CSC 249 Project 1 Report

Chris Dalke

March 1, 2017

1 Introduction

The goal of this project was to familiarize ourselves with numerous filters and operations that can be performed on images to extract useful data. Particularly, the project focused on kernel-based filters and the 2D FFT. This project helped me to build an understanding of image filtering in the space domain and how to compute and interpret a 2D Fourier transform of an image.

1.1 How to Run

If you wish to run my MATLAB program and regenerate the output images, the main file is `Program1.m`. When this script is run, it will load both of the input image files, and save all of the outputs to the `Output` folder.

I've chosen to output to files for all of the pieces of the assignment instead of displaying a figure, so the program will not display a figure when it is run. You can view the progress of the program by examining the command line output, which will display progress indicators.

2 Methods

2.1 Part 1: Basic Matlab Image Functions

For Part 1, I followed the set of instructions given. I chose to additionally save the figure from each step into a file so that I could add it to this report. I saved the figures by using MATLAB's `saveas(gcf,"filename")` which saves the current figure to a specified filename. I used this function consistently throughout the project to save the results of my program instead of displaying them to the screen. The outputted images are all stored in the `Outputs/images/` folder.

2.2 Part 2: Image Filters

2.2.1 Gaussian (Low Pass) Filter

I generated the Gaussian Filter kernel using the `fspecial` command, and the generated filter is shown below:

$$\begin{bmatrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{bmatrix}$$

To test that the filter was separable, I created a separate version of the Gaussian filter for the X and Y axis and applied them in sequence. The resulting image was saved separately after the process was finished, and analyzed in comparison with the 2D Gaussian Filter.

2.2.2 High Pass Filter

To compute the high pass, my algorithm loops through the pixels of an input image row by row, applying the image kernel and saving the result for each pixel into a second image. At each pixel, the algorithm computes a sum of the 3x3 area of samples surrounding the pixel, with each sample multiplied by its respective coefficient in the kernel matrix.

I chose the following kernel for the high pass filter:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

To deal with surrounding pixels, I clamp the X and Y sampling positions to within the image body, so any pixel sampled outside of the image will use the value of the nearest in-image pixel. I chose this instead of zero-padding because it avoids a black border on the edge of the resulting image.

For the output file, I chose to clamp the resulting image pixel values instead of scale the values since this would most effectively show the high-frequency details emphasized by the filter without drowning out the detail because of extreme values in certain positions in the image.

2.2.3 Median Filter

The Median Filter was applied using the `medfilt2` command.

2.2.4 Sobel Edge Filter

I used the following kernel for the Sobel Edge Filter:

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

To create the kernel for the Sobel Edge Filter, I used the `fspecial` command. The Sobel Filter I chose to use emphasizes horizontal edges, similar to the High Pass filter, except only on a single dimension. This is shown graphically in the frequency responses of the filters. I chose to display the Sobel filter using scaled pixel values instead of the clamped pixel version because the scaled values better show the pixels with negative values instead of clamping which would hide them along pixels with a value of 0.

2.3 Part 3: The 2-D Fourier Transform

For Part 3, I chose to use the first option: computing the 2D Fourier Transform using the builtin Matlab functions. I created a function that took in an image, and outputted the FFT array with unscaled and logarithmically scaled versions

in both unshifted and shifted version. To create an image from the array, I took the absolute value of the array and displayed it using the `imagesc` function.

The results of Part 3 are saved into the `Outputs/fft/` folder.

3 Results and Discussion

3.1 Part 1: Basic Matlab Image Functions

Figure 1: Imshow for Image 1

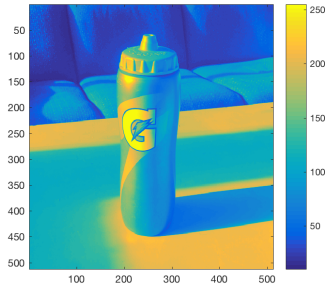


Figure 2: Imagesc for Image 1

The results of `imshow` and `imagesc` are shown above for Image 1. imshow displays an image by directly interpreting the values in the image array as a pixel color. In this case, each of the values is an unsigned integer indicating a greyscale color which is displayed as a pixel value from 0-255. If a pixel is outside this value, it will be clamped. By default, imshow does not show the coordinates on the side of the plot, unlike imagesc which displays the coordinates by default.

`imagesc` displays an image by examining the range of values in the image array. The function then uses a colormap to map the minimum, maximum and intermediate values to a set gradient. The values in the inputted array do not have to be in the 0-255 range to display an image; the scaling will shift the values to a range that is always visible. The colors in the gradient can be shifted using the colormap command, and the colorbar currently in use can be displayed next to the plot using the colorbar command. The result of using a different colorbar is shown below.

