Report on
Image Processing and Interpretation
Project 2

Submitted to
Dr. Bebis
for
CS474

September 21, 2015

By:
Aaron Whitehead
and
Jeremiah Berns

This report examines several introductory computer processing practice programs. The programs cover such things as spacial filtering using correlation, averaging masks, and Gaussian masks, median filtering, and sharpening using several common masks. The project was a great success, and each part of the project runs smoothly and correctly.

Technical Discussion

As the overall structure of the project is broken up into four sections, the technical writeup will also be broken into four sections.

Program one:

The first problem is a spatial filtering using correlation one. The basic idea is to take in any mask and apply it across the entire image and then review the output. The program needed the ability to take in an image and use it as a mask to be applied to a larger image. The basic algorithm goes as thus:

1. Take in the image that will be made into a mask.
2. Put all of the data into a vector while measuring how much padding you will need.
3. Open the image that will be filtered and pad it based on how big the mask was from step 2.
4. Iterate across the target image applying the mask, storing the output pixels in a new image.
5. Print new image for comparison reasons.

Instead of using a smaller picture to create a mask, the program also supports simply inputting your own padding values and corresponding mask values to manually create a mask. An important point to note is that instead of dividing the summed value of the weights times the values, we simply normalize them all to 255.

Program two:

The second program is a spatial filtering problem, using both averaging and Gaussian masks. The basic concept is to generate a mask based on certain criteria and apply it throughout the image, reviewing the results. The algorithm goes as follows:

1. For an averaging filter, simply create a mask filled with 1's. However, for a Gaussian filter, you need to generate a normal distribution depending on some sigma value.
2. Normalize the mask by dividing each cell by the total sum of all of the cells in the mask.
3. Pad the target image based on the size of the mask generated in step 1.
4. Iterate over the target image using the now normalized mask.
5. Store each new pixel value into a new image, and print the new image when done.

Program three:

The third problem is the application of a new median filter, along with applying the averaging filter from problem 2. The goal was to first generate a corrupted image, and then use these two different methods in order to see what the best way to correct the corruptions are. The algorithm is as follows:

1. Iterate through the image, and at each point randomly decide if that pixel should change, and if so, randomly chose black or white. This will apply a “Salt and Pepper” corruption to the image.
2. Create a vector to hold the data in the pixels around each pixel.
3. Iterate through image, storing each neighbor pixel value into said vector.
4. Sort the vector, and place the pixel in the middle into the new image at the current location.
5. Print new image.
6. Perform averaging on the image with the same size mask. Also print this average.

In regards to handling the averaging filter for comparison, see above for the averaging filter algorithm.

Program four:

Program four is very similar to say program two which simply applies different masks to images in order to discern certain information from the result, so the process is very similar.

1. Create a mask.
2. Apply mask to the image, do so twice if you need to do an x and a y gradient.
3. Optionally, combine the gradient masks if necessary.
4. Print the resulting image(s).

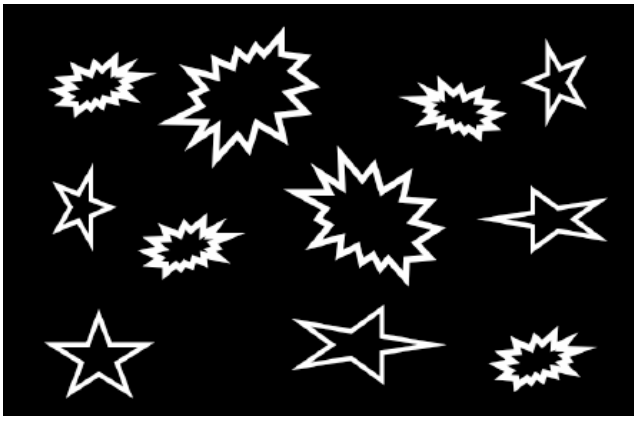
Results and Discussion

Once again, the results section, like the technical discussion section, will be broken into four core parts.

