MONTE CARLO OPTIMIZATION OF NON-LOCAL MEANS DENOISING ALGORITHM

ROBU PETRU-RĂZVAN, VERZOTTI MATTEO-ALEXANDRU, VOAIDEȘ-NEGUSTOR ROBERT-IONUȚ

Abstract. Non-Local Means is a powerful algorithm for image denoising, yet its high computational complexity limits real-time application. This paper replicates the Monte Carlo Non-Local Means Optimization, which utilizes random sampling to accelerate weight calculations without significantly degrading peak signal-to-noise ratio.

Keywords. Denoising, Monte Carlo, Optimization

1. Introduction

Image noise is a random variation of brightness or color information. In most real-life cases, it can be modeled as additive white Gaussian noise:

$$y = x + \eta \tag{1.1}$$

Where y is the noisy pixel, x the pure pixel and $\eta \sim \mathcal{N}(0, \sigma)$ is the noise.

1.1. Denoising Techniques.

Traditional denoising techniques, such as Gaussian smoothing, Median filtering or Local means, operate on the principle of locality, assuming that the true value of the pixel must be similar to the values of its neighbours. This is effective at removing noise, but it also leads to loss of fine textures and blurring of edges. Newer techniques include Convolutional Neural Networks, Wavelet Transforms [9], and Modified Decision-Based Median Filter [10]. Although these methods can achieve impressive results, they often require extensive training data, computational resources, or complex parameter tuning. Non-Local Means (NLM) [3] is remarkable in its simplicty and effectiveness, however its high computational complexity limits real-time application.

1.2. Non-Local Means.

The Non-Local Means estimates the true value of the pixel by computing a wieghted average of different patches, using a weight function that prioritizes pixels with similar structural patterns. The most similar pixels to a given noisy pixel have no reason to be in its immediate vicinity. For example, patterns such as smooth surfaces, periodic textures or repeated structures can appear in different areas of the image. Although the search space is larger, we are still going to use only a neighbourhood of the pixel to limit the computational cost. As aknowledged by the authors themselves [4], the term "non-local" is somewhat misleading, and a more accurate name would be *"semi-local"*.

Definition 1.1. Suppose Ω is an image domain, and let p, $q \in \Omega$ be two points within the image. Then, the algorithm for denoising is defined as:

$$z(p) = \frac{1}{C(p)} \int_{\Omega} w(p,q)y(q)\, dq \tag{1.2}$$

where $z(p)$ is the denoised value at point p, $y(q)$ is the noisy value at point q, $w(p,q)$ is the weight function that measures the similarity between patches centered at p and q, and $C(p)$ is a normalizing factor given by:

$$C(p) = \int_{\Omega} w(p,q)\, dq \tag{1.3}$$

Obviously, in a real-world application we would use a discrete version of this algorithm:

$$z(p) = \frac{1}{C(p)} \sum_{q \in \Omega} w(p,q)y(q) \tag{1.4}$$

where, again, the normalizing factor is given by:

$$C(p) = \sum_{q \in \Omega} w(p,q) \tag{1.5}$$

Given a noisy pixel, the surrounding patch $\mathbf{y}$ will be used to check for similarity with other patches. This object can be flattened into a d-pixel: $\mathbf{y} \in \mathbb{R}^d$. The algorithm requires a set of patches $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ that are obtained from reference images. Given this, the discrete version of NLM can be expressed as:

$$z = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \tag{1.6}$$

where x_i represents the center pixel in the patch $\mathbf{x_i}$, and the weight w_i measures the similarity between the match $\mathbf{y}$ and $\mathbf{x_i}$.

A standard choice for the weight function is:

$$w_i = e^{-||\mathbf{y}-\mathbf{x_i}||^2/(2h_r^2)} \tag{1.7}$$

Where h_r is a scalar parameter that controls the decay of the distribution: small values create a fast decay and ensure a focus on stricter similarity between patches, whereas larger values allow different patches to have a greater weight. And $||\cdot||$ is the Euclidian norm.

The downside of NLM is it high computational complexity. The most expensive step of this algorithms is computing the weights $\mathbf{w}_i$, which leads to $\mathcal{O}(mnd)$ operations, where m is the number of pixels to be denoised, n the number of patches and d the dimensions of the patch. In reality, a study has shown that it's enough to only use a localized window of dimension D around the pixel to be denoised [8], which reduces the complexity to $\mathcal{O}(mD^2d)$. Keep in mind that m and d are also quadratic in nature of the image and patch sizes.

A fun, but not-so-trivial way of improving the computational complexity is to use summed-area tables and the Fast Fourier Transform, which achieves a theoretical complexity of $\mathcal{O}(mD^2\sqrt{d})$. An implementation of this by Jacques

Froment [7], called *Parameter-Free Fast Pixelwise Non-Local Means Denoising*, claims to speed up the process by a factor of 6 to 49.

In reality, both of these implementations, as well as the one below, are easily parallelizable, and can be sped up even further using GPU computing. In 4, we will explore a different approach that uses KD-Trees to accelerate the search for similar patches, at the cost of a slight decrease in the quality of the denoised image.

1.3. **Monte Carlo Non-Local Means (MCNLM).** This paper focuses on the implementation and effectiveness of a method developed in [5]. The main focus of the approach is to approximate 1.6 by selecting randomly k reference pixel and creating a k-subset of weights $\{w_i\}^k$. The computational complexity of this approach is $\mathcal{O}(mkd)$, which is significantly lower for $k \ll n$.

2. Monte Carlo Non-Local Means

In this section, we discuss the full implementation of the algorithm, the correctness of the approach and some other optimizations that can be added to the method.

2.1. **Sampling Patches.** There are two ways to extract reference patches. The first one is based on picking them from the original noisy, image, which is called *internal denoising*. The second method is based on having the patches taken from a large database of other images, which is called *external denoising*. The paper focuses on internal denoising.

The Monte Carlo sampling is done on the set $\mathcal{X} = \{x_1, \ldots, x_n\}$, considering each reference patch independent. The process is determined by a sequence of random variables $\{I_j\}_{j=1}^n$, where $I_j \sim \text{Bernoulli}(p_j)$. The weight will be sampled only if $I_j = 1$. The vector of these probabilities, $\mathbf{p} := [p_1, \ldots, p_n]^T$, is called the *sampling pattern* of the algorithm [5].

The main parameter of this algorithm is ξ, which is the expected value of the random variable which models the ratio between the number of the samples taken and the number of references:

$$S_n = \frac{1}{n} \sum_{j=1}^{n} I_j \tag{2.1}$$

which has:

$$\xi \overset{\text{def}}{=} \mathbb{E}[S_n] = \frac{1}{n} \sum_{j=1}^{n} \mathbb{E}[I_j] = \frac{1}{n} \sum_{j=1}^{n} p_j \tag{2.2}$$

So, an important requirement is that:

$$\sum_{j=1}^{n} = n\xi \tag{2.3}$$

S_n is called the *empirical sampling ratio* and ξ the average sampling ratio [5].

