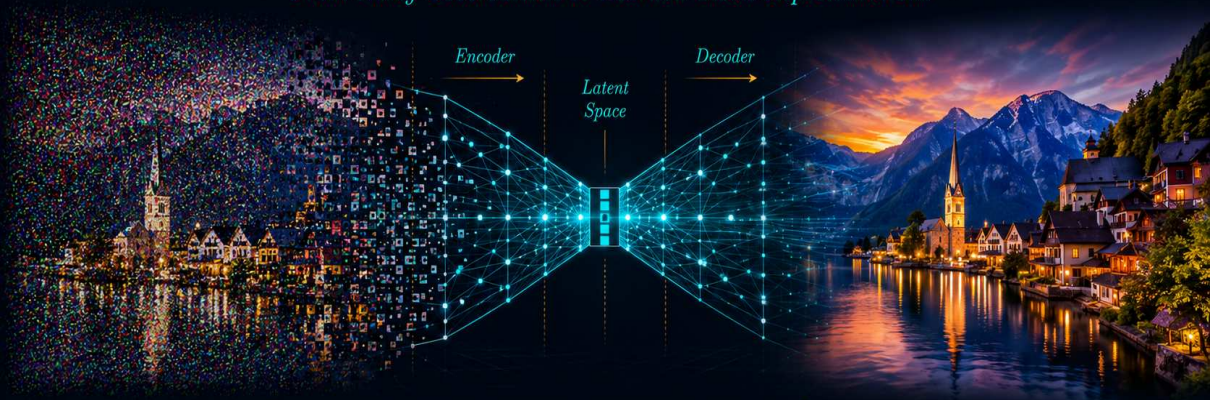
Chapter 4: Denoising and Auto Encoder Networks

From Noisy Observations to Learned Clean Representations



Denoising as a Powerful Building Block

Abstract — from noise removal to a cornerstone of modern imaging

Noisy Observation

Denoised Estimate

$$x = u + e \qquad e \sim \mathcal{N}(0, \sigma^2 I) \qquad \hat{x}(\alpha) = f(x; \alpha) \approx u$$

1. Denoising reduces random fluctuations while preserving essential structure.

2. Modern denoisers approach *theoretical limits* and reveal powerful underlying structure.

3. Denoising now serves as a reusable building block for imaging, inverse problems, and machine learning.

Linear Intuition and Stability

Why symmetric smoothers matter.

1 $f(x, \alpha) = W(\alpha)x$

- The denoiser is a linear operator parameterized by α.

2 $W(\alpha)1 = 1$

- Row-stochastic weights *preserve* the mean local brightness.

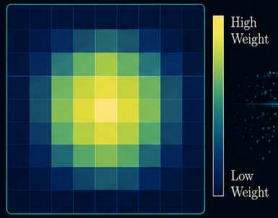
3 $W(\alpha) = W(\alpha)^T$

- Symmetry gives a *conservative*, self-adjoint structure.

4 $\|f(x,\alpha) - f(y,\alpha)\| \leq M(\alpha)\|x - y\|$

- The Lipschitz condition expresses stability to input perturbations.

Averaging Weights $W(\alpha)$



Symmetric & Row-stochastic

Input Image x

Smoothed (Denoised) Output $f(x, \alpha) = W(\alpha)x$

☆ Non-expansive denoiser

$M(\alpha) \leq 1$

No amplification of perturbations.

Key Takeaways

- ✓ Row-stochastic weights *preserve mean* local brightness.
- ✓ Symmetry yields a *conservative*, *self-adjoint* operator.
- ✓ The Lipschitz bound ensures *stability* and robustness.

The Basic Denoising Model

Section 1 — clean signal, noise, and the denoiser.

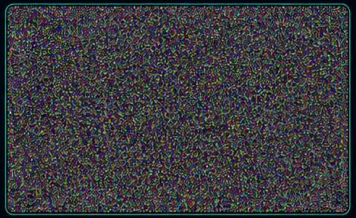
Clean signal (u)

+

Noisy observation (x)

=

Noise (e)



$$x = u + e \quad | \quad e \sim \mathcal{N}(0, \sigma^2 I) \quad | \quad \hat{x}(\alpha) = f(x; \alpha) \approx u \quad | \quad \alpha(\sigma^2) \text{ is monotonic in } \sigma^2$$

1. The observed image is *signal plus noise*.

2. The noise is modeled as zero-mean *Gaussian white noise*.

3. The denoiser estimates the underlying signal using a strength parameter α.

Note: Images are scanned lexicographically (row by row, left to right, top to bottom) and vectorized into x, u, $e \in \mathbb{R}^N$, where N is the number of pixels.

Ideal Denoisers: Two Core Properties

Identity and conservation.

1 Property 1: *Identity*

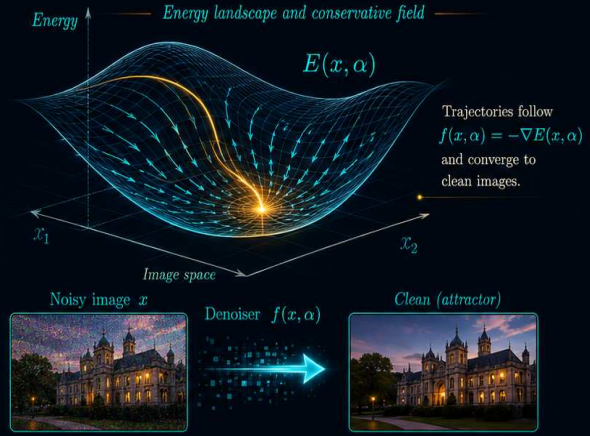
$$f(x, 0) = x, \quad \forall x$$

- Identity means no change when $\alpha = 0$ (no noise).

2 Property 2: *Conservation*

$$\nabla f(x, \alpha) = \nabla f(x, \alpha)^T$$
$$f(x, \alpha) = \nabla E(x, \alpha)$$

- The Jacobian of f is symmetric.
- Hence, f defines a *conservative vector field*.
- Therefore, f is the gradient of a scalar *energy* $E(x, \alpha)$.



Identity preserves clean images. *Conservation* ensures the denoiser is the *gradient of an energy*, guaranteeing stable, structure-respecting, and optimizable behavior.

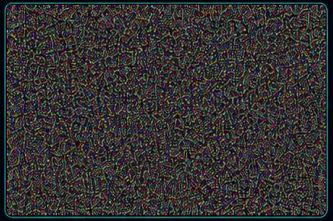
The Basic Denoising Model

Section 1 — clean signal, noise, and the denoiser.

Clean signal (u)

+

Noise (e)



=

Noisy observation (x)

$$x = u + e \quad | \quad e \sim \mathcal{N}(0, \sigma^2 I) \quad | \quad \hat{x}(\alpha) = f(x; \alpha) \approx u \quad | \quad \alpha\,(\sigma^2) \text{ is monotonic in } \sigma^2$$

1. The observed image is *signal plus noise*.

2. The noise is modeled as zero-mean *Gaussian white noise*.

3. The denoiser estimates the underlying signal using a strength parameter α.

Note: Images are scanned lexicographically (row by row, left to right, top to bottom) and vectorized into $x, u, e \in \mathbb{R}^N$, where N is the number of pixels.

Affine Structure and Why It Matters

Ideal denoisers form an affine space.

Affine combination of ideal denoisers

$$g_a(x,\alpha) = \sum_{k=0}^{N} a_k\, f(x,\alpha_k)$$

Affine constraint

$$\sum_{k=0}^{N} a_k = 1$$

Blending multiple denoised versions preserves the ideal-denoiser structure.

Summary: Ideal Denoiser

Property 1:

$$f(x,0) = x$$

Property 2:

$$f(x,\alpha) = \nabla E(x,\alpha)$$

Any affine combination of ideal denoisers is still an ideal denoiser: it equals $\nabla E_g(x,\alpha)$ for some energy $E_g(x,\alpha)$, and satisfies $g_a(x,0) = x$.

Why it matters: Denoising as a reusable building block

Imaging

Enhances image quality, restores detail, and preserves structure.

Inverse Problems

Solves ill-posed problems with prior-driven regularization.

Machine Learning

Improves data, features, and generalization; enables plug-and-play and deep models.

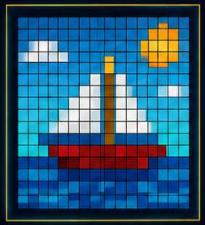
From a theoretical insight to practical impact — denoising has evolved into a universal building block for imaging, inverse problems, and machine learning.

