Bayesian Deblurring with a Known Gaussian Filter

Fall 2023

Abstract – A Bayesian algorithm was made to deblur an image that has been blurred with a known Gaussian filter. Bayes theorem is formulated using per-pixel Gaussian Difference and a Gaussian Neighborhood prior to introduce dependencies between neighboring pixels. The formulation is then solved using Metropolis Hastings MCMC. Finally, we look at the results under different blur conditions and make comparisons to existing deblurring algorithms.

Introduction

Images can often end up blurred unintentionally. We would like to de-blur the image and regain the original information. The general problem of deblurring is extremely difficult as we do not know the type of blur, the size of the blur, or any information about how that image was produced. We can solve a much simpler version of this problem where we try to find the original image with a known Gaussian Blur. To further simplify the problem, we will only work with black and white images instead of three color dimensions. With the problem formulated, let's understand how image blurring works.

Method

Fundamentally, we blur an image by convolving it with a blur kernel. This kernel can vary in size but it will be a 2D Gaussian of the form:

$$N(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

If we have a $\sigma = 1$ and sample a range over -3σ to 3σ, we will get the kernel in Figure 1.

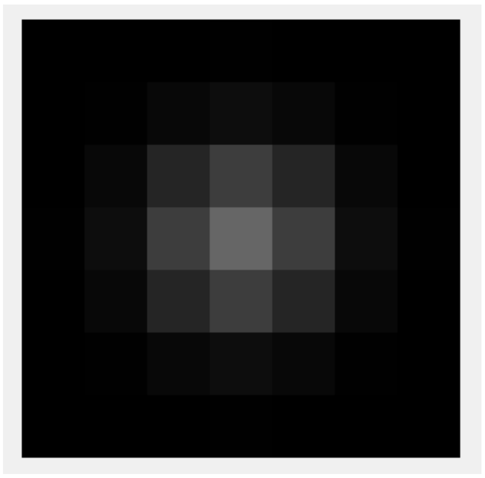

Figure 1: Gaussian Kernel

Figure 1 represents a kernel we can convolve with the image. The forward function of blurring where K is the Gaussian Blur Kernel, Y is the original clear image, and X is the final blurred image.

$$X = Y * K$$

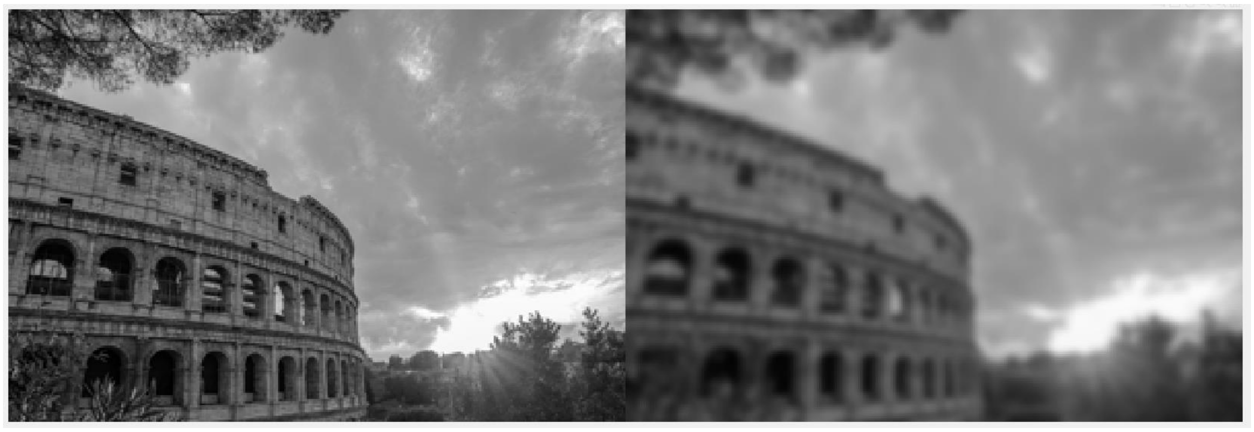

Figure 2: Clear Image (Left), Blurred Image (Right)

In Figure 2 we have a clear image on the right. If we have a blur kernel of size 9x9, and a sigma of 2, we get the blurred image on the right.

The goal of this project is to find the original image Y given a kernel K and a blurred image X. Formulating this as a Bayes' problem we would write:

$$p(Y \mid X, K, I) = P(Y|I) * \frac{p(X|Y, K, I)}{Z}$$

Here we want to find the odds of the hypothesized clean image Y given the blurred image X, the kernel K, and any other prior information. On the right side we have a prior for any given hypothesized image, the likelihood based on the blurred image, and the normalizing evidence term Z. This formula may seem simple but due to the images being matrices with 100’s of rows and columns, this makes the problem explode in dimensionality. To avoid this we can reframe the problem around a single pixel where i and j index the row and column.

$$p\left(Y_{ij} \mid X_{ij}, K, I\right) = p\left(Y_{ij} \middle| I\right) * \frac{p\left(X_{ij} \middle| Y_{ij}, K, I\right)}{Z_{ij}}$$

Doing this does have drawbacks though. By treating each pixel as an independent atomic object, we lose sight of the bigger picture, literally. Pixels are not independent and they do have important relationships with each other. We will address this problem by adding back some dependence between pixels in the prior. Let's breakdown what the likelihood and prior functions will be.

