Graham Skeats

Computer Vision Project 1 Report

For the first of the three main parts of this project, spatial filtering, I implemented a set of functions in a python script that allowed me to specify and apply filters of various sizes and values to images. I found that the trickiest part of this section was figuring out a way to specify the values for the filter. I solved this by creating a function that manually defines filters pixel location by pixel location which can be tedious for large filters but actually made the function transparent since I could visualize the value at each location. This made it a lot of fun to experiment with different values at different locations in the filter. For the larger filters I used built in cv2 methods to either create the filter as in the case for the large gaussian filters or I used an actual median blur for the median filters.

Gaussian and Median Filters:



Figure 1 27x27 Median Blur Puppy

Figure 2 9x9 Median Blur Puppy

Figure 3 3x3 Median Blur Puppy

Figure 4 27x27 Gaussian Puppy

Figure 5 5x5 Gaussian Puppy

Figure 6 3x3 Gaussian Filter

Figure 7 Original Noisy Puppy with Max Noise

What really jumped out at me was the difference that the size of the filter made on the resulting image. It is easy to see here that the 27x27 filters from both the median and gaussian blurs have a much greater effect than that of the smaller 3x3 filters. What also struck me was this color change from the original picture which I believe is a channeling issue since red in the original image is treated as blue in the resulting filtered image. I wasn’t able to fix the channeling issue since I think my OS swaps or interprets the channels as I save the image.

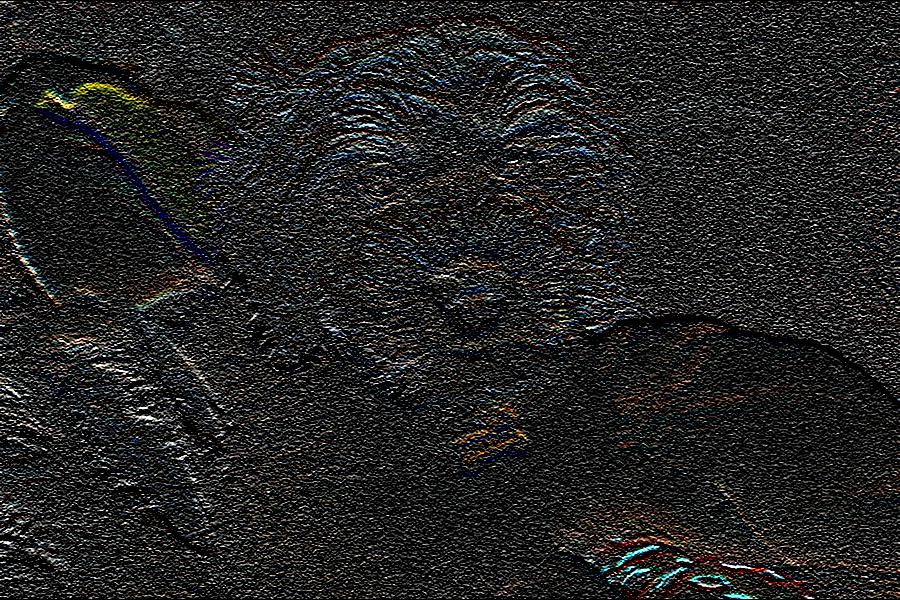
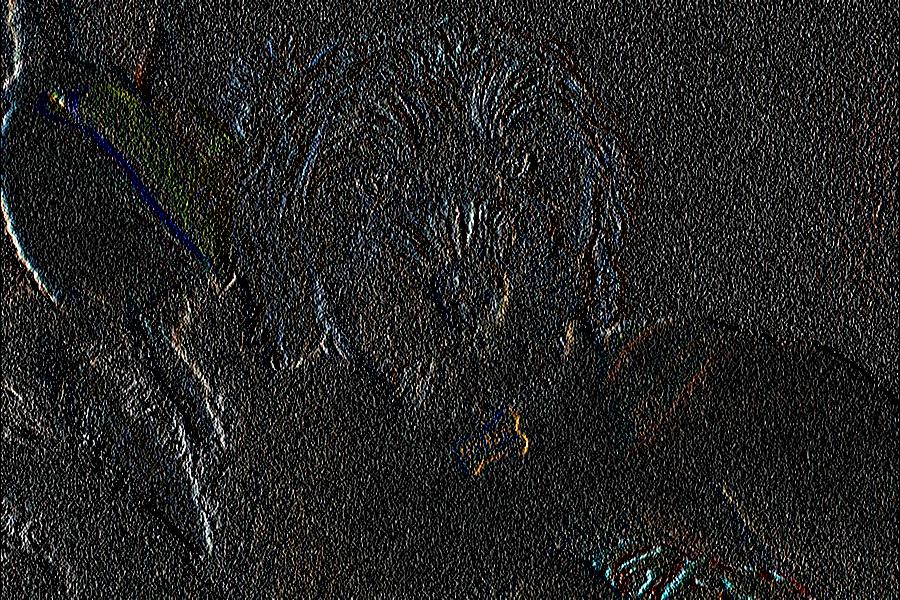
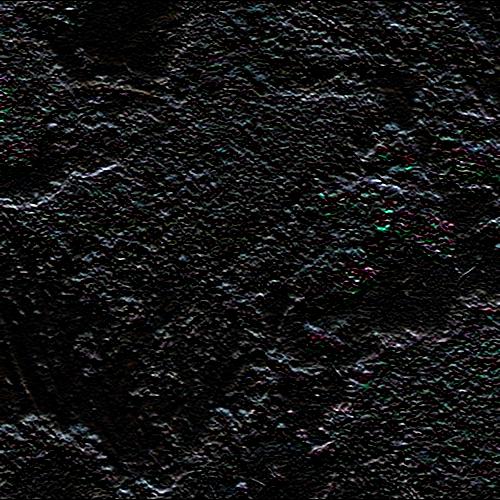
Edge Detection:

Figure Landscape Edge Horizontal

Figure Landscape Edge Vertical

Figure Original Landscape Photo

Figure 11 3x3 Horizontal Edge Noisy Puppy

Figure 12 3x3 Horizontal Edge Quiet Puppy

Figure 13 3x3 Vertical Edge Noisy Puppy

Figure 14 3x3 Vertical Edge Quiet Puppy

Edge detection was a very interesting exercise that was a little challenging. I used two edge detection filters to detect the vertical and horizontal edges in the image data. What is clear even from looking at these small photos, the images with noise are far more resistant to edge detection compared to the noiseless, or quiet, photos. Just looking at the different between the two sets of puppy photos its obvious that clean image data lets spatial filters extract far more information from the data. This suggests that rather than immediately attempting edge detection on a noisy photo, it makes sense to use a denoising filter such as a gaussian or median, to clean up the image data and extract cleaner edges.

For frequency analysis I wrote a separate python script and simply imported my spatial filtering script so that I could reuse any useful code. Here are some sample images from my investigation:

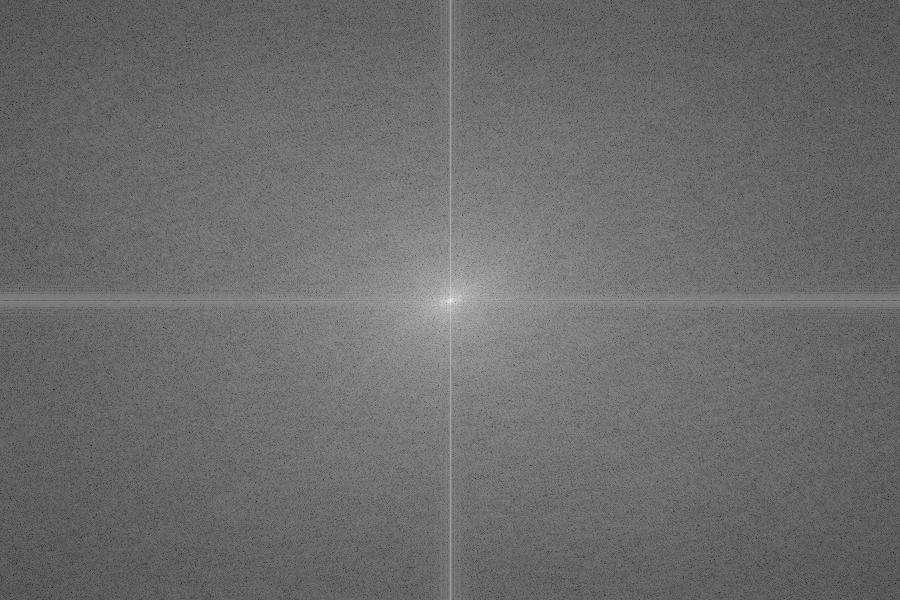
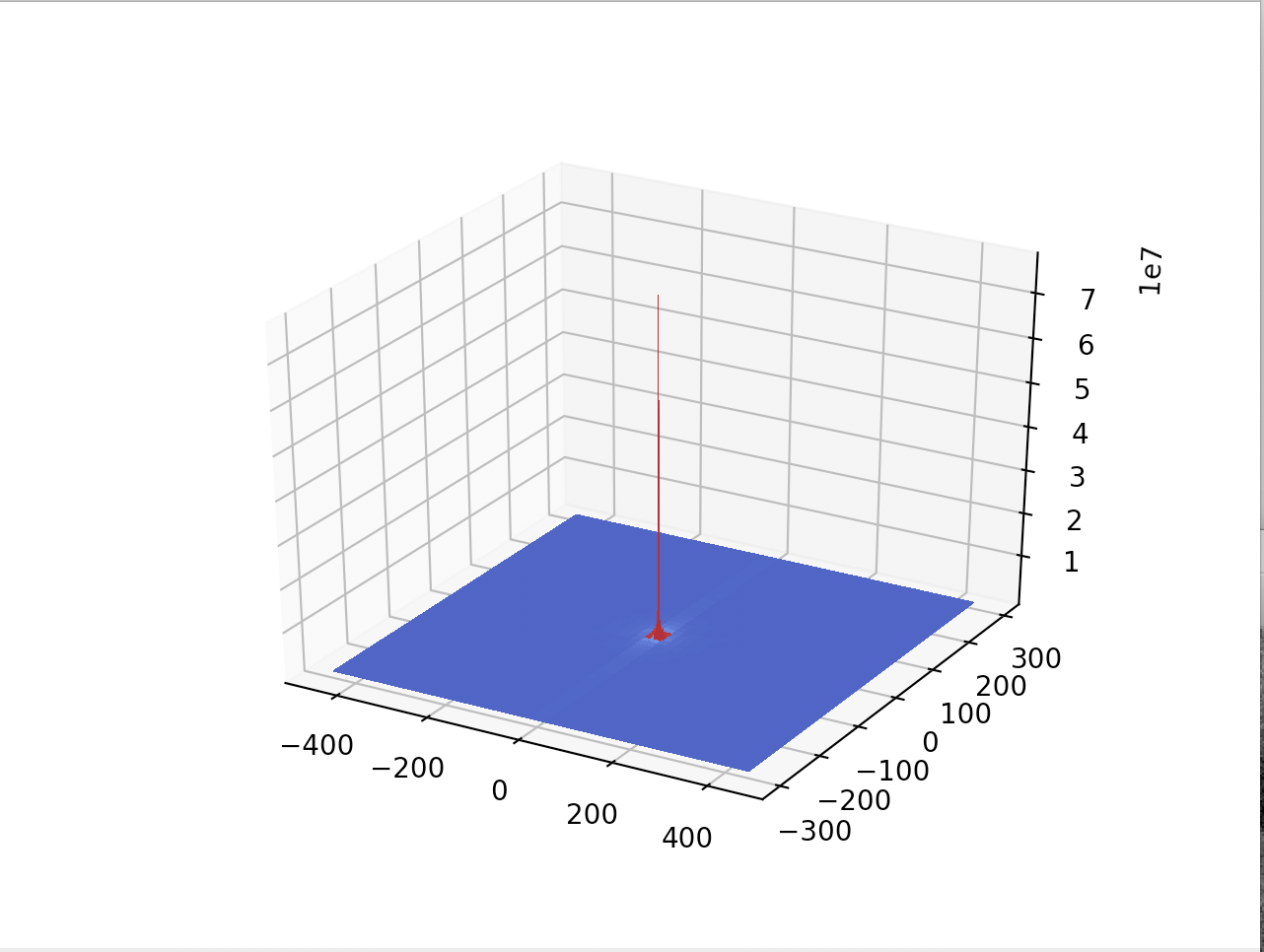
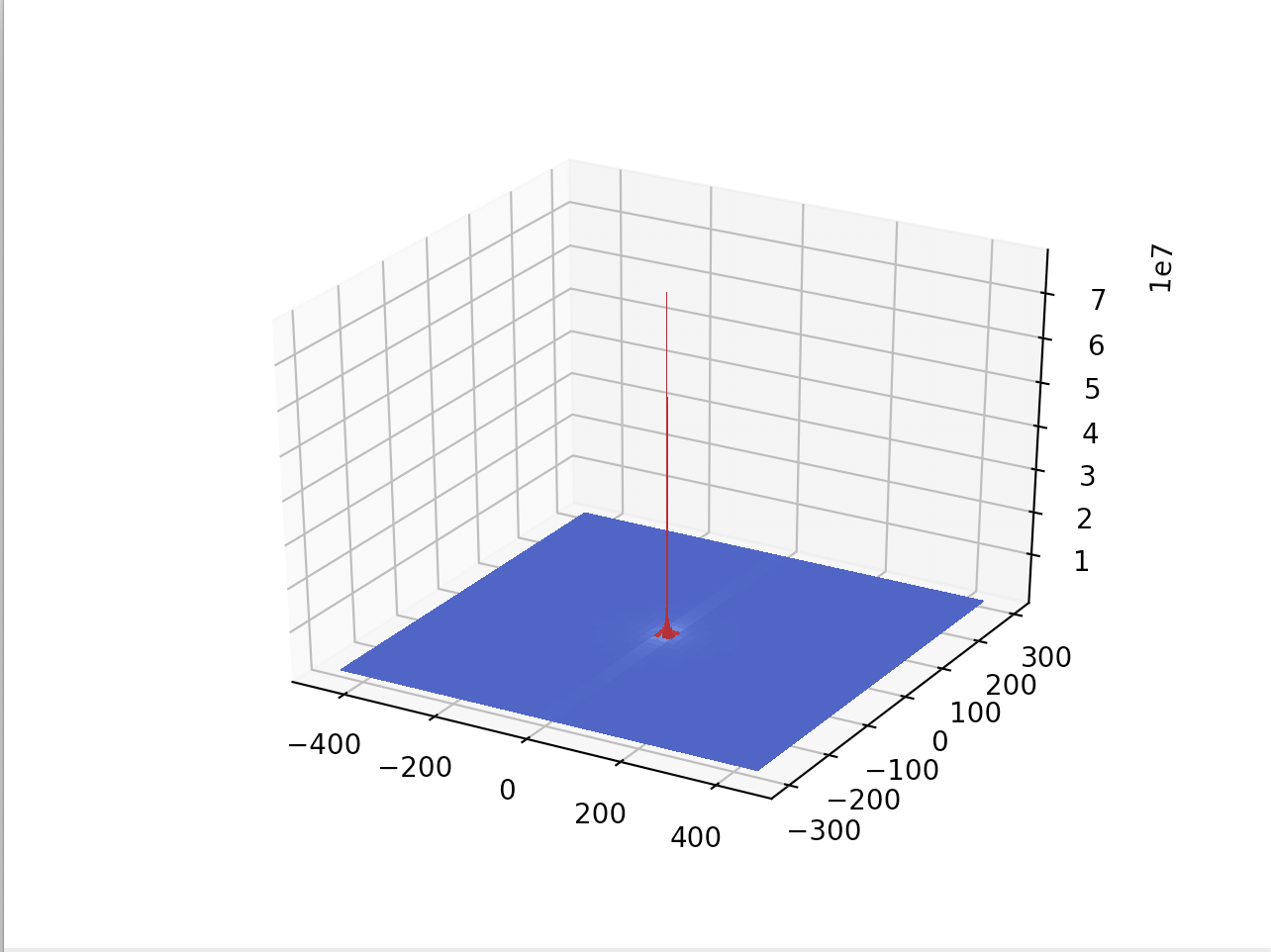


