VIP-IPA: Blur Correction

Final Report

Fall 2024

Jacob Morales, Angela Qian, Brian Ramos, Alessandra Rice, Ramsey Daniel, Jeslyn Yang

Advisors: Professor Carla Zoltowski, Professor Edward J. Delp

Department of Electrical and Computer Engineering

Purdue University

Table of Contents

1. ABSTRACT

Digital images make up a large portion of data in the modern world, however a problem with digital images is that information is lost when images are blurry or otherwise degraded. This study aims to explore methods that we can use to both detect and unblur degraded images. We narrowed the scope of our project to images with out-of-focus blur.

In order to achieve our goals, we used the "Blur dataset" from Kaggle. The dataset contains 350 image pairs of sharp and out-of-focus blurred images, which we used to test our unblurring algorithms. Through testing, we settled on using the Richardson-Lucy algorithm and the Wiener algorithm which we used to unblur images when information such as the point spread function or PSF is unknown.

The unblurring methods we used achieved positive results based on our PSNR and SSIM metrics, which we used to compare the similarity of an unblurred image to its true sharp image. Our focus is to be able to use our program to make digital images clearer both in casual uses like photography to potentially life saving applications in medical imaging and surveillance.

2. DATASET

To test our algorithms against blurry images where we did not know the PSF, we started utilizing a dataset from Kaggle which contained 1050 blurred and sharp images [1]. Eventually, when we began using the PSNR and SSIM metrics for success, we realized the blurred and original images were shifted slightly in different directions so they could not be used for evaluating our algorithms beyond visually, so we switched to adding artificial blur in order to have accurate metrics.

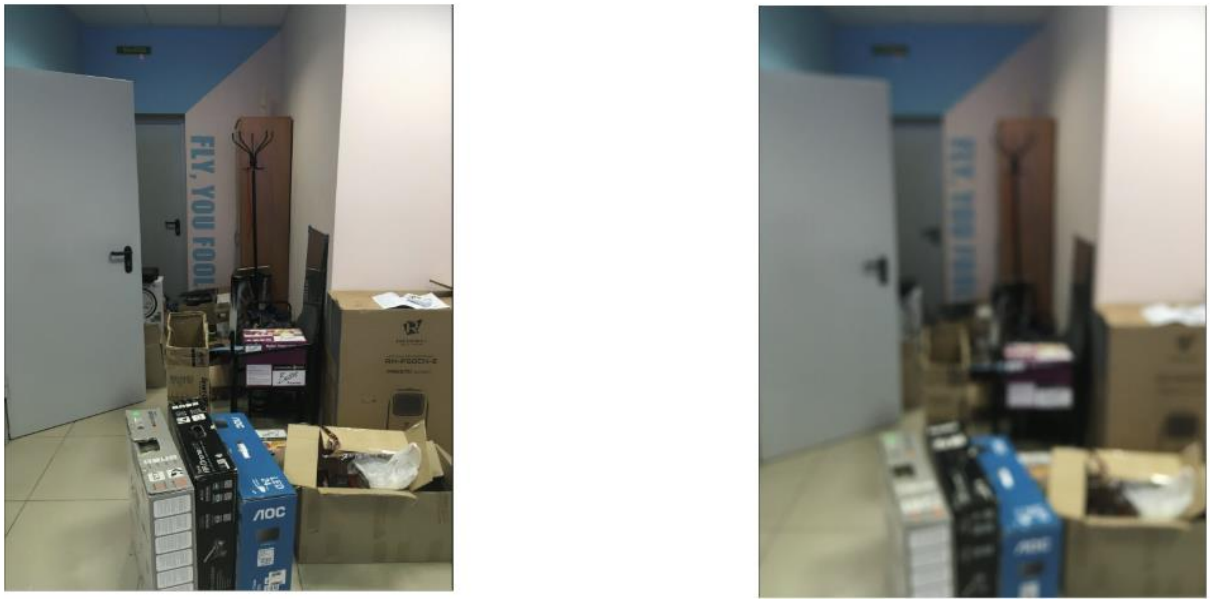


Figure 1. Sample Dataset Picture

3. Background

A. *Image Processing*

We can think of a grayscale digital image as a two dimensional matrix of numbers, where each number in the matrix corresponds to the intensity, or darkness of one pixel.

$$G(u,v) = K(u,v) \otimes f(u,v) + n(u,v)$$

$\otimes$ denotes a convolution operation
$G(u,v) =$ degraded image
$K(u,v) =$ true image
$f(u,v) =$ point spread function
$n =$ noise function

Equation 1. Degradation Model

We can then consider a degraded image as the result of the matrix of a true image with the noise and point spread (blur) functions applied to it. Therefore, when we receive a degraded image, we want to be able to undo the effects of the point spread and noise functions in order to retrieve the true, undegraded image. However, when images are blurred, information is usually lost, so when unblurring images the point spread function and noise function are usually unknown, and this makes undoing these functions difficult.

B. *Point Spread Function*

The PSF (Point Spread Function) is an important part of deconvolution where it plays the integral part of modeling the convolution of an image. The PSF represents the effects of outside factors on a single pixel in an image. Most of these effects depend on the light sensor of the camera and other extraneous circumstances, which in some scientific fields can be difficult to account for all factors. For example in astronomy, the Hubble Space Telescope uses a series of mirrors to focus light in to capture an image. However, when the telescope was first launched it was discovered to have mechanical flaws resulting in blurred images. The spacing of the main

lens was l.3 millimeters off which resulted in a spherical aberration in the primary mirror. Because of the spherical aberration, the PSF of the Hubble Space Telescope became much larger thus demonstrating a stronger blur in its images. [9]

In our paper, we try to estimate the PSF of our blurred images to deconvolve the images. In some of our filters, the algorithm iteratively estimates the PSF and uses it to deconvolve the images while in others we use a premade PSF which was a guess. In deconvolution filters, the accuracy of PSF can directly affect the performance of the filter. However, in real life it is impossible to get the PSF so our filters deal with blind deconvolution where the PSF is unknown.

4. Measurement Scales

A. SSIM

To better evaluate and compare our algorithms and the restored images they produced, we researched methods that would consistently quantify the quality of our results. One of those methods we implemented was the Structural Similarity Index Method, or SSIM. It is a common method for image quality assessment that compares the structural similarity between two images – in this case a restored image and its ground truth. Equation 2 is based on comparing the image in terms of contrast, luminance, and structure. It takes in two images, x and y, and returns a value ranging from -1 to 1, where a 1 indicates a perfect similarity [2].

