Self-Similarity Based Image Denoising: Theory and Applications

Advanced Image Processing Lecture Notes

July 15, 2025

Abstract

This document presents a comprehensive treatment of self-similarity based image denoising methods, focusing on non-local means and Block-Matching 3D (BM3D) algorithms. We explore the theoretical foundations, mathematical formulations, and practical implementation details of these state-of-the-art denoising techniques. The notes emphasize the exploitation of natural image statistics through patch-based similarity measures and collaborative filtering approaches.

Contents

1 Introduction to Self-Similarity in Image Processing

1.1 Mathematical Framework for Image Denoising

The image denoising problem can be formulated as follows. Given a noisy observation y, we seek to recover the underlying clean image x from the degraded measurement:

$$y = x + \eta \tag{1}$$

where η represents additive white Gaussian noise with zero mean and known variance σ^2:

$$\eta \sim \mathcal{N}(0, \sigma^2 I) \tag{2}$$

Limitations of Simple Averaging Consider the classical approach of estimating pixel intensities through local averaging. For a pixel at location (i, j), a naive estimate would be:

$$\hat{x}_{i,j} = \frac{1}{|U_{i,j}|} \sum_{(k,l)\in U_{i,j}} y_{k,l} \tag{3}$$

where $U_{i,j}$ denotes a neighborhood around pixel (i, j), and $|U_{i,j}|$ is its cardinality.

Remark 1.1. *This approach is equivalent to convolution with a normalized box kernel:*

$$\hat{x} = y * h \tag{4}$$

where h is the box kernel with support $U_{i,j}$.

1.2 Fundamental Limitations of Local Smoothing

The primary limitation of local averaging becomes apparent near image discontinuities. Consider an ideal edge scenario where:

$$x_{i,j} = \begin{cases} a & \text{if } (i,j) \in \Omega_1 \\ b & \text{if } (i,j) \in \Omega_2 \end{cases} \tag{5}$$

where Ω_1 and Ω_2 are disjoint regions separated by an edge, and $a \neq b$.

Edge Degradation Analysis For a pixel (i, j) near the edge boundary, the local average becomes:

$$\hat{x}_{i,j} = \frac{1}{|U_{i,j}|} \left(\sum_{(k,l)\in U_{i,j}\cap\Omega_1} y_{k,l} + \sum_{(k,l)\in U_{i,j}\cap\Omega_2} y_{k,l} \right) \tag{6}$$

$$\approx \frac{|\Omega_1 \cap U_{i,j}|}{|U_{i,j}|} a + \frac{|\Omega_2 \cap U_{i,j}|}{|U_{i,j}|} b \tag{7}$$

This results in a blurred edge transition, motivating the need for adaptive filtering strategies.

2 Non-Local Means Algorithm

2.1 Theoretical Foundation

The Non-Local Means (NLM) algorithm addresses the limitations of local averaging by exploiting the *self-similarity* property of natural images. The key insight is that similar patches exist throughout the image, not just in local neighborhoods.

Definition 2.1 (Self-Similarity Prior). *For a natural image x, there exist multiple patches $\{P_k\}$ such that:*

$$\|P_i - P_j\|_2 < \epsilon \tag{8}$$

for some small threshold $\epsilon > 0$, where P_k represents a patch extracted from the image.

2.2 Mathematical Formulation

The NLM estimate for pixel (i, j) is given by:

$$\hat{x}_{i,j} = \sum_{(k,l)\in S_{i,j}} w_{i,j}(k, l) \cdot y_{k,l} \tag{9}$$

where $S_{i,j}$ represents the search window around pixel (i, j), and $w_{i,j}(k, l)$ are the similarity weights.