Figure Quiet Puppy Basic Fourier

Figure Basic Puppy Fourier Noisy

Figure Quiet Puppy Log Plot Vertical

Figure Noisy Puppy Log Plot Vertical

The quiet and noisy images are startlingly different in the way that the magnitude of the Fourier coefficients decay. The noiseless image has strong, clear lines along the x and y axes while the noisy image has much fainter lines along its axes and is much more uniformly grey in the space in each quadrant of the plot. This suggests that noise in images is recognized by the Fourier transform as frequencies that otherwise wouldn’t be simply because random information introduces random frequencies. The log plots show these higher values around the axes but away from the center of the image in the first place because these frequencies are much lower and are more likely to appear in the picture of this resolution than much higher frequencies. This suggests that should the resolution of the same scene be dramatically increased, it would be reflected in these Fourier plots.



Figure Quiet Puppy with several low frequencies set to 0

Figure Quiet Puppy with only the [0][0] frequency set to 0

Figure Quiet Puppy with the entire x axis set to 0 eg.[0]

Filtering out the low frequencies left a ghost like puppy and actually looked similar to the results obtained by performing edge detection. This makes sense since an edge by definition is a high frequency “event” that occurs in the picture so it should remain unaffected by zeroing low frequencies, as we see here. It is also interesting to see the effect of zeroing more than one frequency, with more and more frequencies set to 0 large shapes begin to wash out and the puppy appears even more ghostly, losing the outline of his body and the pillows around him. This result is not surprising since low frequencies should compose the larger object in an image, especially if they are only one color so no changes occur over the area of pixels that represent it.



Figure Quiet puppy with every frequency except the 5 lowest set to 0 (coefficients[5:599])

Figure Quiet puppy with more high frequencies set to 0 (coefficients[50:599])

Figure Quiet Puppy with highest frequency only set to 0 (coefficients[599])

Figure Quiet Puppy with a set of highest frequencies set to 0 (coefficients[500:599])

The high frequency filtering was actually more revealing than the low frequency filtering. I started by only zeroing the highest frequency, with almost no effect on the image. Next I incrementally zeroed greater and greater sections of Fourier coefficients by using the range [:] operator on my 2D array of Fourier coefficients that I had returned from my DFT function. Even by zeroing all but the lowest 50 frequencies I hardly changed the image at all, until finally I got a drastic effect by zeroing only 45 more so that only the lowest five frequencies remained in the image. While surprising to see, this result intuitively makes sense since the low frequencies in images are so drastically higher than a given individual high frequency. It was also interesting to see the low frequency zeroing work in reverse, as first detail washed out of the image leaving only shapes and regions untouched.

By zeroing out Fourier coefficients it is possible to perform both edge detection and smoothing. Even though the pixel manipulation is occurring in the Fourier domain as opposed to the spatial domain the same result can be achieved suggesting interoperability between the two domains, which we now know as the convolution theorem. This also raises the question is there an advantage between performing edge detection and smoothing in the Fourier or the spatial domain. I suspect that besides the ability to more finely tune smoothing and edge detection in the Fourier domain by precisely selecting what to leave out, there is a computational advantage to performing item by item multiplication in the Fourier domain as opposed to picking up and moving a mask around a large image repeatedly.