2.2. **Algorithm.** Given a set of references $\mathcal{X}$ and the sampling ration ξ, we can define the sampling pattern $\mathbf{p}$ in order for it to respect the condition:

$$\sum_{j=1}^{n} p_j = n\xi \tag{2.4}$$

.

We can approximate the numerator and the denominator in 1.6 using two random variables:

$$A = \frac{1}{n}\sum_{j=1}^{n} x_j w_j \frac{I_j}{p_j} \quad \text{and} \quad B = \frac{1}{n}\sum_{j=1}^{n} w_j \frac{I_j}{p_j} \tag{2.5}$$

To show why we scale by $\frac{1}{p_j}$, we calculate the expected value of the estimators. By linearity of expectation:

$$\mathbb{E}[A] = \frac{1}{n}\sum_{j=1}^{n} x_j w_j \frac{\mathbb{E}[I_j]}{p_j} = \frac{1}{n}\sum_{j=1}^{n} x_j w_j \frac{p_j}{p_j} = \frac{1}{n}\sum_{j=1}^{n} x_j w_j \tag{2.6}$$

and

$$\mathbb{E}[B] = \frac{1}{n}\sum_{j=1}^{n} w_j \frac{\mathbb{E}[I_j]}{p_j} = \frac{1}{n}\sum_{j=1}^{n} w_j \tag{2.7}$$

So, A and B are *unbiased estimators* of the true numerator and denominator. In the end, we will aproximate the result z by another random variable:

$$Z = \frac{A}{B} = \frac{\sum_{j=1}^{n} x_j w_j \frac{I_j}{p_j}}{\sum_{j=1}^{n} w_j \frac{I_j}{p_j}} \tag{2.8}$$

However,

$$\mathbb{E}[Z] = \mathbb{E}\left[\frac{A}{B}\right] \neq \frac{\mathbb{E}[A]}{\mathbb{E}[B]} = z \tag{2.9}$$

so, Z is a *biased estimator* of z. However, section 3.2 proves that as $n \to \infty$, the deviation $|Z - z|$ drops exponentially. This shows that, for a larger number of patches, the more efficient MCNLM gives results comparable to that of the simple NLM.

Algorithm 1 Monte Carlo NLM Denoising

Require: Noisy patch $\mathbf{y}$, reference set $\mathcal{X}$, sampling pattern $\mathbf{p}$
Ensure: Denoised pixel z

for j in $1 \dots n$ **do**
    Generate random variable $I_j \sim \text{Bernoulli}(p_j)$
    if $I_j = 1$ **then**
        Compute w_j
    end if
end for
Compute $A = \frac{1}{n} \sum_{j=1}^{n} w_j x_j \frac{I_j}{p_j}$
Compute $B = \frac{1}{n} \sum_{j=1}^{n} w_j \frac{I_j}{p_j}$
return $Z = \frac{A}{B}$

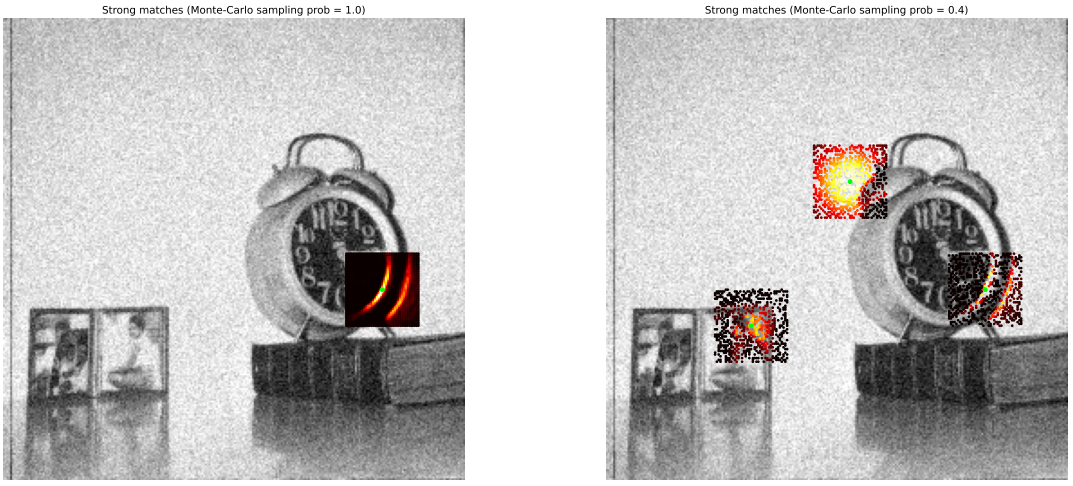
2.3. **Spatial Sampling.** We must also consider that locality matters in images. In a picture of a starry night, for example, a bright star might look like noise if compared to the rest of the dark sky. We can integrate this insight with the ingenious structural handling of MCNLM by incorporating a spatial distance parameter into the weighting function:

$$w_j = w_j^s \cdot w_j^r \tag{2.10}$$

where w_j^r is the weight mentioned at 1.7 and w_j^s is a spatial weight:

$$w_j^s = e^{-(d_2^j)^2/(2h_s^2)} \cdot \mathbb{I}\{d_\infty^j \leq \rho\} \tag{2.11}$$

where d_j^2 is the Euclidean distance between the noisy patch and the reference patch we compare to and d_j^∞ is the Chebyshev distance. The indicator function $\mathbb{I}$ is used to determine a spatial search window of width ρ.