Problem one:

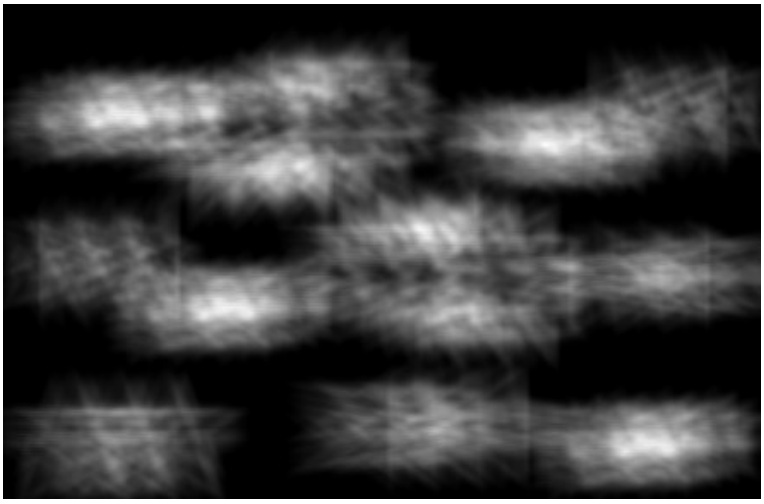
Problem one was the spatial filtering using correlation problem. The desired goal was to use a image that had a series of patterns on it, and then use a smaller image that was a single pattern. By then creating a mask out of the smaller image, and applying it to the overall picture, we should be able to calculate a correlation

coefficient of sorts in order to learn more about where the single pattern may be. This is also known as template matching based on correlation coefficients.



The given image.

The pattern to find.



The output Image.

As one can prove with a small amount of logic, the brighter spots are where there is a higher chance of the pattern being present. This can be understand, since you are multiplying two matrices with 1's and 0's, then the two matrices that have the exact same pattern on 1's and 0's will yield the highest overall value. So as you iterate the mask around the image, the spots where the mask “aligns” better with the underlying pattern will have a higher value, or in regards to the image, a brighter spot. Since moving the mask over slightly will only slightly miss the desired pattern, it will slowly fade away from the bright spot. Therefore, by scanning the image above using a threshold system, I can tell you exactly where the middle of each one of the patterns is at.

Problem two:

Both the averaging filer and the Gaussian filter are softening filters, but they still have a different output. The Gaussian filter, while still seeming just as smooth, seems to have many more discernible details. Below are the images for comparison.

Given SF image.

Given Lenna image

Gaussian filter size 7

Gaussian filter size 7

Averaging filter size 7

Averaging filter size 7

Gaussian filter size 15

Gaussian filter size 15

Averaging filter size 15

Averaging filter size 15

As you may be able to tell by reviewing the images above, the averaging filter at size 15 becomes almost impossible to interpret,

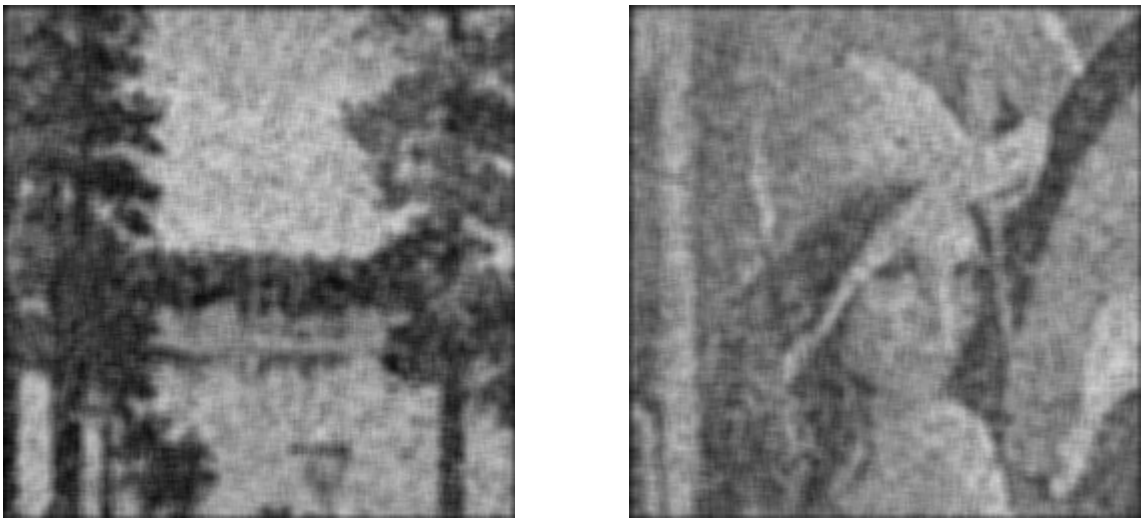
while the Gaussian filter at the same mask, while still quite smoothed, is easily understood to be a picture of a bridge. The Gaussian filter seems to be more of a softened edge approach, while the averaging filter seems to just sort of meld the image into the background.

