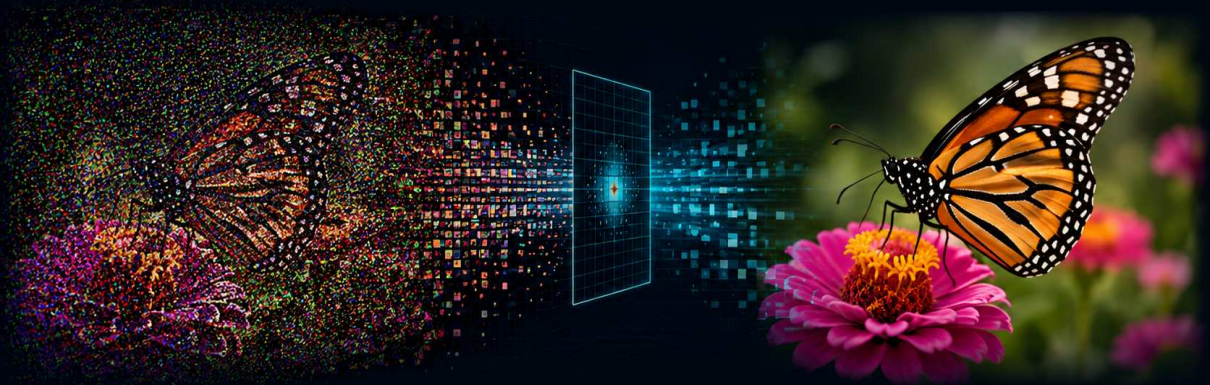
Chapter 2: Filtering and Denoising

From noisy observations to clean images



Histogram Equalization

From low contrast to expanded dynamic range

Original low-contrast image

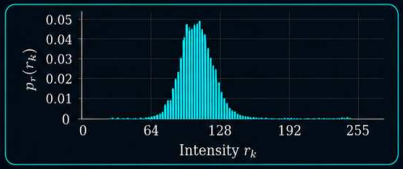
input image

redistribute intensities

Equalized image

output image

input histogram



output histogram

Goal: use the full dynamic range to enhance visual contrast.

$$s = T(r)$$

$$T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

r_k : input intensity level k

s : output intensity level

$p_r(r_j)$: normalized histogram (PDF)

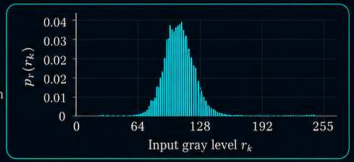
L : number of intensity levels (e.g., 256)

- Histogram equalization redistributes intensities using the cumulative distribution to maximize contrast.

Histogram Equalization — Step 3

Apply the mapping and reconstruct the image

1 Original (low-contrast) image



2 Lookup table / remapping

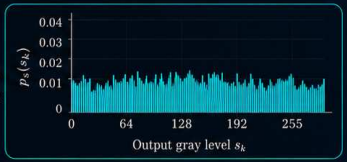
$$s_k = T(r_k)$$

Map each input level r_k to output level s_k

Input level r_k		Output level $s_k = T(r_k)$
0	→	0
32	→	15
64	→	48
96	→	92
128	→	128
160	→	170
192	→	206
224	→	236
255	→	255

Each pixel intensity is replaced by its mapped value from the cumulative distribution.

3 Equalized (enhanced) image



Before vs After

The equalized image uses a wider tonal range (from dark to bright) and reveals more detail in both shadows and highlights.

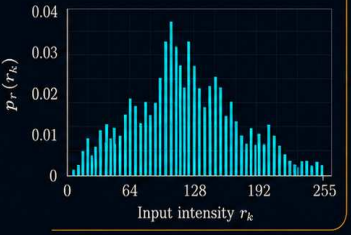
Step 3: remap every pixel through $T(r_k)$ and form the enhanced image.

Histogram Equalization — Step 2

Build the cumulative distribution and transformation

1) Normalized Histogram

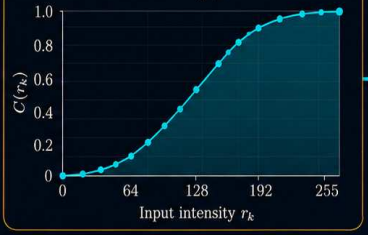
$p_r(r_k)$



Normalized histogram (PDF)
area under the bars = 1

2) Cumulative Distribution (CDF)

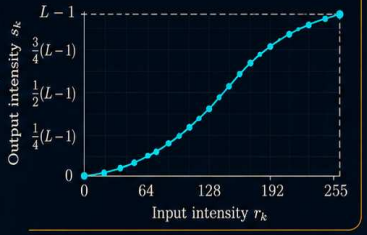
$C(r_k) = \sum_{j=0}^{k} p_r(r_j)$



Cumulative histogram (CDF)
monotonic increase from 0 to 1

3) Mapping Function

$s_k = T(r_k)$



Maps r_k to s_k to redistribute intensities across the full dynamic range $[0, L-1]$

Cumulative distribution function (CDF) accumulates probability up to level k.
This CDF becomes the remapping function.

$$s = T(r) \qquad T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

- r_k : input intensity level
- s_k : output intensity level
- L : number of gray levels

Step 2: accumulate the normalized histogram; the cumulative distribution becomes the intensity remapping function.

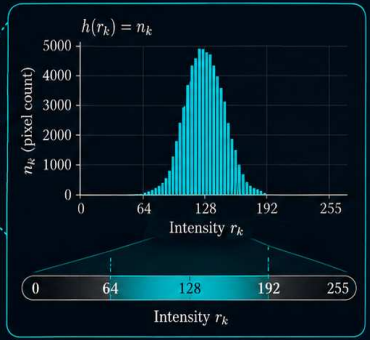
Histogram Equalization — Step 1

Measure the histogram and normalize it

1 Input grayscale image

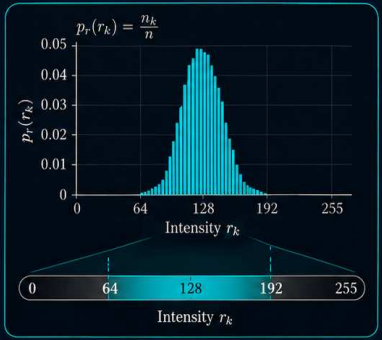
2 Compute discrete histogram

$h(r_k) = n_k$



3 Normalize to obtain PMF

$p_r(r_k) = \frac{n_k}{n}$



Low contrast images have intensities concentrated in a narrow range.

r_k : intensity level k, $k = 0, 1, \ldots, L-1$

n_k : number of pixels at intensity r_k

n : total number of pixels, $n = \sum_{k=0}^{L-1} n_k$

$h(r_k) = n_k$ (histogram)

$p_r(r_k) = \frac{n_k}{n}$ (normalized histogram / PMF)

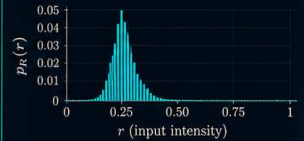
$\sum_{k=0}^{L-1} p_r(r_k) = 1$

Step 1: count pixels at each intensity, then normalize to obtain a probability mass function.

