ENSC 474

Assignment 6 - Fun with Masks

Cyrus WaChong

301306459

cwachong@sfu.ca

Not much to show for part 1, so part 2 is started below.

For this portion, gaussian noise was simply added with the “imnoise()” function. Once noise was added, a constant mask was applied to remove the noise. Below the output of the noise, and the applied masks can be seen.

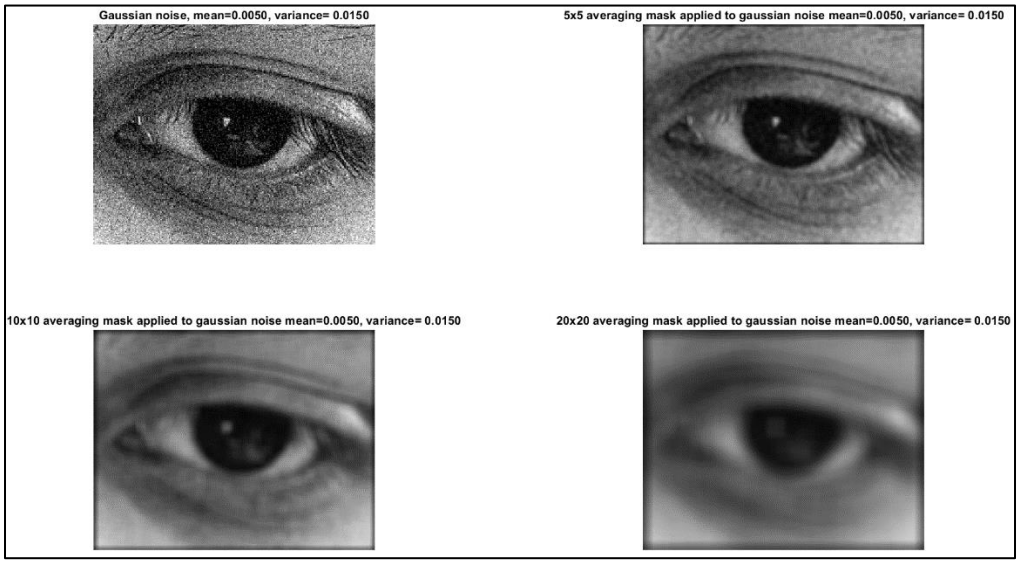

Figure 1: Applied gaussian noise and applied averaging masks.

One can see that multiple sizes of the constant mask were used, 5x5, 10x10, and 20x20. Other values were also tested, but anything larger than 20x20 (~25x25) produced an image that was excessively blurry, and anything smaller than 5x5 (3x3) did not get ride of the noise sufficiently.

With the constant mask, one can see that the larger the mask, the more noise that is removed, but this also causes the image to become blurrier. The larger the image became, the more intensive the calculation became since for every point in the image one must account for many more surrounding values, which in turn caused the computation to be slightly slower. An 5x5 mask must account for the point, and the surrounding 24 cells, but with a 20x20 mask, for every point we must use 400 surrounding cells.

In conclusion the constant mask is not a very good mask at removing noise without ruining the quality of the image itself. The averaging mask is very bad at keeping edges, and the black padding surrounding the image becomes more prevalent and ruins the border of the image (this is due to zero padding).

Applying Salt n Pepper noise with the same “imnoise()” function, two cases were done, one with a decent amount of noise, and one with very high-density noise.