(A) Weight values in search window (B) Weigths for sampled center pixels

FIGURE 1. Main caption describing both images

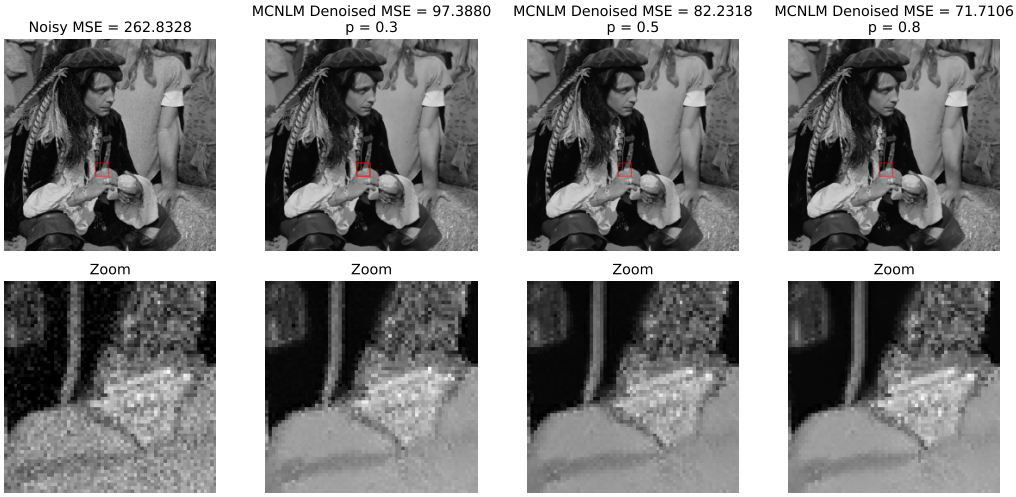


FIGURE 2. MCNLM with $\xi \in \{0.3, 0.5, 0.8\}$

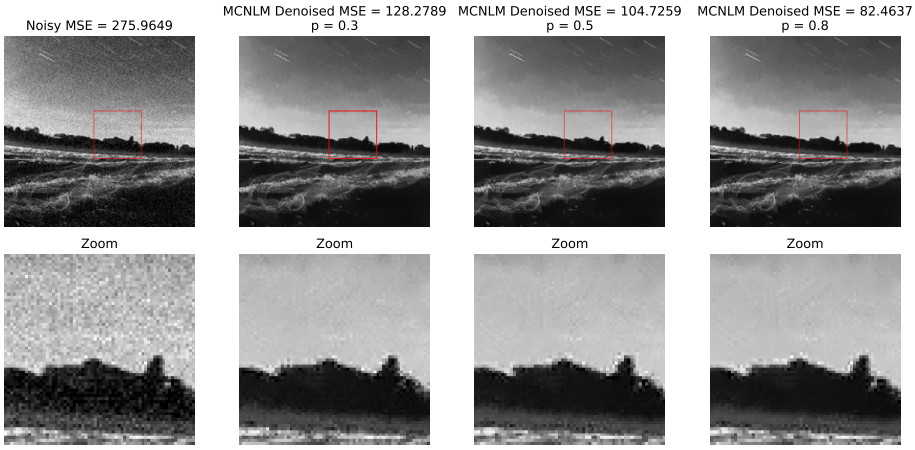


FIGURE 3. MCNLM with $\xi \in \{0.3, 0.5, 0.8\}$

Figure 4 visualizes the weight function within a search window centered on a noisy pixel, where brighter values indicate higher weight magnitudes. This illustrates how the method preserves structure: pixels with similar patches exert a stronger influence on the weighted average, ensuring that structural details are retained in the denoised output.

2.4. **Results.** In the figures 8, 3 we can see how the MCNLM algorithm behaves on an 1024x1024 image ($\approx 10^6$ pixels), with Gaussian noise with $\sigma = 17/255$ (taken into account that we normalize our pixels to have values in the interval $[0, 1]$)/ and zero mean. The comparison is done on 5x5 patches, and on a 20x20 search window arround the noisy pixel. The sampling pattern is chosen as uniform, with $\mathbf{p} = \{\xi, \cdots, \xi\}$. It is obvious that an increase in the sampling ratio only provides a marginally better result.

3. Stochastic Approximation of Non-Local Means

We yield the question whether the Monte-Carlo variant of our algorithm is indeeed reliable and aproaches the full Non-Local Means solution. For this we need to analyze the approximation error $|Z - z|$.

To understand our approximation, we will study the empirical sampling ratio S_n. The law of large numbers states that the empirical mean S_n of a large number of indepent random variable is close to the true mean.

3.1. **Probability Bounds.** For any error bound $\varepsilon > 0$ and any sampling pattern p with $0 < p_j <= 1$ and $\sum_{j=1}^{n} p_j = \varepsilon$ we want to study the probability:

$$\mathbb{P}(|S_n - \mathbb{E}[S_n]| > \varepsilon) \tag{3.1}$$

From the Cebyshev inequality we know that:

$$\mathbb{P}(|S_n - \mathbb{E}[S_n]| > \varepsilon) < \frac{Var[I_1]}{n\varepsilon^2}, \forall \varepsilon > 0 \tag{3.2}$$

This bound provided by the Cebyshev inequality above is too loose, only providing a linear descent of the error probability as $n \to \infty$.

We can use the Bernstein inequality to provide a much tighter exponential bound. The following is true for a sequence of n Bernoulli i.i.d. variables:

$$\mathbb{P}(|S_n - \mathbb{E}[S_n]| > \varepsilon) \leq \exp\left(-\frac{n\varepsilon^2}{2\left(\xi(1-\xi) + \varepsilon/3\right)}\right) \tag{3.3}$$

where $S_n = \frac{1}{n}\sum_{j=1}^{n} I_j$.

3.2. **General Error Probability Bound.** To rigorously quantify the reliability of the Monte Carlo approximation, we refer to the concentration bound derived in [5]. For a sample size n and an error tolerance $\varepsilon > 0$, the probability that the Monte Carlo estimate Z deviates from the exact NLM value z is bounded by:

$$\begin{aligned}\mathbb{P}(|Z - z| > \varepsilon) \leq & \exp\{-n\xi\} \\ & + 2\exp\left\{\frac{-n(\mu_B\varepsilon)^2}{2\left(\frac{1}{n}\sum_{j=1}^{n}\alpha_j^2\left(\frac{1-p_j}{p_j}\right) + \frac{(\mu_B\varepsilon)M_\alpha}{6}\right)}\right\}\end{aligned} \tag{3.4}$$

3.2.1. *Simplification for Uniform Sampling.* While Equation 3.4 accounts for variable sampling probabilities, our implementation utilizes a uniform sampling pattern where $p_j = p$ for all patches. Furthermore we have $\mu_B \leq 1$.

Under these experimental conditions, the bound simplifies significantly. Neglecting the trivial failure term $\exp\{-n\xi\}$ (which approaches zero for our window size of $n = 441$), the failure probability becomes:

$$\mathbb{P}(|Z - z| > \varepsilon) \leq 2\exp\left(\frac{-n\varepsilon^2}{2\left(\rho\sum_{j=1}^{n}\alpha_j^2 + \varepsilon/6\right)}\right) \tag{3.5}$$

where $n = 441$ is the total pixels in the 21×21 search window, $\rho = (1-p)/p$ is the sampling penalty factor and $\alpha_j = \frac{w_j}{\sum_{k=1}^{n} w_k}$ is the normalized weight of the patch j.

3.2.2. *Derivation of Numerical Bounds.* To obtain a concrete probability bound, we will replace the values of the variables in Equation 3.5 with the parameters from our experimental setup. The parameters are, as follows, $n = 441$, for a 21×21 search window, $\rho = (1 - 0.5)/0.5 = 1$. We select the desired error tolerance of $\varepsilon = 0.05$.