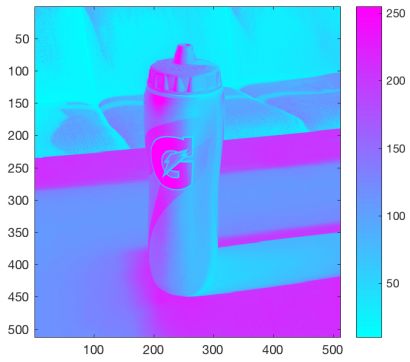


Figure 3: Modified Colormap for Image 1

3.2 Part 2: Image Filters

3.2.1 A Note on Scaling

Some of these filters produce pixel values outside the range of 0-255 in the output image. It's necessarily to decide how to treat these images, and there are two options: Scale the image values or clamp them to valid pixel colors. An example of this is shown below, based on the result of the Sobel Filter:

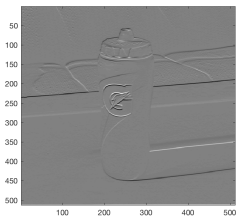


Figure 4: Scaled to 0-255

Figure 5: Clamped to 0-255

For my filter code, I chose to generate both the clamped and scaled versions and I chose to display the image that makes it easier to see the details given by the filter. I've described my choice of clamping or scaling in the Method section for relevant filters.

3.2.2 Gaussian (Low Pass) Filter

The Gaussian Filter blurs the image by removing high-frequency (fine details) from the image and leaving low-frequency pieces of the image. A result of the Gaussian Blur is shown below. Repeated application of the Gaussian Blur will result in more of the high-frequency data being removed, and therefore a higher blur. The blur is shown after 1 iteration and 5 iterations.

Figure 6: Gaussian Blur, 1st Iteration

Figure 7: Gaussian Blur, 5th Iteration

This filter is separable, which means that it can be applied as two separate 1D components and the result will be identical to the 2D filter. This is proven to be true, and the results of applying the horizontal and vertical blurs separately are identical to applying the 2d filter, as shown below.

Figure 8: Result of 2D Gaussian Blur

Figure 9: Result of Successive 1D Blurs

3.2.3 High Pass Filter

The High Pass Filter performs the opposite operation as the Gaussian Filter, only highlighting the high frequency data (fine details) in an image and removing any low frequency data. The result of this filter will be an image that is black except for white lines wherever high frequency patterns are detected.

Figure 10: High Pass on Image 1

Figure 11: High Pass on Image 2

The High Pass Filter is essentially an edge / pattern detection, since edges and patterns are regions of high frequency data. The filter can be ineffective in the presence of a large amount of noise or texture, as seen in the *Effects of Noise on Filters* section.

3.2.4 Median Filter

The Median Filter is effective at reducing very high-frequency data, and smooths out the image while keeping the edges relatively unimpacted. This is especially valuable for some types of noise removal. Below, the results of the Median Filter are shown on an image with no noise, with impulse noise, and with Gaussian noise. These results are discussed further in *Effects of Noise on Filters.*

Figure 12: Result, Normal Image

Figure 13: Result, Impulse Noise

Figure 14: Result, Gaussian Noise

3.2.5 Sobel Edge Filter

The Sobel Edge filter is similar to the High Pass Filter in that it is another way to show high-frequency detail in an image. The Sobel Filter displays high-frequency information on a single axis, in this case, horizontal. The Sobel Filter will generate patterns of lines similar to the high pass filter, but because of the negative signs on one part of the filter kernel as shown in the Methods section, some of the lines will display as black and some will display as white depending on whether the high-frequency data is a change from bright to dark or vice versa. This had an effect on how I displayed the data; I chose to display the result of the Sobel Filter by scaling the pixel values because clamping the values would result in half of the lines (the black lines) becoming invisible as the background and lines would both be clamped to become black.

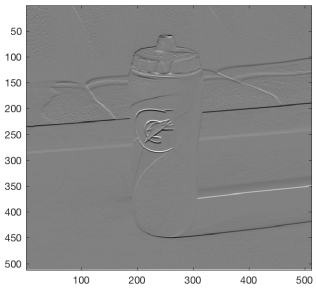


Figure 15: Sobel on Image 1

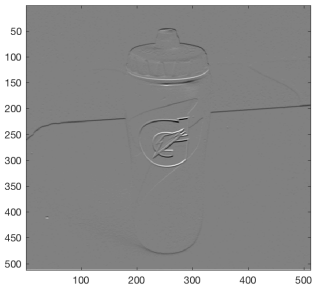


Figure 16: Sobel on Image 2

3.2.6 Effects of Noise on Filters

I applied Gaussian and Impulse noise to both images using `imnoise` and repeated the above filters.

Gaussian Noise is noise in which the values that the noise takes on are in a Gaussian distribution, so extreme pixels (white and black) are less likely than gray pixels.