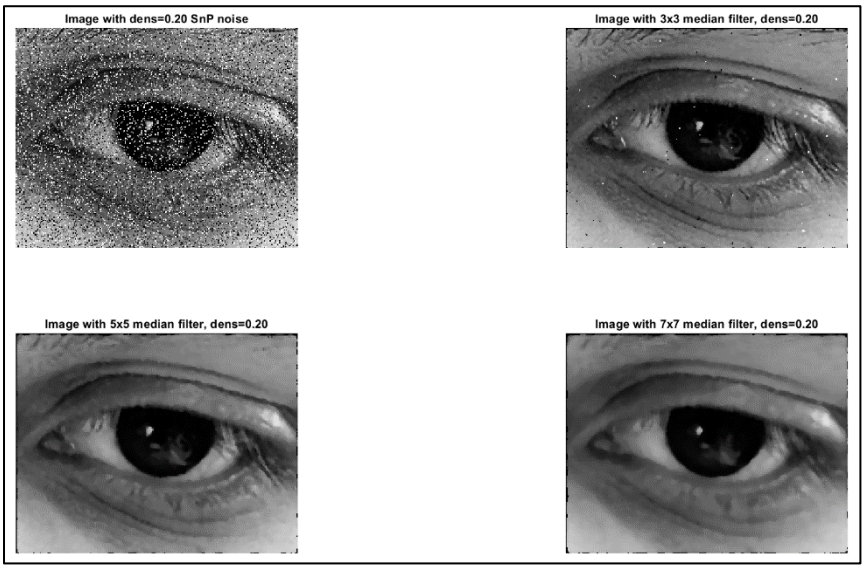

Figure 2: Low density SnP noise filtered with median mask

SnP noise is where random pixels will get set to max (1) or minimum (0). This leads to the Salt n Pepper name, since we see random black and white pixels, the number of noise pixels depends on the density of the noise. As was expected, the median filter is very good at filtering the SnP noise out of the image, since the pixels will be extreme values, so in most cases, the median point is not one of the noise pixels.

The median mask was done by going to each point and taking all values in the range of a mask, finding the median value and equating the current cell to this value.

The median filter is also good at keeping edge integrity and does not smudge the image like the constant mask does. At first glance this seems to be the superior filter for Salt n Pepper noise, which is turns out to be. The larger the mask, the more noise is filtered, but the image becomes slightly blurrier. This turns out to be a very rewarding tradeoff, the larger mask in these cases produces a much better image.

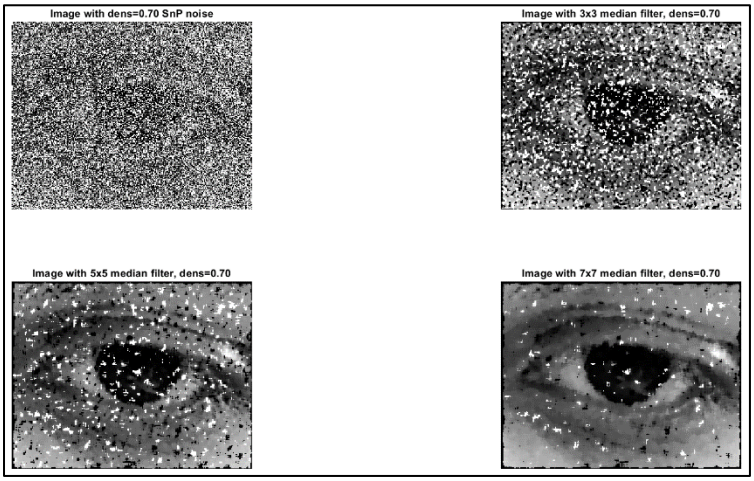

Figure 3: High density SnP noise filtered with median mask

As can be seen, the median filter also works very well with high density SnP noise and can turn an image that is 70% noise to an image that is comprehensible with a large enough mask. For this lab, 7x7 seemed to do the job. As was explained, the larger the filter, the more noise filtered, but the output is blurrier.

The next part of question 3 was to attempt to remove SnP noise with the uniform mask. The attempt can be seen below.