Why Histogram Equalization Works

Transformation of Random Variables

1 Input gray level



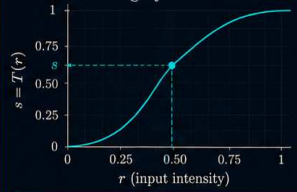
Let $R \in [0, 1]$ be the input gray level with pdf $p_R(r)$.

Low contrast → histogram clustered in a small range.

2 CDF mapping

$$S = T(R) = \int_0^R p_R(w)\, dw$$

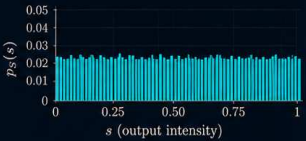
Use the cumulative distribution function as the gray-level transform.



$s = T(r)$ is monotone increasing.

$$\begin{aligned} p_S(s) &= p_R(r) \left| \frac{dr}{ds} \right| \\ &= \frac{p_R(r)}{ds/dr} \qquad \text{(change of variables)} \\ &= \frac{p_R(r)}{p_R(r)} = 1, \quad 0 \le s \le 1 \quad \left(\text{since } \frac{ds}{dr} = p_R(r)\right) \end{aligned}$$

3 Output gray level



Therefore S is uniformly distributed on $[0, 1]$.

Contrast is enhanced because gray levels occupy the full dynamic range.

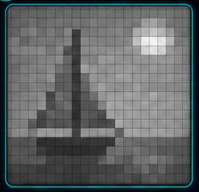
Key idea: histogram equalization is a *CDF-based transform* that maps the original gray-level random variable into an output variable that is (ideally) uniform.

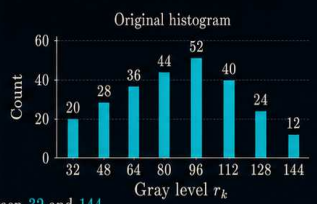
⚠ In discrete digital images, the equalized histogram is only *approximately* uniform because of quantization and finite bin counts.

Histogram Equalization: 16×16 Worked Example

Step-by-step equalization of a low-contrast image

1. Original 16×16 image





Gray levels clustered between 32 and 144

2. Compute PDF and CDF

$M = N = 16, \quad MN = 256$

$$p_R(r_k) = \frac{n_k}{256}$$

$$C_R(r_k) = \sum_{j=0}^{k} p_R(r_j)$$

$$s_k = T(r_k) = \text{round}\left((L-1)\,C_R(r_k)\right), \quad L = 256$$

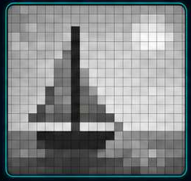
r_k	n_k	$p_R(r_k)$	$C_R(r_k)$	s_k
32	20	0.078	0.078	20
48	28	0.109	0.188	48
64	36	0.141	0.328	84
80	44	0.172	0.500	128
96	52	0.203	0.703	179
112	40	0.156	0.859	219
128	24	0.094	0.953	243
144	12	0.047	1.000	255

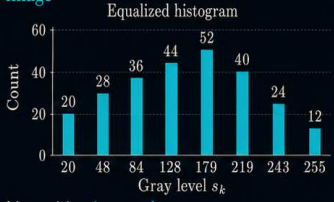
3. Apply the mapping

32→20, 48→48, 64→84, 80→128, 96→179, 112→219, 128→243, 144→255

The monotone CDF transform spreads the occupied gray levels across [0, 255].

4. Equalized 16×16 image





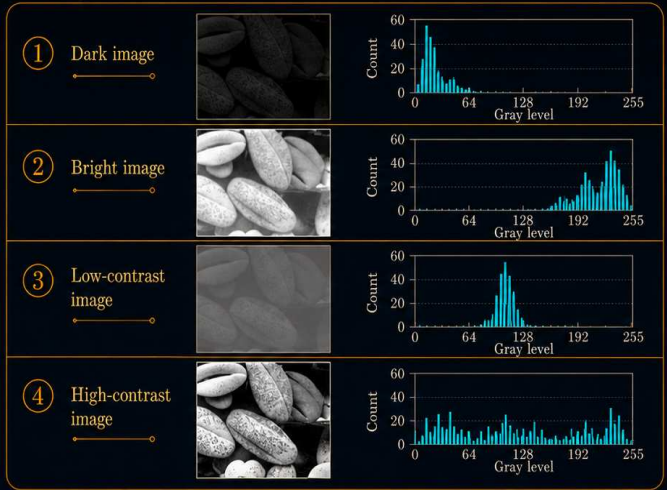
Same pixels, redistributed intensities, improved contrast.

Key idea: histogram equalization replaces each gray level by its cumulative probability mapped into [0, 255], producing a contrast-enhanced image.

In discrete images, the equalized histogram is not perfectly uniform because of quantization and finite counts.

Histogram Processing

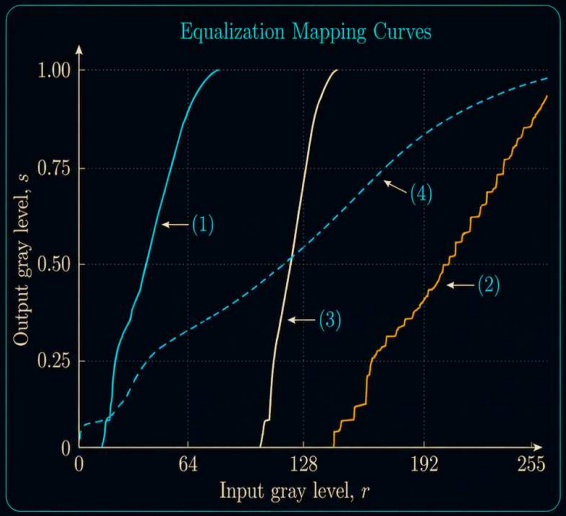
- Light, dark, and low-contrast images have concentrated histograms
- Images with uniform histograms
  - Contain the full range of gray values
  - Have high contrast
  - Better general visual appearance

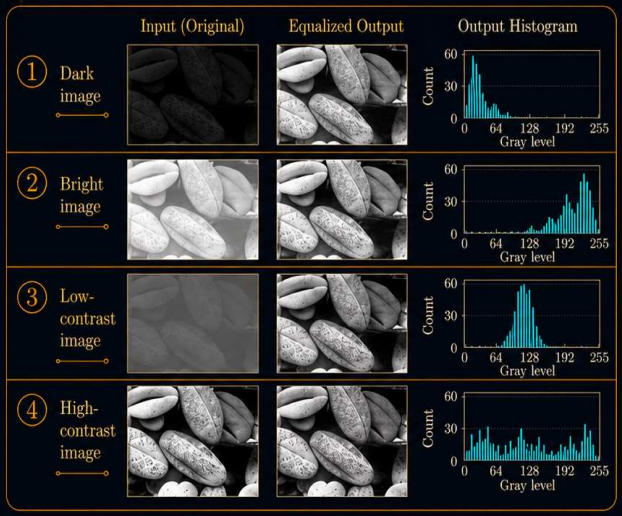


Key idea: Images with histograms that span the full range of gray levels generally have higher contrast and better visual appearance.