Fully Connected Neural Networks

From an image to layered inference

Neural Networks

Input image



$8 \times 8 \times 3$ (RGB)

vectorization / flattening

$x =$

x_1
x_2
x_3
$\vdots$
x_n

$\in \mathbb{R}^n$

Input layer n | Hidden layer 1 128 | Hidden layer 2 64 | Hidden layer 3 32 | Output layer 3

$\hat{y} =$ 0.03 / **0.91** / 0.06

Predicted class: **2**

(highest probability)

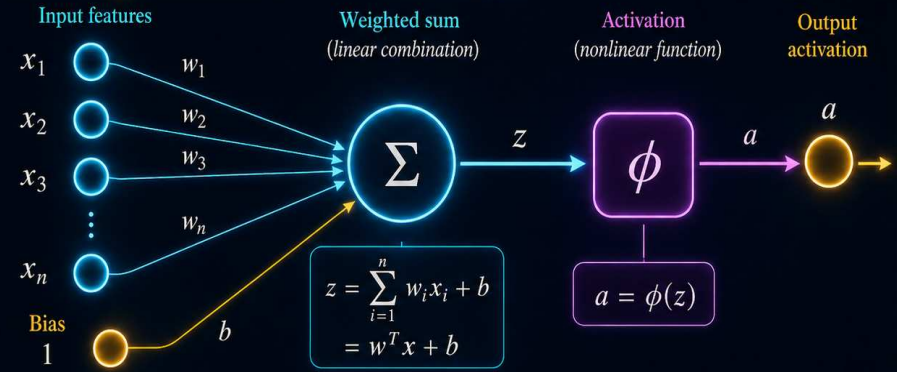
$$a^{(0)} = x$$
$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$$
$$a^{(l)} = \phi(z^{(l)})$$

- Dense connections mix information globally
- Depth enables hierarchical features
- Training adjusts weights to minimize error

A Single Neuron: Weighted Sum + Nonlinearity

Neural Networks

The basic computational unit



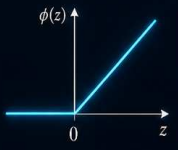
What this neuron does

- **weights** scale each input
- **bias** shifts the response
- **nonlinearity** enables complex decision boundaries

Common activation functions $\phi(z)$

ReLU

$\phi(z) = \max(0, z)$

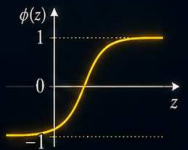


Sigmoid

$\phi(z) = \dfrac{1}{1 + e^{-z}}$

Tanh

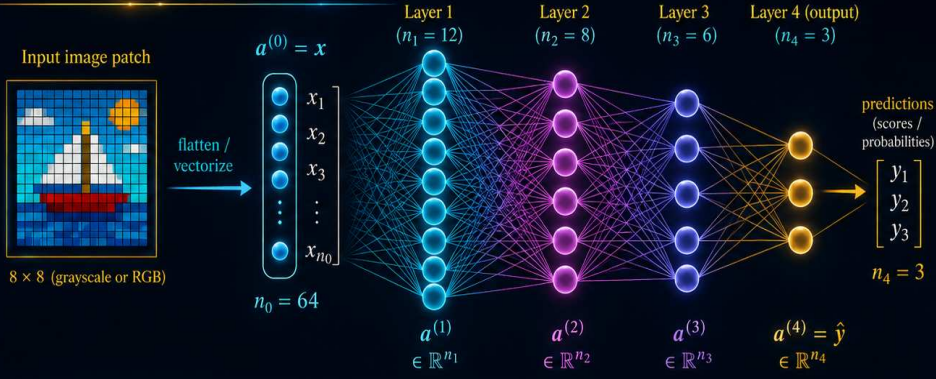
$\phi(z) = \tanh(z)$



Stacking Layers in a Dense Network

From one layer to the next

Neural Networks



KEY TAKEAWAYS

Each layer mixes all outputs from the previous layer.

Depth allows progressively richer features.

The final layer maps to prediction scores or probabilities.

LAYERWISE EQUATIONS

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \phi\left(z^{(\ell)}\right), \qquad \ell = 1, 2, \ldots, L$$

DIMENSIONS

$W^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$,

$b^{(\ell)} \in \mathbb{R}^{n_\ell}$

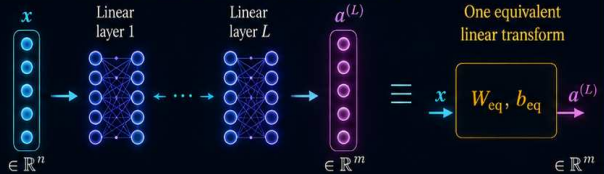
$a^{(\ell)}$: activations (outputs) $z^{(\ell)}$: pre-activations (linear) $W^{(\ell)}$: weights $b^{(\ell)}$: biases ϕ : activation function

Why Nonlinearities Matter

Neural Networks

Without them, deep networks collapse to one linear map

Without nonlinearities: everything stays linear



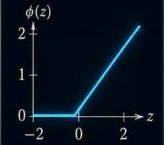
If $\phi(z) = z$, then a deep network is still linear.

$$a^{(L)} = W_{\text{eq}}\, x + b_{\text{eq}}$$

Common nonlinear activation functions

① ReLU

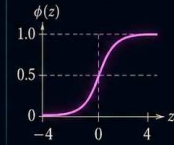
$$\phi(z) = \max(0, z)$$



Pros: sparse and efficient

② Sigmoid

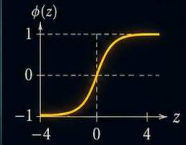
$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Pros: bounded
Cons: can saturate

③ tanh

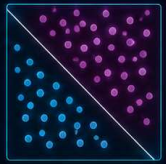
$$\phi(z) = \tanh(z)$$



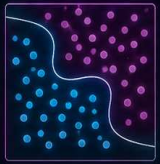
Pros: zero-centered

Linear model: straight decision boundary

Equivalent to one linear map → linear classifier



VS.



Nonlinear network: curved decision boundaries

Nonlinearities unlock expressive representations

Nonlinearities let the network build complex features and decision boundaries.

Takeaway

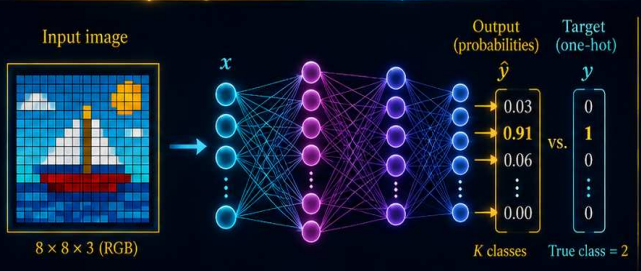
Without nonlinearities ($\phi(z) = z$), depth adds no expressiveness—everything collapses to one linear map.

- ReLU: sparse and efficient
- sigmoid: bounded but can saturate
- tanh: zero-centered

Cost Functions: Measuring Error

Neural Networks

Training seeks parameters that minimize loss



Loss (per example)

1) Regression

Mean Squared Error (MSE)

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Use for continuous targets (e.g., prices, angles, temperatures).

2) Classification

Cross-Entropy (CE)

$$\mathcal{L}_{\text{CE}} = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

Use for class probabilities with one-hot targets.

Dataset-level objective

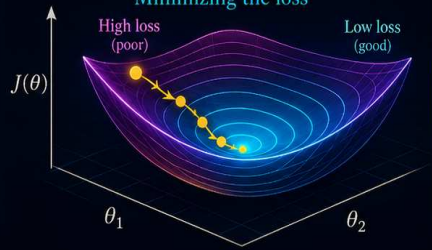
$$J(\theta) = \frac{1}{m}\sum_{j=1}^{m}\mathcal{L}^{(j)}$$

Average loss over all m training examples.

θ are the network parameters (weights & biases).

Training finds θ^* that minimizes $J(\theta)$.

Minimizing the loss



small loss = good prediction
Model output matches the target.