Weight Computation The similarity weights are computed based on patch distances:

$$w_{i,j}(k, l) = \frac{1}{Z_{i,j}} \exp\left(-\frac{d^2(P_{i,j}, P_{k,l})}{h^2}\right) \tag{10}$$

where:

- $P_{i,j}$ and $P_{k,l}$ are patches centered at (i, j) and (k, l) respectively
- $d^2(P_{i,j}, P_{k,l})$ is the squared Euclidean distance between patches
- h is the filtering parameter controlling the decay rate
- $Z_{i,j}$ is the normalization constant ensuring $\sum w_{i,j}(k, l) = 1$

2.3 Patch Distance Computation

The patch distance is computed as:

$$d^2(P_{i,j}, P_{k,l}) = \frac{1}{|P|} \sum_{(u,v)\in P} |y_{i+u,j+v} - y_{k+u,l+v}|^2 \tag{11}$$

where P represents the patch domain and $|P|$ is the number of pixels in the patch.

Noise-Aware Distance In the presence of noise, the distance can be corrected as:

$$d^2_{\text{corrected}}(P_{i,j}, P_{k,l}) = \max\left(d^2(P_{i,j}, P_{k,l}) - 2\sigma^2, 0\right) \tag{12}$$

This correction accounts for the noise contribution to the patch distance.

2.4 Normalization and Properties

The normalization constant is given by:

$$Z_{i,j} = \sum_{(k,l)\in S_{i,j}} \exp\left(-\frac{d^2(P_{i,j}, P_{k,l})}{h^2}\right) \tag{13}$$

Theorem 2.2 (NLM Consistency)**.** *For a noiseless image, the NLM estimate satisfies:*

$$\lim_{\sigma\to 0} \hat{x}_{i,j} = x_{i,j} \tag{14}$$

Proof Sketch. As $\sigma \to 0$, the patch distances approach their true values, and the weight distribution becomes increasingly concentrated around patches identical to the reference patch. □

3 Implementation Details and Practical Considerations

3.1 Algorithmic Framework

Algorithm 1 Non-Local Means Algorithm

```
Require: Noisy image $y$, patch size $p$, search window size $s$, filtering parameter $h$
Ensure: Denoised image $\hat{x}$
1: for each pixel $(i, j)$ in the image do
2:     Initialize weight matrix $W = 0$
3:     Define search window $S_{i,j}$ of size $s \times s$
4:     for each pixel $(k, l)$ in $S_{i,j}$ do
5:         Extract patches $P_{i,j}$ and $P_{k,l}$ of size $p \times p$
6:         Compute distance $d^2(P_{i,j}, P_{k,l})$ using Eq. 11
7:         Compute weight $w_{i,j}(k, l)$ using Eq. 10
8:         $W(k, l) = w_{i,j}(k, l)$
9:     end for
10:    Normalize weights: $W = W / \sum W$
11:    Compute estimate: $\hat{x}_{i,j} = \sum_{(k,l)} W(k, l) \cdot y_{k,l}$
12: end for
```

3.2 Boundary Handling

Near image boundaries, patches may extend beyond the image domain. Common strategies include:

1. **Symmetric Padding**: Reflect image content across boundaries
2. **Periodic Padding**: Wrap image content periodically
3. **Zero Padding**: Extend with zeros (not recommended)

The symmetric padding approach is preferred as it maintains image statistics:

$$\tilde{x}_{i,j} = \begin{cases} x_{i,j} & \text{if } (i, j) \text{ is inside image} \\ x_{2N-i,j} & \text{if } i > N \text{ (bottom boundary)} \\ x_{i,2M-j} & \text{if } j > M \text{ (right boundary)} \end{cases} \tag{15}$$

3.3 Parameter Selection

Typical Parameter Values

- Patch size: $p = 7 \times 7$ (provides good balance between detail and computational cost)
- Search window: $s = 21 \times 21$ (sufficient for finding similar patches)
- Filtering parameter: $h = 10\sigma$ (empirically determined)

Adaptive Parameter Selection The filtering parameter can be adapted based on local image characteristics:

$$h_{i,j} = \alpha \cdot \sigma \cdot \sqrt{\text{LocalVariance}(P_{i,j})} \tag{16}$$

where α is a scaling factor and LocalVariance measures local image activity.