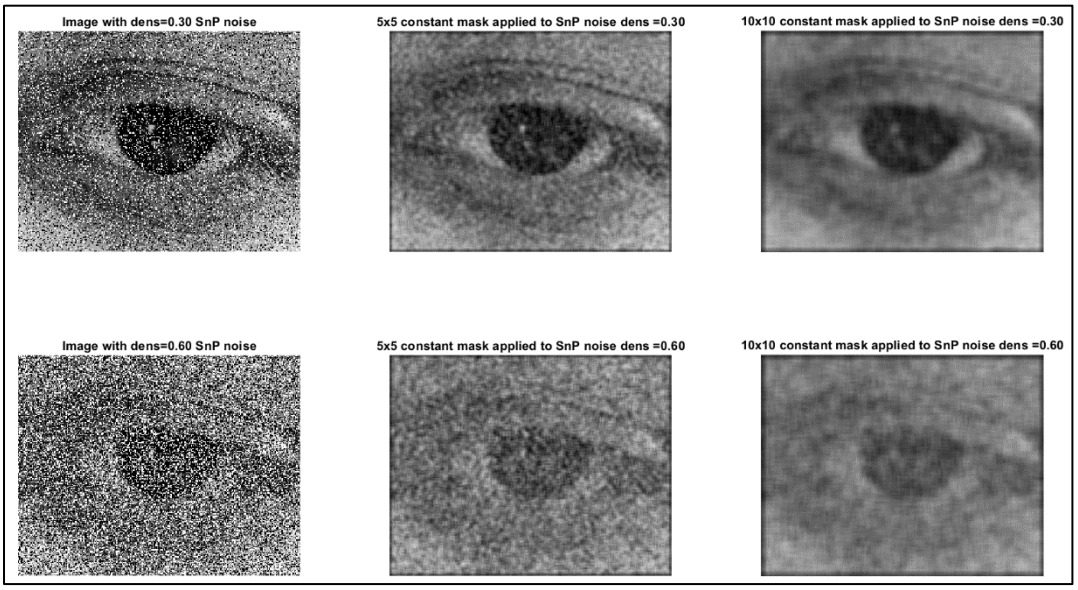

Figure 4: SnP of 2 densities being filtered with a constant mask

As can be seen, the filtered images are not filtered properly, and the constant mask seems to just smudge the image, blending the noise in the output image. This output is much worse than the one using the median filter, even with lower noise density. 70% noise was not included since the outputted image was almost useless, and unrecognizable. Larger masks also caused more blur, with minimal aid in making the noise disappear, so this was not included.

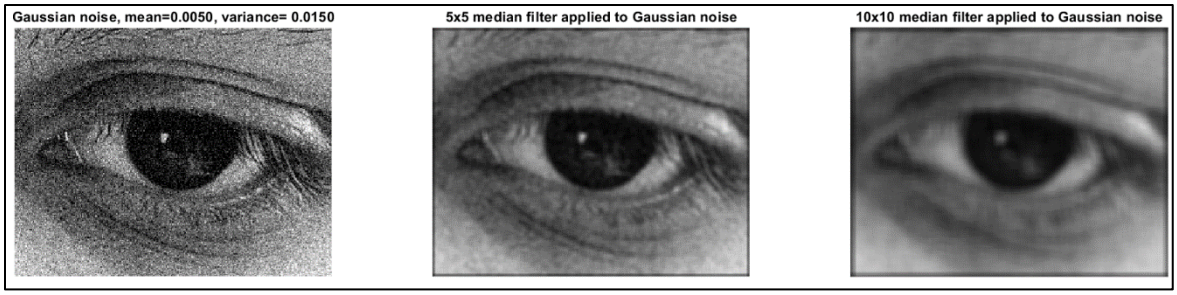

Figure 5: Image with Gaussian noise, filtered with median masks.

Above we can see the median filter being applied to an image with Gaussian noise. This has a similar effect as when the constant mask was applied but does a worse job at filtering the gaussian noise out.

In the end, the median mask works better for SnP noise, and the constant mask works better for Gaussian noise, but only slightly.

Creating the function “n(x,y) = A cos[2*pi*(((u0*x)/M) + ((v0*y)/N))]”, 4 different cases were tested, with different values of u0 and v0. Below you can see the functions as images.

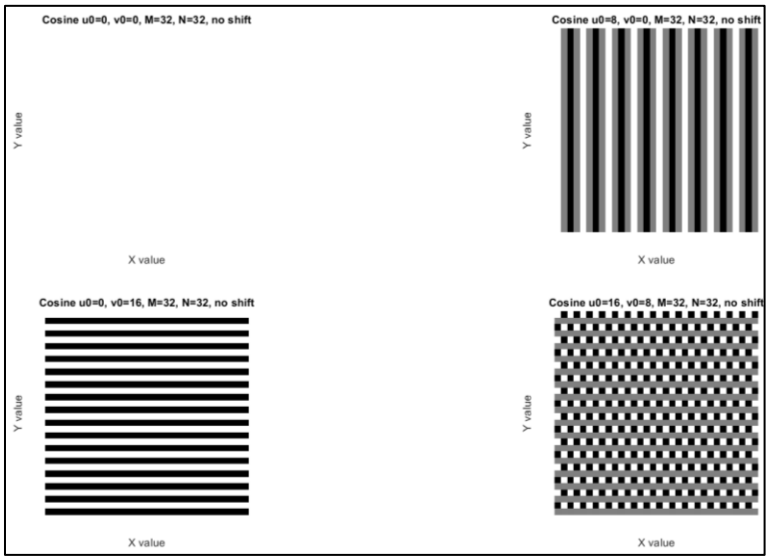

Figure 6: n(x,y) displayed as images (no vertical shift)

This is the sinusoid graphed with no shift upwards. The reason there is gray and black is because the image is rescaled before being outputted. For this case, if the output was not scaled, the gray portions would become black, since those would be points with values of 0, the black has a value of -1, and the white is value 1. This cosine’s values range from -1 to 1, which was ideal for an output, so a second test was done, with the sinusoid shifted upward by 1 unit. This may be seen below.