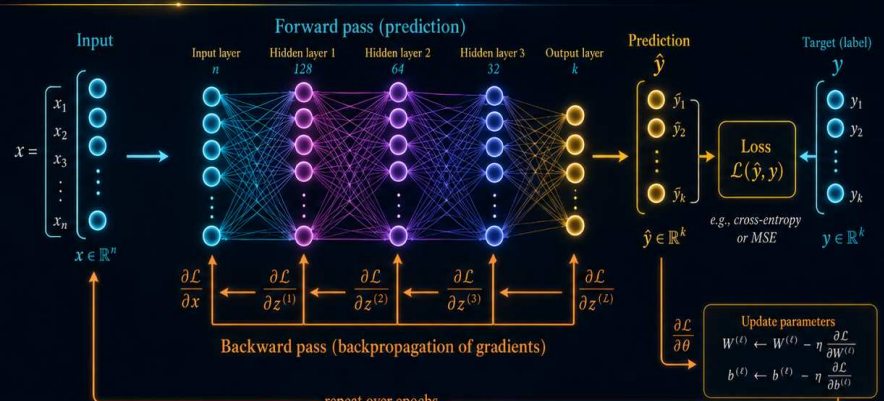
large loss = poor prediction
Model output is far from the target.

the loss guides learning
Gradients point the way to better parameters.

Backpropagation and Learning

Forward pass, error, gradients, update

Neural Networks



Chain rule (core idea)

$$\frac{\partial \mathcal{L}}{\partial W^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial z^{(\ell)}} \frac{\partial z^{(\ell)}}{\partial W^{(\ell)}}$$

Gradient descent update

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(\ell)}}$$

$$b^{(\ell)} \leftarrow b^{(\ell)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(\ell)}}$$

- Forward pass: compute activations
- Backward pass: compute gradients
- Update: adjust parameters

1 Forward propagation

Compute activations layer by layer to get $\hat{y}$.

2 Compute loss

Compare $\hat{y}$ with y to get loss $\mathcal{L}(\hat{y}, y)$.

3 Backpropagate gradients

Use chain rule to compute gradients back through layers.

4 Update parameters

Apply gradient descent with learning rate η to update $W^{(\ell)}, b^{(\ell)}$.

5 Repeat over epochs

Shuffle data, form mini-batches, and repeat until converge.

Backpropagation efficiently *reuses intermediate derivatives via the chain rule.*

Denoising a 9×9 Image with a Fully Connected Neural Network

From noisy patches to learned denoising

Noisy 9×9 patch

23	56	12	78	91	45	33	67	28
48	95	31	64	72	21	80	43	59
17	29	85	66	54	73	19	61	37
62	75	47	90	22	38	71	28	84
36	51	96	41	63	57	24	69	32
15	27	39	68	82	44	53	18	46
71	93	20	34	76	61	88	29	65
42	60	58	14	49	94	31	72	26
33	26	81	55	37	69	25	57	90

9×9 = 81 pixels

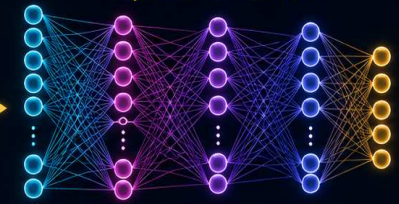
flatten

Flattened input vector

$\mathbf{x} \in \mathbb{R}^{81}$

x_1 x_2 x_3 ⋮ x_{81}

Fully connected neural network



Input layer 81 · Hidden layer 1 128 · Hidden layer 2 64 · Hidden layer 3 32 · Output layer 81

$\hat{\mathbf{y}} = f_\theta(\mathbf{x})$

$\hat{\mathbf{y}} \in \mathbb{R}^{81}$

Reconstructed 9×9 patch

25	44	40	60	72	63	58	45	28
40	66	68	77	80	75	67	55	36
48	74	84	89	92	86	74	59	38
54	78	88	94	95	90	77	61	40
57	81	91	95	96	92	82	62	42
55	79	89	93	94	90	78	60	39
48	70	80	87	89	89	72	55	36
38	58	63	69	73	68	60	45	30
25	36	40	45	48	46	40	32	22

Denoised 9×9 patch

1) Supervised learning from noisy/clean pairs
Learn from many examples of noisy patches and their clean targets.

2) Nonlinear mapping with optimized weights
The network learns $f_\theta : \mathbb{R}^{81} \to \mathbb{R}^{81}$ to remove noise and preserve structure.

3) Apply to unseen noisy images
Denoise any new image by processing its overlapping 9×9 patches.

Applies to real images

Noisy image

Learned denoiser f_θ

Denoised image

Takeaway: A fully connected neural network can learn a powerful, nonlinear mapping from noisy pixels to clean pixels.

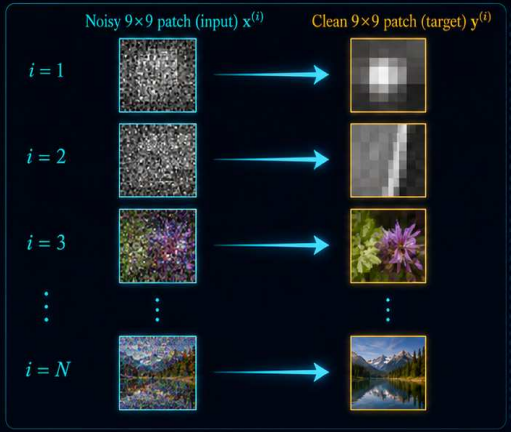
Principle scales to larger patches, deeper networks, and modern architectures.

Training with Many Noisy/Clean 9×9 Examples

Supervised learning from paired image patches

$$\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}, \qquad \mathbf{x}^{(i)}, \mathbf{y}^{(i)} \in \mathbb{R}^{81}$$

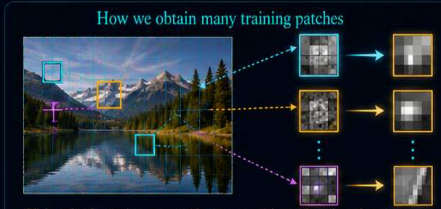
A dataset of N noisy/clean patch pairs. Each 9×9 patch is vectorized into an 81-dimensional vector.



input: noisy patch

target: clean patch

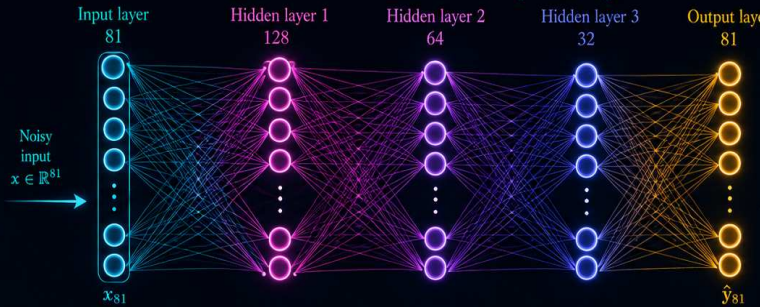
learn from many examples



Takeaway: We collect many overlapping 9×9 noisy/clean pairs from images and learn a mapping $f_\theta : \mathbb{R}^{81} \to \mathbb{R}^{81}$ that transforms noisy patches into clean patches.

Fully Connected Architecture and Learned Weights

How the network maps 81 noisy values to 81 clean values



Input (layer 0)

$$\mathbf{a}^{(0)} = \mathbf{x}$$

$\mathbf{x} \in \mathbb{R}^{81}$

Linear transform (layer ℓ)

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}$$

$\mathbf{W}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$

Nonlinear activation (layer ℓ)

$$\mathbf{a}^{(\ell)} = \phi(\mathbf{z}^{(\ell)})$$

$\mathbf{a}^{(\ell)} \in \mathbb{R}^{n_\ell}$

Output (layer L)

$$\hat{\mathbf{y}} = \mathbf{a}^{(L)}$$

$\hat{\mathbf{y}} \in \mathbb{R}^{81}$

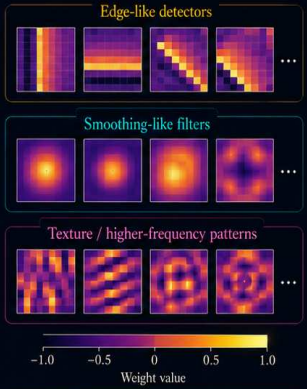
Fully connected
Each neuron connects to all neurons in the previous layer.

Nonlinearity $\phi(\cdot)$:
ReLU or tanh

Optimized parameters
The network learns the weights and biases

$$\theta = \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}$$

Learned first-layer weights $\mathbf{W}^{(1)}$
Each column reshaped to a 9×9 patch

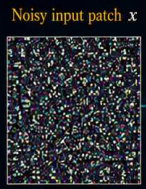


The learned weights are meaningful:
edges, smoothers, and texture detectors arise from data-driven optimization.