Histogram shape indicates how gray levels are distributed across the intensity range $[0, 255]$.

Histogram EQ Mappings

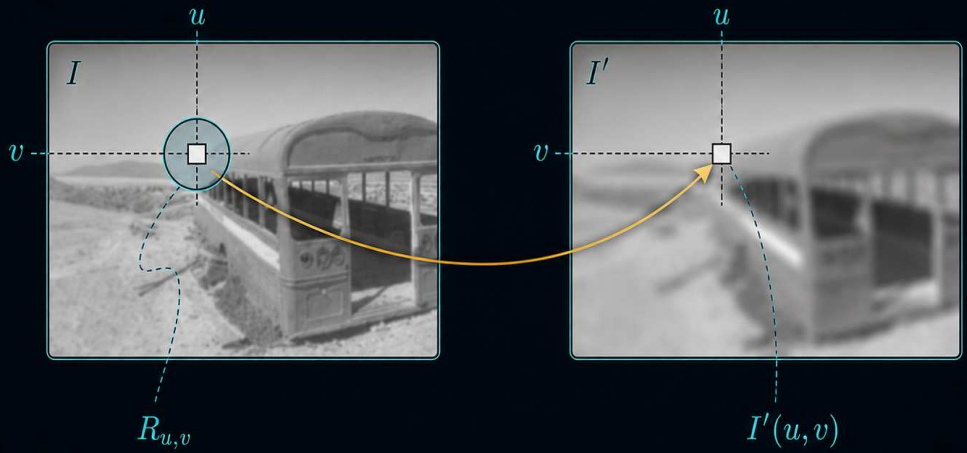




Key idea: Equalization maps intensities through the cumulative distribution, expanding occupied gray levels and improving visual contrast.

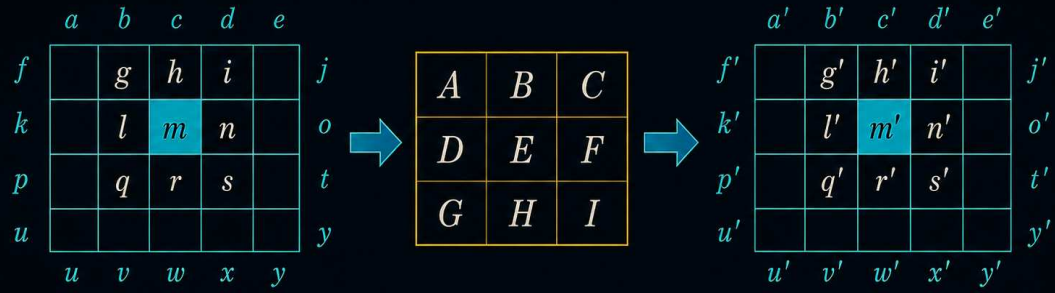
From Point Mapping to Filtering

1. Each new pixel value $I'(x, y)$ is computed as a function of the pixels in a corresponding region in the original image I.



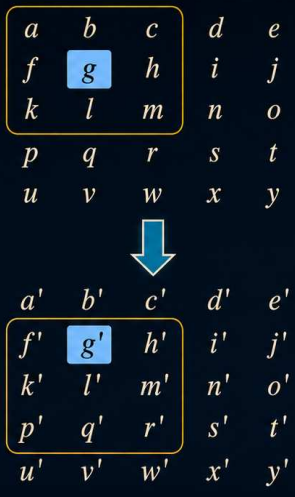
Filtering

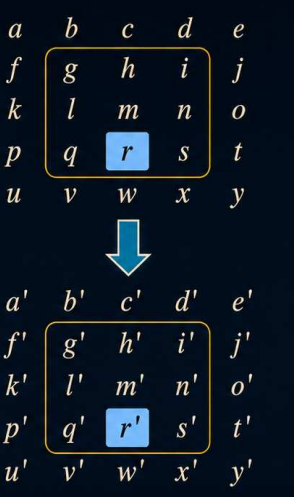
To sharpen or filter an image that was taken out-of-focus and is blurred, each pixel is replaced with a linear combination of its neighbors.



$$m' = A \cdot g + B \cdot h + C \cdot i + D \cdot l + \cdots$$
$$\cdots + E \cdot m + F \cdot n + G \cdot q + H \cdot r + I \cdot s$$

Filtering





A local window (neighborhood) in the input determines the corresponding output pixel.

Simple Smoothing Masks

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Simplest linear filter: spatial average: reduces noise, but introduces blurring
- Distance weight samples
  - Centrally located samples are more important
  - Reduces blurring (somewhat)
  - Example above: simple integer arithmetic
- Alternative approach: utilize spectral characteristics

3×3 Averaging (Box) Filter

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Each output pixel is the average of the 3×3 neighborhood.

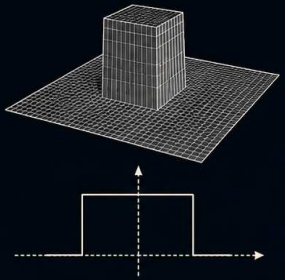
Original
(Sharp)

After 3×3 Averaging
(Smoothed)

Linear Filters

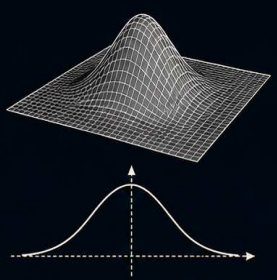
Examples of linear filters. The “box” filter (a) and the Gauss filter (b) are both smoothing filters with all-positive coefficients. The “Laplace” or “Mexican hat” filter (c) is a difference filter.