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(2\sigma_x^2 + \sigma_y^2 + C_2)}$$

Equation 2. Structural Similarity Index Equation where

- μ_x = average of x
- μ_y = average of y
- σ_x = variance of x
- σ_y = variance of y
- σ_{xy} = covariance of x and y
- $C_1 = (k_1L)$ to stabilize division with weak denominator
- $C_2 = (k_2L)$ to stabilize division with weak denominator
- L = dynamic range of the pixel-values (255 in this case)
- $k_1 = 0.01$ and $k_2 = 0.03$ by default

B. PSNR

We also utilized the Peak Signal to Noise Ratio or PSNR to evaluate the success of our algorithms. It is another common metric for evaluating the success of deblurring algorithms which compares the modified image to the ground truth (sharp) image. Using the following equation, we calculate the mean squared error (MSE) by taking the squared difference between the original and sharp image and dividing it by the amount of pixels in the image. Then, we compute the PSNR by dividing the maximum value in the image, taking the log of it, and multiplying it by 20 [3].

$$\mathrm{MSE} = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(x,y) - g(x,y)\|^2$$

$$\mathrm{PSNR} = 20\log_{10}\left(\frac{\max_f}{\sqrt{\mathrm{MSE}}}\right)$$

Equation 3. PSNR Equation

- F(x,y) is the original image.
- g(x,y) is the new image.
- Max_f is the maximum value of the image, in our case it is 255 since we are working with grayscale images
- m is the number of rows in the image
- n is the number of columns in the image

5. Blurry & Noisy Images

A. Introduction

Before we start to de-blur images, we need to understand how to add blur into images. We decided to explore various types of blur and noise to understand how these problems occur in images. For image noise, we focused on shot noise, gaussian noise, and salt and pepper noise, as those are the most common types of noise to run into in photography. By understanding how these noise types occurred, we were better prepared to combat them with our denoising techniques.

B. Gaussian Noise

Gaussian noise is additive noise caused by random thermal fluctuations or electrical interference within cameras that follows a normal distribution in accordance with the following equation

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma}}$$

Equation 4. Gaussian Noise Formula

where

- σ is the standard deviation of the noise and positively correlated with the amount of noise in the image.
- x is the pixel in the image
- μ is the mean.

C. Shot Noise

Shot noise is noise caused by fluctuations in light photons hitting the lens of the camera. It follows a poisson distribution as given below

$$P(X = x) = \frac{\mu^x e^{-\mu}}{x!}$$

Equation 5. Shot Noise Formula

where P(X = x) represents the probability that the pixel value is equal to x, and μ is the average pixel value

D. Salt and Pepper Noise

Salt and Pepper noise is noise caused by transmission errors, corrupted pixels, or faulty memory storage, resulting in black and white pixels to appear randomly throughout the image.

E. Gaussian Blur

Throughout our project gaussian blur was the most common type of blur we added to images. Gaussian blurring mimics the type of blurring attained when a photo is taken of an image that is out of focus by spreading the PSF in a circular fashion by applying a gaussian filter which weights the value of the pixel by its distance from the center to smooth and blur the image.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Equation 6. Gaussian Blur Formula

G(x,y) is the position in the image and σ is the standard deviation of the blur, creating a uniform blur across the image.

6. Denoising Methodology

A. Median Filter

We used the median filter to get rid of speckling and noise in images, while still keeping major details in the image. This is done by taking an n x n square section around a pixel, and replacing the pixel value with the median value of the pixels that had been selected. This is extremely effective for getting rid of salt and pepper noise, as well as mixed noise

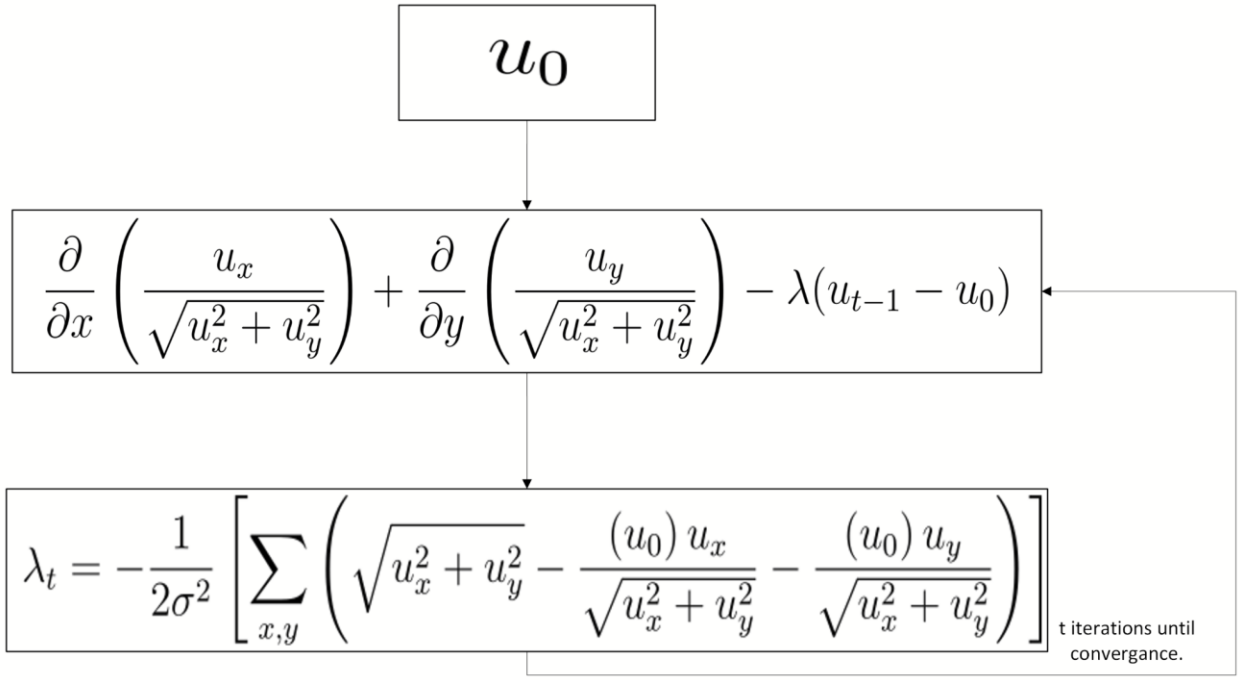
B. Box Blur

Box Blurring is good for smoothing over noisy images, as well as removing speckles in images, but has the downside of removing details from images. We implemented this by selecting an n x n box around a pixel, and replacing the pixel value with the average value of the pixels that were selected. Box blur is effective for fixing shot noise, as well as mixed noise.

C. Total Variation Noise Removal

Total Variation is a method of noise removal wherein the sharp edges of the image are better preserved. This means the image will require less deblurring in the later stages of image correction. We also decided to use it because it can denoise images after they have been deblurred without significantly destroying the quality. It works by decreasing the total amount of variation in the image, thereby removing noise fluctuations from the image but keeping the

overall quality intact [4]. The formula for total variation is given by



Equation 7. Total Variation Noise Removal Formula

- U_0 is the initial image
- U_t is the current iteration of the image being denoised
- $\frac{\partial}{\partial x}$ is the partial derivative in the x direction
- $\frac{\partial}{\partial y}$ is the partial derivative in the y direction
- λ denotes the regularization function given by the second equation.
- σ equates to the standard deviation of the image, and in this case also affects the step size.

D. Results

Overall, all of our deblurring methods have seen distinct improvement in at least one metric of image quality as seen in the photos and graph at the end of this section.