The remaining term, $\sum_{j=1}^{n} \alpha_j^2$, is non-trivial as it depends on the image content. We can, however, derive bounds for two limiting cases:
Case 1: Homogeneous Regions. In flat image regions (e.g., sky or smooth walls), the NLM weights are distributed nearly uniformly across the search window.

$$w_j \approx w \quad \forall j \implies \alpha_j \approx \frac{1}{n}$$

Substituting this into the variance term:

$$\sum_{j=1}^{n} \alpha_j^2 \approx \sum_{j=1}^{n} \left(\frac{1}{n}\right)^2 = n \cdot \frac{1}{n^2} = \frac{1}{441} \approx 0.0023$$

From 3.5, we calculate that the resulting probability is $\mathbb{P}(|Z - z| > \varepsilon) < 10^{-20}$, confirming that random sampling is statistically safe in smooth regions.
Case 2: Structured Regions. In textured regions or near edges, the weights concentrate on a small subset of patches that have a high similarity to the patch we want to denoise. We estimate this by defining an *effective neighbor count* k_{eff}. If the algorithm finds only $k_{\text{eff}} \approx 10$ strong matches in the window (a conservative estimate for detailed texture), the normalized weights approximate $\alpha_j \approx 1/10$ for the matches and 0 otherwise.

$$\sum_{j=1}^{n} \alpha_j^2 \approx \sum_{j=1}^{10} \left(\frac{1}{10}\right)^2 = 10 \cdot \frac{1}{100} = 0.1$$

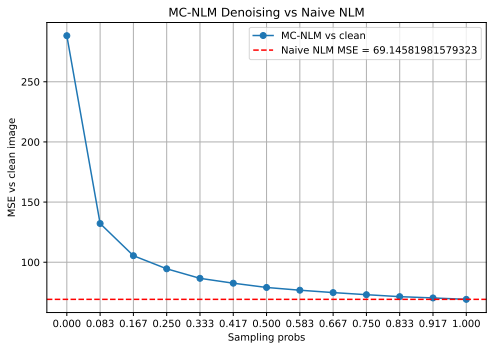
Using this estimate in 3.5:

$$\text{Exponent} = \frac{-441 \cdot (0.05)^2}{2(1 \cdot 0.1 + 0.05/6)} \approx \frac{-1.1025}{0.216} \approx -5.1$$

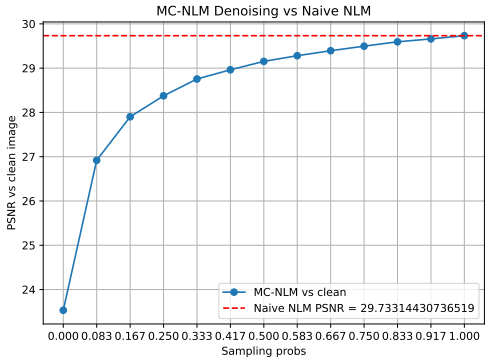
therefore

$$\mathbb{P}(|Z - z| > \varepsilon) \leq 2e^{-5.1} \approx 0.012$$

This shows that even in worst-case scenarios, the probability of the Monte Carlo estimate deviating by more than 5% is strictly bounded below 1.2%.



(A) MSE across different sampling probabilities



(B) PSNR across different sampling probabilities

FIGURE 4. Monte-carlo converge results

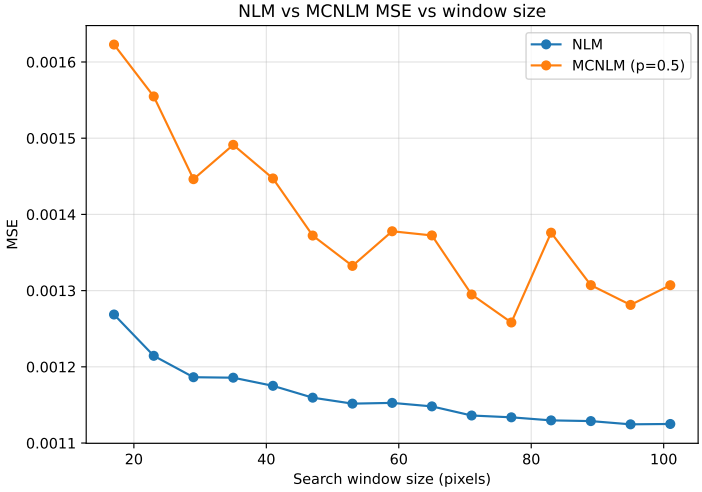


FIGURE 5. NLM vs MCNLM mse for different window sizes

4. KD-TREE ACCELERATED NON-LOCAL MEANS

This section describes a possible improvement of the basic Non-Local Means algorithm that would improve its theoretical time complexity by using a KD-Tree data structure to speed up the finding of similar patches, at the cost of slight accuracy loss. While the theoretical time complexity of this approach is better, the naive algorithm is laughably simple to parallelize, meanwhile this approach does have some bottlenecks in the query phase that are harder to do so; as such, the actual runtime improvement may be less than expected. However, it is still an interesting approach to explore and implement, and can be used as a middle ground between the naive NLM and more heuristic approaches like Monte Carlo NLM.

More complex approaches that are based on this idea have been presented by Adams et al. in the SIGGRAPH 2009 paper [1], that explores the idea of a *Gaussian KD-Tree*, and by Brox et al. [2] who introduced the notion of *Cluster*

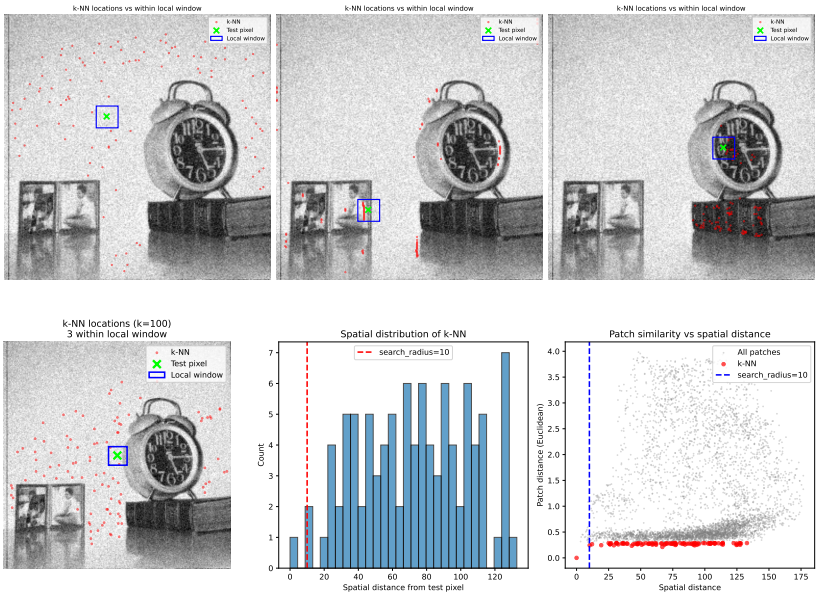


FIGURE 6. MCLNM vs KD-Tree NLM patch selection comparison

Trees. The approach proposed by Adams et al., is a more advanced data structure that combines KD-Trees with Gaussian Mixture Models to further speed up the search for similar patches. This approach is more difficult to implement, but can provide even better performance, since the original authors implemented this using a GPU with CUDA support.

4.1. **Algorithm Description.** Unlike the Monte Carlo NLM algorithm, which randomly samples patches from the search window, this approach builds a KD-Tree from all the patches in the search window, and then queries it for the K nearest neighbors of the current patch. This way, we can find the most similar patches more efficiently than by brute-force searching through all patches in the search window. More specifically, for each pixel p in the image, we extract its patch $B(p, f)$ and consider a point in $\mathbb{R}^{(2f+1)^2}$ corresponding to the flattened patch. This $(2f+1)^2$-dimensional point is then inserted into the KD-Tree. After building the KD-Tree for all patches in the search window, we can query it for the K nearest neighbors of the patch $B(p, f)$, and use these patches to compute the weights and denoise the pixel p. The weight computation and denoising steps remain the same as in the original NLM algorithm, but we only consider the K nearest neighbors instead of all patches in the search window. Below is a high-level pseudocode of the proposed idea.