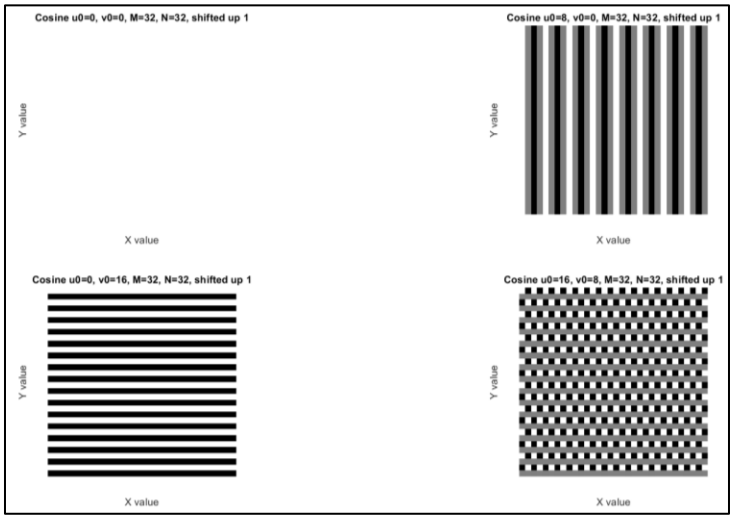

Figure 7: n(x,y) displayed as images (shifted vertically)

The values now range between 0-2. When the output is rescaled, the output is identical to the non shifted output, but if not rescaled, the gray part (which would have been black in the first case) would have been white in this case, but it was decided that it being rescaled would be a better output.

The non-shifted image would be darker than the light image, and the rescaled version allows us to see that without a very odd output.

Creating a matrix of ones, and padding the matrix with 0's, the result what a 2d rect function. If the total size of the image was MxN, then the size of the rect function was (M/2) x(N/2). The ones matrix was multiplied by 255 to make the inside of the rect 255, and the outside is 0. Below the output can be found.

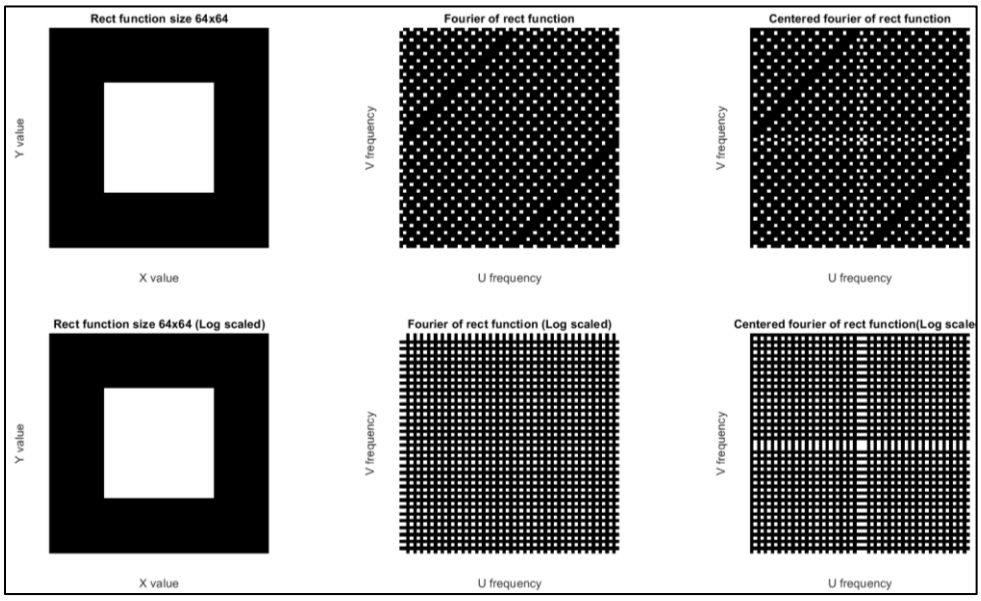

Figure 8: Rect function with its Fourier transforms, and the function log scaled

The left diagrams are r(x,y) as an image. The graphs to the right is the fourier transform, and the centered fourier transform. The bottom 3 figures are just the log scaled versions of the ones ontop of them.