For the final component of the project I again wrote a separate python script called FrequencyFiltering and simply imported my other two scripts to reuse some of their code. The low pass filtering was easy to implement but Butterworth and specifically high pass filtering required a little more thought. Eventually I simply added a parameter that allowed me to get a high pass filter by adding negative one to my Butterworth filtered coefficients which would leave only the high pass information left to reconstitute the image.

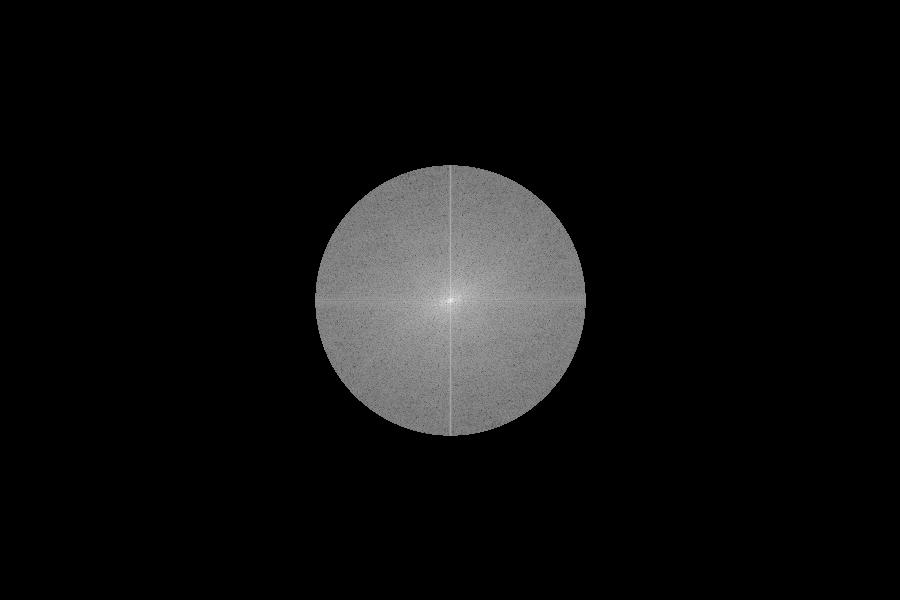


Figure Low Pass at cutoff at.25 magplot for Quiet Puppy



Figure Low pass filter at cutoff at.25 applied to Quiet Puppy

This first filter with the ideal low pass produced the expected results. By decreasing the value of many high frequencies values little change occurred in the image just as we observed in the zeroing of coefficients. However, since this ideal low pass filter affects frequencies in all directions and not just along the axes no artefacts are introduced into the image as we saw when we simply zeroed a set of coefficients along a line. This suggests that the low pass filtering might actually be an even more effective way to manipulate frequencies in an image since its effect is more distributed to many parts of the photo. By shrinking the cutoff significantly we can attain a similar dramatic effect as in zeroing almost all of the coefficients earlier but the image will again be cleaner and with fewer artefacts since frequencies in all four quadrants are affected evenly(the filter produces a smaller but circular area that frequencies can get through).