Likelihood Function

To keep things simple, we will choose a Gaussian likelihood that compares the hypothesized pixel value and the blurred pixel value:

$$p\left(X_{ij} \middle| Y_{ij}, K, I\right) = \frac{1}{\sqrt{2\pi}\sigma_L} \exp\left(-\frac{\left(X_{ij} - (K * Y)_{ij}\right)^2}{2\sigma_L^2}\right)$$

First we generate a hypothesis image, then we blur that guess with the known blur filter. Finally we compare each pixel with the Gaussian where the given blur image is the mean. If our guess image was the original image, it would blur exactly to X and maximize the likelihood. Other guesses can do a similarly good job so we need our prior to help shrink the search space.

Prior Functions

Choosing a prior for an image is a lot less simple. It can be one of the most important steps when there are so many possible hypothesized images. Reading into some of the literature shows a variety of priors. We can ask what qualities of a hypothesized image would make it more or less probable. Most methods try to find commonalities in images that some hypothesized image should have. For example, [1] uses image gradient statistics from an image dataset to produce an informed prior. Papers [2], and [3] also use image gradient distributions to define a prior. This is how we are able to get a single pixel to be aware of the whole image. We can do this more directly using a neighborhood prior found in [4]. This method compares some hypothesized pixel to its neighbors above, below, left and right. The idea is that pixels close to each other are often similar, so we define a function to decrease the prior probability based on the difference of a central pixel to its neighbors.

$$p\left(Y_{ij}|N_{ij},I\right) \propto \exp\left(-\frac{1}{2\sigma_P^2}\sum_{m,n\in N_{ij}}\left(Y_{ij}-Y_{m,n}\right)^2\right), \qquad \text{where } \mathrm{N_{ij}} \text{ is the neighborhood of } Y_{ij}$$

In this case, we use a Gaussian but any symmetrically increasing function around Y_{ij} could work such as the absolute value.

The complete Bayes theorem with the evidence and Gaussian Coefficients taken as a constant of proportionality results in:

$$p\left(Y_{ij} \mid X_{ij},K,I\right) \propto \frac{1}{\sigma_L^2\sigma_P^2}\exp\left(-\frac{1}{2\sigma_P^2}\sum_{m,n\in N}\left(Y_{ij}-Y_{m,n}\right)^2\right) * \exp\left(-\frac{\left(X_{ij}-(K*Y)_{ij}\right)^2}{2\sigma_L^2}\right)$$

The sigmas of each Gaussian were chosen through experimentation and ended up being $\sigma_L = 1, \sigma_P = 100$. A lower sigma means a larger uncertainty or a lower cost to guessing wrong. The likelihood was the most important factor in making sure the image generated actually produced a blurred image that matched the given blurred image. The likelihood alone cannot make the image clearer as it would produce images that blur properly but look random to the eye. This is where the softer guidance of the prior helped. It was found that larger a larger sigma helped, and incorrect patterns would be formed at lower sigmas. Keep in mind that the pixel values are in between 0 and 255. A sigma of 100 allows for decent variation in pixel value where as a sigma of 1 means nothing outside a deviation of 5 from the mean has any chance.

Numerical Technique

Despite simplifying the problem, solving Bayes' Theorem analytically would be difficult if not impossible. The prior's dependence with neighboring pixels makes this extra difficult as well as needing to do this for every pixel in a given image. For this reason we will use a Markov Chain Monte Carlo technique to approach the correct posterior distribution for each pixel.

The technique used will be the Metropolis-Hastings algorithm. This algorithm works by slightly modifying a previous guess, testing if the new guess is better, and then accepting this with some uniform probability. Let's say a given hypothesized pixel value is y. We first sample the normal distribution scaled by some step value, s, to produce a guess, g.

$$g = y + s \cdot \mathcal{N}(x)$$

The step size is how fast we explore a pixels possible values. The larger it is, the faster we can try different things, but the easier it is to jump past an optimual value. The smaller it is, the more precise of a guess we can make to find the peaks, but the easier it becomes to get stuck in a local optimum.