Problem three:

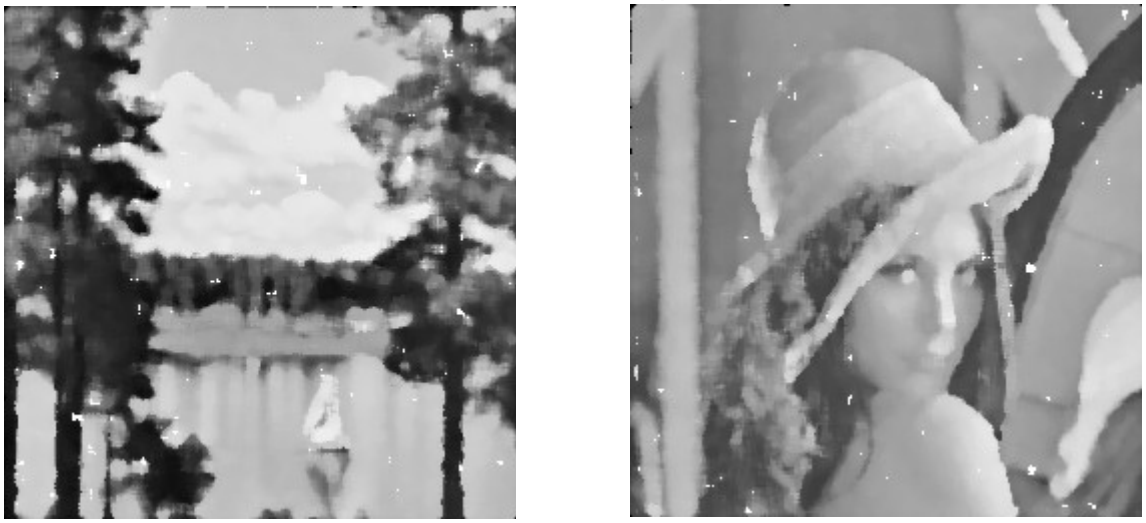
The third problem was very visually appealing to review, as the results are very stark and obvious. The problem asked me to first corrupt some data with salt and pepper noise, and then apply both average and Gaussian filters to soften them and to review the results. Below are the images.