(a)



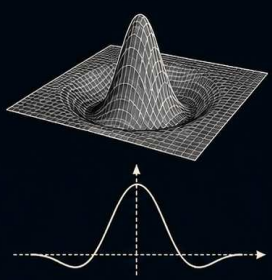
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

(b)



0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

(c)



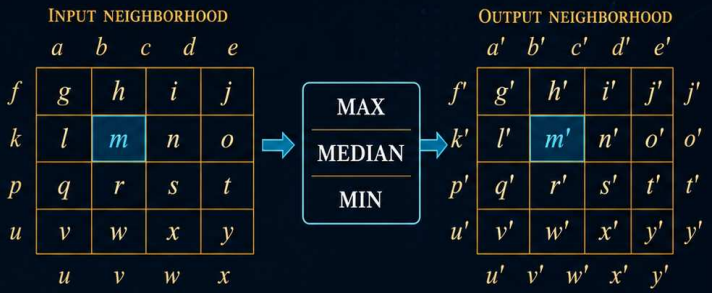
0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Order-Statistic Filters

- Linear (weighted sum) filters
  - Blur edges and details
  - Are susceptible to outliers
- Order-statistic filters (nonlinear)
  - Preserve edges and details
  - Are less susceptible to outliers
- Spatially ordered samples: $z_1, z_2, \cdots, z_N$
- Rank-ordered samples: $z_{(1)}, z_{(2)}, \cdots, z_{(N)}$
- Selection-type filter

$$z_{(1)} \leq z_{(2)} \leq \cdots \leq z_{(N)}$$

$$MED[z_1, z_2, \cdots, z_N] = z_{((N+1)/2)}$$



Minimum Filter

▶ **Purpose**

Suppress bright impulse noise (salt noise) by replacing each pixel with the minimum value in its neighborhood.

Minimum Filter (Gray-Scale)

The output at location (x, y) is

$$g(x, y) = \min_{(s,t) \in S_{xy}} f(s, t)$$

where

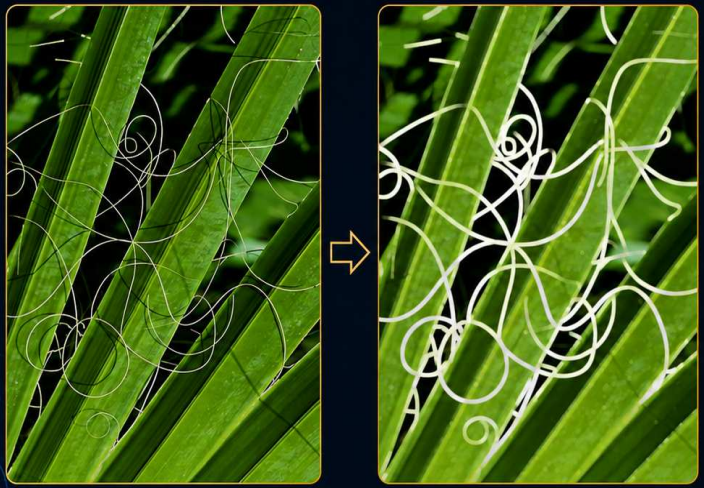
- $f(s, t)$ is the input image,
- S_{xy} is the neighborhood centered at (x, y).

▶ **Effect**

- Removes bright (salt) impulses.
- May darken thin bright structures and fine details.

Original *After minimum filter*

Maximum Filter



Original *After maximum filter*

Definition

The maximum filter replaces each pixel by the maximum value in a neighborhood.

$$g(x,y) = \max_{(s,t)\in S_{xy}} f(s,t)$$

where $f(s,t)$ is the input image,
$g(x,y)$ is the filtered image, and
S_{xy} is the neighborhood centered at (x,y).

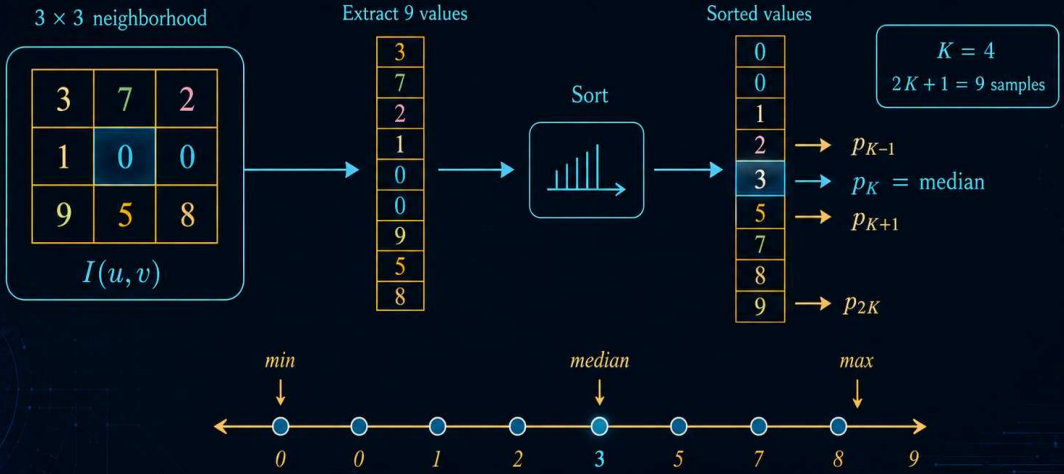
Properties

- Suppresses dark impulse noise (pepper noise).
- Can enlarge bright structures (edges/thin lines become thicker).

Use case: Effective for removing isolated dark pixels while preserving overall brightness and structure.

Median Filter

Computation of a 3×3 median filter. The nine pixel values extracted from the window are sorted and the center value is the median.



Median Filter

- Removes impulse (salt-and-pepper) noise while preserving edges better than linear smoothing filters.
- Operation (over neighborhood S_{xy}):

$$g(x,y) = \underset{(s,t)\in S_{xy}}{\text{median}} \; f(s,t)$$

where

- $f(s,t)$: input image
- $g(x,y)$: output (filtered) image
- S_{xy}: neighborhood centered at (x,y) (often a $(2k+1)\times(2k+1)$ window)

Original

After median filter

Weighted Median Filters (WMF)

The median filter can be generalized to weighted order statistic filters.

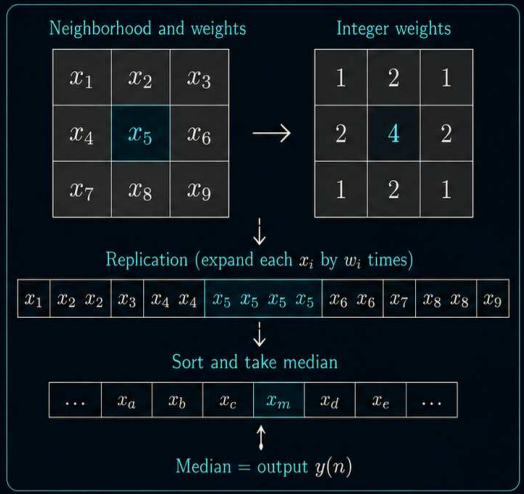
Given $x(n) = [x_1, \ldots, x_N]$, and a set of weights $w_1, \ldots, w_N$, the WMF output is given by

$$y(n) = \text{MEDIAN}\,[x_1 \lozenge w_1, x_2 \lozenge w_2, \ldots, x_N \lozenge w_N] \qquad (1)$$

where $x_i \lozenge w_i$ denotes replication:

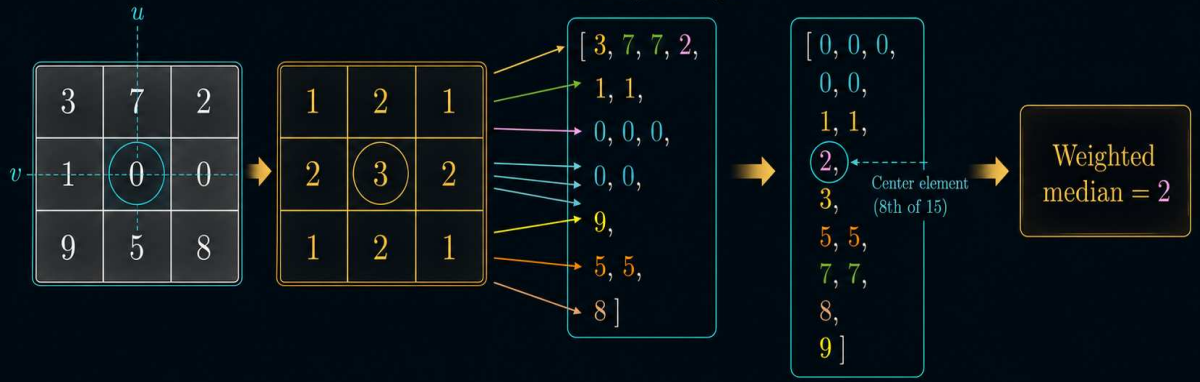
$$x_i \lozenge w_i = \overbrace{x_i, x_i, \ldots, x_i}^{w_i \text{ times}} \qquad (2)$$

and $w_i \in \mathbb{Z}^+$



★ Weighted median filtering preserves edges better than linear averaging while reducing impulse noise.

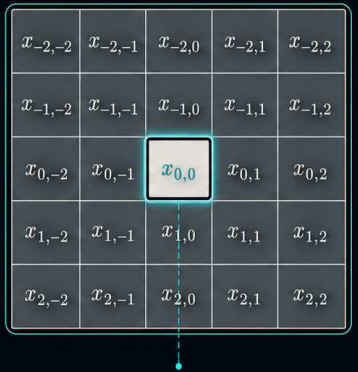
Weighted Median Example



The center pixel value 0 is repeated three times, the neighboring 0 is repeated twice, and the median of the replicated list becomes 2.

Center Weighted Median Filter (CWMF)

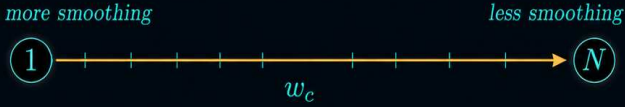
A special case is the center weighted median filter (CWMF), where the center sample receives extra emphasis.



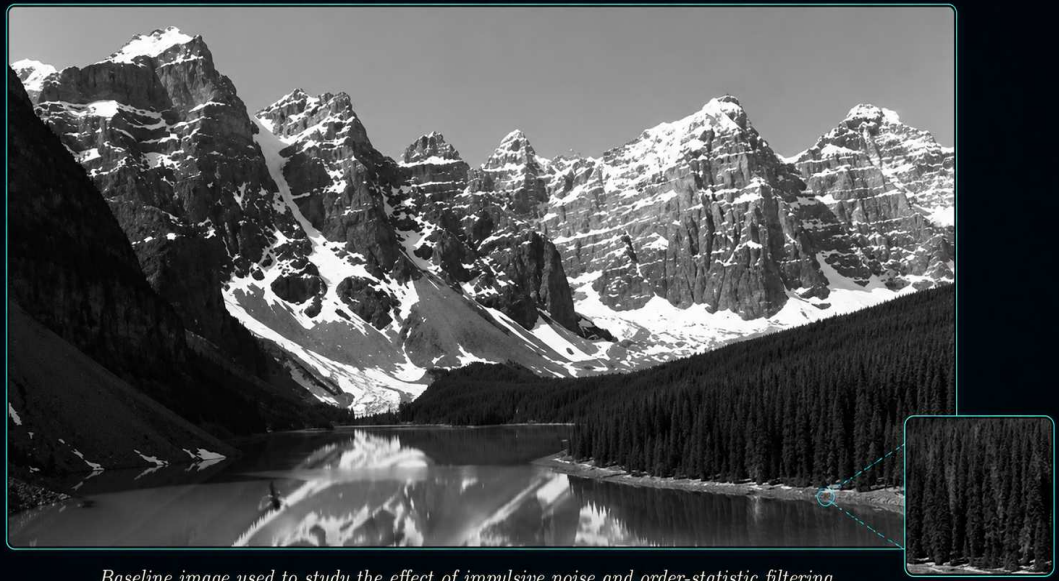
$$y(n) = \text{MEDIAN}[x_1, x_2, \ldots, x_{c-1}, x_c \lozenge w_c, x_{c+1}, \ldots, x_N] \quad (3)$$

where $c = (N+1)/2 = N_1 + 1$ is the index of the center sample, for $N = 2N_1 + 1$.

1. For $w_c = 1$, the CWMF reduces to a median filter.
2. For $w_c \geq N$, the CWMF is an identity filter.



Original Test Image



Baseline image used to study the effect of impulsive noise and order-statistic filtering.

Salt-and-Pepper Noise

- Impulse noise flips random pixels to black or white corrupting fine structure.

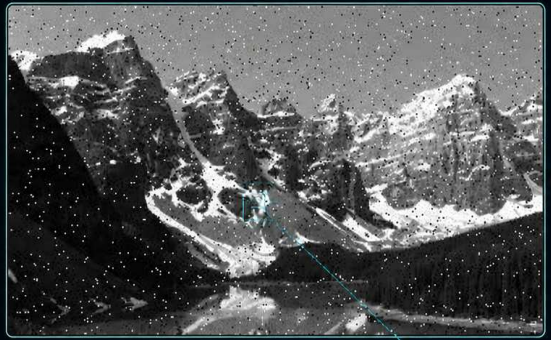
Original

Noisy

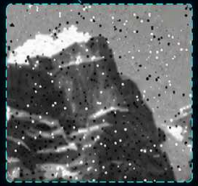
- *Order-statistic filters are especially effective for this type of degradation.*

5×5 CWMF Result — $w_c = 1$

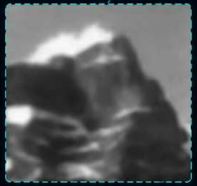
Noisy Input (Salt-and-Pepper)



Filtered Output (5×5 CWMF, $w_c = 1$)



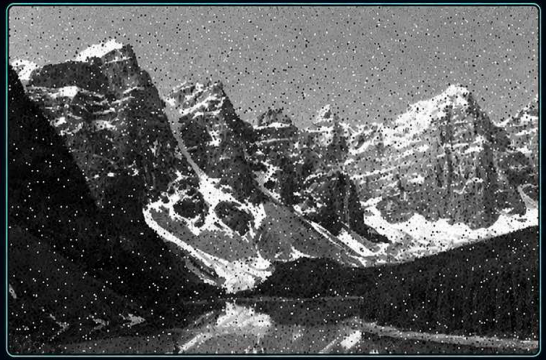
Zoom



Strong impulse-noise suppression, but with noticeable smoothing and detail loss.