For RGB images, we would need to also consider the color channels, resulting in points in $\mathbb{R}^{3(2f+1)^2}$. However, as before, we will focus on grayscale images for simplicity.

Algorithm 2 KD-Tree Accelerated Non-Local Means Denoising

Require: Noisy image Y, patch half-size f, filtering parameter h, number of neighbors K

Ensure: Denoised image Z

$\mathcal{P} \leftarrow$ Extract all patches $B(p, f)$ from image Y and flatten them into vectors in $\mathbb{R}^{(2f+1)^2}$
KD $\leftarrow$ BuildKDTree($\mathcal{P}$)
for each pixel p in image Y **do**
    $B_p \leftarrow$ Extract and flatten patch $B(p, f)$
    $\mathcal{N}_p \leftarrow$ KD.Query(B_p, K)
    $Z(p) \leftarrow 0$, $W \leftarrow 0$
    for each patch $B(q, f)$ in $\mathcal{N}_p$ **do**
        $d \leftarrow$ ComputeDistance(B_p, $B(q, f)$)
        $w \leftarrow \exp\left(-\frac{d^2}{h^2}\right)$
        $Z(p) \leftarrow Z(p) + w \cdot Y(q)$
        $W \leftarrow W + w$
    end for
    $Z(p) \leftarrow Z(p)/W$
end for
return Z

4.2. **Experimental Results.** We compared the two approaches (Monte Carlo NLM and KD-Tree NLM) with the `clock` image from the `SIPI Image Database`, corrupted with Gaussian noise with $\sigma = 17$. We can see that both approaches have very similar performance in terms of PSNR and MSE, even though they look pretty different. The Monte Carlo approach works very well with large uniform spaces, but struggles near rough edges (such as under the clock, or next to the numbers), while the KD-Tree approach can outperform it in these areas, but looks more noisy in large uniform areas. This is to be expected, since the Monte Carlo approach is more likely to be able to find good patches in large similar areas, and struggle in very dense, high-frequency ones. The KD-Tree approach, on the other hand, is not restricted by a fixed window search area, and can find good patches even in high-frequency areas, as long as they are present somewhere in the image. However, the noise in the background can be attributed to the fact that it's behaving too good — since there are many patches similar to the noisy one, it's likely to pick the ones with similar noise patterns, resulting in less effective denoising.

An idea for improvement would be to only use the KD-Tree approach in high-frequency areas, and use the Monte Carlo approach in low-frequency ones, to get the best of both worlds. Or, as another idea, we could build the KD-Tree locally for each search window and have a hybrid approach that uses both methods. However, due to time constraints, we were not able to explore these ideas further.

4.3. **Small Conclusion.** The KD-Tree Accelerated Non-Local Means algorithm is an interesting approach to improve the efficiency of the NLM algorithm using

FIGURE 7. Comparison between KD-Tree NLM and MC-NLM

the idea from Monte Carlo NLM and improving on it. It does not provide a significant improvement, especially considering the bottlenecks in parallelization which can be a problem for large images, but it's still a valid approach that can be explored further. The paper by Adams et al. [1] provides a more advanced approach that can be implemented for even better performance, adding new techniques to improve the speed of the algorithm and accuracy. It involves *Monte Carlo sampling* and *Splat-blur-slice* for faster search. It also solves issues such as the curse of dimensionality by using Principal Component Analysis (PCA) to reduce the dimensionality of the patches before inserting them into the KD-Tree. While this method was also tried in our implementation, it did not provide a significant improvement in performance or image quality.

5. Noise Estimation and Algorithm Implementation

In the above algorithm it is assumed that the noise standard deviation σ is known. However, in real-life applications this is not always the case. There are multiple methods for estimating this value, but we are going to focus on a simple one based on the Fast Fourier Transform. The idea is that, in the frequency domain, the high-frequency components are more likely to represent noise rather

than actual image details. By analyzing these components, we can estimate the noise level. A pseudo-code of the algorithm is given below.

Algorithm 3 Gaussian Noise Standard Deviation Estimation using FFT

Require: Input image
1: $F \leftarrow$ FFT(image)
2: Shift the zero-frequency component F to the center of the spectrum
3: Define a high-frequency mask M that selects the high-frequency components
4: $H \leftarrow F \cdot M$
5: $noise \leftarrow$ Inverse FFT Shift of H
6: Keep only the real part of $noise$ and remove the mean.
7: **return** $\sigma \leftarrow std(noise)$

While this obviously doesn't yield perfect results like already knowing the noise level, it provides a reasonable estimate that can be used in the denoising process. We can see that the estimated noise standard deviation is close to the actual value, which is sufficient for our denoising algorithm to perform effectively. However, the blurring effect can starts to be noticeable when we compare the two images side by side.

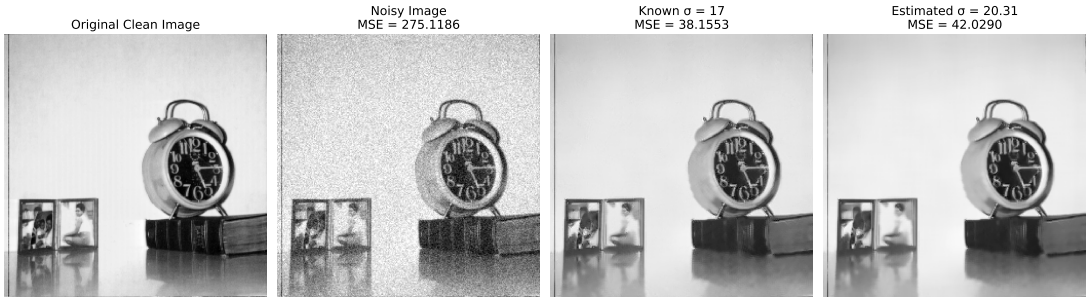


FIGURE 8. Comparison between using a known σ and an estimated one

6. Hashed Nonlocal Means

As we have seen so far, Nonlocal Means algorithms are computationally heavy. The main reason NLM is a "heavyweight" algorithm boils down to its exhaustive searching strategy. In a truly non-local naive NLM algorithm, we have to brute force through all pixels of the image. For every pixel, compare it against **every other pixel**. If N is the number complexity, we obtain $O(N^2)$ search complexity. NLM means works by selecting patches of size D x D. The total complexity of NLM is $O(N^2D^2)$.

However, when we move from theory to a practical implementation, the complexity is usually expressed relative to a search window, as we do in the Monte carlo algorithm and our naive implementation. Considering the search window size S x S, we obtain a total complexity of $O(NS^2D^2)$.