Given a guess we can find the acceptance ratio which is simply the ratio of the posteriors. As the evidence is a constant across all hypthoeses, it cancels out:

$$\text{Acceptance Ratio} = \text{AR} = \frac{p(g|I) * p\left(X_{ij}|g, K, I\right)}{p(y|I) * p\left(X_{ij}|y, K, I\right)}$$

We also clamp this between 0 and 1 with $min(1, AR)$

If the AR is closer to one, the guess is better, and opposite for an AR closer to zero. At this point, we would have a procedure that would climb to the nearest optimum value for each pixel. This is a problem since we may get stuck at a local optimum instead of finding the global one. To avoid this we must accept worse guesses some of the time. This is implemented by sampling from the uniform distribution. If this sample is less than the clamped Acceptance Ratio, we accept the guess.

$$\text{Accept} = U[0,1] < \min(1, AR)$$

Producing the Final Image

As we run the Metropolis-Hastings algorithm, we settle at a general image structure. However, due to the random nature of the class of Markov Chain Monte Carlo Algorithm, each pixel will still have some small acceptance rate which produces noisy changes over every iteration. Because the Markov Chain approaches the posterior, we can take the mean of our algorithm's samples to approximate the mean of the posterior. This will eliminate the noise of the algorithm to produce a clearer image. Note, we seed the initial guess with the blurred image. Finally, we need some metric by which we exit the loop. We can choose a maximum iteration of 10,000 which was chosen to be very large based on experimenting with the algorithm. Of course the algorithm could settle at a solution way before we reach 10,000 iterations. We will measure the RMS Error of each iteration (Equation Below), and if we see that this RMS isn't changing a lot between iterations, we can break the loop.

$$\text{RMS Error} = \sqrt{\frac{1}{i \cdot j} \sum_{j} \sum_{i} \left(Y_{ij}\right)^{2}}$$

The chosen method was to measure the RSM difference between every 1000 iterations which avoids the local variations between iterations. Experimentally, a difference of less than 0.05 is considered to be settled and will break the loop. As a final note, the RMS error often spikes up, the goes down in the first thousand iterations (See Figure 6). To avoid breaking the loop early, we wait for 2000 iterations to do this comparison. This is because the initial error and some error 1000 iterations later will be similar but this doesn't represent any improvement in the image. The initial error spike is just the Markov Chain "burning in" approaching the posterior.

Results

Let's see how the algorithm fairs with some different images. We start with a Gaussian blur that is 9 by 9 and a sigma of 2. In Matlab, a Gaussian Blur Kernel is sized as $2(2\sigma) + 1$, which makes the kernel an odd size and extends to 2σ of the Gaussian.

If we observe the algorithm run we can see it evolve in Figure 3, Figure 4, and Figure 5. These represent a snapshot of the algorithm at 250, 1000, and 5000 iterations. Early on, the algorithm has got the general image completed already as it is seeded with the blurred image, but it is quite noisy at this point. As the algorithms settles, the noise averages out and the images become clearer at 1000 and 5000 iterations. These figures also show the blurred guess image which is what the likelihood function compares. We can see the difference of the images in the bottom right of each figure. It is mostly noisy as algorithm has gotten the structure correct already. In Figure 6 we can see how the error evolves over time. It spikes in the beginning as the numerical algorithm burns in. As the algorithm further progresses the RMS Error drops until it settles. Due to the log axis, the last tail appears to have a more dramatic slope than it really does.