Salt & Pepper Noise Results:

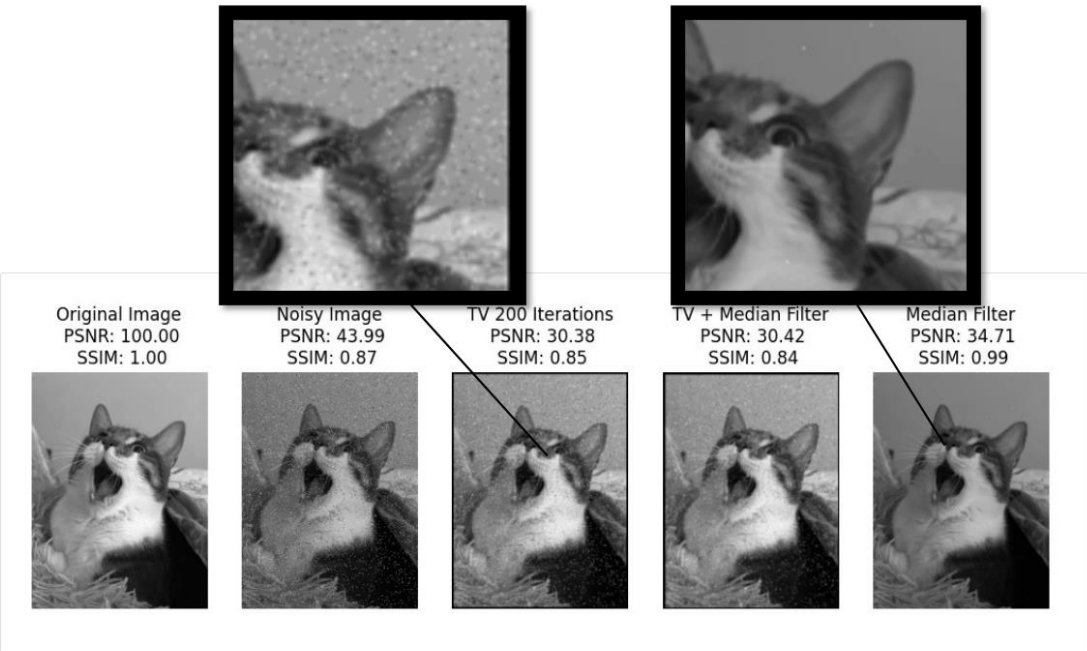


Figure 2. Salt and Pepper Noise Results

Gaussian Noise Results:

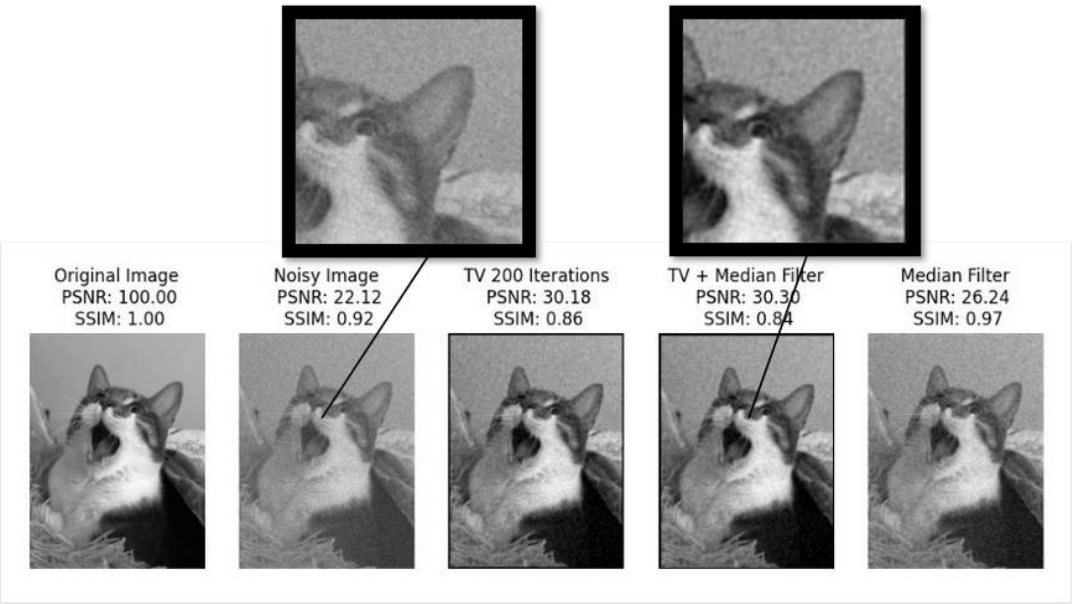


Figure 3. Gaussian Noise Results

Shot Noise Results:

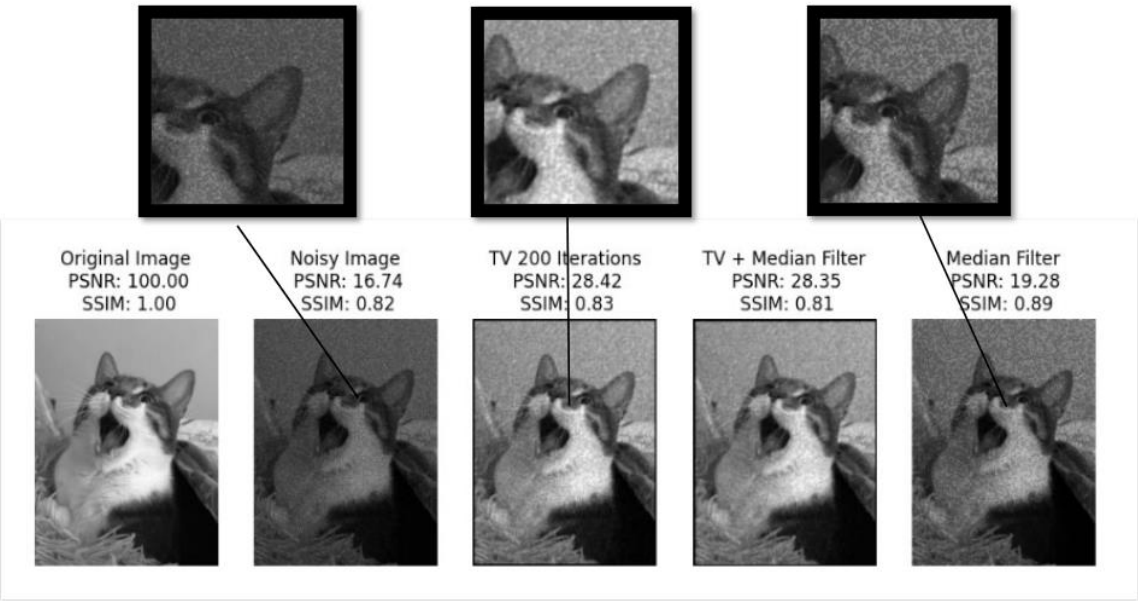


Figure 4. Shot Noise Results

Mixed Noise Results:

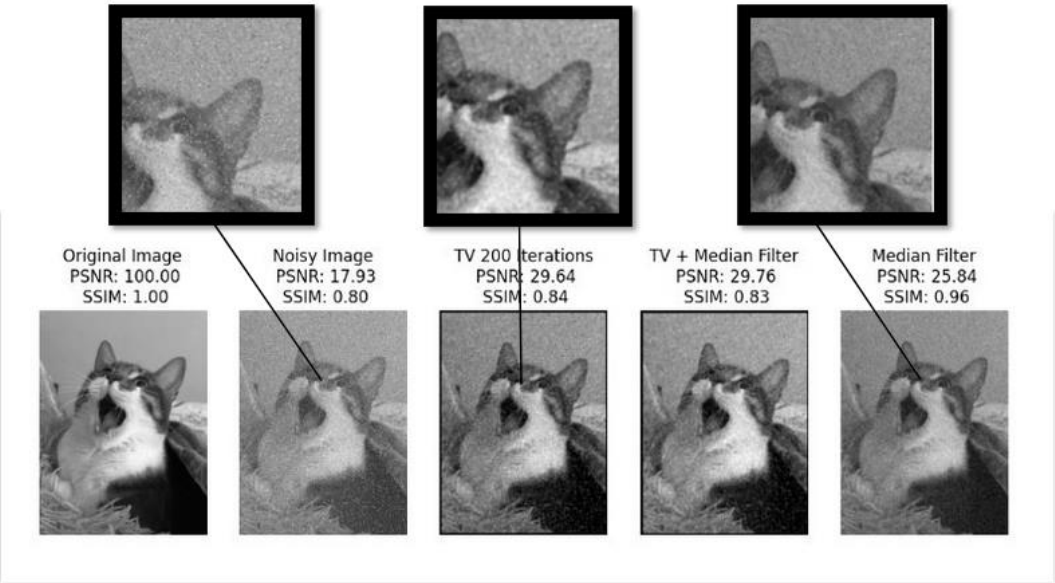


Figure 5. Mixed Noise Results

Box Blurring Results:

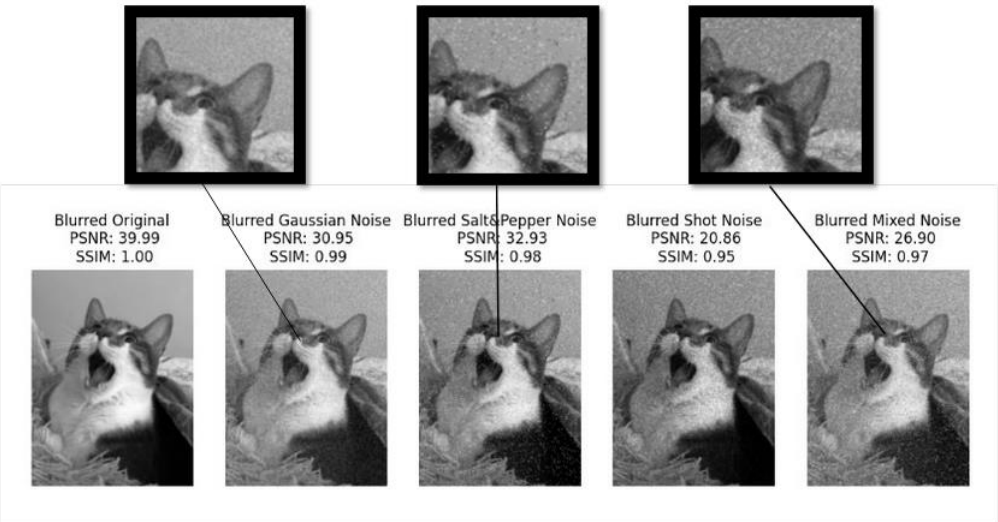


Figure 6. Box Blurring Results

Using the equation below we can calculate the improvement we saw over the noisy image according to the PSNR and SSIM for all the types of noise we were attempting to remove.

$$\text{Improvement} = 100\frac{\text{Image Metrics - Noisy Image Metrics}}{\text{Noisy Image Metrics}}$$

Equation 8. Improvement Metrics

	Noisy Image	**Total Variation**	**Median Filter**	**Total Variation + Median Filter**	**Box Blurring**
Gaussian Noise	PSNR: 22.12 SSIM: 0.92	PSNR: 30.18 (+36%) SSIM: 0.86 (-7%)	PSNR: 26.24 (+19%) SSIM: 0.97 (+5%)	PSNR: 30.30 (+37%) SSIM: 0.84 (-9%)	PSNR: 30.93 (+40%) SSIM: 0.99 (+8%)
Shot Noise	PSNR: 17.93 SSIM: 0.80	PSNR: 29.64 (+65%) SSIM: 0.84 (+5%)	PSNR: 25.84 (+44%) SSIM: 0.96 (+20%)	PSNR: 29.76 (+66%) SSIM: 0.83 (+4%)	PSNR: 20.84 (+16%) SSIM: 0.95 (+19)
Salt & Pepper Noise	PSNR: 43.99 SSIM: 0.87	PSNR: 30.38 (-31%) SSIM: 0.85 (-2%)	PSNR: 34.71 (-21%) SSIM: 0.99 (+14%)	PSNR: 30.42 (-31%) SSIM: 0.84 (-3%)	PSNR: 32.90 (-25%) SSIM: 0.98 (+13%)
Mixed Noise (Gaussian + Salt & Pepper)	PSNR: 16.74 SSIM: 0.82	PSNR: 28.42 (+70%) SSIM: 0.83 (+1%)	PSNR: 19.28 (+15%) SSIM: 0.89 (+9%)	PSNR: 28.35 (+70%) SSIM: 0.81 (-1%)	PSNR: 26.90 (+61%) SSIM: 0.97 (+18%)

According to our results, we found success in almost every denoising algorithm we tried. Salt and pepper noise was the hardest to combat due to the drastic shift in intensity of the pixels, however we did find success according to the SSIM with the median filter and box blurring. Both of these options did result in the image becoming darker and therefore decreasing the PSNR, but the image is undoubtedly clearer. We found that the best approach to denoising when there are multiple kinds of noise or the type of noise in unknown is to utilize both the total variation and median filters, or simply box blur the image. We also found success in decreasing every type of noise with the box blur, however it does come with the tradeoff of significantly reducing image clarity which means we must process it further with our denoising algorithms. Overall, our results are very promising.

7. Deblurring Methodology

A. Unsharp masking with Laplacian Filter

I. Introduction

Unsharp masking is a popular technique for enhancing the edges and sharpness of an image. In this application, we use it to sharpen and thus deblur slightly out-of-focus images. It can be implemented using many different filters, including a Laplacian filter, to enhance the edge details and suppress the less noticeable parts of an image.

II. Laplacian Filter

The Laplacian filter is a second-order derivative filter that identifies edges by finding zero-crossings in the rate at which the first derivatives change in an image. It is mainly used for edge detection and is defined by the following kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Equation 9. Laplacian Filter

III. Unsharp Masking with Laplacian filter

As seen in Equation 9, unsharp masking sharpens an image by applying a Laplacian filter to a blurry image, then subtracting that from the blurry image to a set strength, resulting in a restored image [5].