If a small enough value of M and N were used (both were set to 32 for a clear output) there was no need to scale values. Below the non-centered Fourier transform of the cosine function can be found n(x,y)).

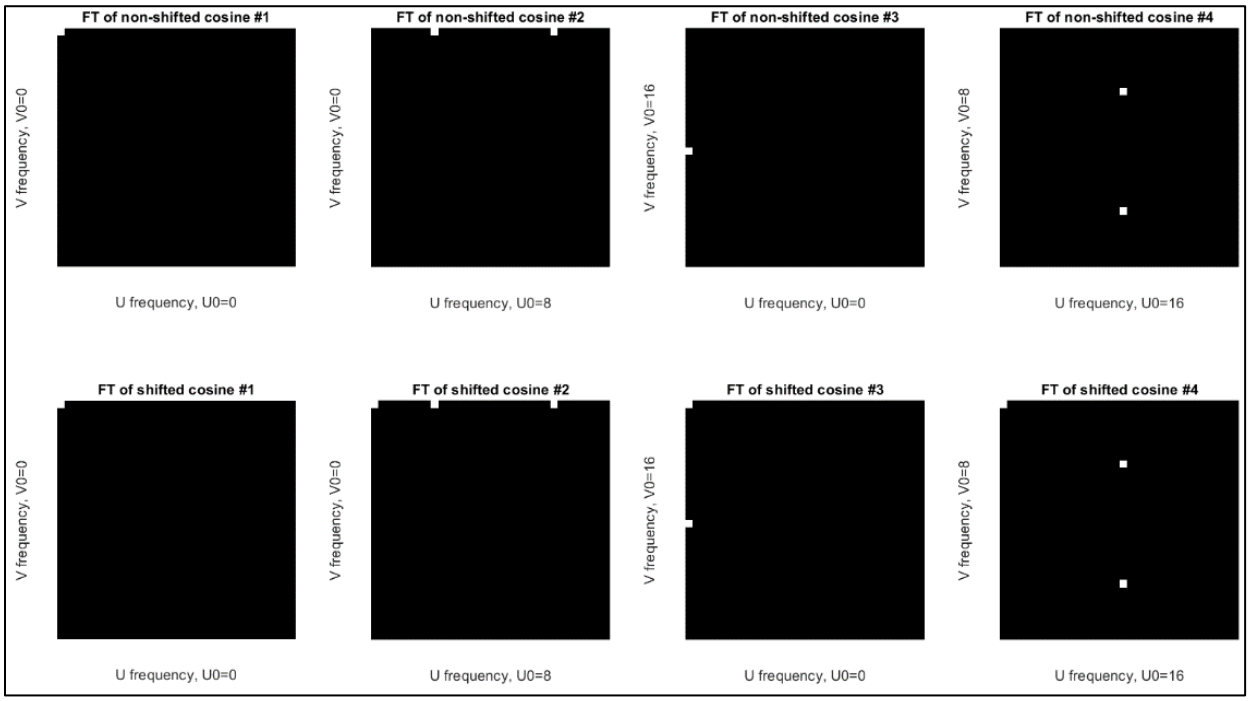

Figure 9: non-centered cosine Fourier transform for the 4 cases

For the non shifted case, we can see only the first case (where cos becomes a constant) has a DC value, the other terms just have frequency components, and nothing at the zero frequency point. The only difference between the shifted (cosine that was shifted up 1 unit) and non shifted cosines is that the shifted cosine has a fixed DC offset, due to the shift. The rest of the values are the same.

The outer portions of the square are the low frequencies, and the center of the square holds the highest frequencies, which s why we see the dots closest to the middle is when U0 is at its max value, and V0 has a non zero value. Centering the frequencies with “fftshift()” flips this around; the center now holds low frequencies, and the outer part hold the high frequencies. The following figure shows this.