Impulse Noise is noise in which random pixels are switched completely on (white) or completely off (black). Impulse noise seems more disruptive to the image, which happened to also make it easier to remove from the image since the Gaussian noise is closer to high-frequency data that would be likely to be present in an image.

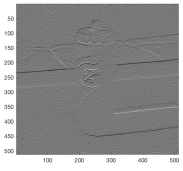
I've shown the results on Image 1 for each filter and type of noise. The results are also present for Image 2 in the `Output/filters/` folder.

Gaussian Noise

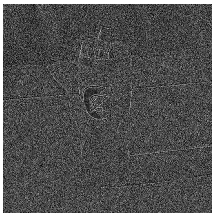
Noise Image

Gaussian Filter

Median Filter



Sobel Filter



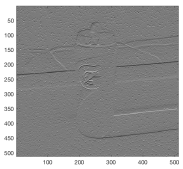
High Pass Filter

Impulse Noise

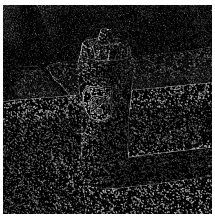
Noise Image

Gaussian Filter

Median Filter



Sobel Filter



High Pass Filter

The Sobel Edge Detector and High Pass filter respond badly to noise, since they both detect high frequency data in an image, and noise adds high frequency data.

The Median Filter is very effective at removing impulse noise because impulse noise pixels are outliers and therefore will rarely be the median of a pixel's neighborhood. The filter is less effective at removing Gaussian Noise because Gaussian Noise is more likely to be an intermediate brightness value and therefore noise pixels more frequently end up as the median of a pixel's neighborhood and are chosen to be displayed.

The Gaussian Blur is less effective at removing either type of noise because a noise pixel with the incorrect value will be taken into account in the resulting image as part of the weighted average of the surrounding area, where the Median Filter completely removes outliers present because of noise.

3.2.7 Reducing Affect of Noise on High-Pass Filters

A solution to reduce the effects of noise when performing High Pass Filtering or detecting edges would be to run the image through a Median Filter before passing it through the other filters. The Median Filter does a good job at reducing all of the impulse noise in the image. For the Gaussian noise, an effective method to remove noise might be to apply the median filter followed by the Gaussian blur, or to invoke either of the low-pass filters (Gaussian or Median) repeatedly until noise is sufficiently eliminated.

3.2.8 Frequency Responses of Filter

The frequency responses of the filters are shown below. The frequency responses roughly show what type of data from the image is kept by the filter. The area in the center of the plot around the origin represents low frequency data, and the area on the edges of the plot represent high frequency data. The Y axis represents the relative amount a given frequency is taken into account by the filter.

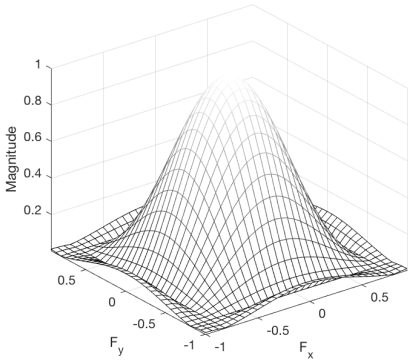


Figure 17: Frequency Response of Gaussian Filter

The Gaussian Filter is a low pass filter, meaning that it keeps low-frequency (larger scale data) while removing high-frequency data (finer details). This makes sense because the blur effectively removes all the small details from an image.

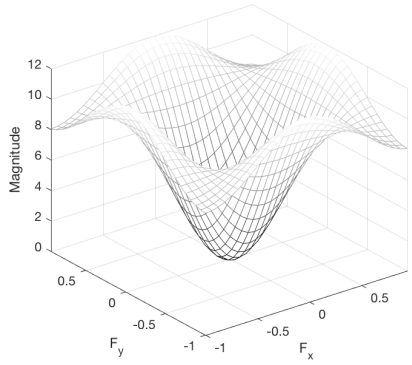


Figure 18: Frequency Response of High Pass Filter

In contrast, the high pass filter keeps little of the low frequency data, shown by the depression in the middle of the frequency response, and keeps the high-frequency details, shown by the raised edges of the frequency response curve.

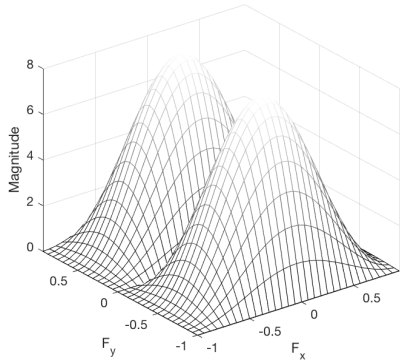


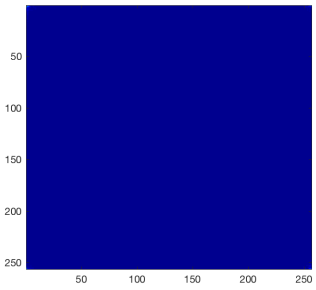
Figure 19: Frequency Response of Sobel Filter

The Sobel filter is similar to the High Pass filter's frequency response but it only shows high-frequency data on the horizontal axis, which shows up as two peaks. This curve is similar to the High Pass Filter but only on a single axis. If the peaks were also on the other axis, they would combine to show a similar frequency response as the High Pass filter.