$$s(x,y) = i(x,y) - f(l(x,y))$$

Equation 10. Unsharp masking formula with Laplacian filter where

- s = restored image
- i = blurry image
- f = strength of filter

- l = image after Laplacian filter

IV. Results

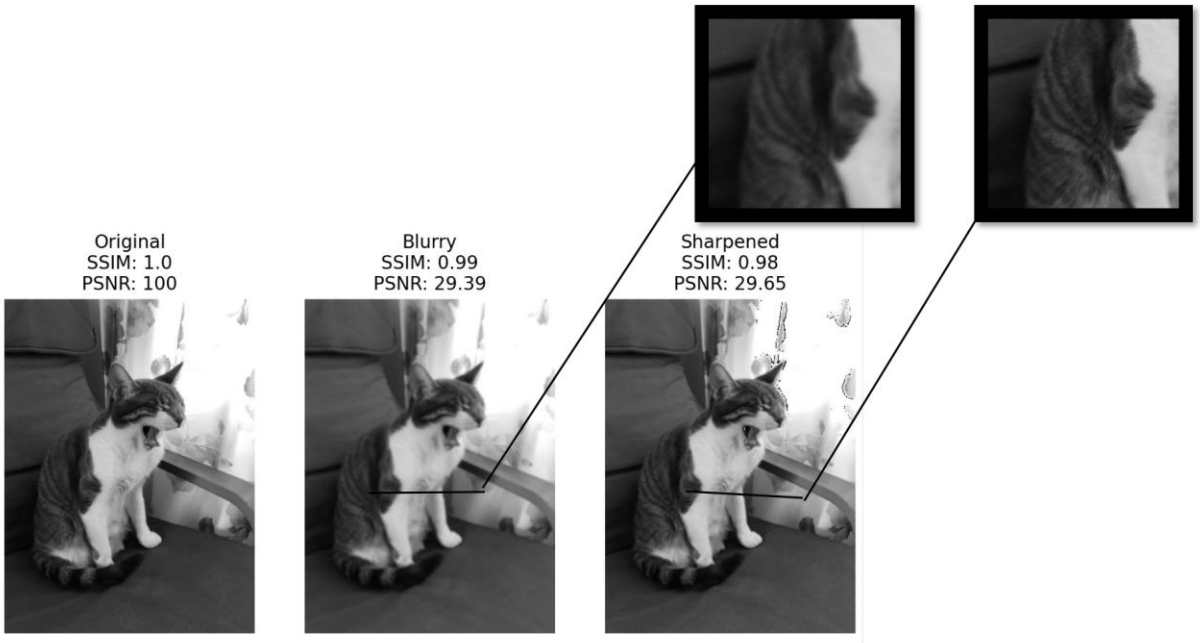


Figure 7. Unsharp Masking with Laplacian Filter Results

As seen in figure 7, the algorithm results in a sharpened image and shows an increase in PSNR values, from 29.39 to 29.65. However, it has its limitations and drawbacks. It is only effective for slightly blurry images and can create unwanted artifacts in the restored image.

B. Unsharp masking with Gaussian Blur

I. Introduction

Similar in concept to the method used above, unsharp masking sharpens and deblurs images that are slightly out of focus. However, in this case, instead of using the Laplacian filter, we will instead be using the Gaussian filter to increase contrast around the edges of the photos [6].

II. Gaussian Blur

The Gaussian Blur filter is a filter that blurs images by also incorporating values from the surrounding pixels. It mainly smooths over images and according to the following kernel:

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

Equation 11. Gaussian Blur

III. Unsharp Masking with Gaussian Blur

As seen in Equation 11, unsharp masking sharpens an image by applying a Laplacian filter to a blurry image, then subtracting that from the blurry image to a set strength, resulting in a restored image.

$$s(x,y) = i(x,y) + f(i(x,y) - g(x,y))$$

Equation 12. Unsharp masking formula with Gaussian blur where

- s = restored image
- i = blurry image
- f = strength of filter
- g = image after Gaussian blur

IV. Results

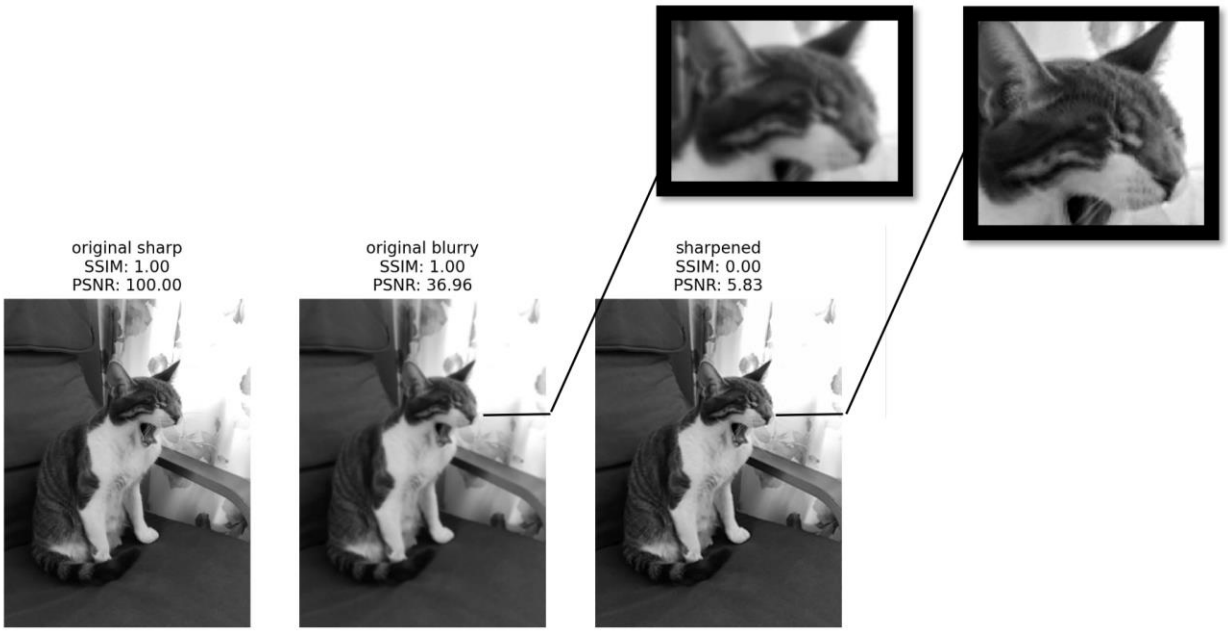


Figure 8. Results from unsharp masking with Gaussian filter

As seen in figure 8, the algorithm results in a sharpened image, however there is a decrease in both the SSIM and PSNR values. While the image does seem much sharper to the human eye, the dark values have been darkened, and the light values have been lightened, which means that while the image is much sharper, it is also less similar to the original image in terms of pixel values.

C. Constrained Least Squares

I. Equation

The Constrained Least Squares filter has a similar structure to the Wiener filter later in the paper. Because of noise becoming a problem when using deconvolution, the CLS filter uses the laplacian operator to remove noise in an image. Similar to how we used the laplacian operator in the unsharp masking, the CLS adds the fourier transform of the laplacian operator in place of the PSNR used in the Wiener Filter [8, p. 363].

$$F(u,v) = \frac{G*(u,v)}{|G(u,v)|^2 + \gamma|C(u,v)|^2} B(u,v)$$

Equation 13. Constrained Least Squares Formula

- F(u,v) = Fourier Transform of Restored Image
- G(u,v) = Fourier Transform of PSF
- G(u,v) = complex conjugate of G(u,v)
- C(u,v) = Fourier Transform of Laplacian Kernel
- γ = Adjustable Constant