4 Block-Matching 3D (BM3D) Algorithm

4.1 Overview and Motivation

The Block-Matching 3D (BM3D) algorithm extends the self-similarity concept by combining it with sparsity-based denoising. The key innovations include:

1. **Grouping**: Collect similar patches into 3D arrays
2. **Collaborative Filtering**: Process groups jointly using 3D transforms
3. **Aggregation**: Combine processed patches back into the image

4.2 Mathematical Framework

Patch Grouping For a reference patch $P_{i,j}$, we define the group $G_{i,j}$ as:

$$G_{i,j} = \{P_{k,l} : d^2(P_{i,j}, P_{k,l}) < \tau\} \tag{17}$$

where τ is a similarity threshold.

3D Array Construction The group is arranged as a 3D array $\mathcal{G} \in \mathbb{R}^{p\times p\times K}$, where $K = |G_{i,j}|$ is the number of patches in the group.

4.3 3D Transform Domain Processing

The BM3D algorithm applies a 3D transform to exploit both spatial and inter-patch correlations:

$$\mathcal{T} = \mathcal{W}_{3D} \cdot \mathcal{G} \tag{18}$$

where $\mathcal{W}_{3D}$ represents the 3D transform operator.

Separable Transform The 3D transform is typically implemented as a separable transformation:

$$\begin{aligned} \mathcal{T} &= \mathcal{W}_1 \cdot (\mathcal{W}_2 \cdot (\mathcal{W}_3 \cdot \mathcal{G})) && (19) \\ &= (\mathcal{W}_1 \otimes \mathcal{W}_2 \otimes \mathcal{W}_3) \cdot \mathcal{G} && (20) \end{aligned}$$

where $\mathcal{W}_1$ and $\mathcal{W}_2$ are 2D transforms (e.g., DCT) applied to each patch, and $\mathcal{W}_3$ is a 1D transform applied across the grouping dimension.

4.4 Hard Thresholding Stage

In the first stage, BM3D applies hard thresholding to the transform coefficients:

$$\hat{\mathcal{T}} = \mathcal{H}_\lambda(\mathcal{T}) \tag{21}$$

where the hard thresholding operator is defined as:

$$\mathcal{H}_\lambda(t) = \begin{cases} t & \text{if } |t| > \lambda \\ 0 & \text{if } |t| \leq \lambda \end{cases} \tag{22}$$

Threshold Selection The threshold is typically chosen as:

$$\lambda = \beta \cdot \sigma \tag{23}$$

where β is a parameter controlling the aggressiveness of denoising (typically $\beta = 2.7$).

4.5 Collaborative Filtering Stage

The second stage performs collaborative filtering using both the noisy and first-stage estimates:

$$\hat{\mathcal{T}}^{(2)} = \frac{|\mathcal{T}^{(1)}|^2}{|\mathcal{T}^{(1)}|^2 + \sigma^2} \cdot \mathcal{T}^{(0)} \tag{24}$$

where:

- $\mathcal{T}^{(0)}$ represents the transform of the noisy group
- $\mathcal{T}^{(1)}$ represents the transform of the first-stage estimate
- $|\cdot|^2$ denotes element-wise squared magnitude

Remark 4.1. *This is a Wiener filter formulation that optimally combines the noisy observations with the first-stage estimate based on their respective reliabilities.*

4.6 Aggregation and Weighting

After processing, patches must be aggregated back into the image. Due to overlapping patches, multiple estimates exist for each pixel:

$$\hat{x}_{i,j} = \frac{\sum_{G \ni (i,j)} w_G \cdot \hat{x}_{i,j}^{(G)}}{\sum_{G \ni (i,j)} w_G} \tag{25}$$

where the sum is over all groups G containing pixel (i,j).

Sparsity-Aware Weighting The weights are chosen based on the sparsity of the processed group:

$$w_G = \frac{1}{\|\mathcal{T}_G\|_0} \tag{26}$$

where $\|\cdot\|_0$ denotes the number of non-zero coefficients.