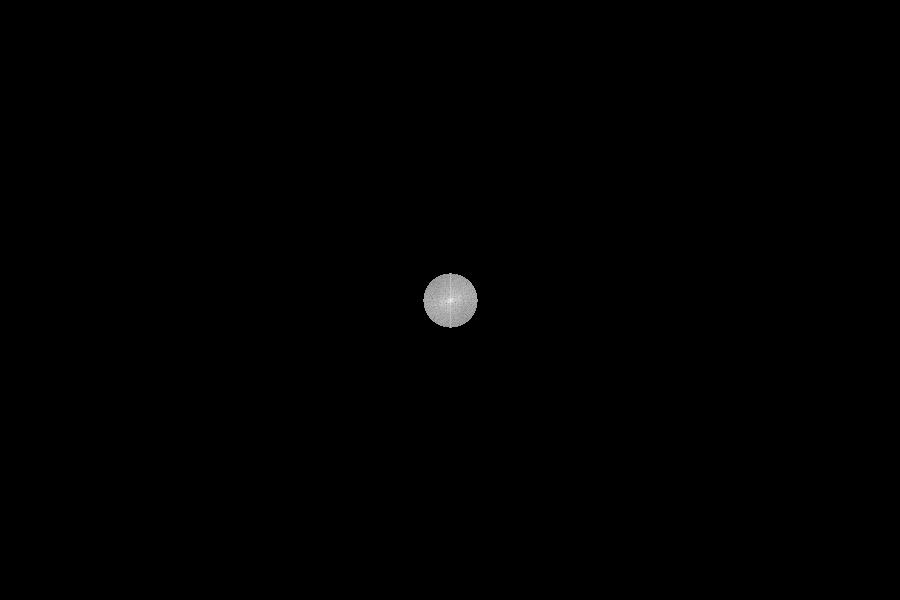


Figure Quiet puppy with ideal low pass filter, cutoff set to .05

Figure Mag plot of the ideal low pass filter with cutoff set to .05

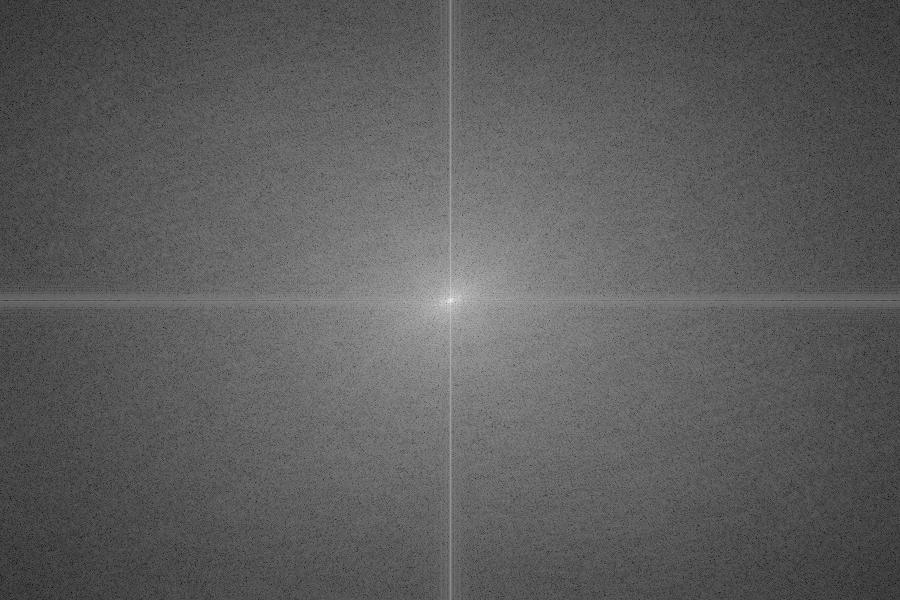
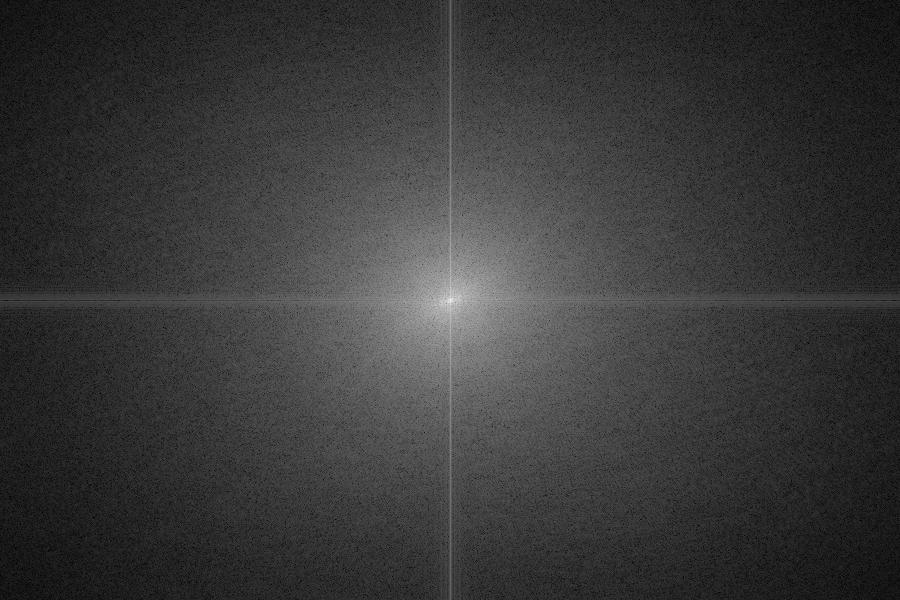
 Finally, the Butterworth and high pass filters continued the recent trend of producing more evenly blurred and sharpened images. By more smoothly dealing with the frequencies at the edges of its filter, the Butterworth was able to even more evenly alter the component frequencies of the images, preventing artefacts from entering the reconstructed image.

