

CS 474  
Image Processing  
Programming Assignment 2

Kenneth Peterson  
Zachary Estreito

October 2nd, 2023

Problems 1 and 3:  
Kenneth Peterson

Problems 2, 4 and 5:  
Zachary Estreito

## Theory

---

### 1. Correlation(Kenneth Peterson):

Correlation is a process of comparing a filter or template with an image to identify regions that match said filter or template. Particularly in the use of spatial filtering, where the kernel(or mask) is a small matrix that defines the operations to happen on an image where the filter can be linear or nonlinear, correlation is computed using a convolution operation with a kernel(KxK) that is a mirrored, or 'flipped', version of the filter or template. The correlation highlights areas where the filter or template matches regions of an image that are similar.

The equation for spatial filtering is as follows:

$$I_{\text{filtered}}(x,y) = \sum(a, i = -a) (\sum(b, j = -b) ( I(x + i, y + j) * H(i, j) ) )$$

- $I_{\text{filtered}}(x,y)$ : our filtered pixel value at (x,y)
- $I(x + i, y + j)$ : pixel value at position (x + i, y + j)
- $H(i, j)$ : filter kernel/mask at position (i,j)
- a and b: sizes of the filter kernel/mask

The equation for correlation is quite similar:

$$C(x,y) = \sum(a, i = -a) (\sum(b, j = -b) ( I(x + i, y + j) * T(i, j) ) )$$

- $C(x,y)$ : is correlation value at (x,y) in the output image
- $I(x + i, y + j)$ : the pixel value at position(x + i, y + j)
- $T(i, j)$ : the template at (i, j)
- a and b: size of the template.

Essentially, this is a sliding dot product between a filter(such as a pattern) and an image for tasks like feature extraction, pattern matching, and edge detection.

## 2. Averaging and Gaussian Smoothing:

The averaging smoothing filter involves calculating the mean pixel values of an  $N \times N$  mask, with  $N$  representing the size of the mask, then applying it to the center pixel. This is iterated over every single pixel in the image, producing a blurred result. There are a couple concerns with this approach however. The first concern is that the mask cannot be applied normally to pixels that are too close to the edge because the mask would extend past the outer bounds of the image. To address this issue, I treated edge pixels as if their values extended past the outer boundaries of the image. The second concern of the averaging filter is that it inherently creates a blocky blur due to the fact that every pixel within the mask size is weighed equally. This would not be as big of a problem if the mask used was a round shape, but that still would not solve it. In order to address the blocky blur, Gaussian smoothing can be used. Gaussian smoothing uses a similar approach as average smoothing, with an  $N \times N$  sized mask. The key difference, however, is that the pixels within the mask area are weighted differently. The pixels within the mask area are instead weighted based on a Gaussian (aka normal) distribution. This also means that the Gaussian filter includes the sigma parameter to represent the standard deviation. A higher sigma value will give a greater weight to the pixels around the edge of the mask, while a lower sigma value will give a greater weight to the pixels in the center. Because the normal distribution is applied in both the X and Y directions, there is no blockiness to the blur effect.

### 3. Median Filtering(Kenneth Peterson):

Median filtering is a non-linear form of noise reduction that is used to preserve edges in an image. The main operation of median filtering is to replace the pixel value at a location with the median value of pixel values within a surrounding neighborhood of pixels. The operation is meant to reduce noise while retaining central values of an image. With the use of a small sliding window, that is typically square or rectangular in shape, we attempt to move across an image with noise. As the window shifts, for each position in the window, the median value of the pixel values within the window is computed then assigned to the center pixel of that window.

The following is the technique behind sliding window:

$$I_{\text{filtered}}(x, y) = \text{Median}\{I(x, y) \text{ and its neighboring pixels}\}$$

- $I_{\text{filtered}}(x, y)$  : filtered pixel at position (x, y) in output image
- $I(x, y)$ : pixel value in the center of sliding window
- Median: Calculates median value among pixel values within the window.
  - Window size and pixels vary so it's been simplified to a set{ }.

The noise used for demonstrating this median filtering is known as salt and pepper noise. This is also known as impulse noise but it is essentially irregular noise that corrupts images. There's many causes for this noise to be present in pictures but this time it is simulated to images for the purpose of demonstration.