We want an algorithm that doesn't compromise the non-locality of NLM and is also reasonably fast. This is motivated by denoising 3D images (especially 3D medical images from MRIs and scans that contain noise), where the complexity of the naive NLM or other NLM methods is not fulfilling. This subject is considered in the paper titled "Hashed Nonlocal Means for Rapid Image Filtering" by Nicholas Dowson and Olivier Salvado [6], which addresses this exact problem of denoising 3D images. The algorithm and method they proposed can be well applied in our situation of denoising 2D grayscale images.

Nicholas Dowson and Olivier Salvado introduce a significantly faster way to remove noise from digital images. The core innovation is the transition from an exhaustive search-based filtering approach to a signal-processing approach using discretized frequency distributions (hash spaces).

6.1. **Standard NLM.** Standard NLM assumes that a pixel i should resemble other pixels j throughout the image that have similar surrounding intensity patches. The smoothed intensity $\hat{I}[x_i]$ is traditionally calculated as:

$$\hat{I}[x_i] = \frac{\sum_{j \in X} w(f_i, f_j) I[x_j]}{\sum_{j \in X} w(f_i, f_j)}$$

So, here for every pixel in the image, we ask how similar is this pixel to every other pixel in the image.

6.2. **From Global Summation to Feature Space Integration.** The classic NLM approach is slow, searching through every single pixel j in the image to see if its surrounding patch f_j looks like our current pixel's patch f_i.

To speed this up, we stop looking at where pixels are and start looking at what they look like. We do this by introducing a **feature space**. This features of this space are 4-dimensional vectors that contain the direct neighbours of our pixel (up, down, left, right). Let's consider now g, a generic neighbor pattern and an indicator function, $\delta(g)$. δ is similar to a dirac function: it only equals 1 when our generic pattern g exactly matches a pixel's actual pattern f_j.

By integrating over the entire Feature Space $\mathcal{F}$ (the set of all possible neighbor patterns), we can rewrite the smoothed intensity $\overline{I}[x_i]$ like this:

$$\overline{I}[x_i] = \frac{\sum_{j=1}^{|X|} \int_{g \in \mathcal{F}} w_K(f_i - g) \delta(g - f_j) I[x_j] dg}{\sum_{j=1}^{|X|} \int_{g \in \mathcal{F}} w_K(f_i - g) \delta(g - f_j) dg} \tag{6.1}$$

6.3. **Hash Spaces.** Let's consider:

$$H_1(g) = \sum_{j=1}^{|X|} \delta(g - f_j)$$

$$H_f(g) = \sum_{j=1}^{|X|} \delta(g - f_j) \cdot I[x_j]$$

- $H_1(g)$ is a **frequency map**. It counts how many times the pattern g appears in the whole image.

- $H_f(g)$ is an **intensity map**. It adds up all the intensity values for every pixel that shares the pattern g. By using these hash spaces, we no longer care where a pixel is located in the image. We only care about its descriptor f_i.

Since we are just adding and multiplying, we can swap the order of the sum and the integral in the formula at (6.1). The final denoising formula simplifies into a ratio of two integrals:

$$\tilde{I}_i = \frac{\int_{g\in\mathcal{F}} H_f(g) \cdot w_K(f_i - g)dg}{\int_{g\in\mathcal{F}} H_1(g) \cdot w_K(f_i - g)dg}$$

6.4. **Discretization.** Having defined the hash spaces above, our algorithm needs to store them. The problem is that our spaces are continous and we can't model them on the computer. The solution is to discretize those features and map them to bins. Every pixel's 4D descriptor f_j is hashed into a specific bin by normalizing with a smoothing factor and rounding its values. By mapping multiple pixels into the same bin, the algorithm effectively groups terms in the NLM summation.

6.5. **Smoothing factor.** The amount of smoothing is governed by the smoothing parameter h, which is calculated based on the estimated noise standard deviation σ in the image. The relationship is defined by the formula $h^2 = 2\sigma^2|P|\beta$, where $|P|$ represents the number of features in the patch descriptor. The tuning parameter β is included to refine the filter's performance; while some methods set β to one, experiments show that it should ideally decrease as the number of samples increases. Selecting an appropriate β is critical, as a value that is too large can lead to overblurring and the merging of separate image structures, while a value that is too small results in insufficient denoising.

6.6. **Weighting via Separable Convolution.** Once we have the frequecy hashes populated the algorithm must account for the similarity between different bins. In standard NLM, this is done by calculating weights between every pair of pixels, but in the hashed approach, this is achieved by convolving the hash spaces with the weighting kernel w_k.

The intuition behind this is that each bin contains pixels with specific neighbourhood patterns. We need to do averaging of similar pixels (this is what NLM does), so this is a convolution with the weight kernel, allowing pixels to be influenced by **nearby bins**. The transition from a disorganized collection of pixels to a structured 4D grid allows us to treat the NLM weighting process as a signal processing task by convolving.

Because the weighting kernel is axis-aligned (treating each neighbor intensity as independent), we can use **separable convolution**. Instead of one massive 4D operation, we perform four successive 1D convolutions, one along each axis of the 4D grid. The result of this step is two "blurred" hash spaces, H'_1 and H'_f, which now contain the weighted information from neighboring bins.

$$\tilde{I}_i = \frac{(w_K * H_f)(\mathbf{f}_i)}{(w_K * H_1)(\mathbf{f}_i)} = \frac{H'_f(\mathbf{f}_i)}{H'_1(\mathbf{f}_i)}.$$

6.7. **Reconstruction.** We reconstruct the denoised image using **direct nearest neighbor (NN)**. We divide H'_1 and H'_f to obtain our denoised pixel:

$$\tilde{I}_i = \frac{H'_f(f_i)}{H'_1(f_i)}$$

Direct NN mapping is the most computationally efficient, but least accurate approach. For every pixel x_i, the filtered intensity is derived by rounding the descriptor to the nearest integer coordinates and performing a direct lookup in the convolved hash spaces.

6.8. **Artifacts.** Nearest neighbour has significant drawbacks in terms of image quality. This discretization leads to the formation of stepping artifacts, which appear as blocky, homogeneous regions of intensity separated by sharp, artificial edges in areas that should vary smoothly. These artifacts take on a specific "star" appearance, because we are considering 4D features consisting of the neighbours of one pixel (which are "star" shaped: up, down, left, right).

