f(point1(1,2)-border:point1(1,2)+border,:) = 0;
f(:,point1(1,3)-border:point1(1,3)+border) = 0;

f(:,point2(1,3)-border:point2(1,3)+border) = 0;


f(point1(2,1)-border:point1(2,1)+border,:) = 0;
f(point1(2,2)-border:point1(2,2)+border,:) = 0;
f(:,point1(2,3)-border:point1(2,3)+border) = 0;

f(:,point2(2,3)-border:point2(2,3)+border) = 0;



% figure; imagesc(log(abs(f)));
im = ifft2(ifftshift(f));
% figure; imagesc(im);
figure; imshow(uint8(im))
```

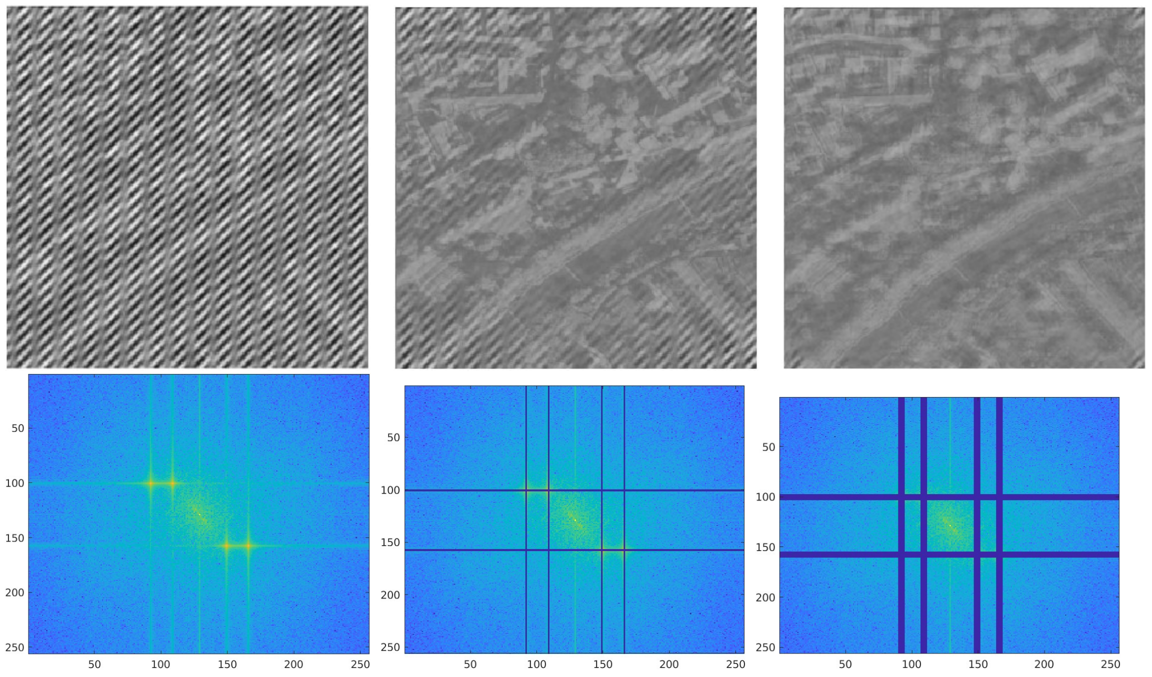

Figure 9: Top left: original figure, Top 2 and 3: IFFT of filtered FFT below. Bottom: the Fast Fourier transformed version for every respective figure (With and without filters, filter width of 1 and 5 respectively)

Q14

This computer exercise felt a lot easier to follow and was easier to draw knowledge from as the questions were more clearly stated and required more explanations/learning to be done from me!