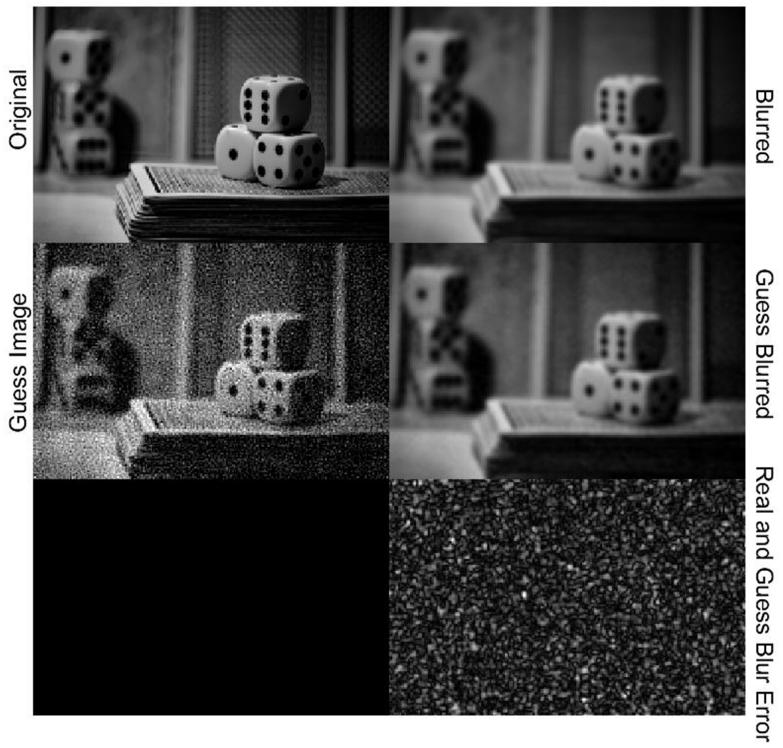

Figure 3: Algorithm at 250 iterations

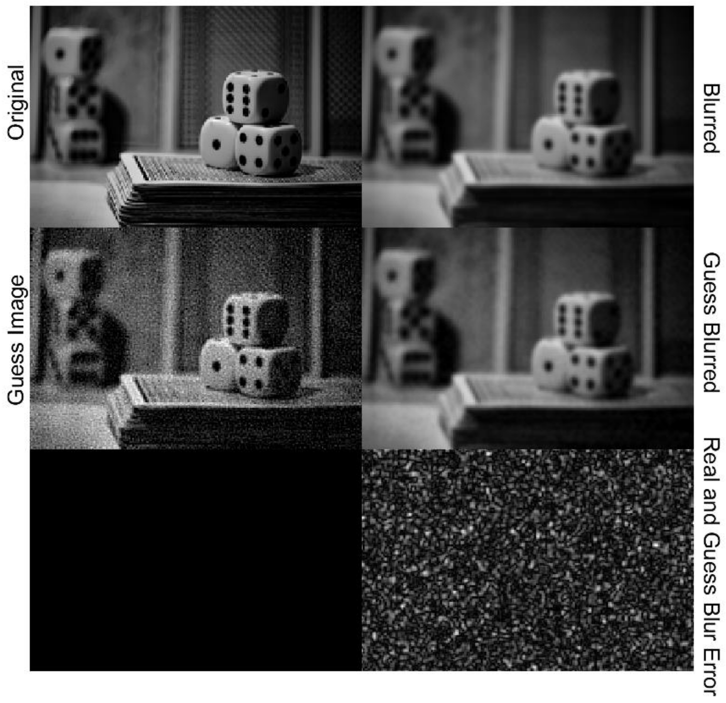

Figure 4: Algorithm at 1000 iterations

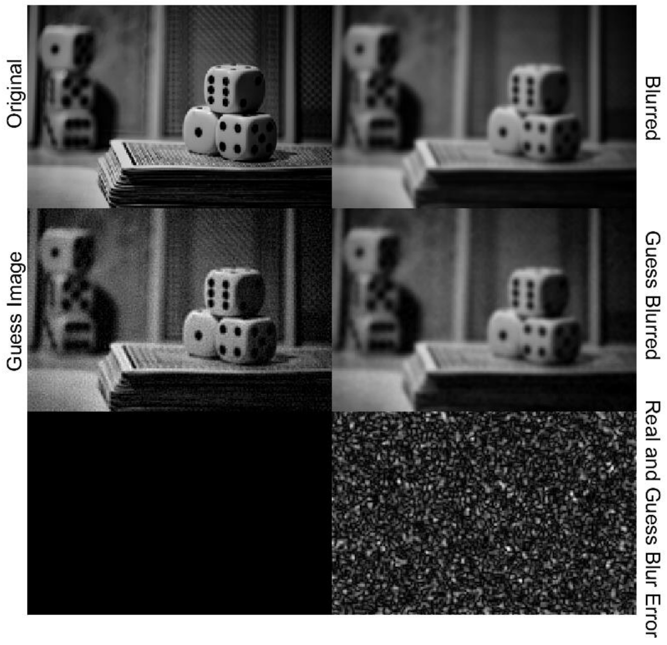

Figure 5: Algorithm at 5000 iterations

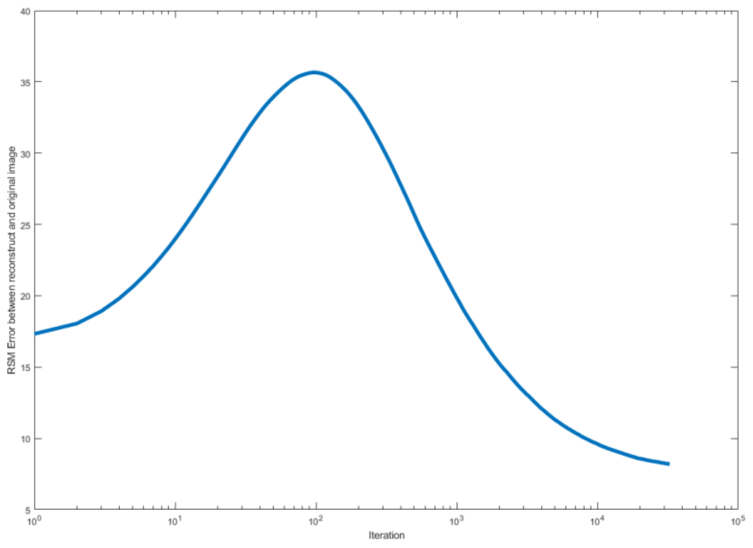

Figure 6: Plot of the RMS error over the number of iterations (log plot)

If we look at Figure 7 and Figure 8, we have a table of images. The top row represents the original image, the second row represents the blurred image, and the final row is the reconstructed image from this report's algorithm. Under each column, the RMS Error between the original and reconstructed image is shown, and the number of iterations it took to complete. Visually, the reconstructed images have more details than the blurred but still aren't as detailed as the original. The reconstructed images also have some noise in them. Despite this, the reconstruction clears the image up nicely. We can compare the initial and final RMS Errors in Table 1 below. Initial being the error between the correct and blurred image, and final being the error between the correct and reconstructed image.