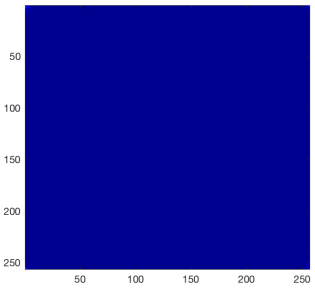
Since the Median Filter is not a linear system, the frequency response does not exist and cannot be displayed like the other filters.

3.3 Part 3: The 2-D Fourier Transform

3.3.1 FFT Display: No Scaling



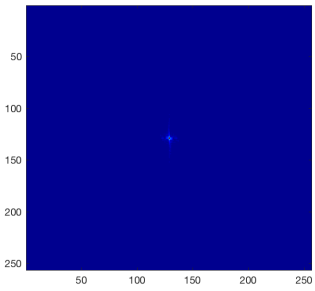
Unscaled, Unshifted Image 1



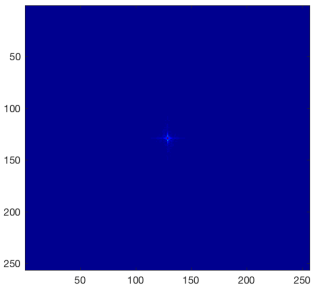
Unscaled, Unshifted Image 2

The unshifted magnitude of both images are shown above. I've colored the FFT in false color to make it easier to look at visually. Even then, almost nothing shows up on the FFT because it must be scaled properly in order to effectively see the data. The origin is located in the corners of the image.

Using the `fftshift` command shifts the origin to the center of the image, as shown below. The effects of this are better shown in the images that have been scaled logarithmically so you can properly see the data.



Unscaled, Shifted Image 1



Unscaled, Shifted Image 2

3.3.2 FFT Display: Logarithmic Scaling and Shift

The figures below show the unshifted and shifted magnitudes of the two images, but this time the FFT values have been scaled using `log` and a fixed offset has been added to each value.

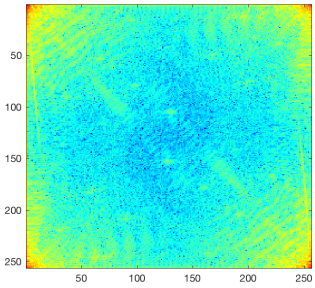


Figure 20: Scaled, Unshifted Image 1

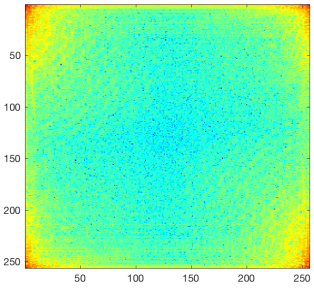


Figure 21: Scaled, Unshifted Image 2

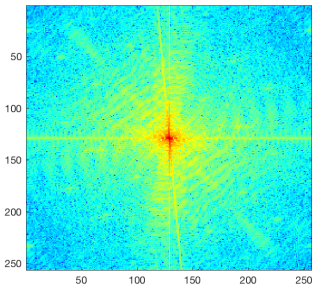


Figure 22: Scaled, Shifted Image 1

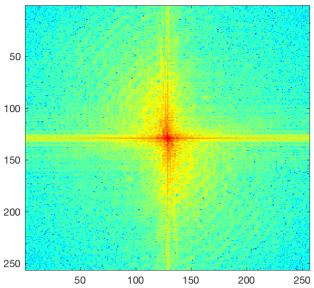


Figure 23: Scaled, Shifted Image 2

I prefer the set of images that uses a logarithmic scale, because this more effectively shows the full range of magnitudes in frequency responses. The original unscaled version is almost impossible to analyze for my set of images, so the scaling is essential to use of the FFT results.

Examining the particular results for the images, the FFT for both images shows a strong cross shape, indicating that there is pattern repetition on both the horizontal and vertical axes. In the FFT for Image 1, there are two faint 45-degree lines, indicating that there is periodic repetition at an angle. This is likely picking up on the pattern of dots on the couch, which is not visible in Image 2.

4 Appendix

The figures are displayed inline with the writeup. These images are also available in full resolution in the `Output/` folder. I've chosen not to include the images with the Appendix since I already included them as a part of the main document body.

5 Academic Honesty

I did not collaborate on this project and all work is my own.