Takeaway: A fully connected network learning powerful nonlinear transformations. Dense connections + nonlinearities = expressive mappings from noisy to clean.

Cost Function and Backpropagation

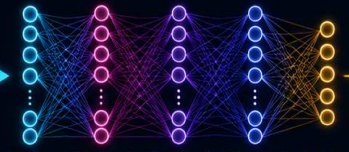
Training the denoiser by minimizing reconstruction error



Noisy input patch x

9×9 patch

Denoiser (neural network) f_θ



Predicted clean patch

$\hat{y} = f_\theta(x)$

Predicted 9×9 patch

Compare (MSE loss)

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left\|f_\theta(x^{(i)}) - y^{(i)}\right\|_2^2$$

Mean Squared Error

True clean target patch y

True 9×9 patch

Backpropagation (backward pass)

compute gradients $\nabla_\theta J(\theta)$ → update weights → repeat over epochs

Gradient Descent Update

$$\theta \leftarrow \theta - \eta\,\nabla_\theta J(\theta)$$

η = learning rate

How training works (one mini-batch)

1) **Forward pass**
Input noisy patches are passed through the network to produce predictions.

2) **Compare with target**
Compute the mean squared error between predictions and clean targets.

3) **Propagate error backward**
Backpropagate the error to compute gradients with respect to the weights.

4) **Update weights**
Adjust weights via gradient descent to reduce the reconstruction error.

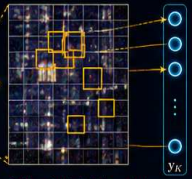
Takeaway: We minimize reconstruction error between the predicted and true clean patches by iteratively updating the network weights using backpropagation and gradient descent.

Applying the Trained Network to a Different Noisy Image

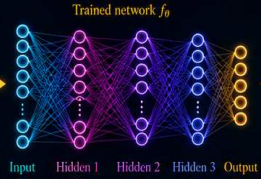
Generalization to unseen images

1) Noisy test image (unseen)

2) Extract overlapping 9×9 patches



3) Denoise each patch with the trained fully connected network



4) Denoised patches

Overlap–add reconstruction (see equation below)

Reconstructed clean image

Patch aggregation (overlap–add)

$$\hat{\mathbf{x}}_k = f_\theta(\mathbf{P}_k\mathbf{y}), \quad \hat{\mathbf{X}} = \left(\sum_k \mathbf{P}_k^T\mathbf{P}_k\right)^{-1} \sum_k \mathbf{P}_k^T\hat{\mathbf{x}}_k$$

- $\mathbf{y}$: vectorized noisy image
- $\mathbf{P}_k$: extracts the k-th 9×9 patch (selection matrix)
- $\hat{\mathbf{x}}_k$: denoised k-th patch (81×1 vector)
- $\hat{\mathbf{X}}$: reconstructed clean image (vectorized)

Grayscale zoom-in

Color zoom-in

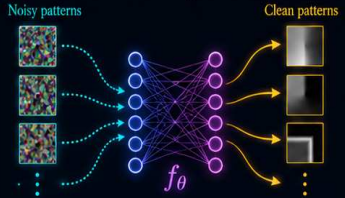
Takeaway: Trained on many noisy/clean 9×9 patch pairs, the network generalizes to a completely different unseen image and produces a high-quality denoised result.

Why It Works and Practical Considerations

Learned nonlinear denoising from data

1 Why it works

Similar noisy patterns are mapped toward clean patterns.



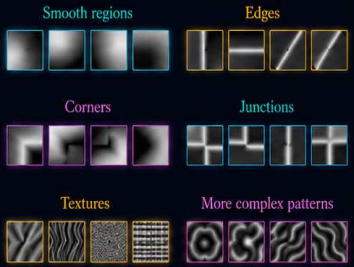
Learns from examples: captures complex relationships in natural images.

Nonlinear mapping: models rich, image-dependent structure beyond fixed filters.

Preserves structure: better edges, textures, and fine details than fixed linear rules.

2 What the network learns

The network discovers and uses a rich set of visual patterns.



By combining these *primitives*, the network can reconstruct clean versions of a vast variety of *natural image content*.

3 Efficiency and limitations

Inference is fast after training.

Can process patches in parallel (highly scalable).

Larger networks improve representation power (at higher computation/memory cost).

Fully connected patch denoisers do not exploit translation invariance as well as CNNs.

Complexity (per patch):

per patch inference:
matrix multiplications + nonlinearities

Summary equations:

$$f_\theta : \mathbb{R}^{81} \to \mathbb{R}^{81} \quad | \quad \theta^* = \arg\min_\theta J(\theta)$$

Maps a noisy 9×9 patch to a clean patch.
Learned by minimizing a loss $J(\theta)$ on training pairs.

Takeaway: Fully connected denoisers learn powerful nonlinear mappings from data—an important *bridge* from classical methods to modern *deep learning*.

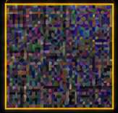
PATCH-BASED FULLY CONNECTED NN DENOISING

256 × 256 image restored from overlapping noisy patches

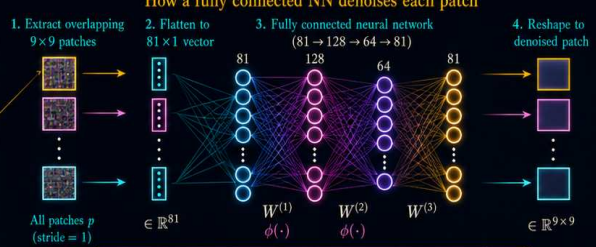
Noisy 256 × 256 image Y

Example noisy patch (9 × 9)

$\mathbf{y}_p \in \mathbb{R}^{9\times 9}$



How a fully connected NN denoises each patch



Denoised 256 × 256 image $\hat{X}$

Training objective (over many noisy/clean patch pairs $\{(\mathbf{y}_p, \mathbf{x}_p)\}$)

$$\theta^* = \arg\min_{\theta} \sum_{p} \left\| f_\theta(\mathbf{y}_p) - \mathbf{x}_p \right\|_2^2$$

where $f_\theta(\cdot)$ is the NN mapping and $\theta = \{W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)}\}$

Inference (with optimized parameters θ^*): $\hat{\mathbf{x}}_p = f_{\theta^*}(\mathbf{y}_p)$

Aggregate overlapping patch estimates

$$\hat{X}(i,j) = \frac{1}{C(i,j)} \sum_{p \ni (i,j)} \hat{\mathbf{x}}_p(i,j)$$

where $C(i,j)$ is the number of patches covering pixel (i,j)

1. Train on many noisy/clean patch pairs

Learn from large datasets of aligned noisy and clean patches.

2. Apply the optimized NN patch by patch

Extract overlapping patches from a new image and denoise each with f_{θ^*}.

3. Average overlapping outputs to form the final image

Aggregate estimates by averaging to reduce blocking artifacts and preserve details.

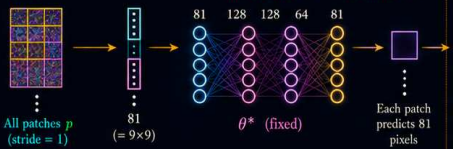
Takeaway: A fully connected neural network learned on clean pairs maps each noisy 9×9 patch to its clean counterpart. Sliding the network densely across the image and averaging overlaps yields a smooth, high-quality 256 × 256 restoration.

PATCH-BASED FCNN DENOISING: RESULTS AND ARTIFACTS

Noisy 256×256 image Y

Gaussian noise / visible corruption
(e.g., $\sigma = 25$, $\sigma^2 = 625$)

Patch-based FCNN denoising pipeline



$$\hat{x}_p = f_{\theta^*}(y_p)$$

Predicted clean patch from noisy patch y_p

$$\hat{X}(i,j) = \frac{1}{C(i,j)} \sum_{p \ni (i,j)} \hat{x}_p(i,j)$$

Average of all overlapping patch predictions covering pixel (i,j)

θ^* are optimized weights learned from many noisy/clean patch pairs.
Inference is done patch-by-patch with fixed θ^*.

Denoised 256×256 image $\hat{X}$

Noise strongly reduced,
but artifacts remain

1. oversmoothing (fine texture lost)
2. patch averaging blur (edges softened)
3. residual / structured artifacts (faint blotchy / checkerboard-like remnants)

1. Noise strongly reduced

Random noise is largely removed, yielding a visually clean result.