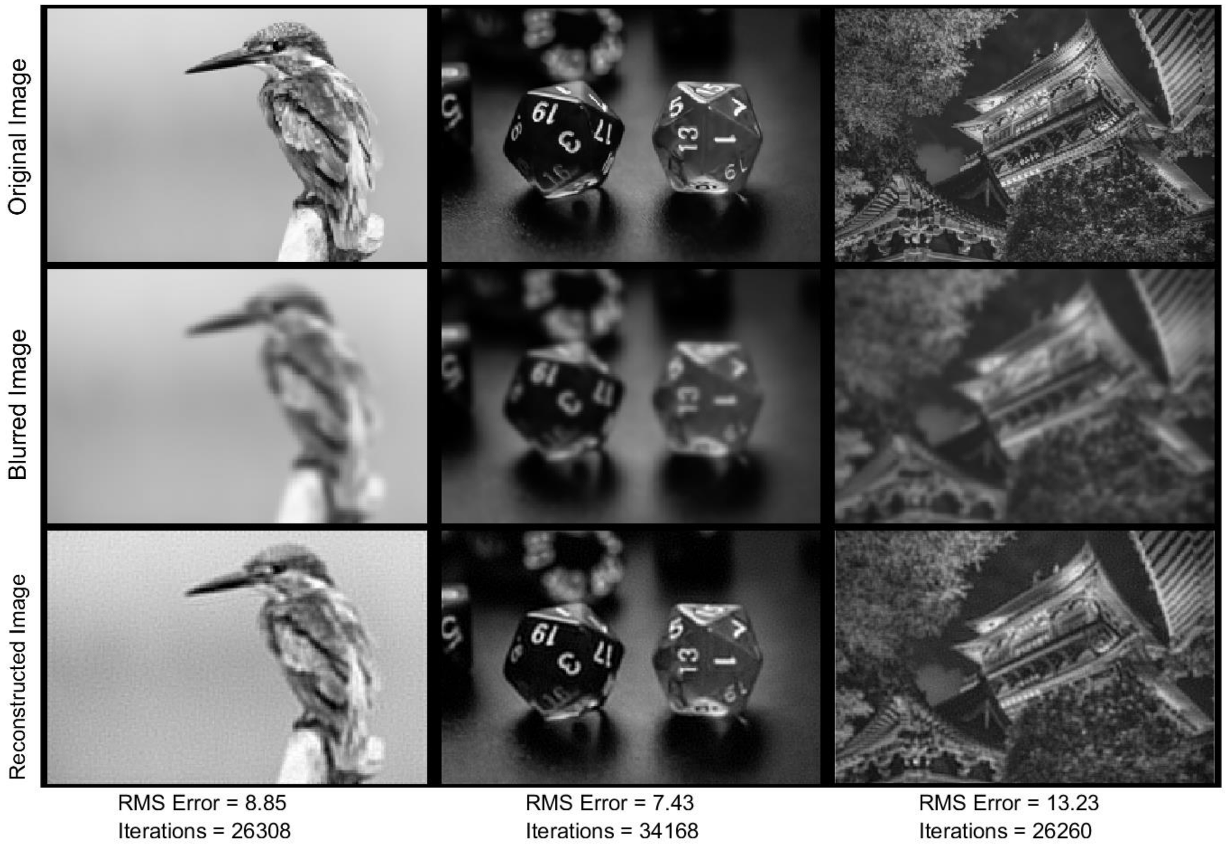

Figure 7: Image Set 1 containing the original, blurred, and reconstructed images.