Figure Quiet Puppy with Butterworth filter, cutoff at .05

Figure Quiet Puppy Butterworth filter mag plot with cutoff at .25

Figure Quiet puppy with butterworth filter, cutoff at .25

Figure Quiet Puppy with butterworth filter, cutoff at .05

Here it is easy to see the relationship between the ideal low pass and Butterworth as shrinking the size of the filter again more dramatically affected the image as more high frequencies were filtered out. However, it is also important to notice that the mag plot of the fourier coefficients look very different. Instead of circles the Butterworth gently decreases the values of the coefficients as they move beyond the size of the filter. This is why the Butterworth blurs the image even more smoothly than the ideal low pass.

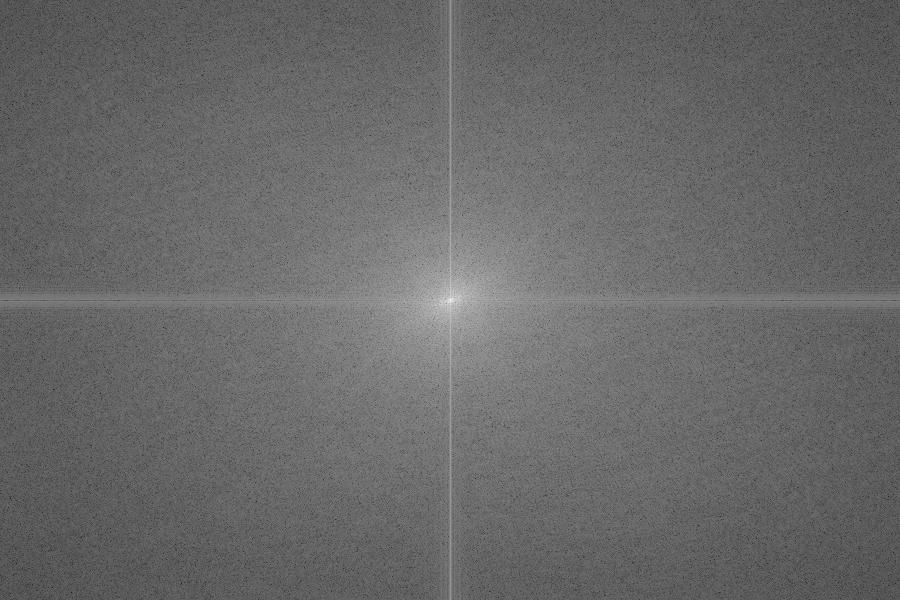


Figure Quiet Puppy high pass filter cutoff at .05

Figure Quiet Puppy high pass filter cutoff at .05

Finally, as we would expect, subtracting a filter than blurs edges from 1, what remains are sharp high frequency edges. Looking closely at the mag plot it is there are dark lines along the axes where our tight cutoff eliminated the low frequencies of the image. So in order to produce a more dramatic effect we can actually expand our cutoff so that more low frequencies are left out, which is the inverse of what we did earlier. Interestingly, I had some issues with saving the high pass filtered image which I believe after some research is due to the way the JPG files are stored since by convention they filter out low frequency values in order to remove information that the human eye can’t detect. This makes JPG images smaller without leaving a noticeable impact on the way the image appears.