2. Edges mostly preserved

Major structures and object boundaries are well maintained.

3. Textures may be oversmoothed

Fine textures and subtle details can be smoothed out by the network.

4. Patch-based inference can leave aggregation artifacts

Overlap averaging can introduce faint blotchy or checkerboard-like artifacts.

Takeaway: A fully connected network denoises each local patch effectively, but because it has limited spatial context, the reconstructed 256×256 image may exhibit texture loss and mild patch-related artifacts.

ZOOMED VIEW: BLURRING AND BLOCKING ARTIFACTS

Reconstruction from a patch-based fully connected network (applied patch-by-patch)

Denoised reconstruction (patch-based FCNN)
1024 × 768 image

Overall result is clean and pleasing,
but subtle artifacts are visible when zoomed in.

1 Blurring (loss of detail)

Fine edges and textures are smoothed.
Window frames, roof edges, and small details lose sharpness.

2 Blocking artifacts (patch boundaries)

Independently processed patches produce slight intensity discontinuities along patch boundaries, visible as a grid pattern.

3 Texture smearing (detail suppression)

Repetitive textures (leaves, stone, shingles) appear waxy or smeared; micro-contrast is reduced.

Takeaway: Patchwise fully connected denoisers effectively suppress noise, but limited spatial context and independent patch reconstruction can introduce blur, block artifacts, and texture smearing.
Larger context, overlap with aggregation, or specialized architectures are needed to reduce these artifacts.

Denoising as a Powerful Building Block

Abstract — from noise removal to a cornerstone of modern imaging

Noisy Observation

Denoised Estimate

$$x = u + e \qquad e \sim \mathcal{N}(0, \sigma^2 I) \qquad \hat{x}(\alpha) = f(x; \alpha) \approx u$$

1. Denoising reduces random fluctuations while preserving essential structure.

2. Modern denoisers approach *theoretical limits* and reveal powerful underlying structure.

3. Denoising now serves as a reusable building block for imaging, inverse problems, and machine learning.

Linear Intuition and Stability

Why symmetric smoothers matter.

1 $f(x, \alpha) = W(\alpha)x$

- The denoiser is a linear operator parameterized by α.

2 $W(\alpha)1 = 1$

- Row-stochastic weights *preserve* the mean local brightness.

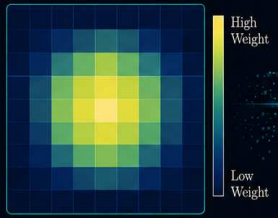
3 $W(\alpha) = W(\alpha)^T$

- Symmetry gives a *conservative*, self-adjoint structure.

4 $\|f(x,\alpha) - f(y,\alpha)\| \leq M(\alpha)\|x - y\|$

- The Lipschitz condition expresses stability to input perturbations.

Averaging Weights $W(\alpha)$



Symmetric & Row-stochastic

Input Image x

Smoothed (Denoised) Output $f(x, \alpha) = W(\alpha)x$

☆ Non-expansive denoiser

$M(\alpha) \leq 1$

No amplification of perturbations.

Key Takeaways

- ✓ Row-stochastic weights *preserve mean* local brightness.
- ✓ Symmetry yields a *conservative*, *self-adjoint* operator.
- ✓ The Lipschitz bound ensures *stability* and robustness.

The Basic Denoising Model

Section 1 — clean signal, noise, and the denoiser.

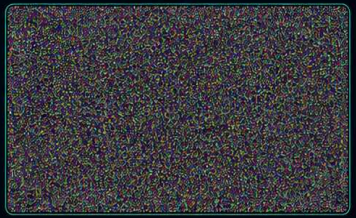
Clean signal (u)

+

Noisy observation (x)

=

Noise (e)



$$x = u + e \quad | \quad e \sim \mathcal{N}(0, \sigma^2 I) \quad | \quad \hat{x}(\alpha) = f(x; \alpha) \approx u \quad | \quad \alpha(\sigma^2) \text{ is monotonic in } \sigma^2$$

1. The observed image is *signal plus noise*.

2. The noise is modeled as zero-mean *Gaussian white noise*.

3. The denoiser estimates the underlying signal using a strength parameter α.

Note: Images are scanned lexicographically (row by row, left to right, top to bottom) and vectorized into x, u, $e \in \mathbb{R}^N$, where N is the number of pixels.

Ideal Denoisers: Two Core Properties

Identity and conservation.

1 Property 1: *Identity*

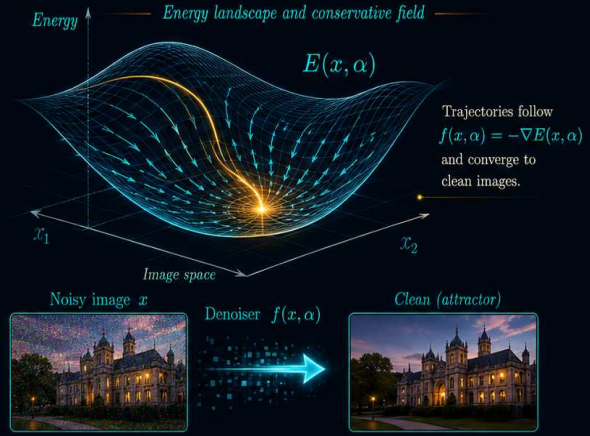
$$f(x, 0) = x, \quad \forall x$$

- Identity means no change when $\alpha = 0$ (no noise).

2 Property 2: *Conservation*

$$\nabla f(x, \alpha) = \nabla f(x, \alpha)^T$$
$$f(x, \alpha) = \nabla E(x, \alpha)$$

- The Jacobian of f is symmetric.
- Hence, f defines a *conservative vector field*.
- Therefore, f is the gradient of a scalar *energy* $E(x, \alpha)$.



Identity preserves clean images. *Conservation* ensures the denoiser is the *gradient of an energy*, guaranteeing stable, structure-respecting, and optimizable behavior.

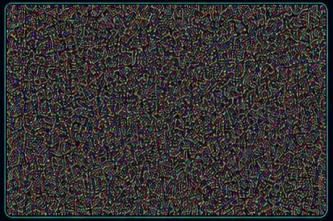
The Basic Denoising Model

Section 1 — clean signal, noise, and the denoiser.

Clean signal (u)

+

Noise (e)



=

Noisy observation (x)

$$x = u + e \quad | \quad e \sim \mathcal{N}(0, \sigma^2 I) \quad | \quad \hat{x}(\alpha) = f(x; \alpha) \approx u \quad | \quad \alpha\,(\sigma^2) \text{ is monotonic in } \sigma^2$$

1. The observed image is *signal plus noise*.

2. The noise is modeled as zero-mean *Gaussian white noise*.

3. The denoiser estimates the underlying signal using a strength parameter α.

Note: Images are scanned lexicographically (row by row, left to right, top to bottom) and vectorized into $x, u, e \in \mathbb{R}^N$, where N is the number of pixels.

Affine Structure and Why It Matters

Ideal denoisers form an affine space.

Affine combination of ideal denoisers

$$g_a(x,\alpha) = \sum_{k=0}^{N} a_k\, f(x,\alpha_k)$$

Affine constraint

$$\sum_{k=0}^{N} a_k = 1$$

Blending multiple denoised versions preserves the ideal-denoiser structure.

Summary: Ideal Denoiser

Property 1:

$$f(x,0) = x$$

Property 2:

$$f(x,\alpha) = \nabla E(x,\alpha)$$

Any affine combination of ideal denoisers is still an ideal denoiser: it equals $\nabla E_g(x,\alpha)$ for some energy $E_g(x,\alpha)$, and satisfies $g_a(x,0) = x$.

Why it matters: Denoising as a reusable building block

Imaging

Enhances image quality, restores detail, and preserves structure.

Inverse Problems

Solves ill-posed problems with prior-driven regularization.

Machine Learning

Improves data, features, and generalization; enables plug-and-play and deep models.

From a theoretical insight to practical impact — denoising has evolved into a universal building block for imaging, inverse problems, and machine learning.