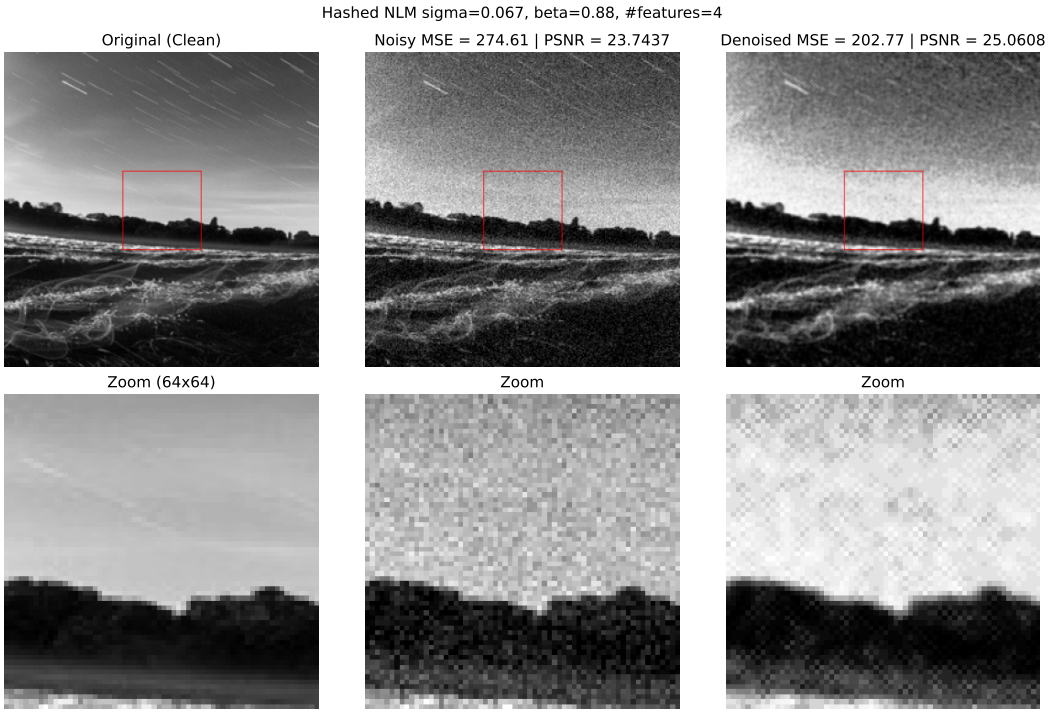


FIGURE 9. Hashed NLM Artifacts

6.9. **Reconstruction via Marginal-Linear Interpolation.** After convolving, we are left with two blurred hash spaces, H'_f and H'_1, which contain the weighted numerator and denominator for the NLM formula. However, because the neighbor intensities in our descriptor f_i are continuous values, they usually fall between the integer centers of our discrete bins. To reconstruct the final image without steep jumps caused by assigning pixels to the nearest single bin, we must interpolate.

Because interpolation in our 4D feature space is computationally expensive, the authors of the paper propose **marginal linear interpolation**. Marginal-Linear Interpolation only looks at the gradients along each of the 4 axes independently.

We take the value of the nearest bin center $H[\text{round}(f)]$ and add the slopes (gradients) of the data for each of the 4 dimensions:

$$H(f_i) \approx H[\text{round}(f_i)] + \sum_{k=1}^{4} (f_{ik} - \text{round}(f_{ik})) \frac{\partial H}{\partial f_{ik}}$$

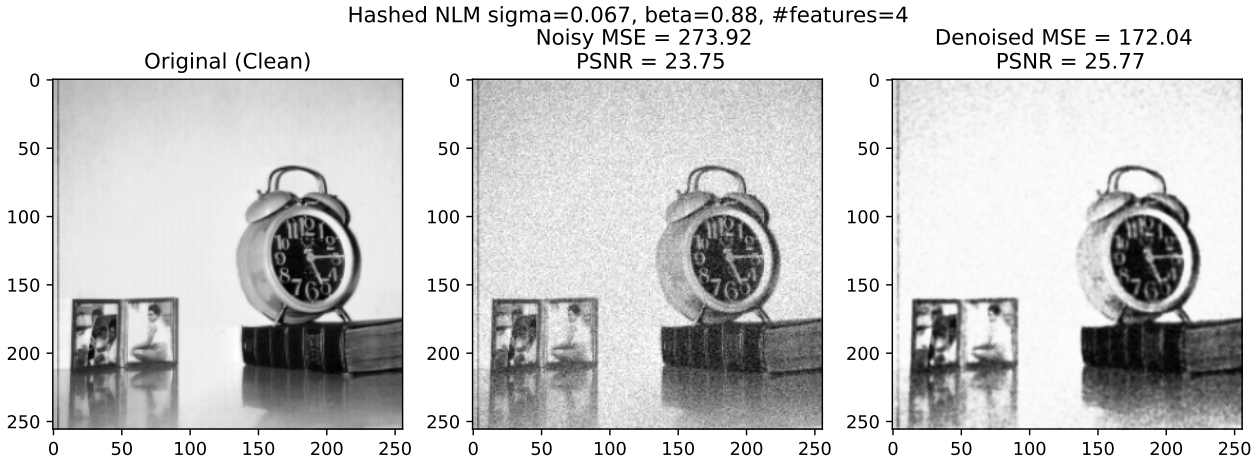


FIGURE 10. Hashed NLM Results

6.10. **Exploring different feature spaces.** The method described above focused on using 4 neighbours of the pixel. We can choose to work with higher dimensional spaces, which of course require more memory. For instance, we can select 8 neighbours, adding the diagonals (top-left, top-right, bottom-left, bottom-right) and apply the same algorithm. While in theory this should give better results, it is not often the case. This 8D space in the case of 2D images results in bins too sparse with too few values.

6.11. **Adding locality.** We see that this method is truly non-local, it is not like other NLM algorithms which are actually "semi-local". We are traversing the entire grid and assigning pixels to bins. Nonetheless, adding locality to NLM can yield good results. We see that our MCNLM uses this. Locality is relevant for denoising. For example, consider a background or a sky. Locality is not so advantageous in the case of edges.

How can locality be added to hashed NLM. Locality can be added by simply adding two more dimensions to the feature vector. This 2D vector, (x, y) that we use to extend our vector are the normalized coordinates of the pixel. In this fashion, the feature space is more complex, but a criterion for similarity between pixel is spatiality as well.