5×5 CWMF Result — $w_c = 3$

Noisy Input (Salt-and-Pepper Noise)



5×5 CWMF Output ($w_c = 3$)

A balanced trade-off: effective noise removal with improved edge and structure preservation.

Center weight w_c (larger → less smoothing)

1 — 3 — 5 — 7

5×5 CWMF Result — $w_c = 5$

Noisy Input (Salt-and-Pepper Noise)

After 5×5 CWMF ($w_c = 5$)

Greater center emphasis preserves more detail, but residual impulse noise may remain.

Filter Trend (Center Weight w_c)

$w_c = 1$ → Strongest Smoothing
best noise removal, less detail

$w_c = 5$ → Strongest Detail Preservation
more detail, possible residual noise

Bilateral Filtering

ELEG404/604

Edge-preserving smoothing for grayscale and color images

A bilateral filter smooths within regions while preserving sharp edges by combining **spatial proximity** and **intensity similarity**.

$$\hat{I}(p) = \frac{1}{W_p} \sum_{q \in \Omega} G_{\sigma_s}(\|p - q\|)\, G_{\sigma_r}(|I(p) - I(q)|)\, I(q)$$

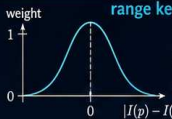
$$W_p = \sum_{q \in \Omega} G_{\sigma_s}(\|p - q\|)\, G_{\sigma_r}(|I(p) - I(q)|)$$

spatial kernel

Weights neighbors by spatial distance

$G_{\sigma_s}(\|p - q\|)$

+

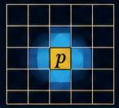


range kernel

Weights neighbors by intensity similarity

$G_{\sigma_r}(|I(p) - I(q)|)$

=



normalized weighted average

Combine both kernels and normalize to get the output

$\hat{I}(p)$

Input

Noisy grayscale

Noisy color

Bilateral output

Grayscale (smoothed, edges preserved)

Color (smoothed, edges preserved)

Why it matters: unlike Gaussian blur, bilateral filtering reduces noise without washing out edges.

The Two Kernels Behind Bilateral Filtering

ELEG404/604

Spatial closeness + photometric similarity

① **Spatial kernel:**

$$G_{\sigma_s}(\|p-q\|) = \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right)$$

② **Range kernel:**

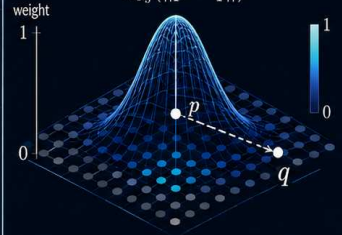
$$G_{\sigma_r}(|I(p)-I(q)|) = \exp\left(-\frac{|I(p)-I(q)|^2}{2\sigma_r^2}\right)$$

③ **Combined unnormalized weight:**

$$w(p,q) = G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I(p)-I(q)|)$$

① Spatial kernel

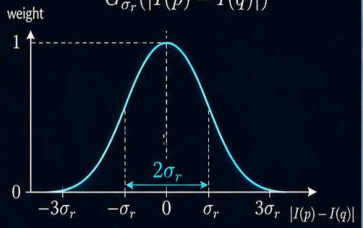
$G_{\sigma_s}(\|p-q\|)$



Nearby pixels receive larger weights;
σ_s controls spatial extent.

② Range kernel

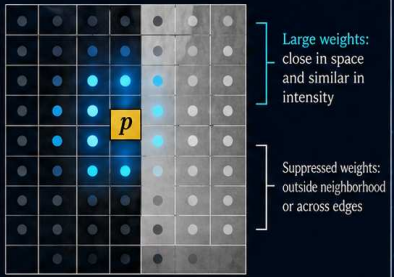
$G_{\sigma_r}(|I(p)-I(q)|)$



Similar intensities receive larger weights;
σ_r controls edge sensitivity.

③ Combined effect

$w(p,q) = G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I(p)-I(q)|)$



Key idea: weights are large only when pixels are both **close in space** and ***similar in intensity***.

Why Bilateral Filtering Preserves Edges

ELEG404/604

Gaussian smoothing blurs edges; bilateral filtering does not

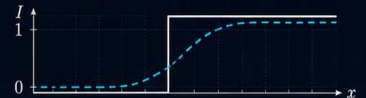
$$\hat{I}(p) = \frac{1}{W_p} \sum_{q \in \Omega} G_{\sigma_s}(\|p - q\|)\, G_{\sigma_r}(|I(p) - I(q)|)\, I(q),$$
$$W_p = \sum_{q \in \Omega} G_{\sigma_s}(\|p - q\|)\, G_{\sigma_r}(|I(p) - I(q)|)$$

Gaussian blur

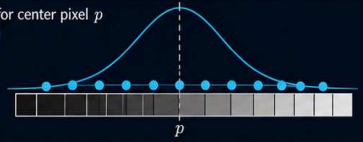
1D intensity profile

— Original (step edge)

- - - After Gaussian blur



Neighborhood weights for center pixel p (only spatial kernel G_{σ_s})



Similar weights across the edge

Pixels across the edge still contribute significantly.

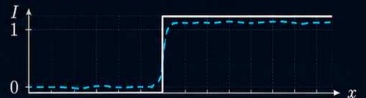
VS

Bilateral filter

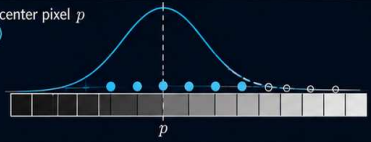
1D intensity profile

— Original (step edge)

- - - After bilateral filter



Neighborhood weights for center pixel p (spatial G_{σ_s} × range G_{σ_r})



Weights drop sharply across the edge

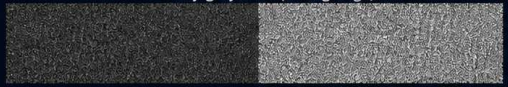
Large intensity differences → small range weights → negligible contribution.

1) Within a homogeneous region: range weights are high, so noise is averaged out.

2) Across an edge: intensity differences are large, so weights drop sharply.

3) Result: smooth interiors and preserved boundaries.

Noisy grayscale (strong edge)



Bilateral-filtered result

Bilateral filtering is an **adaptive, data-dependent** smoother: edge pixels are protected because **similarity** matters.