II. Results

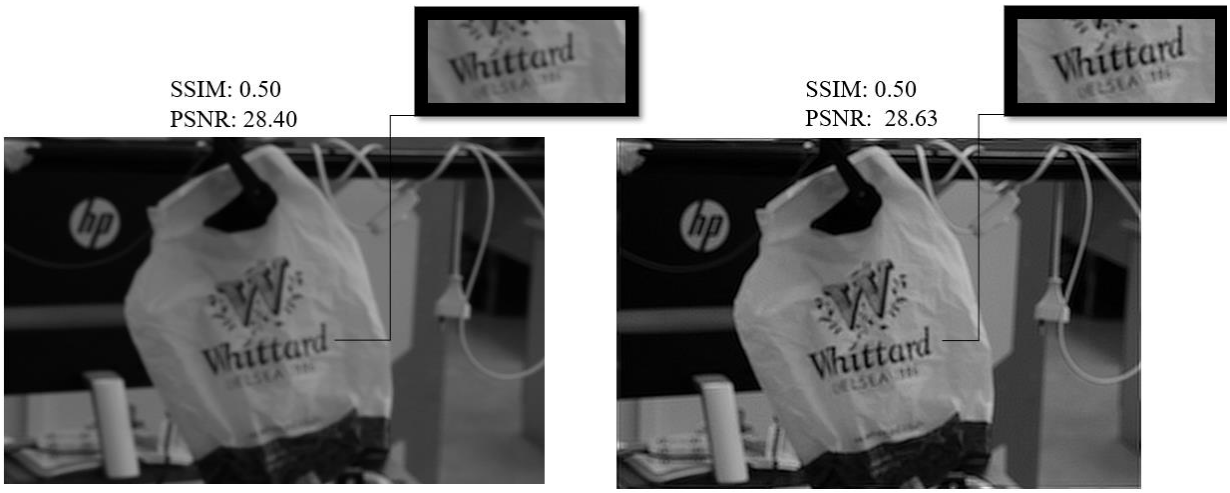


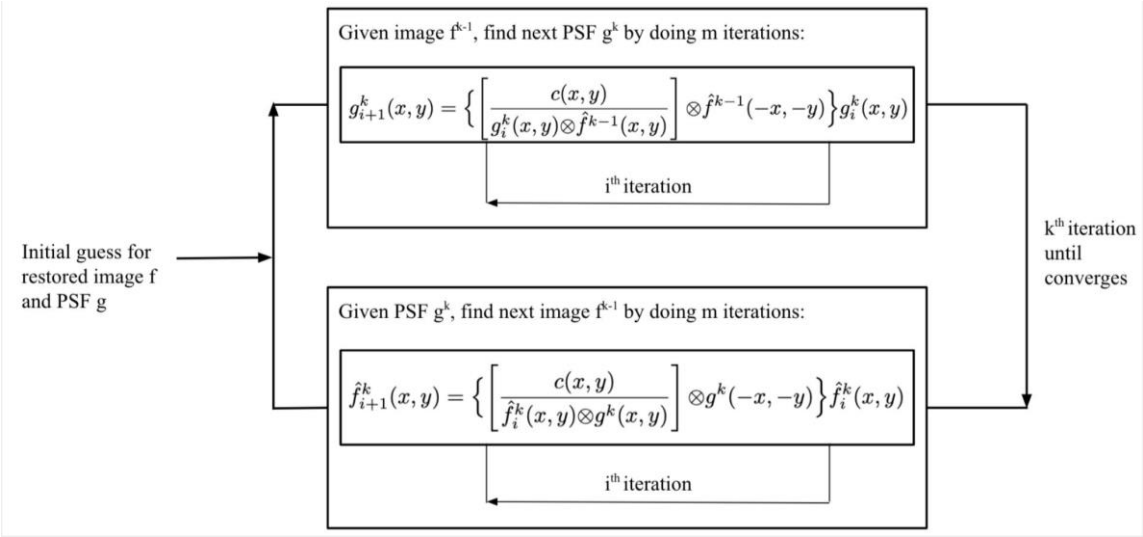
Figure 9. CLS Filter Results

The results in Figure 9 show the CLS application in the dataset. The original blurry image does not have a strong PSF so the filter did not have to be strong to achieve satisfactory results. The specific parameters were σ = 5 and γ = 1 which did not create strong ringing artifacts. Visually, the deblurred image looks better to the naked eye and the text is much clearer. However, the SSIM and PSNR do not significantly shift when the image is filtered. As noted in the dataset, this is because the ground truth image and blurred images are shot in different angles.

D. Richardson Lucy Algorithm

I. Richardson-Lucy Algorithm

The Richardson-Lucy algorithm is an iterative method in which an unblurred image is derived from a blurry image where the point spread function and the noise function are unknown. The algorithm works by first estimating both the restored image and the point spread function. In our case, the initial guess for the image is simply the degraded image we are given and the point spread function is a gaussian kernel to simulate the out-of-focus blur that we are trying to remove [7].



Equation 14. Richardson-Lucy Algorithm Formula

- f(x, y) = restored image
- g(x, y) = PSF g
- c(x, y) = degraded image
- i = inner loop index
- k = outer loop index
- $\otimes$ = convolution operation

As seen in Equation 14, the algorithm starts by having an initial guess for a restored image and a point spread function (PSF). Then, given this restored image guess, it uses the upper equation to calculate a new PSF and continues to recalculate for a set amount of inner iterations. Now, given this new PSF guess, it uses the lower equation to calculate a new restored image and continues to recalculate for the same set amount of inner iterations. This algorithm will perform these two steps for a set amount of outer iterations, or until the values converge.

II. Tikhonov Regularization

A problem with the Richardson Lucy algorithm is that iterating the function until the image converges does not usually produce the best visually looking result. This is because artifacts are gradually introduced to the image in high iterations. In order to solve this, we employ a stopping criteria to the algorithm which optimizes the amount of iterations that occur. We do so with a modified version of Tikhonov regularization, which gives each iteration of the function an error as shown below [10].

$$e^{(k)} = ||g - p \otimes \hat{f}^{(k)}||_2^2 + \alpha||\frac{\hat{f}^{(k)} - g}{g}||_2^2$$

$\otimes$ denotes a convolution operation
e = Error
k = Current iteration
g = Initial degraded image
p = Initial PSF
$\hat{f}$ = Current estimation for image
α = Regularization parameter

Equation 15. Tikhonov regularization stopping criteria formula

When the error of an iteration is higher than that of the previous iteration, the algorithm stops iterating. In doing so, the amount of iterations that occur is optimized and the algorithm produces the best visually looking results as possible.

III. Results

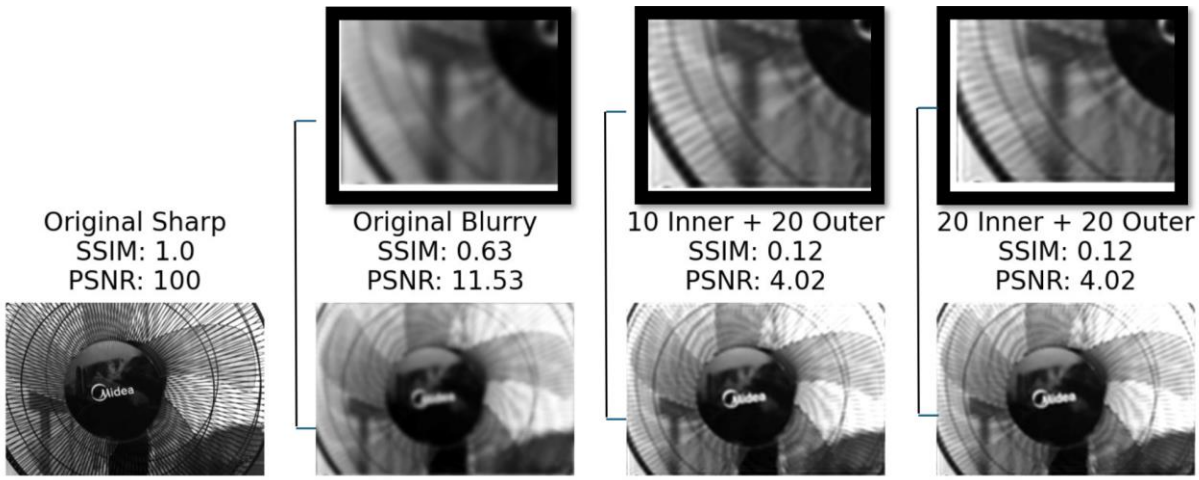


Figure 10. Richardson-Lucy Algorithm Results

The results from Figure 10 show that the Richardson Lucy algorithm visually sharpens a blurry image after enough inner and outer iterations. However, the SSIM and PSNR values decreased significantly, indicating that the image quality worsened according to our metrics. This error could be due to the brightness that we introduce to the image.