#### 4. Unsharp Masking and High Boost Filtering

Unsharp masking and high boost filtering are both image sharpening methods. They both work exactly the same, but unsharp masking strictly uses a  $k$  value of 1, while high boost filtering uses  $k$  values greater than 1. The formula for these sharpening methods is as follows:

$$g(x,y)=f(x,y)+kg_{\text{mask}}(x,y)$$

$$\text{Where } k \geq 0 \text{ and } g_{\text{mask}}(x,y)=f(x,y)-f_{\text{LP}}(x,y)$$

The application of this mask works as such: First, a smoothed (low pass) version of the original image called  $f_{\text{LP}}(x,y)$  is created, generally using a Gaussian smoothing filter. This smoothed version of the image is then subtracted from the original image, leaving the mask  $g_{\text{mask}}(x,y)$ . Because this mask represents the difference between the original image and the blurred version, adding the mask to the original image will sharpen it. This is called unsharp masking. For high boost filtering, a  $k$  value greater than 1 will be multiplied to the mask to further exaggerate the sharpening effect. It is possible for the end product image to contain pixel values outside of the allowed range. To fix this, pixel values are clamped to the allowed range.

## 5. Gradient and Laplacian

The Prewitt and Sobel masks used for question 5 are similar in their application. They are both used to find edges in an image. They accomplish this by finding the partial derivatives of an image in both the X and Y directions, then multiplying the results together to find the gradient magnitude. For Prewitt and Sobel, a 3x3 mask is generally used. Once again, edges are dealt with by treating the areas of the mask that exceed the image boundaries as if the true edge pixel values extended outwards. This represents the rate of change of pixel values. There are actually two masks for each - one for the X axis and one for the Y axis. These masks are iterated over each pixel to calculate the X and Y partial derivatives. A high rate of change indicates an edge, because the pixel value has changed rapidly. The visualization of these derivatives will almost always contain values outside of the normal pixel value range, so in order to visualize the results, the pixel values must be normalized to the allowed range (0 to 255). This normalization can be accomplished by offsetting all values so that the minimum pixel value becomes 0, then dividing by the maximum pixel value, then multiplying by the quantization max (255). This shifts the resulting pixel range to occupy the full allowed range of 0 to 255.

After calculating the Prewitt and Sobel X and Y partial derivatives by iterating each one over all the pixels in the image, the gradient magnitude can be found by multiplying the two partial derivatives together. This gradient magnitude will show all of the edges clearly, while the X partial derivative only shows edges along the X axis and the Y partial derivative only shows edges along the Y axis.

The Laplacian mask differs from the Prewitt and Sobel masks because it instead calculates the second derivative, which represents the rate of change of the rate of change of pixel values. For Laplacian, a single mask is used and applied to the image twice. Once again, values can be normalized to their allowed range (0 to 255) using the same approach as above.

## Results and Discussion

---

### 1. Correlation(Kenneth Peterson):

For this demonstration of correlation, we were tasked with first making a function that takes in the size and weight of a given filter. That filter came in the form of a pattern that was moved along an image to check for correlation.