Given Images

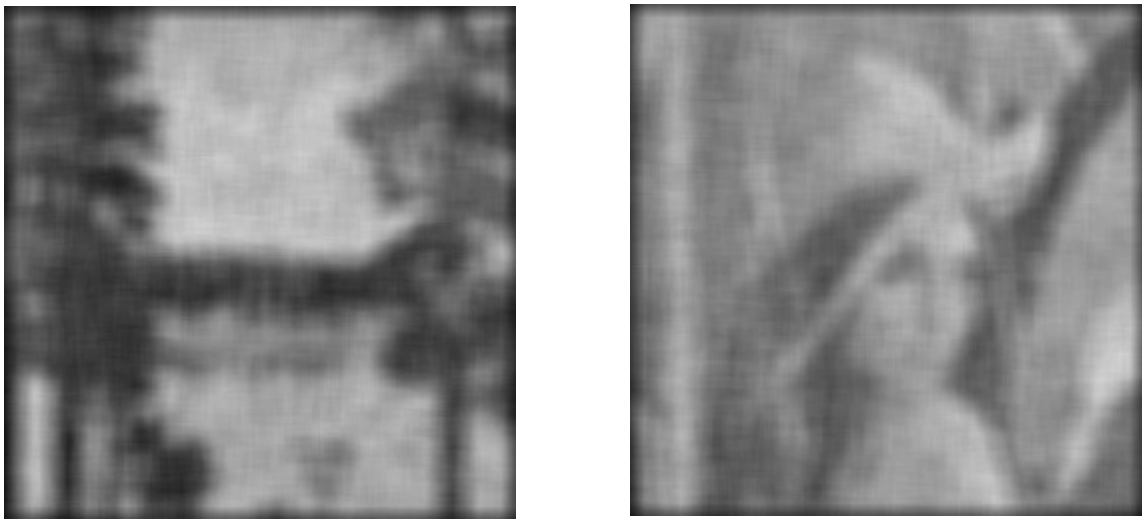

30% salt and pepper filter



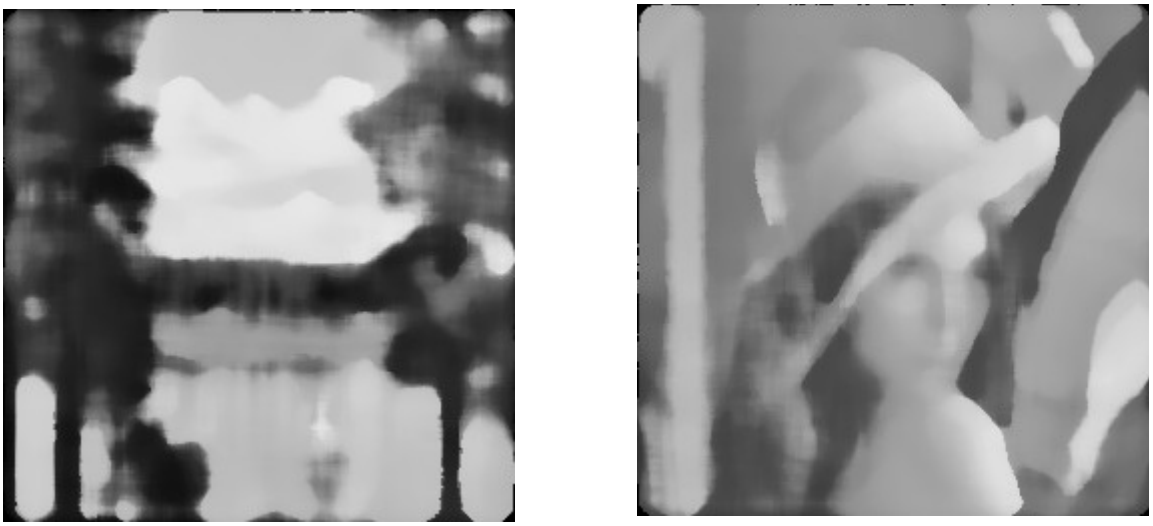
30% salt, size 7 averaging filter



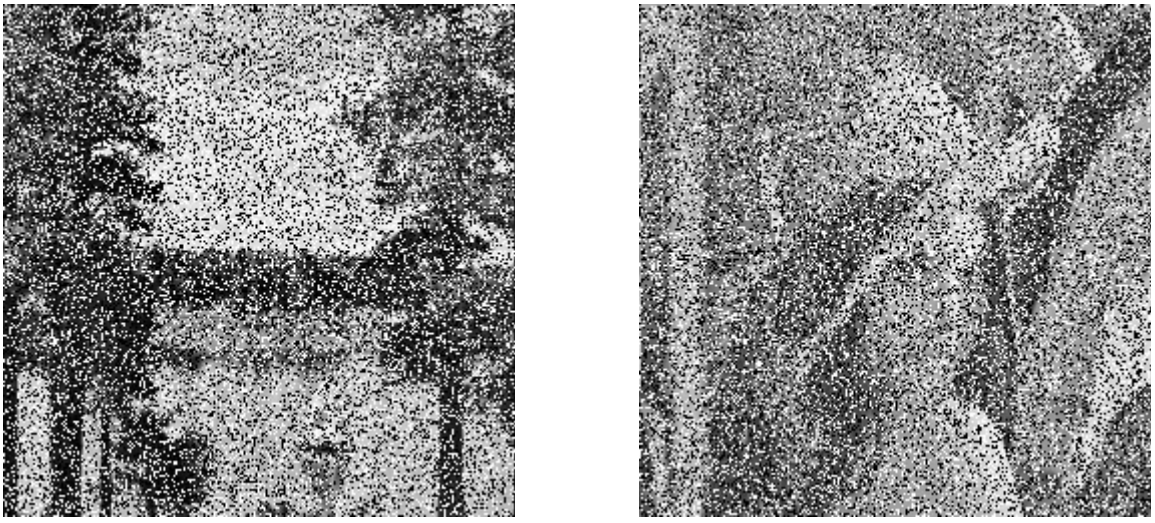
30% salt, size 7 median filtering



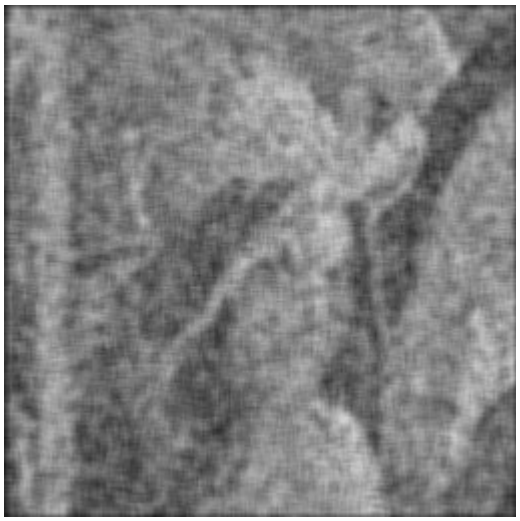
30% salt, size 15 averaging filter



30% salt, size 15 median filter

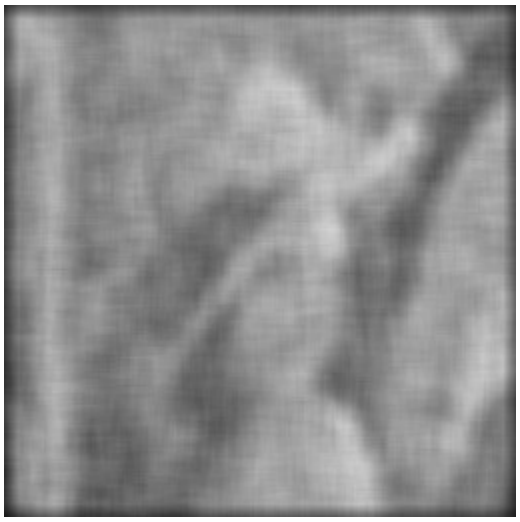


50% salt filter

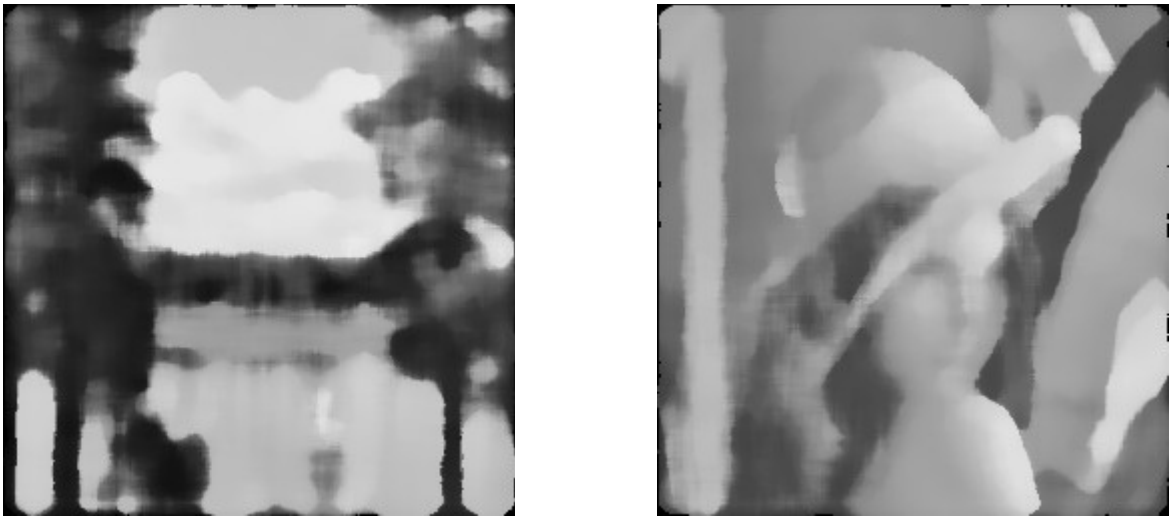



50% salt, size 7 averaging filter


50% salt, size 7 median filter

50% salt, size 15 averaging filter



50% salt, size 15 median filter

So several key observations to point out. Median filtering definitely seems to be the more apt way to remove salt and pepper noise. The reason that this seems to be true is that unlike an averaging filter, median filtering keeps most of the boundary lines of objects, such as the hat in the Lenna image. This also attributes to such details as the sun light gradient on the hat in Lenna also. However, in the cases where the noise was very high (50% in this case) and the mask size was not big enough (7) it can be seen that median filtering ends up with gaps of blank space in the image. However, this downfall is definitely overshadowed by the fact that the images are much easier to understand, and that they preserve boundary lines. Also, it is important to note that averaging, since it takes in all the data, actually uses the noise to calculate its outcome, which is why the average filtered images still have salt and pepper issues, even through the spots are greatly softened.

Problem four:

The last problem, problem four, was a bit of a simple problem to code, in that you simply use the gradient masks to iterate over the image and generate the overall image. Below are the images.