Step-by-Step Example on a Grayscale Patch

① Input patch (I)

48	50	52
47	51	53
46	49	55

Center pixel p (blue)

$I(p) = 51$

② Spatial kernel $G_{\sigma_s}(p, q)$

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

$\sigma_s = 1.0$

③ Range kernel $G_{\sigma_r}(|I(p) - I(q)|)$

0.882	0.980	0.835
0.923	1.000	0.803
0.960	0.961	0.691

$\sigma_r = 10.0$

④ Combined weights $w(p, q)$

0.066	0.122	0.063
0.115	0.204	0.100
0.072	0.119	0.052

$$w(p, q) = G_{\sigma_s}(p, q)\, G_{\sigma_r}(|I(p) - I(q)|)$$

⑤ Normalized weighted sum

$$\hat{I}(p) = \frac{\sum_q w(p, q)\, I(q)}{\sum_q w(p, q)}$$

Weighted sum $\sum_q w(p, q)\, I(q)$	=	91.48
Sum of weights $\sum_q w(p, q)$	=	1.7942
Normalized result $\hat{I}(p)$	=	**50.99**

⑥ Output

The filtered value at the center pixel p stays near the dark region value.

$$\hat{I}(p) \approx 51$$

Takeaway: Pixels across the edge are rejected by the range kernel, so the output does not get pulled across the boundary.

Results on Grayscale and Color Images

ELEG404/604

Comparison with Gaussian smoothing

Grayscale example

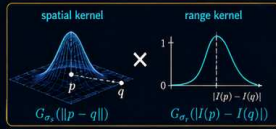
Color example

Metric	Gaussian smoothing	Bilateral filtering
Noise reduction	High	High
Edge preservation	Weak	Strong
Color bleeding	More	Less

$$\hat{I}(p) = \frac{1}{W_p} \sum_{q \in \Omega} G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I(p)-I(q)|)\, I(q)$$

$$W_p = \sum_{q \in \Omega} G_{\sigma_s}(\|p-q\|)\, G_{\sigma_r}(|I(p)-I(q)|)$$

The range kernel prevents averaging across large intensity/color differences.



In both grayscale and color images, bilateral filtering offers a strong visual compromise between denoising and edge preservation.

Step-by-Step Example on a Grayscale Patch

ELEG404/604

How the bilateral filter computes one output pixel

$$w(p,q) = \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right)\exp\left(-\frac{|I(p)-I(q)|^2}{2\sigma_r^2}\right)$$

p: center pixel (output pixel)
q: neighbor pixel in the window
σ_s: spatial std. dev. σ_r: range std. dev.

1 Neighborhood values

5×5 grayscale patch straddling a vertical edge. Center pixel p (on dark side near the edge) $I(p) = 52$.

45	47	51	195	201
46	49	53	198	202
44	48	52 p	197	199
47	50	54	196	203
49	52	55	194	205

0 — 255

2 Spatial weights

Gaussian over spatial distance ($\sigma_s = 1.0$)

0.018	0.082	0.135	0.082	0.018
0.082	0.368	0.607	0.368	0.082
0.135	0.607	1.000	0.607	0.135
0.082	0.368	0.607	0.368	0.082
0.018	0.082	0.135	0.082	0.018

0 — 1

3 Range weights

Similarity in intensity ($\sigma_r = 10$)

Neighbors with intensities near $I(p) = 52$ get high weights. Bright neighbors (~200) get tiny weights.

0.803	0.670	1.000	4.5e-90	1.2e-118
0.726	0.449	0.911	1.9e-92	2.5e-120
0.882	0.670	1.000	2.2e-91	6.1e-119
0.607	0.368	0.726	5.0e-91	1.0e-121
0.449	0.327	0.607	1.3e-90	8.5e-123

0 — 1

4 Combined weights

Elementwise product: $w(p,q) = w_s(p,q)\, w_r(p,q)$

Bright neighbors across the edge receive almost zero combined weight.

0.014	0.055	0.135	3.7e-91	2.2e-120
0.059	0.165	0.553	6.0e-92	2.0e-121
0.119	0.406	1.000	1.3e-91	8.2e-120
0.050	0.135	0.441	1.8e-91	8.2e-123
0.008	0.027	0.082	1.1e-90	1.5e-122

0 — 1

5 Normalized weighted sum

$$\hat{I}(p) = \frac{\sum_q w(p,q)\, I(q)}{\sum_q w(p,q)}$$

Because weights on bright pixels are ~0, they contribute almost nothing to the sum.

Weighted sum $\sum_q w(p,q)I(q)$	52.48
Sum of weights $\sum_q w(p,q)$	1.7942
Normalized result $\hat{I}(p)$	29.24

(Bright side contributes ~0 to both sum and weight.)

6 Output

Output stays near the dark region value.

$$\hat{I}(p) \approx 51$$

Bilateral filtering preserves the edge and avoids pulling the output across it.

Pixels across the edge are **rejected** by the range kernel, so the output **does not get pulled** across the boundary.

Non-Local Means Filtering

ELEG404/604

Edge-preserving denoising through nonlocal self-similarity

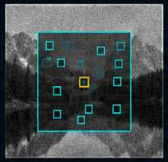
1 Noisy image

Observed noisy image v

2 Reference patch

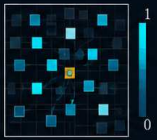
Patch centered at i

3 Search window



Candidates $j \in \Omega$

4 Similarity weights

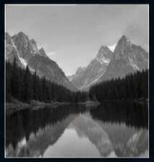


Larger weight = more similar

5 Weighted average

Aggregate using $w(i, j)$

6 Clean image



Denoised estimate $\hat{u}$

$$\hat{u}(i) = \sum_{j \in \Omega} w(i, j)\, v(j), \qquad \sum_{j} w(i, j) = 1$$

NLM replaces each pixel by a weighted average of many pixels whose surrounding patches look similar.

Uses **nonlocal redundancy** across the whole image.

Preserves **edges** and **fine details** better than local averaging.

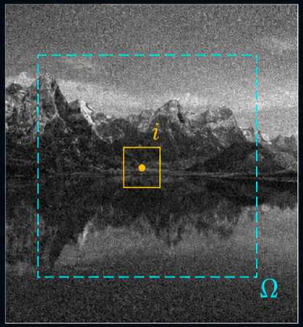
Reduces noise effectively while **avoiding oversmoothing**.