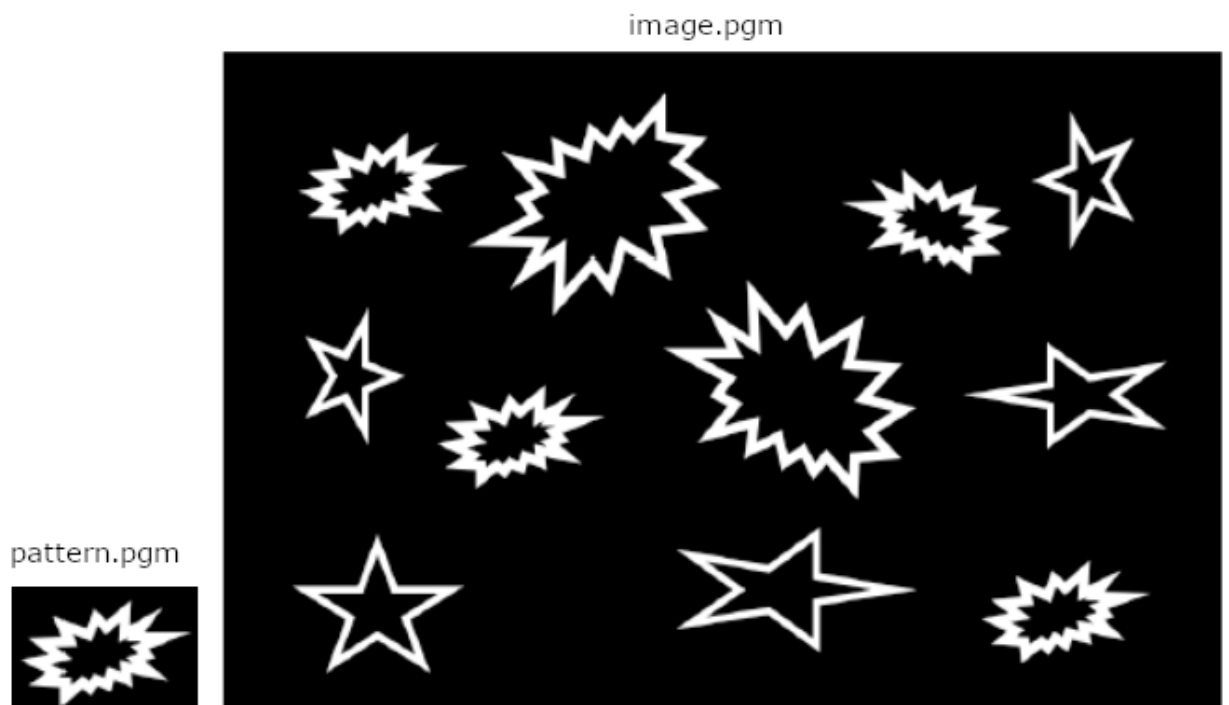


Fig. 1. Models for demonstration, pattern.pgm on the left applied to image.pgm on the right.

With the use of the models seen in Fig. 1., the goal is to capture the weights of pattern.pgm and apply the size and weights to image.pgm in order to get a resulting output image that describes the correlation between the two images. In image.pgm, it is clear that some patterns resemble pattern.pgm that have been mirrored or flipped and/or resized. It would be expected that any correlation to the exact orientation of the pattern would have bright values indicating large correlation, the resized patterns would have less values indicating correlation and the stars even more so. The results on the other hand were quite far from that.

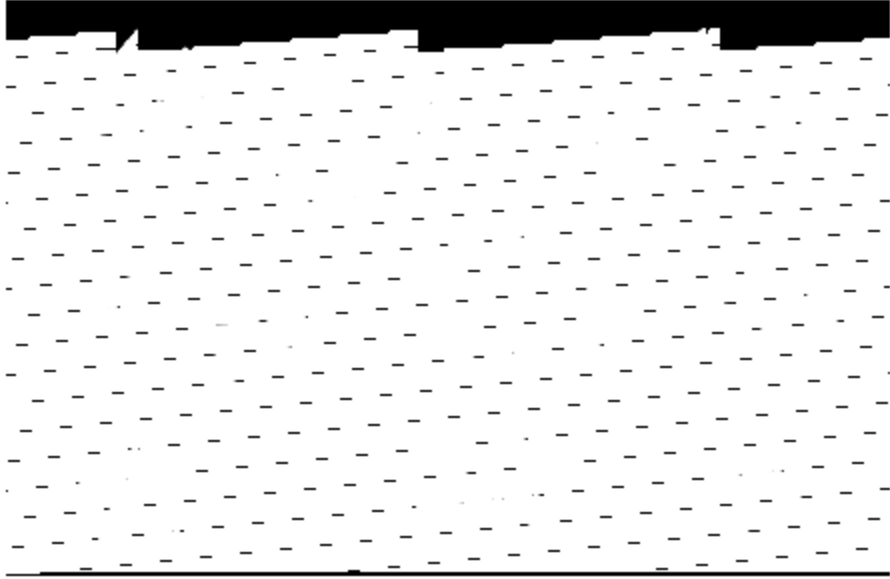


Fig. 2. Output image result of pattern.pgm applied to image.pgm after size and weight extraction

In Fig.2., there is hardly any recognizable image to what was given, the pattern's size and corresponding pixel weights did not mingle well with image.pgm. However, there are some spots that take previous shapes of image.pgm if you were to squint, but it is impossible to determine what is and isn't corresponding to image.pgm. So the next course of action was figuring out why so then I tried tuning the mask to be of a smaller size and the weights would be padded so that they were equal across the mask.



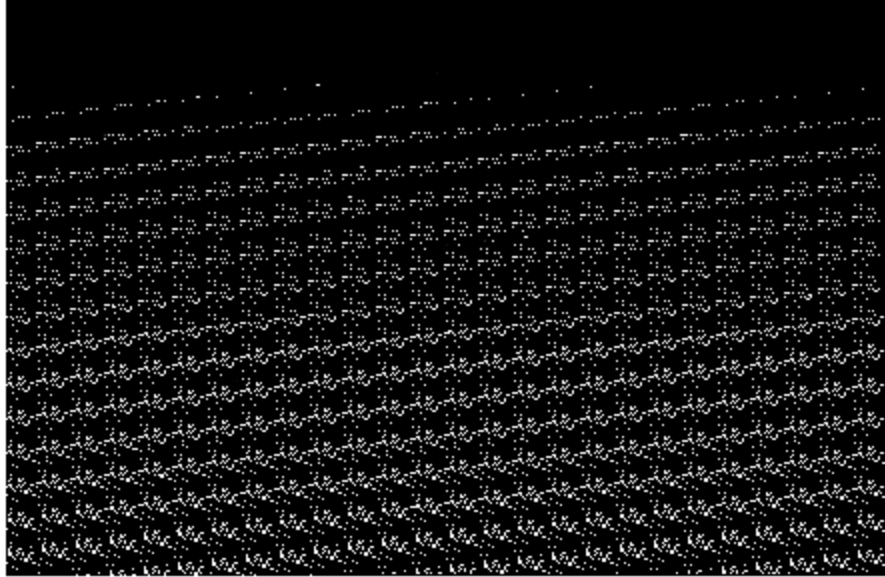


Fig. 3. Output image result of tuning size of pattern.pgm to a smaller factor and padding the weights