Section 2: Denoising as a Natural Decomposition

From denoisers to multiscale image layers

$$x = f(x, \alpha) + [x - f(x, \alpha)] \quad (8)$$

Original image x

Denoiser $f(x, \alpha)$

Smooth (base) layer $f(x, \alpha)$

Low-pass / structure

+

Residual (detail) layer $r_0(x, \alpha) = x - f(x, \alpha)$

High-pass / details

- $f(x, \alpha)$: denoised or smoothed image
- $x - f(x, \alpha)$: high-pass residual
- Perfect reconstruction: base + residual = original

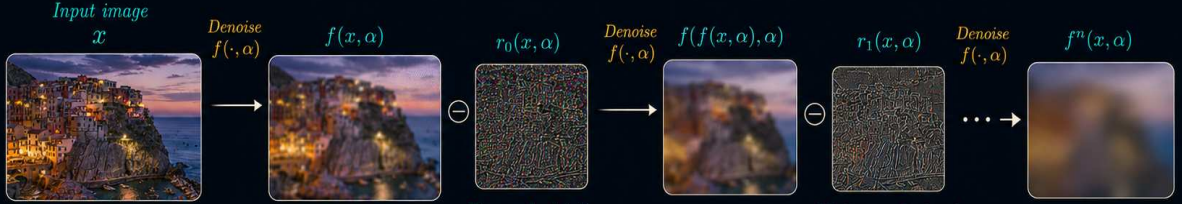
Iterated Decomposition and Multiscale Residuals

Section-2 idea: repeatedly apply the denoiser to peel away details at multiple scales

$$
\begin{aligned}
x &= f(f(x,\alpha),\alpha) + [f(x,\alpha) - f(f(x,\alpha),\alpha)] + r_0(x,\alpha) \\
&= f(f(x,\alpha),\alpha) + r_1(x,\alpha) + r_0(x,\alpha) \\
&\vdots \\
&= f^n(x,\alpha) + \sum_{k=0}^{n-1} r_k(x,\alpha) \qquad (9)
\end{aligned}
$$

Perfect reconstruction for any n

f^n behaves like a diffusion process

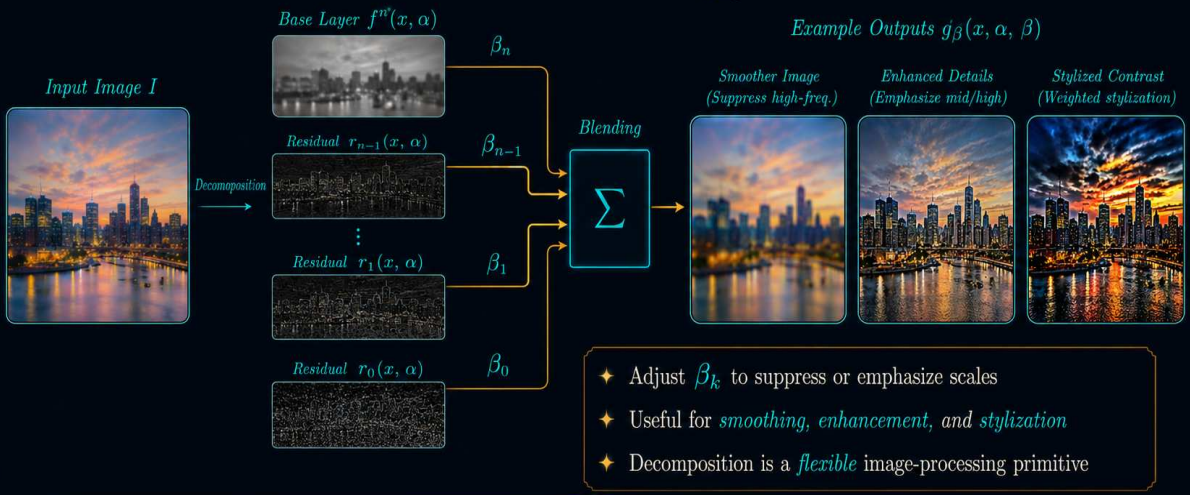


Multiscale residuals capture what the denoiser removes at each step

$$x = f^n(x,\alpha) + \sum_{k=0}^{n-1} r_k(x,\alpha)$$

Recombining Components for Image Processing

$$g_\beta(x, \alpha, \beta) = \beta_n f^n(x, \alpha) + \sum_{k=0}^{n-1} \beta_k r_k(x, \alpha) \tag{10}$$



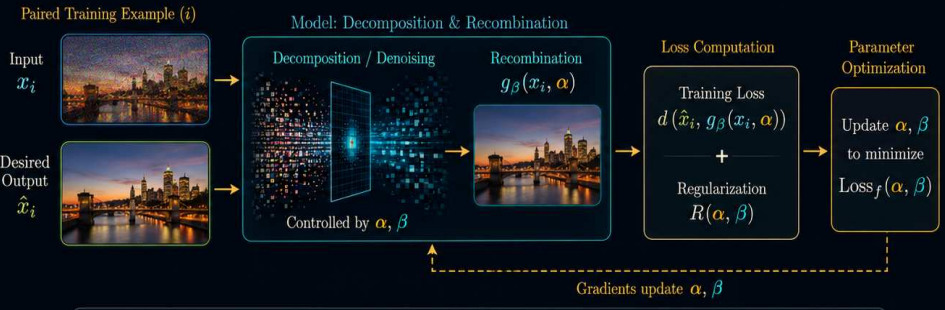
- Adjust β_k to suppress or emphasize scales
- Useful for *smoothing, enhancement, and stylization*
- Decomposition is a *flexible* image-processing primitive

Learning the Decomposition Parameters

We learn the decomposition parameters α and β from paired training examples by minimizing a loss between the desired output and the model's reconstruction.

$$\text{Loss}_f(\alpha, \beta) = \frac{1}{N} \sum_{i=1}^{N} d\left(\hat{x}_i,\ g_\beta(x_i, \alpha)\right) + R(\alpha, \beta) \qquad (11)$$

x_i is the input and $\hat{x}_i$ is the desired target output.



Supervised Learning of the Decomposition

- Learn from paired examples $(x_i, \hat{x}_i)$ of noisy inputs and clean targets.
- Optimize α and β to best reconstruct the clean output.
- Generalize to new images for effective filtering and denoising.

- **Training loss d:** Measures reconstruction error
- **Regularization R:** Encourages good solutions (e.g., smoothness, sparsity)
- **Learnable parameters:** α (decomposition strength / filtering), β (recombination / denoising behavior)

Residual Thinking: Why Decomposition Matters

Instead of learning the full transformation, learn only what's missing.

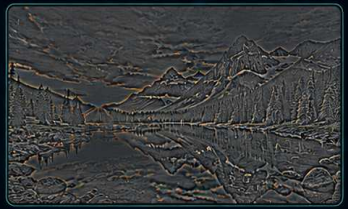
Original Image x

$\approx$

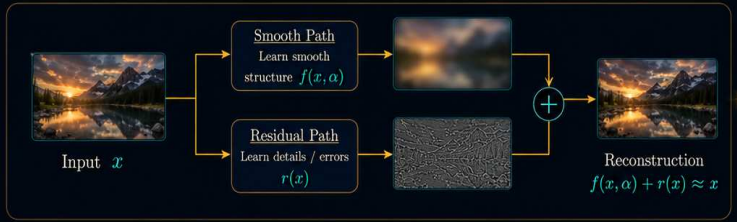
Denoised / Smooth Approximation $f(x, \alpha)$

$-$

Residual / Detail $r(x) = x - f(x, \alpha)$



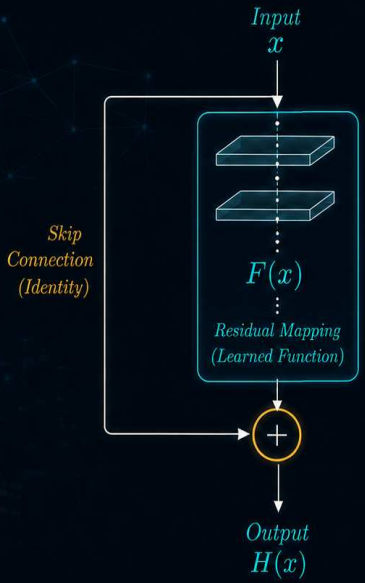
$$\textit{Residual} = \textit{Original} - \textit{Smooth Part}$$



- Residuals isolate fine details and errors
- Residual mappings are often easier to model
- This idea motivates modern *residual architectures*

Model the hard part (what's left), not the whole.