5 Implementation Guidelines and Parameter Tuning

5.1 Computational Complexity Analysis

Non-Local Means Complexity The computational complexity of NLM is $O(N^2 \cdot s^2 \cdot p^2)$ where:

- N^2 is the number of pixels in the image
- s^2 is the search window size
- p^2 is the patch size

BM3D Complexity BM3D has complexity $O(N^2 \cdot s^2 \cdot p^2 \cdot \log(K))$ where K is the average group size.

5.2 Optimization Strategies

Fast Patch Matching

1. **Integral Images**: Use integral images for rapid patch norm computation
2. **Early Termination**: Stop patch comparison when distance exceeds threshold
3. **Hierarchical Search**: Use coarse-to-fine matching strategies

Memory Optimization

- Process image in overlapping blocks to reduce memory requirements
- Use sliding window techniques for patch extraction
- Implement in-place transformations where possible

5.3 Quality Assessment Metrics

Objective Metrics

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\mathrm{MSE}} \tag{27}$$

$$\mathrm{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{28}$$

where MSE is the mean squared error and SSIM parameters are computed from local statistics.

6 Advanced Topics and Extensions

6.1 Adaptive Patch Sizes

Recent research has explored adaptive patch sizing based on local image characteristics:

$$p_{i,j} = f(\text{LocalComplexity}(i, j)) \tag{29}$$

where f is a mapping from complexity measures to patch sizes.

6.2 Multi-Scale Processing

Multi-scale approaches decompose the image into different resolution levels:

$$\hat{x} = \sum_{l=0}^{L} \mathcal{U}_l(\mathcal{D}_l(\hat{x}_l)) \tag{30}$$

where $\mathcal{D}_l$ and $\mathcal{U}_l$ are downsampling and upsampling operators.

6.3 Learning-Based Enhancements

Modern approaches incorporate learned components:

- **Learned Similarity Metrics**: Replace Euclidean distance with learned metrics
- **Adaptive Thresholding**: Learn threshold functions from data
- **Neural Patch Matching**: Use neural networks for patch similarity assessment

6.4 Applications Beyond Denoising

Self-similarity principles extend to various image processing tasks:

1. **Image Inpainting**: Fill missing regions using similar patches
2. **Super-Resolution**: Enhance resolution using patch recurrence
3. **Compression**: Exploit redundancy for efficient encoding
4. **Deblurring**: Combine with motion models for blur removal

7 Mathematical Notation and Symbol Glossary

Symbol	Definition
x	Clean (noise-free) image
y	Noisy observed image
$\hat{x}$	Denoised image estimate
η	Additive noise
σ^2	Noise variance
$P_{i,j}$	Image patch centered at (i,j)
$U_{i,j}$	Local neighborhood around (i,j)
$S_{i,j}$	Search window around (i,j)
$w_{i,j}(k,l)$	Similarity weight between pixels
$d^2(\cdot,\cdot)$	Squared patch distance
h	Filtering parameter
$G_{i,j}$	Group of similar patches
$\mathcal{T}$	Transform domain representation
$\mathcal{W}_{3D}$	3D transform operator
λ	Threshold parameter
$\mathcal{H}_\lambda$	Hard thresholding operator
$\|\cdot\|_0$	Number of non-zero elements
$\|\cdot\|_2$	Euclidean norm

Table 1: Mathematical notation used throughout the document

8 Conclusion

Self-similarity based denoising algorithms represent a fundamental paradigm shift in image processing, moving from local operations to global patch-based methods. The Non-Local Means algorithm introduced the concept of exploiting patch recurrence, while BM3D extended this through collaborative filtering in transform domains.

Key Contributions

- Exploitation of natural image statistics through self-similarity
- Adaptive filtering based on patch-wise similarity measures
- Collaborative processing of similar image structures
- State-of-the-art denoising performance across various noise levels

Future Directions Current research focuses on:

1. Integration with deep learning architectures
2. Real-time implementation strategies
3. Extension to video and volumetric data
4. Adaptive parameter selection mechanisms

The principles established by these algorithms continue to influence modern image processing and computer vision applications, providing a foundation for numerous advanced techniques.