As can be seen here in Fig. 3., there was again not a whole lot of image.pgm left in the output image. It seems like the result of the picture couldn't be further from what was needed. I felt like my approach to this was rather simple, extract the weights and the sizes of pattern.pgm and apply the mask over image.pgm so that there would be some highlighted correlation around the output image but unfortunately there was no resemblance to the expected output. It's likely that my  $C(x,y)$  was calculated wrong in my code, resulting in dot products that massively upscaled the resulting pixel. Then when it came time to resize and visualize the output image for the results to have correlation localized in areas where it would be brighter depending on the strength of the correlation to be scaled too far up or too far down, resulting in essentially nonsense.

## 2. Averaging and Gaussian Smoothing

The results of the averaging and Gaussian filters can be seen below. As expected, the averaging filter caused a blocky blurring effect, while the Gaussian filter caused a much smoother blurring effect.

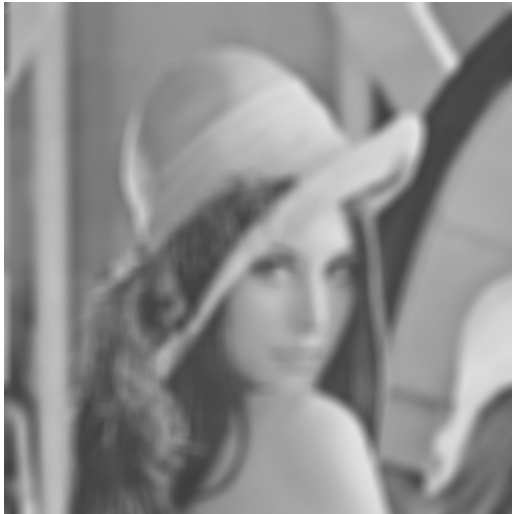
Lenna original



SF original



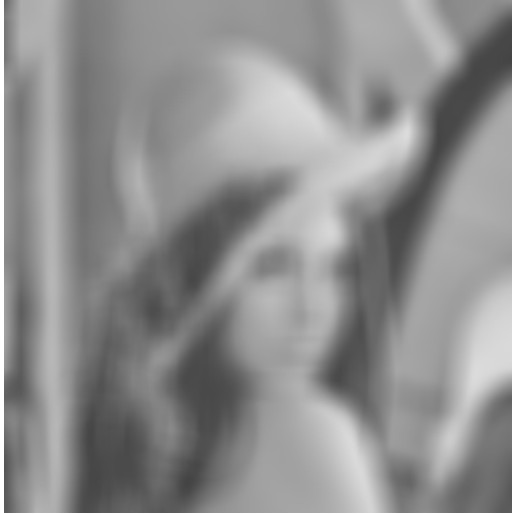
Lenna Average 7x7



SF Average 7x7



Lenna Average 15x15



SF Average 15x15



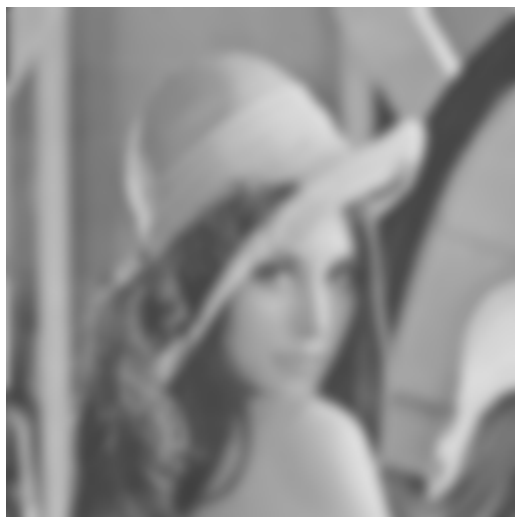
Lenna Gaussian 7x7



SF Gaussian 7x7



Lenna Gaussian 15x15



SF Gaussian 15x15



### 3. Median Filtering(Kenneth Peterson):

The first task given for median filtering was to have a function that passed in the size of our filter. In order to demonstrate our median filtering function, we then had to apply some artificial noise to two images, lenna.pgm and boat.pgm, in the form of salt and pepper noise. The goal for the noise was simple: make a case where we change around 30% of the pixels in an image, save the result, then have a separate case that changes 50% of the pixels in an image and save that result.