Connection to Residual Network Architectures



$$H(x) = x + F(x) \qquad (12)$$

Connection to Denoising Decomposition

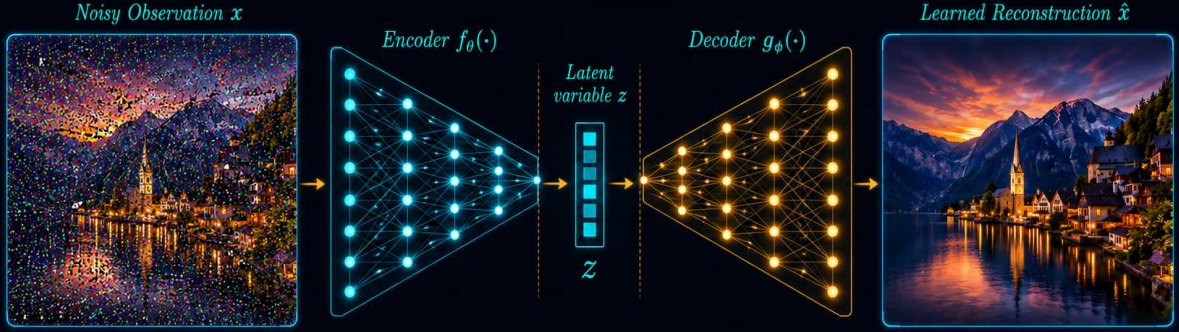
$$x = \underbrace{f(x,\alpha)}_{\text{Denoised / Base Component}} + \underbrace{[x - f(x,\alpha)]}_{\text{Residual Term}}$$

Residual term ↔ learned residual mapping

- ResNets learn residuals instead of full mappings
- Skip connections preserve information
- Optimization is *easier*
- Helps *mitigate vanishing gradients*

AUTOENCODERS FOR DENOISING

From latent representations to residual learning



Learns a compact latent representation of structure

Maps noisy observations to clean reconstructions

Provides a bridge to modern residual denoisers

AUTOENCODER ARCHITECTURE

Encoder, bottleneck, and decoder

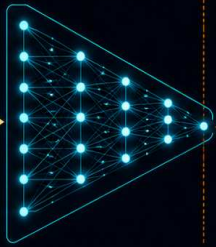
Input image patch

$x \in \mathbb{R}^n$

Vectorize

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

Encoder $f_\theta(\cdot)$



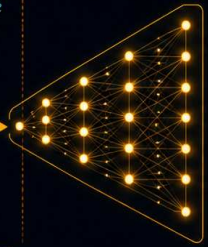
Latent code

$z \in \mathbb{R}^m$

z

$m \ll n$

Decoder $g_\phi(\cdot)$

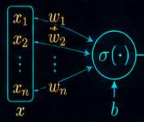


Reconstruction

$\hat{x} \in \mathbb{R}^n$

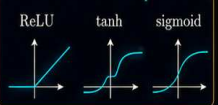
$$z = f_\theta(x)$$
$$\hat{x} = g_\phi(z)$$
$$\hat{x} = g_\phi(f_\theta(x))$$
$$m \ll n$$

Single neuron



$$a = \sigma(w^T x + b)$$

Activation examples



Compression

Maps high-dimensional input $x \in \mathbb{R}^n$ to a compact latent code $z \in \mathbb{R}^m$.

Feature Extraction

The encoder $f_\theta(\cdot)$ learns informative representations captured in z.

Reconstruction

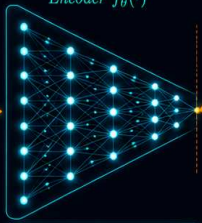
The decoder $g_\phi(\cdot)$ maps z back to the input space to reconstruct $\hat{x} \approx x$.

LATENT VARIABLES AND THE BOTTLENECK

Why compression can suppress noise

Noisy observation $\tilde{x}$

Encoder $f_\theta(\cdot)$

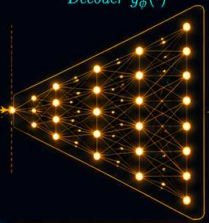


Compressed latent space

$z \in \mathbb{R}^m, \quad m \ll n$

z

Decoder $g_\phi(\cdot)$



Denoised reconstruction $\hat{x}$

$\tilde{x} = x + n$
(noisy observation)

$z = f_\theta(\tilde{x})$
(encode and compress)

$z \in \mathbb{R}^m, \quad m \ll n$
(low-dimensional code)

$\hat{x} = g_\phi(z)$
(decode and reconstruct)

$\hat{x} \approx x$
(denoised reconstruction)

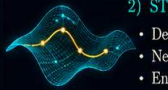
The encoder keeps the dominant structure and discards part of the random perturbation.

$$x \approx g_\phi(z), \quad z = \arg\min_z \|x - g_\phi(z)\|_2^2$$ (project onto a low-dimensional manifold learned by the decoder)

★ A bottleneck encourages the network to model structure rather than memorize pixel noise.

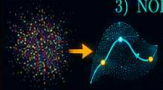
1) COMPACT REPRESENTATION

- Bottleneck forces a short code.
- Only essential information fits.
- $m \ll n \rightarrow$ compression.



2) STRUCTURAL PRIORS

- Decoder defines a smooth manifold.
- Nearby codes → similar outputs.
- Encodes strong inductive biases.



3) NOISE REJECTION

- Random noise lies off the manifold.
- Projection removes off-manifold noise.
- Reconstruction is cleaner and stable.

DENOISING AUTOENCODER TRAINING

Supervised learning from noisy-clean pairs

Training Data: Noisy–Clean Pairs $\{(\tilde{x}^{(i)}, x^{(i)})\}_{i=1}^{N}$

Noisy Inputs $\tilde{x}^{(i)}$ *(corrupted)*

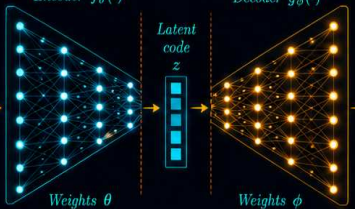
Clean Targets $x^{(i)}$ *(ground truth)*

Corruption model (additive noise)

$$\tilde{x}^{(i)} = x^{(i)} + n^{(i)}$$

$n^{(i)} \sim$ Noise distribution (e.g., $\mathcal{N}(0, \sigma^2 I)$)

Denoising Autoencoder



$$\hat{x}^{(i)} = D_\Theta(\tilde{x}^{(i)}) = g_\phi\left(f_\theta(\tilde{x}^{(i)})\right)$$

with $\Theta = (\theta, \phi)$

Notes

- Backpropagation through the entire network
- Train with mini-batches for efficiency
- Weights $\Theta = (\theta, \phi)$ are optimized to minimize $L(\Theta)$

Reconstruction & Loss

Reconstruction $\hat{x}^{(i)}$ *(output)*

Target $x^{(i)}$ *(ground truth)*

Loss functions (per mini-batch of size N)

$$L_{\text{MSE}}(\Theta) = \frac{1}{N}\sum_{i=1}^{N} \left\|\hat{x}^{(i)} - x^{(i)}\right\|_2^2$$

$$L_{\text{MAE}}(\Theta) = \frac{1}{N}\sum_{i=1}^{N} \left\|\hat{x}^{(i)} - x^{(i)}\right\|_1$$

$$L(\Theta) = \lambda_2 L_{\text{MSE}} + \lambda_1 L_{\text{MAE}}$$

$\lambda_1, \lambda_2 > 0$ are weighting coefficients

Gradient Update (e.g., Adam / SGD)

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta L(\Theta)$$

learning rate $\eta > 0$

Optimized Weights Θ^*

for high-quality reconstructions

DISTORTION METRICS FOR DENOISING

How do we measure reconstruction quality?

MSE

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2$$

RMSE

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$$

MAE

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|x_i - \hat{x}_i|$$

PSNR

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{L^2}{\mathrm{MSE}}\right) = 20\log_{10}\left(\frac{L}{\mathrm{RMSE}}\right)$$

For 8-bit images, $L = 255$

	Ground Truth (x)	*Noisy Observation (y)*	*Candidate A (Sharper)*	*Candidate B (Smoother)*
Visual				
MSE (↓)		1274.45	48.73	32.18
RMSE (↓)		35.71	6.98	5.67
MAE (↓)		26.11	4.77	3.62
PSNR (↑)		17.08 dB	31.25 dB	33.06 dB

INTERPRETATION

- *Lower* MSE / MAE is better
- *Higher* PSNR is better
- Metrics quantify average distortion, but may miss perceptual quality

Metric	Candidate A (Sharper)	Candidate B (Smoother)	What it means
MSE (↓)	48.73	32.18	Candidate A (Sharper) ≠ Candidate B (Smoother)
RMSE (↓)	6.98	5.67	
MAE (↓)	4.77	3.62	
PSNR (↑)	31.25 dB	33.06 dB	Smoother image may have lower MSE / higher PSNR, but less texture and fine detail.