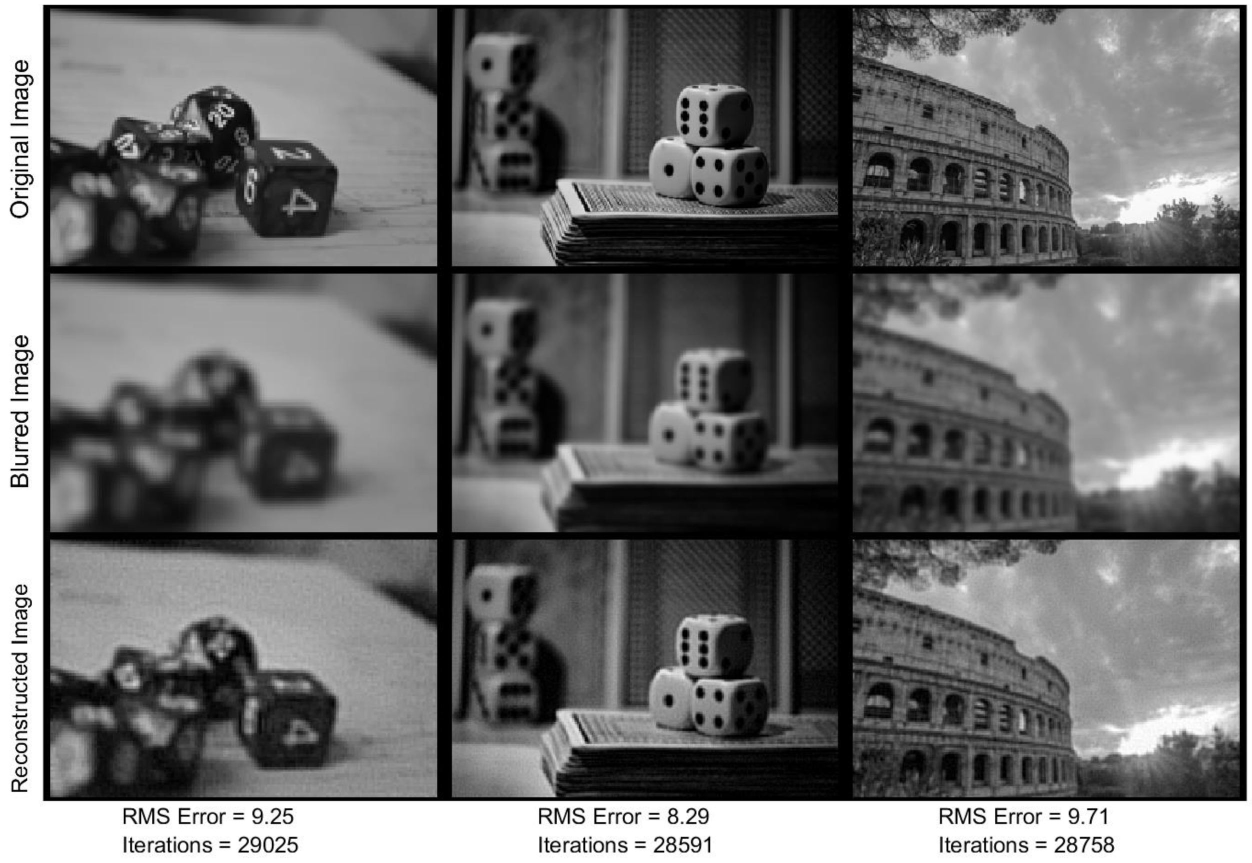

Figure 8: Image Set 2 containing the original, blurred, and reconstructed images

Table 1	**Bird**	**Two Dice**	**Pagoda**	**Dice Rolled**	**Dice Stacked**	**Colosseum**
RMS Blurred	13.84	14.05	18.98	14.66	13.07	13.87
RMS Reconstructed	8.85	7.43	13.23	9.25	8.29	9.71

To get a more quantifiable understanding of how this algorithm performs we can compare it to some existing deblurring/deconvolution algorithms. Matlab has a few built in including a Blind Deblurring Algorithm, the Lucy-Richardson Algorithm, and another Regularized Filter Deconvolution algorithm. Figure 9 shows each algorithms reconstruction including this reports', and the far left shows the original image. The bottom row shows the difference between the reconstruction and original image, amplified so the differences can be seen. Finally, the RMS Error is displayed on the bottom. Our algorithm performs on par with existing ones, and in this particular case, actually performs the best.

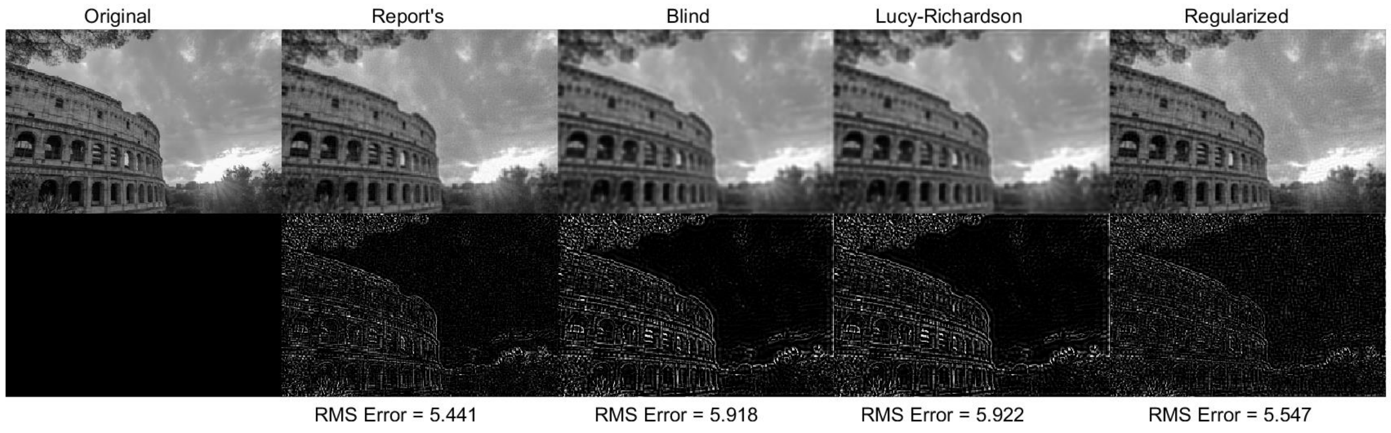

Figure 9: Comparison of Matlab deblurring algorithms to this report's. The top row is the reconstructed image. The bottom row is the exaggerated difference between reconstructed and actual.