Fig. 1. Applying salt and pepper noise to boat.pgm and lenna.pgm, one case with 30% pixels changed and another with 50% pixels changed.

In Fig. 1., we apply our salt and pepper noise to each image tasked, first where we apply 30% changed pixels and second where we apply 50% changed pixels. The results are rather clear, the larger the percent of pixels changed, the more corrupted the image becomes. Now that we have noise in our images, we can apply median filtering and see how it fairs with other forms of filtering.

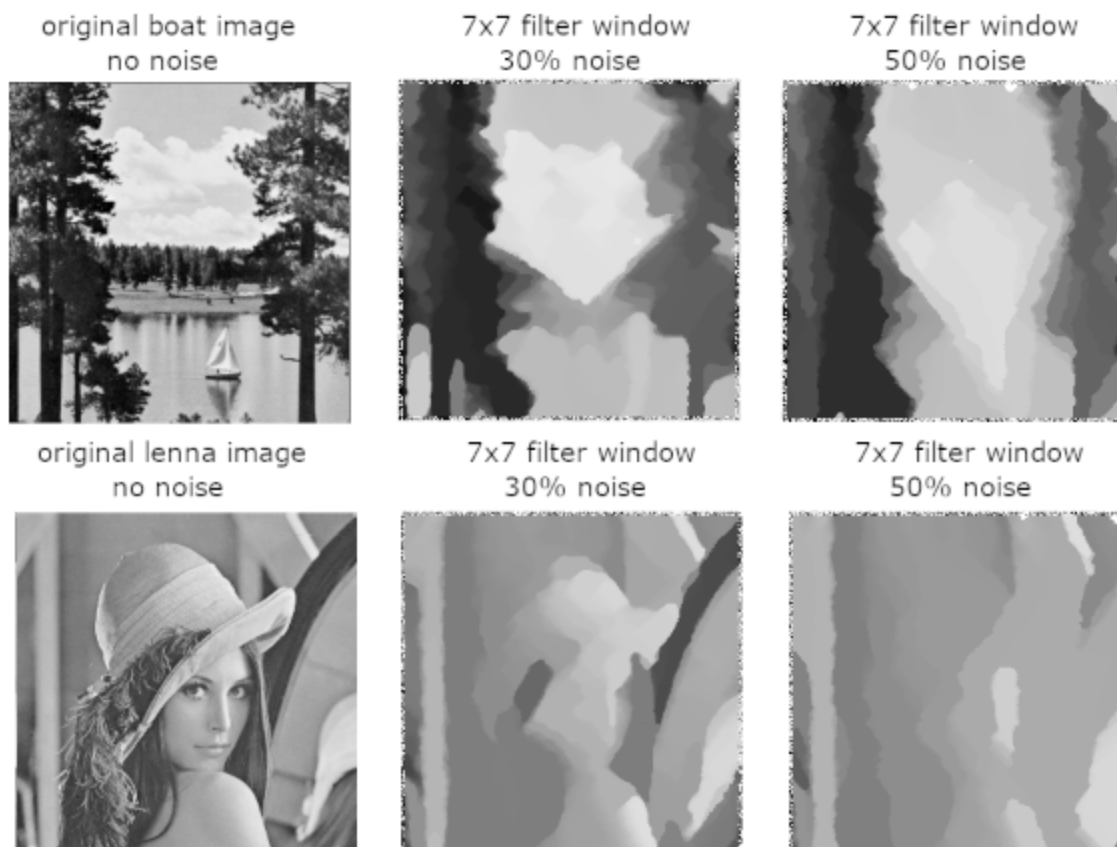


Fig. 2. Applying a 7x7 window to noisy images with original image for comparison.

In Fig. 2., we can see that with a filter window size of 7x7, it's not much of a challenge to recognize the original image from 30% noise corruption. Where it does start to become a challenge is when 50% of the image is corrupted, overarching values and some conceptual parts about the images are still visible but most of the finer details have been lost in comparison.