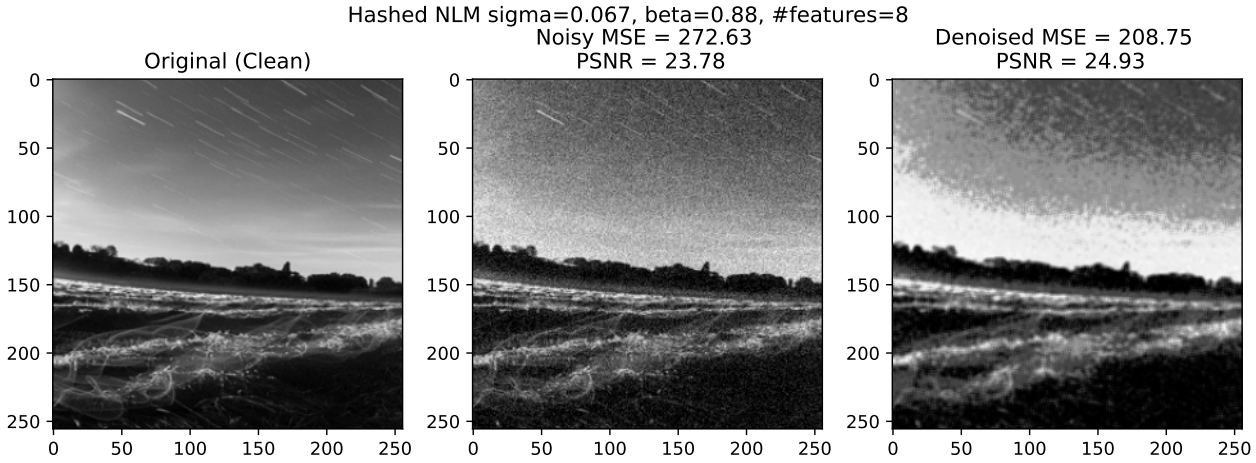


FIGURE 11. Hashed NLM Results With Larger Feature Space

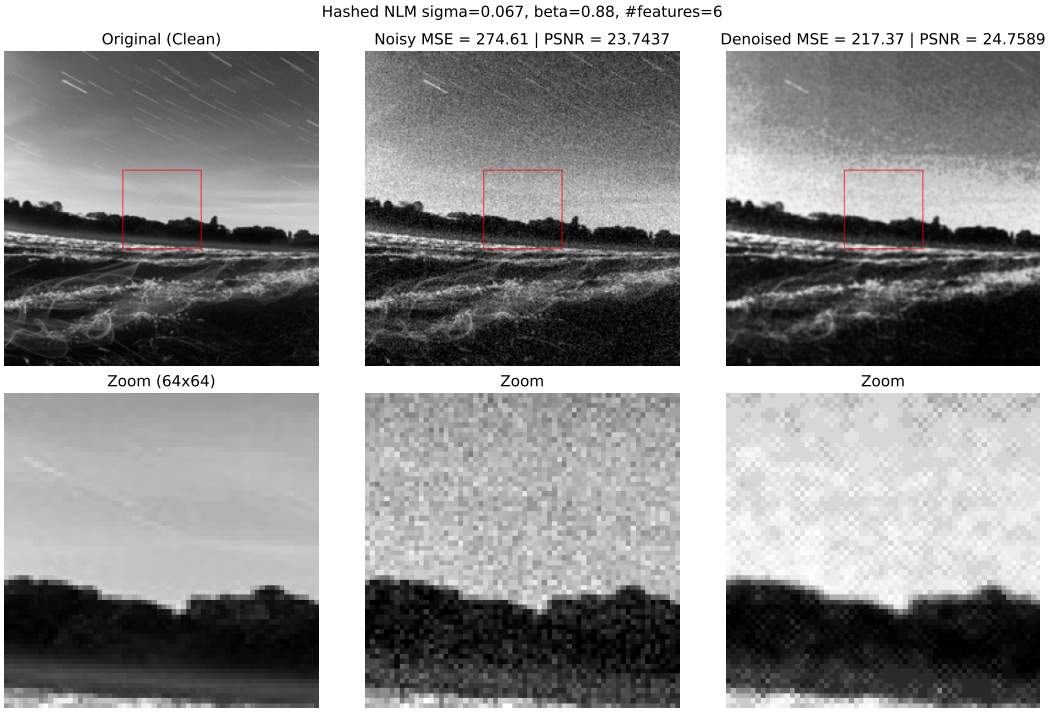


FIGURE 12. Hashed NLM Results with Locality

Algorithm 4 Hashed Non-Local Means Denoising

Require: Noisy image I, smoothing factor h, feature space dimension P
Ensure: Denoised image $\tilde{I}$

$h^2 \leftarrow 2\sigma^2|P|\beta$ ▷ Calculate smoothing parameter
$\mathcal{F} \leftarrow$ Create empty feature space in$|P|$-dimensional space
$H_1, H_f \leftarrow$ Initialize frequency and intensity hash spaces to zero
for each pixel $x \in I$ **do** ▷ Populate Hash Spaces
    $f \leftarrow$ Extract descriptor at x
    $g \leftarrow \text{round}(f/h)$ ▷ Map to discrete bin index
    $H_1[g] \leftarrow H_1[g] + 1$
    $H_f[g] \leftarrow H_f[g] + I[x]$
end for
$H_1' \leftarrow \text{SeparableConvolution}(H_1, w_K)$ ▷ Weighted sum of counts
$H_f' \leftarrow \text{SeparableConvolution}(H_f, w_K)$ ▷ Weighted sum of intensities
for each pixel $x \in I$ **do** ▷ Reconstruction
    $f \leftarrow$ Extract descriptor at x
    $\tilde{I}[x] \leftarrow H_1'/H_f'$
end for
return $\tilde{I}$

6.12. **Complexity.** Hashed NLM improves the standard NLM complexity of $O(N^2D^2)$ or $O(NS^2D^2)$. The complexity is essentially linear with respect to the number of pixels:

- Hashing: $O(n)$
- Convolution: $O(PKB)$ - performs P 1D convolution across B bins with a kernel size of K.
- Reconstruction: $O(n)$

References

1. Andrew Adams, Natasha Gelfand, Jennifer Dolson, and Marc Levoy, *Gaussian kd-trees for fast high-dimensional filtering*, ACM SIGGRAPH 2009 papers, 2009, pp. 1–12.
2. Thomas Brox, Oliver Kleinschmidt, and Daniel Cremers, *Efficient nonlocal means for denoising of textural patterns*, IEEE Transactions on Image Processing **17** (2008), no. 7, 1083–1092.
3. A. Buades, B. Coll, and J.-M. Morel, *A non-local algorithm for image denoising*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) (San Diego, CA, USA), vol. 2, IEEE, 2005, pp. 60–65.
4. Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, *Non-Local Means Denoising*, Image Processing On Line **1** (2011), 208–212, https://doi.org/10.5201/ipol.2011.bcm_nlm.
5. Stanley H. Chan, Todd Zickler, and Yue M. Lu, *Monte carlo non-local means: Random sampling for large-scale image filtering*, IEEE Transactions on Image Processing **23** (2014), no. 8, 3711–3725.
6. Nicholas Dowson and Olivier Salvado, *Hashed nonlocal means for rapid image filtering*, IEEE Transactions on Pattern Analysis and Machine Intelligence **33** (2011), no. 3, 485–499.
7. Jacques Froment, *Parameter-Free Fast Pixelwise Non-Local Means Denoising*, Image Processing On Line **4** (2014), 300–326, https://doi.org/10.5201/ipol.2014.120.
8. Simon Postec, Jacques Froment, and Béatrice Vedel, *Non-local means est un algorithme de d\'ebruitage local (non-local means is a local image denoising algorithm)*, arXiv preprint arXiv:1311.3768 (2013).
9. Chunwei Tian, Menghua Zheng, Wangmeng Zuo, Bob Zhang, Yanning Zhang, and David Zhang, *Multi-stage image denoising with the wavelet transform*, Pattern Recognition **134** (2023), 109050.
10. Faiz Ullah, Kamlesh Kumar, Tariq Rahim, Jawad Khan, and Younhyun Jung, *A new hybrid image denoising algorithm using adaptive and modified decision-based filters for enhanced image quality*, Scientific Reports **15** (2025), no. 1, 8971.