E. Semi-blind Wiener Algorithm

I. Introduction

This section explores the usage of the semi-blind wiener algorithm and how we used it to deblur out-of-focus images given in our dataset. This technique was chosen because it is useful for blind deconvolution. In practical cases, it is nearly impossible to determine the point spread function of the blurred image, and the wiener algorithm can take an estimate of the point spread function and use it to create a sharper image.

II. Wiener Filter

$$f(u, v) = \left(\frac{g^*(u, v)}{|g(u, v)|^2 + k} \right) c(u, v)$$

Equation 16. Wiener Filter Formula

The formula in equation 16 is the formula for the wiener filter, where the description of the variables are as follows:

- f(u, v) = Fourier transform of recovered image
- c(u, v) = Fourier transform of degraded image
- g(u, v) = Fourier transform of degradation function
- g*(u, v) = Complex conjugate of g(u, v)
- K = Constant estimating the noise-to-signal power ratio

This formula works by guessing the value of g(u, v) and K to ultimately remove the noise in the image while also retaining the features of the original image. Furthermore, this formula assumes that the value for K is second-order stationary.

III. Semi-blind Wiener Algorithm

$$f(u, v) = \left(\frac{g_i^*(u, v)}{|g_i(u, v)|^2 + k} \right) c(u, v)$$

Equation 17. Semi-blind Wiener Algorithm Formula

The formula shown in equation 17 is the formula for the semi-blind wiener filter algorithm. The variables displayed in the equation are the same as in the wiener filter formula, and the formula itself is nearly identical to the wiener filter formula. The only difference is that the initial estimate value for the point spread function changes after each iteration.

Hence, $g_i(u, v)$ represents the fourier transform of the degradation function at i^{th} iteration, and $g^*_i(u, v)$ represents the complex conjugate of $g_i(u, v)$ at the i^{th} iteration.

We implemented this algorithm because it uses iterative estimation to approximate the value of the point spread function. Therefore, after each iteration the point spread function changes until it converges to the best possible value which gives us the best possible picture.

IV. Results

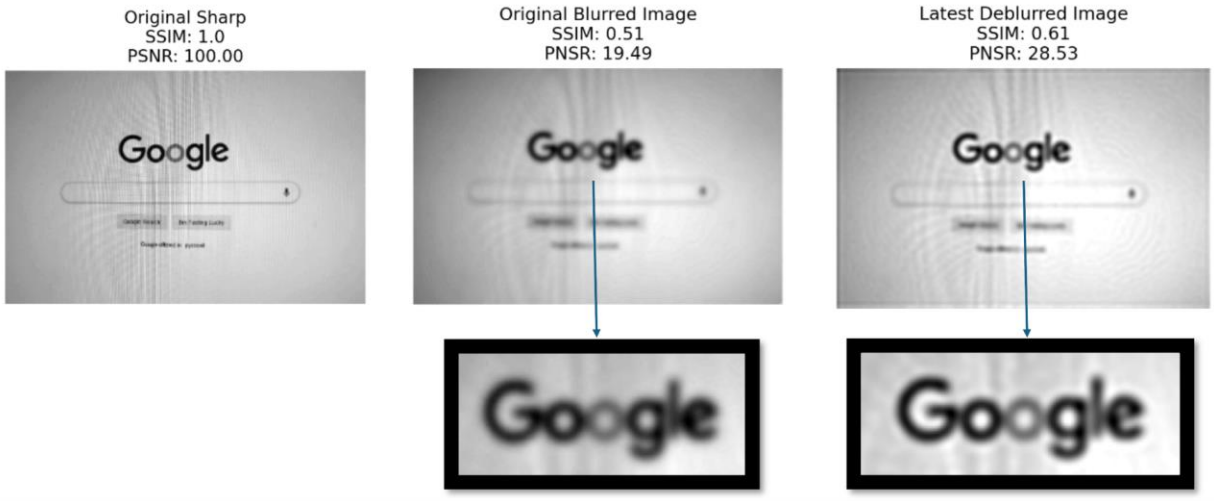


Figure 11. Results From Implementing The Semi-blind Wiener Algorithm

The results from figure 11 are the output from running the blurred picture through the Semi-blind Wiener algorithm. The picture on the very left represents the ground truth, which is the original sharp picture from the dataset, the picture in the middle is the original blurred picture that we will be attempting to deblur, and on the very right is the result from using the algorithm. By closely examining the text in the picture, we can see that the text is indeed clearer than on the blurred picture. Furthermore, by using the metrics we decided to use, our results are indeed sharper than the blurry picture.

However, it is worth noting that the output results are not as sharp as the ground truth, and there are still ringing artifacts around the borders of the picture. There are limitations offered with this algorithm; for example, this algorithm relies heavily on an initial guess for the value of the point spread function. While it does iterate the value for a specified amount, it still requires a good value for the point spread function for it to work optimally. Therefore, the algorithm's reliance on an accurate value for the point spread function prevents it from deblurring images with incomplete information. While the algorithm is not perfect, it still displayed positive progress. Future work and implementations could be studied and used to enhance the performance of the semi-blind wiener algorithm.

8. Conclusions

Our team made significant progress this semester, as all of our algorithms were able to successfully deblur images at a certain scale with noticeable change. Moreover, we ensured that the algorithms were also able to take into account the various types of noise that are in blurry images. Our team also made sure that while the output images appear clearer to the naked eye, we also used two measurement scales, SSIM and PSNR, to validate the clarity of the new output images.

While the Semi-blind Wiener algorithm and the constrained least squares filter displayed positive results when their respective output images were measured using the SSIM and PSNR, the Richardson-Lucy algorithm did not not show positive results. We speculate that this is due to artificially brightening the original blurry image because the algorithm made the output image darker, and therefore it hindered the calculation for the SSIM and PSNR. Lastly, the wiener algorithm also produced an output image with ringing artifacts on the edges of the image.

While our algorithms are not perfect, they represent meaningful progress toward the goal of successfully finding the best methods to sharpen degraded images, specifically out-of-focus images.

9. Future Work

Based on the foundation that we established in Fall 2024 semester, we wanted to delve deeper into the deblurring process and focus on several key areas. Firstly, we plan to understand more complex algorithms such as RCNN, region-based convolutional neural networks and utilize its performance and capabilities in image deblurring. Additionally, we aim to explore other types of algorithms that could address the limitations presented by the algorithms that we implemented in this semester.