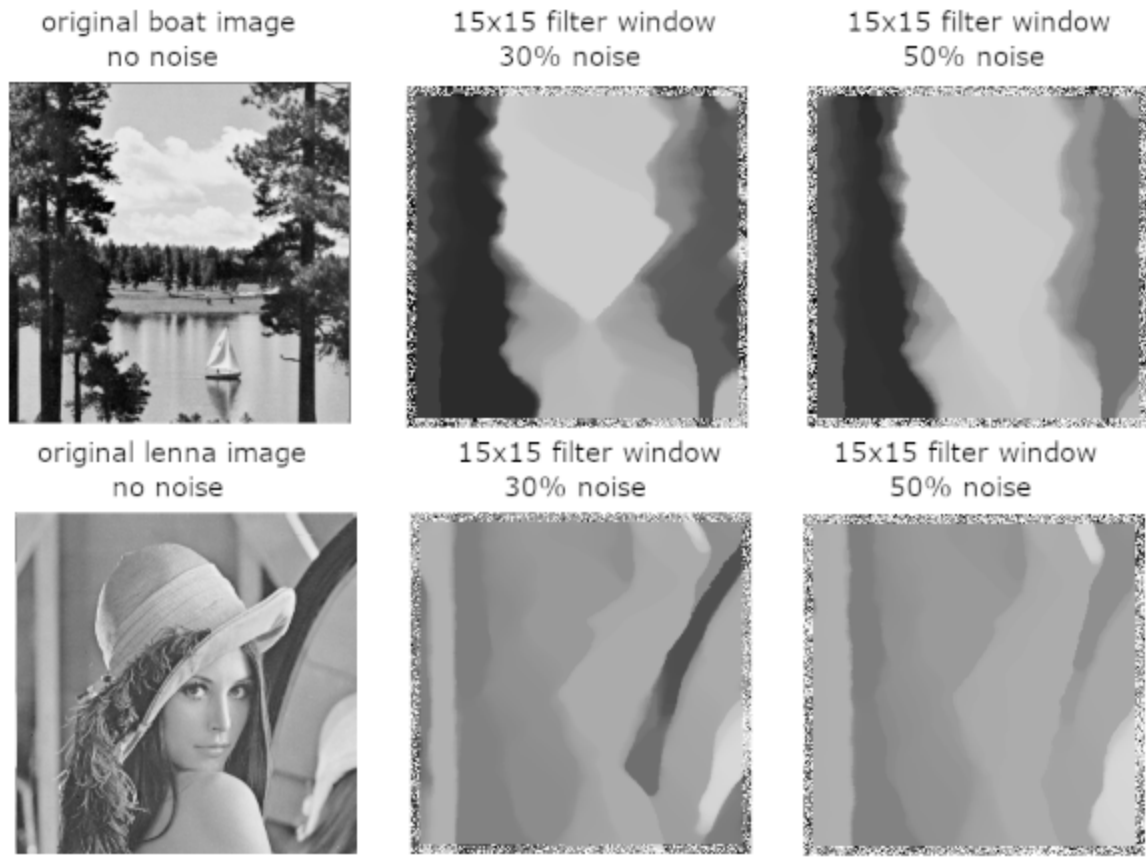


Fig. 3. Applying a 15x15 window to noisy images with original image for comparison.

As can be seen in Fig. 3., if we apply a 15x15 window to the tasked images, it is clear that more detail is lost than the previous window. Some shapes and colors and overarching concepts are still retained but a good proportion of the image is lost when it reaches around 50% noise.