Patch Similarity and Weights

ELEG404/604

How the NLM kernel is built

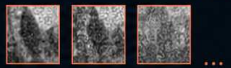
Noisy image v



Search window Ω

Similar patches (high weight)

Dissimilar patches (low weight)



- Reference patch centered at i (size $p \times p$)
- Search window Ω around i
- Similar patches ⇒ large weights
- Dissimilar patches ⇒ tiny weights

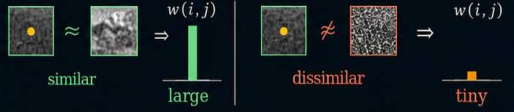
$$w(i,j) = \frac{1}{Z(i)} \exp\left(-\frac{\|P_i - P_j\|_{2,a}^2}{h^2}\right)$$

$$Z(i) = \sum_{j \in \Omega} \exp\left(-\frac{\|P_i - P_j\|_{2,a}^2}{h^2}\right)$$

$$\hat{u}(i) = \sum_{j \in \Omega} w(i,j)\, v(j)$$

- P_i, P_j: patches centered at i and j
- $\|\cdot\|_{2,a}$: Gaussian-weighted patch distance
- h: filtering parameter
- $Z(i)$: normalization

Visual intuition



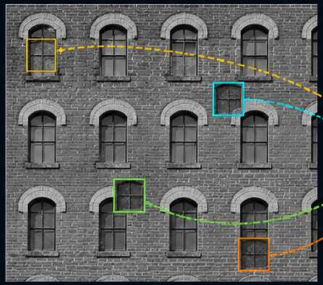
NLM replaces each pixel by a weighted average of all pixels in Ω, with weights that decay with patch dissimilarity.

Why Non-Local Means Works

ELEG404/604

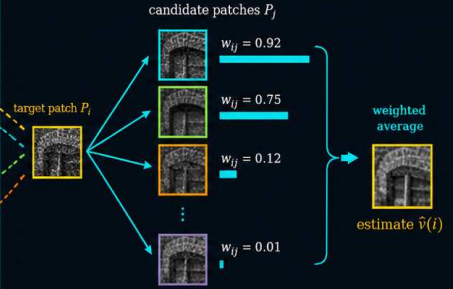
Self-similarity is more powerful than local averaging

Images contain repeated patterns

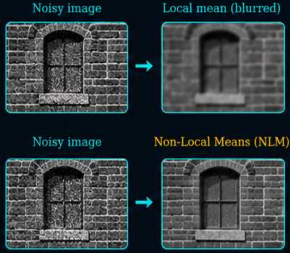


Similar patches can appear far apart

Compare many patches to the target



Local mean blurs edges.



NLM preserves structure by averaging only similar neighborhoods.

Gaussian-weighted patch distance

$$\|P_i - P_j\|_{2,a}^2 = \sum_{k \in K} a(k) \left[v(i+k) - v(j+k) \right]^2$$

$v(\cdot)$: image intensity

K : patch window

$a(k)$: Gaussian kernel (center-weighted)

1. Images contain repeated patterns (self-similarity).
2. Averaging only over similar patches reduces noise without mixing unrelated structures.
3. Edge preservation comes from low weights across dissimilar patches.
4. NLM can use information from far-away but similar neighborhoods.

NLM Results: Grayscale and Color

ELEG404/604

Visual denoising performance

Grayscale example

Noisy

Gaussian blur (local smoothing)

Non-Local Means

Color example

Noisy

Gaussian blur (local smoothing)

Non-Local Means

Key takeaways

- better texture preservation
- less edge blurring
- uses patch redundancy

$$\text{PSNR}_{\text{NLM}} > \text{PSNR}_{\text{Gaussian}}$$

NLM achieves higher fidelity than Gaussian smoothing

Parameters and Practical Implementation

ELEG404/604

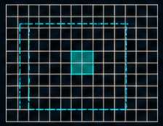
Patch size, search window, and computational cost

Nonlocal Means (NLM) averages all pixels in a search window with weights that decay with patch dissimilarity.

Patch (K)
Small patch captures local context.

e.g., 5×5 or 7×7

Search window (Ω_i)
Larger window enables nonlocal comparisons.



e.g., 11×11 or 21×21

Weights $w(i,j)$
Weights decay with patch dissimilarity.

NLM estimator

$$\hat{u}(i) = \sum_{j \in \Omega_i} w(i,j)\, v(j)$$

$v(j)$: noisy pixel at location j
Ω_i: search window around i
$w(i,j)$: weight based on patch dissimilarity
Weights are normalized: $\sum_{j \in \Omega_i} w(i,j) = 1$

Computational cost

$$\text{cost} \approx O\left(N\ |\Omega_i|\ |K|\right)$$

N: number of pixels, $|\Omega_i|$: search window size,
$|K|$: patch size (number of pixels in the patch)

- **patch size controls context:** larger patches are more discriminative but costlier
- **search window controls nonlocal reach:** larger windows exploit more redundancy
- **h controls smoothing strength:** larger h increases averaging (more smoothing)
- **acceleration** via restricted search, precomputation, integral images, or fast approximations

Typical choices (depends on image and noise level)

Patch size (K)	5×5 or 7×7
Search window (Ω_i)	11×11 or 21×21

h (filter parameter): tuned to noise level, e.g., $h \in [5,15]$

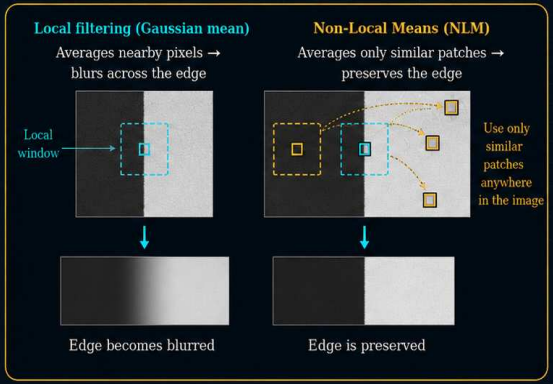
Takeaway: Choose the smallest patch that preserves structure, the largest search window your budget allows, and tune h to balance detail preservation and noise suppression.

Summary: Non-Local Means vs. Local Filtering

ELEG404/604

What students should remember

	Local mean / Gaussian	Non-Local Means (NLM)
Averaging domain	Local neighborhood (nearby pixels)	Whole image (all pixels)
Similarity criterion	Spatial proximity (distance only)	Patch similarity (intensity patterns)
Edge preservation	Blurs edges (smears across edge)	Preserves edges (avoids dissimilar patches)
Texture preservation	Over-smooths textures (removes fine details)	Preserves textures (averages only similar patterns)
Computational cost	Low ($O(N)$ to $O(N \log N)$)	High ($O(N \cdot S)$ to $O(N^2)$) (S = search window size)



NLM = weighted average over similar patches, not merely nearby pixels.

$$\hat{u}(i) = \sum_j w(i,j) v(j)$$

$\hat{u}(j)$: noisy pixel at location j
$w(i,j) \geq 0$: weight based on patch similarity
$\sum_j w(i,j) = 1$

★ **NLM succeeds because natural images contain repeated patterns and self-similarity.**