9 Appendix: Detailed Algorithm Implementations

9.1 Non-Local Means Pseudocode

9.2 BM3D Implementation Framework

The BM3D implementation involves several key components:

Stage 1: Hard Thresholding

```
1: function BM3D_Stage1($y$, $\sigma$, $\tau_1$)
2: for each reference patch location $(i, j)$ do
3:     $G \leftarrow$ FindSimilarPatches$(y, i, j, \tau_1)$
4:     $\mathcal{T} \leftarrow$ Apply3DTransform$(G)$
5:     $\hat{\mathcal{T}} \leftarrow$ HardThreshold$(\mathcal{T}, 2.7\sigma)$
6:     $\hat{G} \leftarrow$ Inverse3DTransform$(\hat{\mathcal{T}})$
7:     AggregatePatches$(\hat{G}, \text{weights})$
8: end for
9: return $\hat{x}^{(1)}$
```

Stage 2: Collaborative Filtering

```
1: function BM3D_Stage2($y$, $\hat{x}^{(1)}$, $\sigma$, $\tau_2$)
2: for each reference patch location $(i, j)$ do
3:     $G_{\text{noisy}} \leftarrow$ FindSimilarPatches$(y, i, j, \tau_2, \hat{x}^{(1)})$
4:     $G_{\text{basic}} \leftarrow$ ExtractCorrespondingPatches$(\hat{x}^{(1)}, G_{\text{noisy}})$
5:     $\mathcal{T}_{\text{noisy}} \leftarrow$ Apply3DTransform$(G_{\text{noisy}})$
6:     $\mathcal{T}_{\text{basic}} \leftarrow$ Apply3DTransform$(G_{\text{basic}})$
7:     $\hat{\mathcal{T}} \leftarrow$ WienerFilter$(\mathcal{T}_{\text{noisy}}, \mathcal{T}_{\text{basic}}, \sigma)$
8:     $\hat{G} \leftarrow$ Inverse3DTransform$(\hat{\mathcal{T}})$
9:     AggregatePatches$(\hat{G}, \text{weights})$
10: end for
11: return $\hat{x}^{(2)}$
```

Algorithm 2 Detailed Non-Local Means Implementation

```
1: Input: Noisy image $y$ of size $M \times N$, patch size $p$, search size $s$, filtering parameter $h$
2: Output: Denoised image $\hat{x}$
3:
4: // Pad image for boundary handling
5: $\tilde{y} \leftarrow \text{SymmetricPad}(y, p/2)$
6:
7: for $i = 1$ to $M$ do
8:    for $j = 1$ to $N$ do
9:       weightSum $\leftarrow 0$
10:      pixelSum $\leftarrow 0$
11:
12:      // Define search window
13:      $i_{\min} \leftarrow \max(1, i - s/2)$
14:      $i_{\max} \leftarrow \min(M, i + s/2)$
15:      $j_{\min} \leftarrow \max(1, j - s/2)$
16:      $j_{\max} \leftarrow \min(N, j + s/2)$
17:
18:      for $k = i_{\min}$ to $i_{\max}$ do
19:         for $l = j_{\min}$ to $j_{\max}$ do
20:            // Extract patches
21:            $P_{ref} \leftarrow \text{ExtractPatch}(\tilde{y}, i, j, p)$
22:            $P_{cur} \leftarrow \text{ExtractPatch}(\tilde{y}, k, l, p)$
23:
24:            // Compute patch distance
25:            $d^2 \leftarrow \frac{1}{p^2} \sum_{u,v} |P_{ref}(u,v) - P_{cur}(u,v)|^2$
26:            $d^2 \leftarrow \max(d^2 - 2\sigma^2, 0)$ // Noise correction
27:
28:            // Compute weight
29:            $w \leftarrow \exp(-d^2/h^2)$
30:            weightSum $\leftarrow$ weightSum $+ w$
31:            pixelSum $\leftarrow$ pixelSum $+ w \cdot y_{k,l}$
32:         end for
33:      end for
34:
35:      // Normalize and store result
36:      $\hat{x}_{i,j} \leftarrow$ pixelSum/weightSum
37:    end for
38: end for
```