Given

Sobel x gradient

Sobel y gradient

Sobel Filtered Lenna

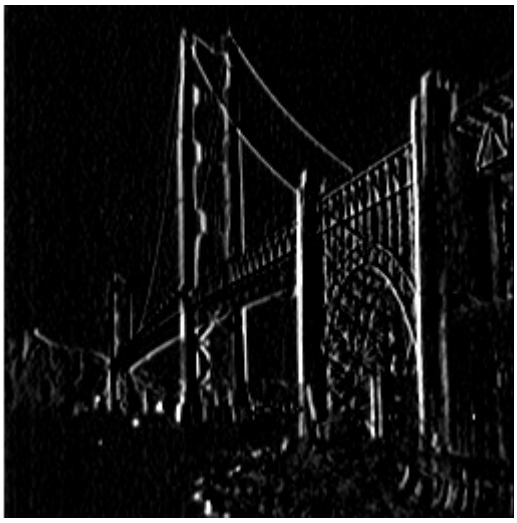
Prewitt x gradient

Prewitt y gradient

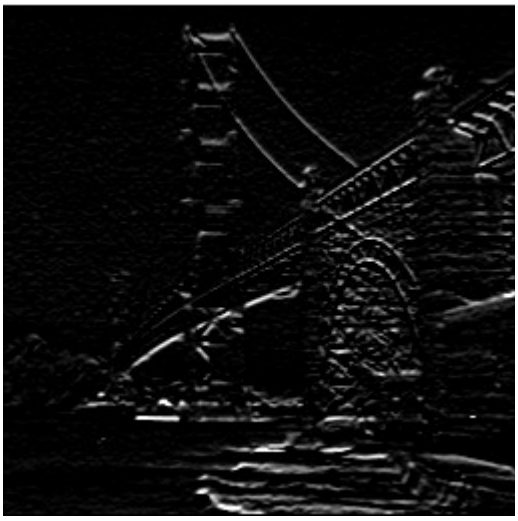
Prewitt filter Lenna

Laplacian Lenna

Given

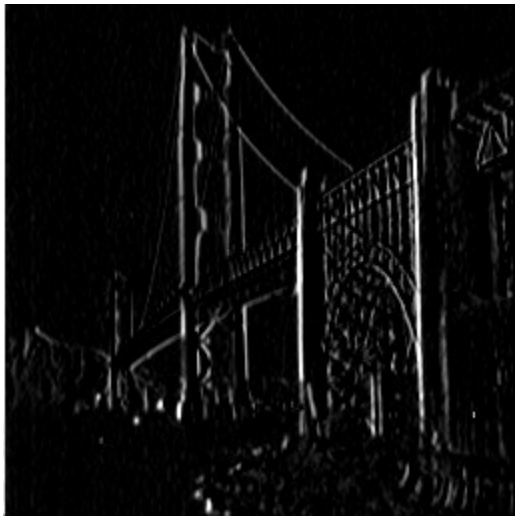


SobelX SF

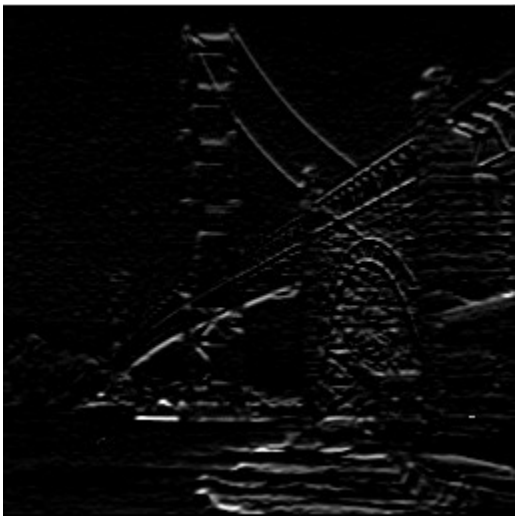


SobelY SF

Sobel Filtered SF

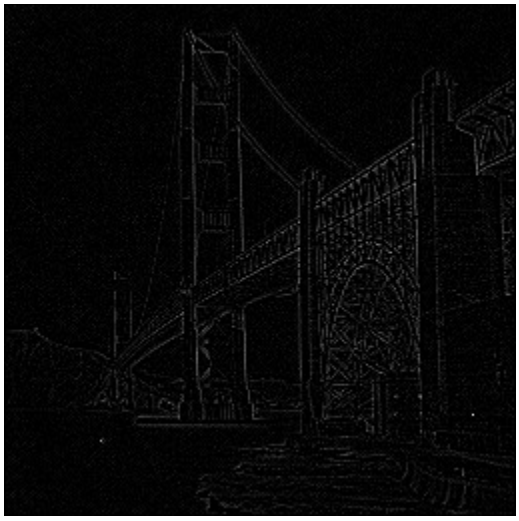


PrewittX SF



PrewittY SF

Prewitt filtered SF



Laplacian Filtered SF

So as can be seen by reviewing the output, the procedure was a definite success. In regards to the Lenna image, the differences between the prewitt and sobel filters are indiscernible by the naked eye. However, the laplacian filter is a great example of optimally sharpened image, in that all of the hard lines are still present but all of the background details have been removed, which would be perfect for an object detection program or several other common applications. This can be also seen on the SF image, however, the laplacian filtered image is a bit dark, simply due to the fact that the given image is a bit dark to begin with. The prewitt and sobel filters look quite a bit better for the SF image than the laplacian filter, but once again I believe the laplacian is a better filter here in that it has much less “noise” and is much more refined looking, even if it is slightly dark.

Division of Labor

It is slightly difficult to say exactly who did what, it can be said that both Aaron and Jeremiah worked equally on the write up, while Aaron took a lead on problems one and two while Jeremiah took the lead on problems three and four. Using GitHub and voice chat while working means that the labor was very evenly split.