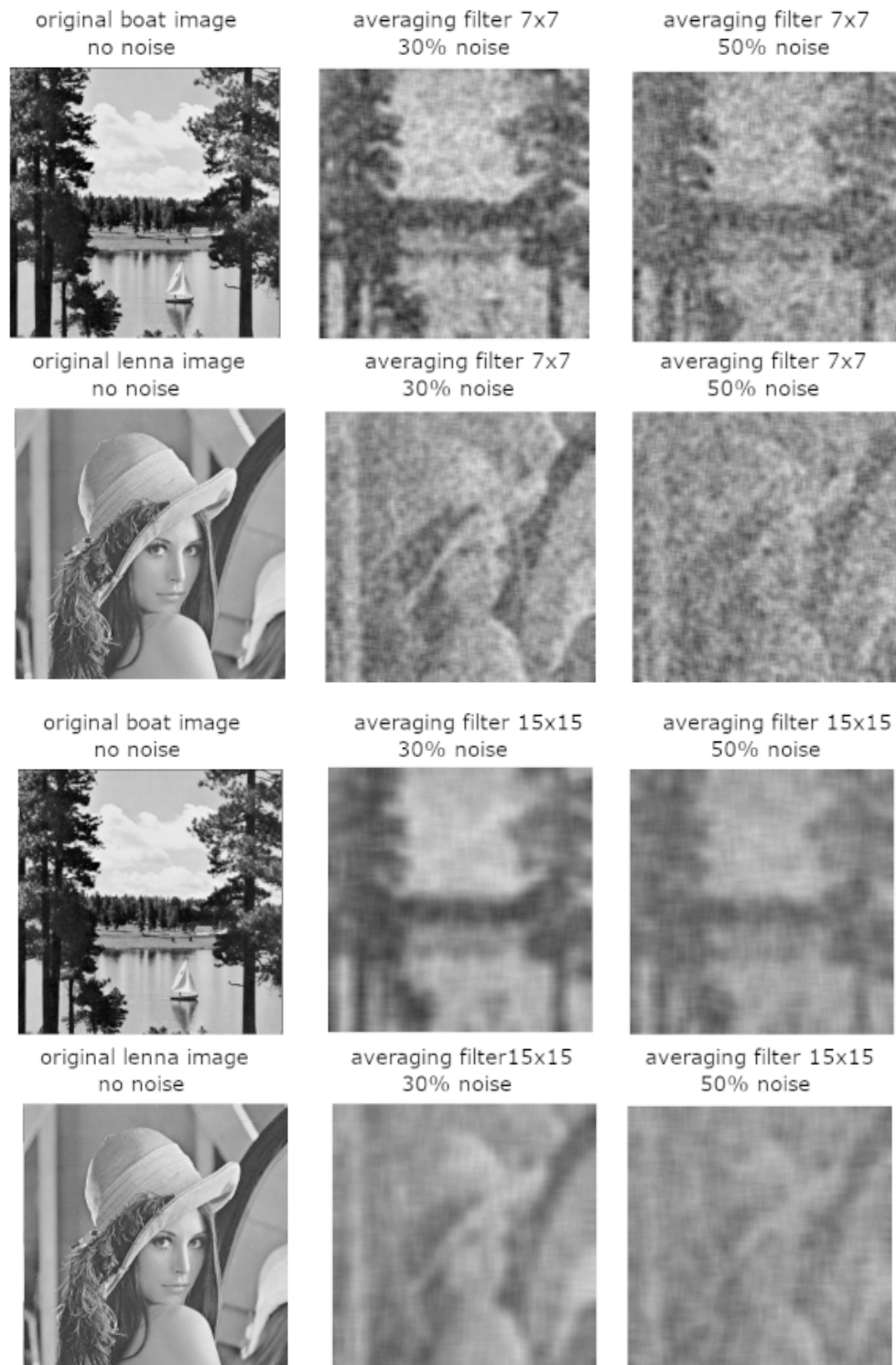


Fig. 4. Applying average filtering to the same noisy images with the same filter sizes for comparison purposes.



For the sake of comparing filters, in Fig. 4., we were also tasked with using the averaging filter. The goal for this averaging filter is to reduce noise on similar spectrums as the median filter, where the sizes of the window and noise percentages remain the same but the filtering is different. Upon examination, averaging seems to keep a lot more of the finer details in comparison to median filtering. Where median filtering seems to be holding a lot of the main shapes, average filtering tends to gravitate towards keeping details. A coupling of the two would probably give a messy but strikingly similar resemblance to the original image if they were applied together. Something these two filters both share is that the window sizes affect the noise similarly, the larger the window, the less comprehensible it tends to be, but some pieces still remain such as overall shapes and maybe a sprinkle of finer details here and there when it comes to the averaging filter. When it comes to determining which filter resembles more closely to the original image, it could be more of personal take, but average filtering seems to more closely resemble the actual image than median filtering since the median filtered images look more like an abstract painting of the original image.

#### 4. Unsharp Masking and High Boost Filtering

Unsharp masking and high boost filtering brought out details that were difficult to see in the original images. This is useful for images whose original images were blurry or washed out. In the case of these two images, I personally found that Lenna looks best with a  $k$  value of 1, while F\_16 looks best with a  $k$  value of 2. This is likely because the original F\_16 image is blurrier than the original Lenna image.

Lenna original



F\_16 original



Lenna boost  $k=1$



F\_16 boost  $k=1$



Lenna boost  $k=2$



F\_16 boost  $k=2$



Lenna boost  $k=3$




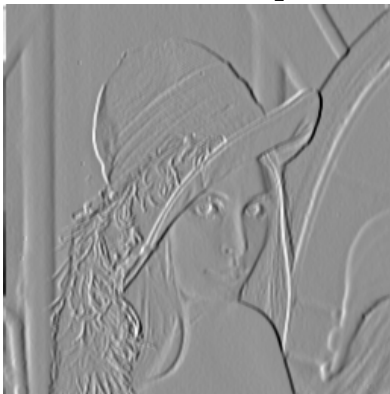
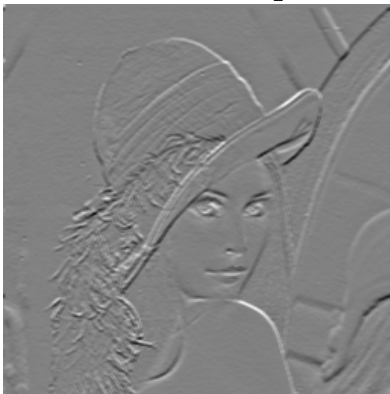