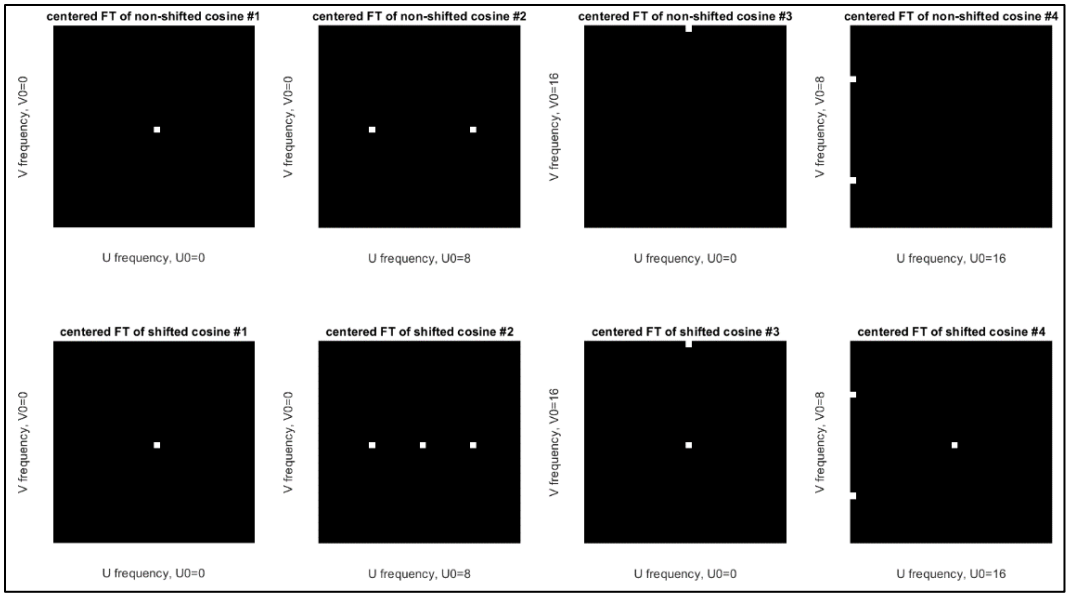

Figure 10: centered cosine Fourier transform for the 4 cases

As can be seen above, the low frequencies got pushed to the center of the graph, and the high frequencies got pushed to the sides. This is right, since the lowest frequency we have, which is DC, can be found directly in the center of the diagram. The Fourier transforms of the rect function can be found above, with its scaled outputs also.

The following diagram shows the low pass filter that was made. Since this LPF is being applied to a centered spectrum, we need a makeshift rect function, with adjustable size, to choose how selective the low pass filter is. Below and example of the low pass filter is outputted.

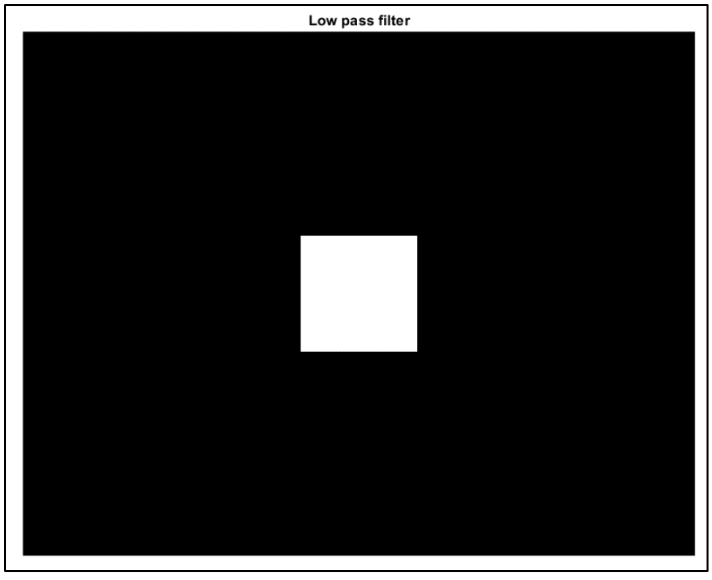

Figure 11: Example of low pass filter used for the lab

The eye image had a very wide range of frequencies, which made it perfect for testing the low pass image.