We can rerun the same tests with a kernel of 5 by 5 with a sigma of 1 to see how it performs at lower blur levels. Again Figure 10 and Figure 11 show the original, blurred, and reconstructed images with the RMS and number of iterations at the bottom. Because the blur is weaker, a lot more detail is in the final image which is why the RMS Error is much smaller. The algorithm isn't perfect and as information is lost during the blur, the reconstruction is still somewhat noisy and less detailed. Same as before, the initial and final RMS values are in Table 2 below.

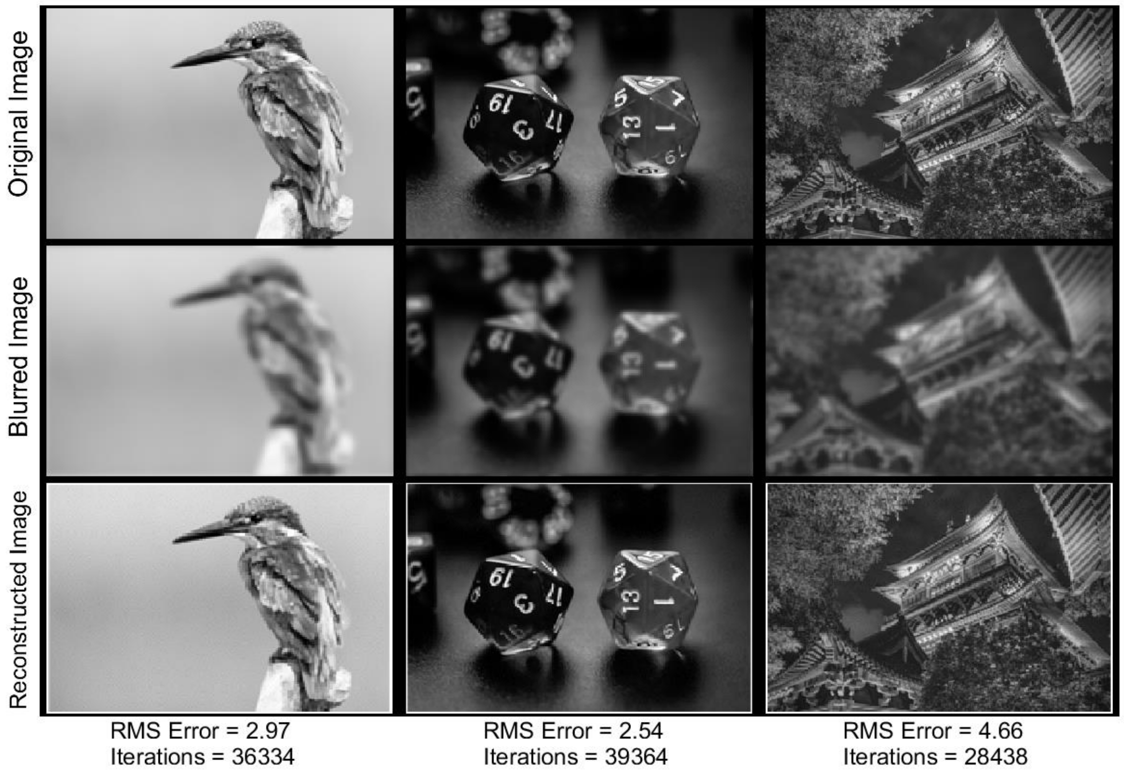

Figure 10: Image set 1 of the original image, blurred image and reconstructed images.