F\_16 boost  $k=2$



## 5. Gradient and Laplacian

Below are the results for my implementation of the Prewitt, Sobel, and Laplacian masks. All of these masks are useful for finding edges in an image. When it comes to sharpening images however, the results are mixed. Using edge detection masks to sharpen images results in the areas around edges increasing in detail, while the areas between edges suffer a loss of details. When it comes to actual edge detection, I believe the Laplacian filter to be the best due to the second derivative making it easy to find exact edge locations. While Prewitt and Sobel masks may make it easier to visualize the edges, the Laplacian filter makes it easier to detect those edges in code.

<p>Lenna original</p> 	<p>Lenna Prewitt sharp</p> 	<p>Lenna Sobel sharp</p> 
<p>Lenna Prewitt X partial</p> 	<p>Lenna Prewitt Y partial</p> 	<p>Lenna Prewitt gradient</p> 

Lenna Sobel X partial



Lenna Sobel Y partial



Lenna Sobel gradient



SF original



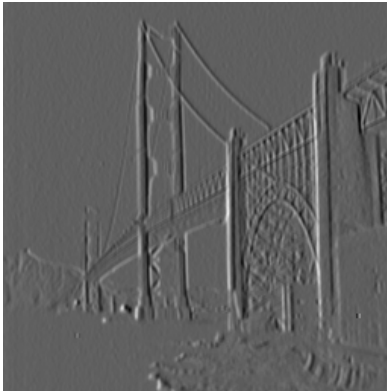
SF Prewitt sharp



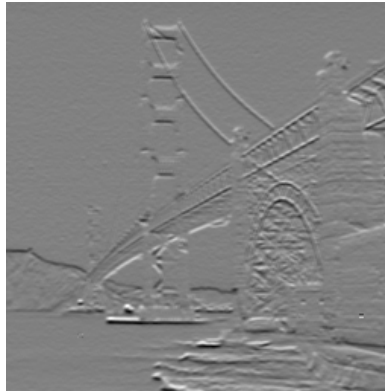
SF Sobel sharp



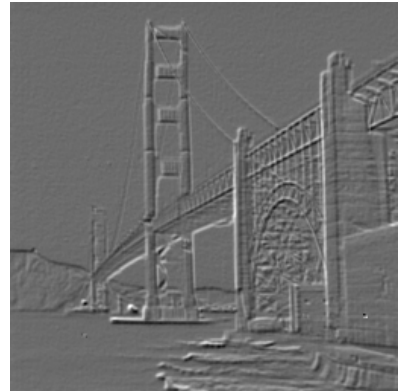
SF Prewitt X partial



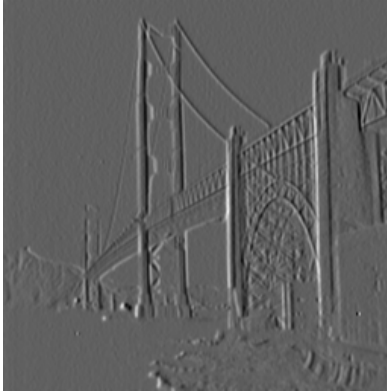
SF Prewitt Y partial



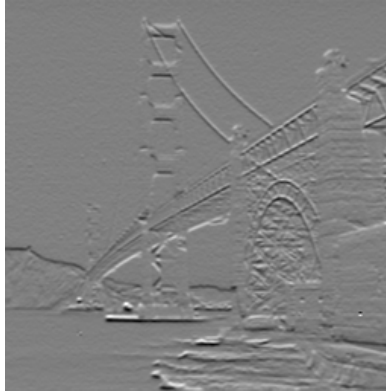
SF Prewitt gradient



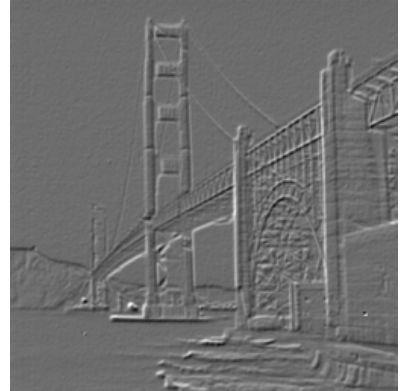
SF Sobel X partial



SF Sobel Y partial



SF Sobel gradient



Lenna original



Lenna Laplace mask



Lenna Laplace sharp



SF original



SF Laplace mask



SF Laplace sharp