Lastly, we plan to expand our research beyond out-focus-blur and attempt to find other deblurring methods for other types of degradation on images. In particular, we want to focus on motion blur as part of our agenda for next semester. With these initiatives, our goal is to refine our methodologies in the field of image restoration.

10. References

[1] Aleksey Alekseev, “Blur dataset,” Kaggle.com, 2019. https://www.kaggle.com/datasets/kwentar/blur-dataset/data (accessed Sep. 03, 2024).

[2] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.

[3] Wang, Zhou, and Alan Bovik. “Mean Squared Error: Love It or Leave It?” *IEEE SIGNAL PROCESSING MAGAZINE*, Jan. 2009.

[4] Rudin , Leonid I, et al. “Nonlinear Total Variation Based Noise Removal Algorithms.” Physica D: Nonlinear Phenomena, North-Holland, 1992, www.sciencedirect.com/science/article/abs/pii/016727899290242F.

[5] Unsharp masking with python and opencv. Instruments & Data Tools. (2021, May 5). https://www.idtools.com.au/unsharp-masking-with-python-and-opencv/

[6] Sharpening: Unsharp Mask. Sharpening Using an Unsharp Mask. (n.d.). https://www.cambridgeincolour.com/tutorials/unsharp-mask.htm

[7] Fish, D. A., Walker, J. G., Brinicombe, A. M., & Pike, E. R. (1995). Blind deconvolution by means of the Richardson–Lucy algorithm. Journal of the Optical Society of America A, 12(1), 58. https://doi.org/10.1364/josaa.12.000058

[8] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. New York, Ny: Pearson, 2018.

[9] “Hubble's Mirror Flaw.” NASA. science.nasa.gov/mission/hubble/observatory/design/optics/hubbles-mirror-flaw/ (accessed December 6, 2024)

[10] Khetkeeree, Suphongsa. (2020). Optimization of Lucy-Richardson Algorithm Using Modified Tikhonov Regularization for Image Deblurring. Journal of Physics: Conference Series. 1438.

11. Appendix

Jacob Morales

As an FYE engineering student, this was my first semester working in VIP as well as working with image processing. I have a small background in coding in various different languages such as Java, Python, and C#. In addition, my main interest in engineering is computers and hardware so learning how to manipulate images was a fresh topic to me.

Within the group, I worked on modeling the PSF and creating an algorithm that outputs the desired PSF. This algorithm would be used or built upon for the degradation function in all of the filters. Moveover, I modeled the constrained least squares filter which used the laplacian operator to account for noise. This was one of many filters we modeled and used for our deblurring filters.

As a result, I learned a lot about fourier transforms and even created an algorithm that turned an image in its fourier domain. However because fourier transforms are expensive computationally, I used the OpenCV fourier transform as it used the fast fourier transform (FFT) which was miles faster. Overall, I had a great time with my group and learning about image processing concepts.

Angela Qian

This semester was my first time working with the Image Processing and Analysis team on blur correction. While I have a strong background in coding and am proficient in several programming languages, including Python, my experience with image processing concepts was limited before this class. To close this gap, I spent the early part of the semester familiarizing myself with foundational image processing concepts by completing the provided tutorials.

Next, I shifted my focus on blur detection. I tried applying a Laplacian filter and calculating the variance of an image to determine how blurry it is, but since this method was not standard practice, it wasn't included in our final results. I then turned my focus onto developing an unsharp masking algorithm that used a Gaussian Filter to sharpen the edges in an image as well as increase contrast. I also worked on both recreating and fixing noisy images, specifically salt and pepper noise, shot noise, and mixed noise. To try to fix the noise in the images, I looked into and implemented the median filter, box blur, as well as also implementing the unsharp masking to sharpen the slightly blurred de-noised images. By the end of the semester, I was able to recreate noise in an image, reduce noise, as well as deblur images.

Brian Ramos

This was my first semester in vertically integrated projects and image processing similar to my team. In the initial stages of the project I had to learn the basics of image processing which included how to perform convolutional operations on images as well as understanding different kernels. Once I was able to implement an edge detection program using convolution and sobel kernels which was used as our initial way to detect blurred images, I began to research further ways to complete our project objective. My research brought me to the Wiener filter, an equation that used fourier transforms and a point spread function in order to unblur an image. I first had to research fourier transforms and understand how they worked, and also how point spread functions were used to degrade images. Once I learned these I was able to implement a non blind wiener algorithm that would unblur an image given the degradation function. This program required me to implement fourier transforms and my own point spread function generator My final contributions to the project include the Richardson-Lucy algorithm, an iterative method which performs blind deconvolution, and further optimized this method by implementing a stopping criteria using Tikhonov regularization.

Alessandra Rice

This was my first semester in Image Processing and VIP. Before joining, I had no experience with any form of coding language. This meant I spent the first several weeks of the semester working to learn the language. Additionally, as a freshman, I spent a lot of time trying to grasp the underlying calculus behind image processing.

While I did not start the semester on the Blur Correction team, I joined halfway through the semester and did my best to quickly catch up on the team's work prior to my joining. After that, I was able to contribute to the team initially by working on the richardson lucy algorithm, then quickly moving to implement total variation denoising - a task that severely tested my knowledge of math and took several weeks to fully understand and implement. I also created and implemented our PSNR function, and worked on debugging the problems we had implementing image metrics into our existing code especially with the Richardson-Lucy.

Ramsey Daniel

This was my first semester joining the VIP image processing team (IPA). I already had prior experience in several coding languages, including Python. However, my background in python isn't that extensive. Be that as it may, I worked throughout the first four weeks of the semester to get a good grasp and understanding of how Python can be used in computer vision, specifically image processing. After studying for the first few weeks, I made long strides in understanding image processing.

Afterwards, I was able to contribute to the team by using a deblurring method known as the wiener filter. Then I spent several weeks fully understanding how this filter works so that I can properly write it in code. Thankfully, I was successful in doing so as I managed to deblur multiple pictures and made them sharper than they originally were.

To improve the quality of the pictures deblurred using the wiener filter, I decided to use an algorithm in tandem with the wiener filter so that I can achieve the best picture possible. During the remainder of the semester, I took the time to carefully study how the algorithm would perform, as it meant that I needed to be able to make an initial guess for the value of the point spread function. In the end, I was able to successfully write a code for the semi-blind wiener algorithm which successfully deblurred an image to make it sharper.

Jeslyn Yang

This was my first semester working with the Image Processing and Analysis team for blur correction. Although I have a thorough background in coding and already know several coding languages, including Python, I had limited experience in image processing concepts until this class. To bridge this knowledge gap, I spent the beginning of the semester becoming familiar with basic image processing concepts by completing the provided image processing tutorial such as implementing a box blur and Sobel Edge detection.

I then moved on to researching other methods more related to blur detection. This included a blur detection method involving the application of a Laplacian filter and calculating the variance of an image to detect if it was blurry or not. However, this method is not a standard practice and therefore did not make it to our final results. I was also able to develop an unsharp masking algorithm that used a Laplacian filter, an iterative Richardson-Lucy algorithm, and an accurate SSIM algorithm, which were all included in our final results. These were all methods I spent time researching through papers and debugging in the latter half of the semester.

By the end of the semester, I learned a lot about image processing and different blur detection and correction methods. This will be useful in my software engineer career if I choose to further explore or pursue computer vision.