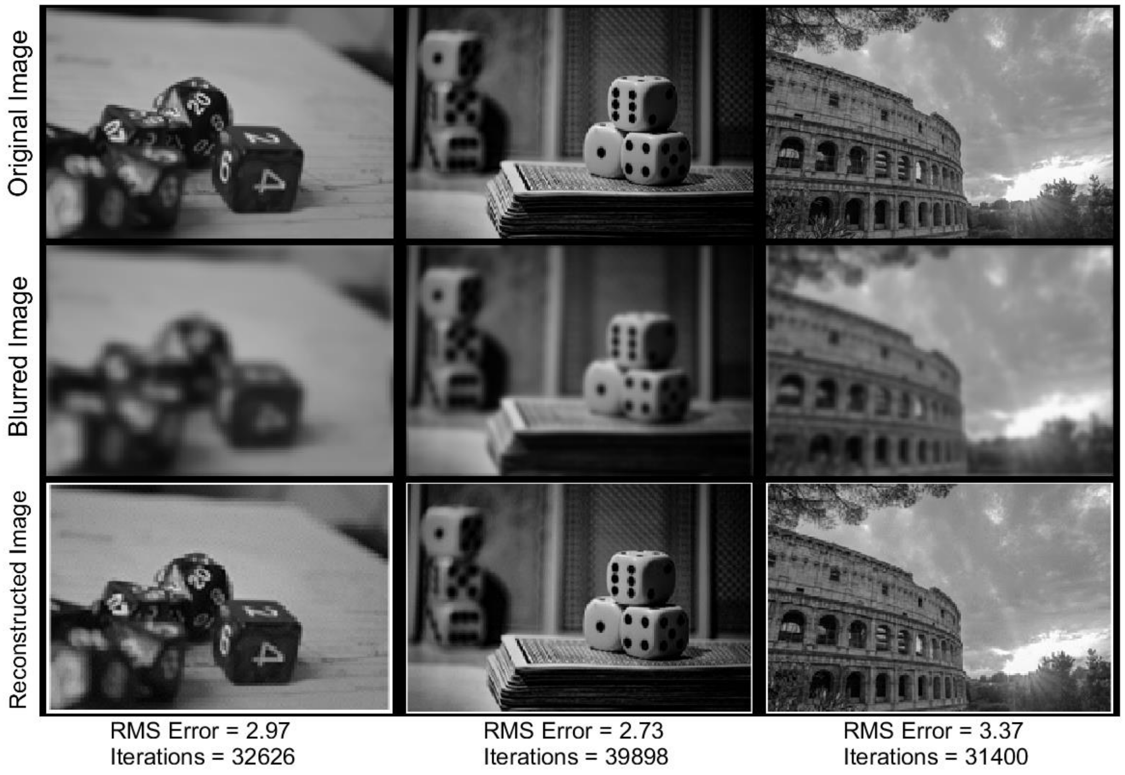

Figure 11: Image set 2 with original image, blurred image, and reconstructed image

Table 2	**Bird**	**Two Dice**	**Pagoda**	**Dice Rolled**	**Dice Stacked**	**Colosseum**
RMS Blurred	8.60	7.96	13.17	9.10	8.09	9.29
RMS Reconstructed	2.97	2.54	4.66	2.97	2.73	3.37

Making the same comparisons as in Figure 9 we do the same in Figure 12. In this case, the report’s algorithm is on par with existing solutions, though doesn’t perform the best.

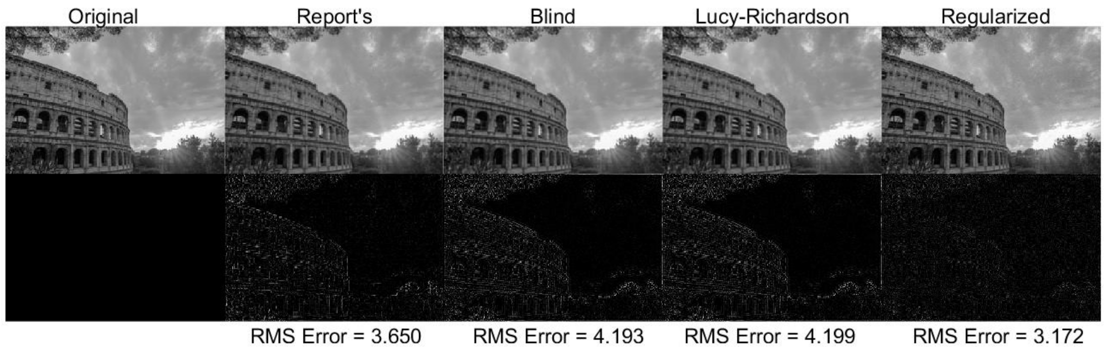

Figure 12: Comparison of this report's algorithm with other Matlab Deblurring algorithms

Conclusion

A Bayesian formulation of the deblurring problem was created with known Gaussian blur. Using Metropolis-Hastings to iterate all of the pixels, we were able to approximate the pixel posteriors. The results of the deblurring worked well as measured by the RMS Error. Visually, the images had more details than the blurred but still has some noise and blur compared to the true original image. Comparing the algorithm to existing deblurring algorithm, we found ours to be comparable in performance using the RMS Error.

References

[1] H. Yang, X. Su, J. Wu, and S. Chen, “Non-blind image blur removal method based on a Bayesian hierarchical model with hyperparameter priors,” *Optik*, vol. 204, p. 164178, Feb. 2020, doi: 10.1016/j.ijleo.2020.164178.

[2] D. Krishnan and R. Fergus, “Fast Image Deconvolution using Hyper-Laplacian Priors,” in *Advances in Neural Information Processing Systems*, Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, Eds., Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2009/file/3dd48ab31d016ffcbf3314df2b3cb9ce-Paper.pdf

[3] A. Levin, R. Fergus, F. Durand, and W. Freeman, “Deconvolution using natural image priors,” *Mass. Inst. Technol. Comput. Sci. Artif. Intell. Lab.*, [Online]. Available: https://groups.csail.mit.edu/graphics/CodedAperture/SparseDeconv-LevinEtAl07.pdf

[4] J. Besag, “Digital Image Processing: Towards Bayesian image analysis,” *J. Appl. Stat.*, vol. 16, no. 3, pp. 395–407, Jan. 1989, doi: 10.1080/02664768900000049.