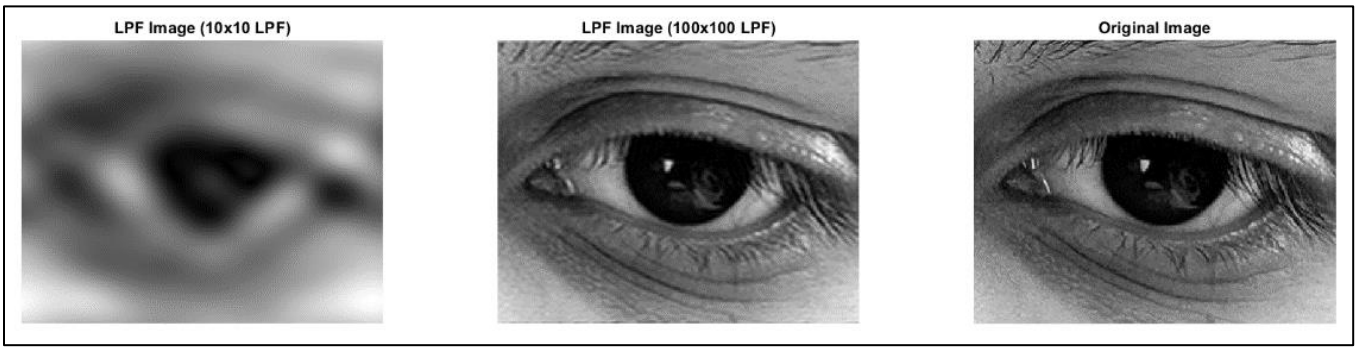

Figure 12: LPF of different size applied to eye image

As can be seen above, the mask applied causes the image to blur. The smaller the mask that is applied, the blurrier the output image is, because of the lack of frequencies that are passed to represent the image. When a mask of 100x100 is applied, the output image is almost identical to that of the input, only a slight bit blurrier.

The only zero padding that was necessary was that on the rect function, to make the entire image holding the rect function the same size at the image being filtered. No zero padding was necessary on the picture of the eye itself.

For the high pass filter, the same filter was applied, but to the uncentered spectrum. Since in the centered spectrum the low frequencies are held in the center, those are the ones passed. In the non centered version, the high frequencies are in the middle, so if the filter is applied, the high frequencies are passed.

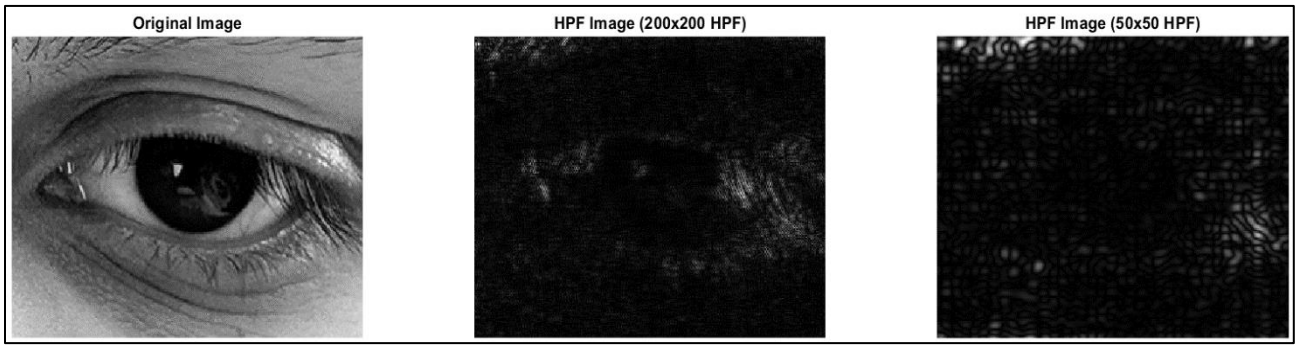

Figure 13: HPF applied to eye image

The 200x200 HPF passes a substantial amount of the frequencies, which allows it to show the general shape of the eye. In the 50x50 HPF, we can see there is no shape at all, and does not even remotely appear like the eye image.

The zero padding is only necessary in the filter image, to make it the same size as the eye image. There is no need to pas the eye image if both images are the same size.

This lab was incredibly useful to see the use of the fft2 function in MATLAB, and the use of using the frequency domain over purely spatial domain modifications. The sizes of filters are a large influence on the outputted image, as well as if the inputted frequencies are centered or not, making it either a high pass or low pass filter.

Dealing with noise is a large part of image processing and getting rid of it is crucial to keeping image accuracy. In this lab, we analyzed the pros and cons of the median filter and constant mask with noise like the Salt and Pepper and Gaussian noise.