AUTOENCODERS FOR IMAGE DENOISING

Results on grayscale and color images

Clean ground truth x | Noisy observation $\tilde{x}$ | Denoised output $\hat{x}$ (from optimized autoencoder) | *Residual / error map* $|x - \hat{x}|$

Grayscale example
Mountain landscape

MSE: 0.0152 MAE: 0.0857 PSNR: 18.21 dB | MSE: 0.0007 MAE: 0.0201 PSNR: 31.82 dB

Color example
City landscape

MSE: 0.0124 MAE: 0.0743 PSNR: 18.92 dB | MSE: 0.0006 MAE: 0.0186 PSNR: 32.36 dB

Model equations

$$\tilde{x} = x + n$$

$$\hat{x} = D_\Theta(\tilde{x})$$

Takeaway

Autoencoders learn nonlinear mappings from noisy observations to clean reconstructions.

RECONSTRUCTION ARTIFACTS

Where denoising autoencoders can fail

Noisy Input $\tilde{x}$

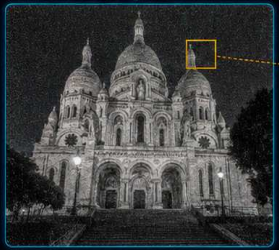
Denoised Output $\hat{x}$

Zoomed Region Comparison

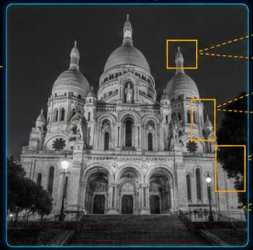
Noisy Input $\tilde{x}$ *Denoised Output* $\hat{x}$

- *over-smoothing* — *fine details are washed out*
- *blur at sharp edges* — *edges lose definition and appear soft*
- *texture loss / regression to the mean* — *textures are simplified and lose variance*
- *faint blocking or checkerboard artifacts* — *subtle grid-like patterns may appear*

Noisy Input $\tilde{x}$

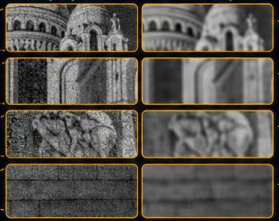


Denoised Output $\hat{x}$



Zoomed Region Comparison

Noisy Input $\tilde{x}$ *Denoised Output* $\hat{x}$



- *over-smoothing* — *fine details are washed out*
- *blur at sharp edges* — *edges lose definition and appear soft*
- *texture loss / regression to the mean* — *textures are simplified and lose variance*
- *faint blocking or checkerboard artifacts* — *subtle grid-like patterns may appear*

Concepts & Equations

$$\hat{x} = D_\Theta(\hat{x})$$

$$e = x - \hat{x}$$

If the model is overly smooth, $\hat{x}$ tends toward conditional means

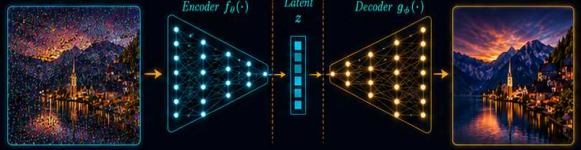
Key Takeaway

Pixelwise losses such as MSE often favor average-looking solutions, which can reduce variance but blur fine details.

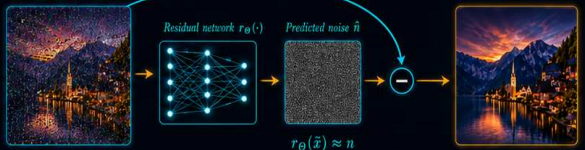
FROM AUTOENCODERS TO RESIDUAL DENOISERS

Learning the noise instead of the image

Traditional Denoising Autoencoder



Residual Denoiser (*Predicts the Noise*)



$$\tilde{x} = x + n$$
$$r_\Theta(\tilde{x}) \approx n$$
$$\hat{x} = \tilde{x} - r_\Theta(\tilde{x})$$

Residual learning often simplifies optimization

Residual block (core idea)

$$y = x + F(x; \Theta)$$

Skip connections help preserve low-frequency content and focus learning on corrections.

Residual learning can preserve edges and textures better

- ✗ *Over-smoothing*
- ✗ *Blurred edges*
- ✗ *Lost textures*

- ✓ *Sharper edges*
- ✓ *More textures*
- ✓ *Better details*

Why it works

Model learns to remove noise while preserving what *already looks good.*

Easier optimization
Learning small residuals is simpler than learning the whole image.

Stronger gradient flow
Skip connections provide short paths for gradients.

Sharper reconstructions
Focus on corrections leads to better fidelity and detail.

Supervised Training with Noisy–Clean Pairs

Autoencoders learn a denoising mapping from examples

$$\mathcal{D} = \{(\tilde{x}^{(i)}, x^{(i)})\}_{i=1}^{N}$$

$$\tilde{x}^{(i)} = x^{(i)} + n^{(i)}, \qquad n^{(i)} \sim \mathcal{N}(0, \sigma^2 I)$$

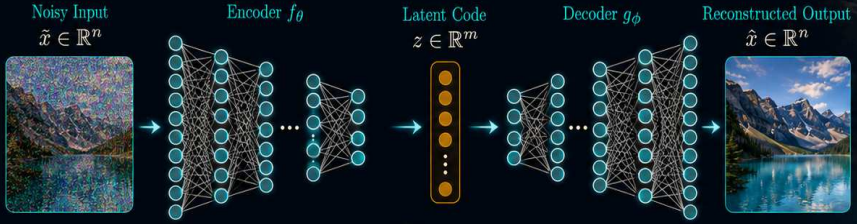
$$D_\Theta : \tilde{x} \mapsto \hat{x} \approx x$$

$$\Theta^\star = \arg\min_\Theta \frac{1}{N} \sum_{i=1}^{N} \ell\left(D_\Theta(\tilde{x}^{(i)}), x^{(i)}\right)$$

- Many *noisy observations* are paired with their *clean targets*
- The network sees examples and *learns parameters*
- *Supervision* defines what counts as a good reconstruction
- After training, the same model can *denoise unseen images*

Autoencoder Forward Model

encoder, latent code, and decoder define the reconstruction mapping



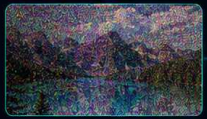
1. $z = f_\theta(\tilde{x})$
2. $\hat{x} = g_\phi(z) = g_\phi(f_\theta(\tilde{x}))$
3. $D_\Theta(\tilde{x}) = \tilde{x}, \quad \Theta = \{\theta, \phi\}$
4. $m = \dim(z) \ll n = \dim(\tilde{x})$
5. $a^{(\ell)} = \sigma\left(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}\right)$

- The encoder compresses the observation into a *low-dimensional latent representation*.
- The decoder *reconstructs the clean signal* from the latent code.
- The bottleneck encourages *structure retention* and *suppresses random noise*.

Supervised Loss Functions

the objective function determines the denoising behavior

Noisy Input $\tilde{x}^{(i)}$



Denoised Output $D_\Theta(\tilde{x}^{(i)})$

Clean Target $x^{(i)}$

1. $$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(D_\Theta(\tilde{x}^{(i)}), x^{(i)}\right) + \lambda \left\|\Theta\right\|_2^2$$

2. $$\ell_{\text{MSE}}(\hat{x}, x) = \frac{1}{n}\sum_{p=1}^{n} (\hat{x}_p - x_p)^2$$

3. $$\ell_{\text{MAE}}(\hat{x}, x) = \frac{1}{n}\sum_{p=1}^{n} |\hat{x}_p - x_p|$$

4. $$\text{PSNR} = 10\log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right)$$

- *MSE* strongly penalizes large errors and often promotes smooth reconstructions.
- *MAE* is more robust to outliers and preserves contrast better in some cases.
- *Regularization* stabilizes learning and improves generalization.
- *PSNR* summarizes fidelity in decibels and is derived directly from the MSE.

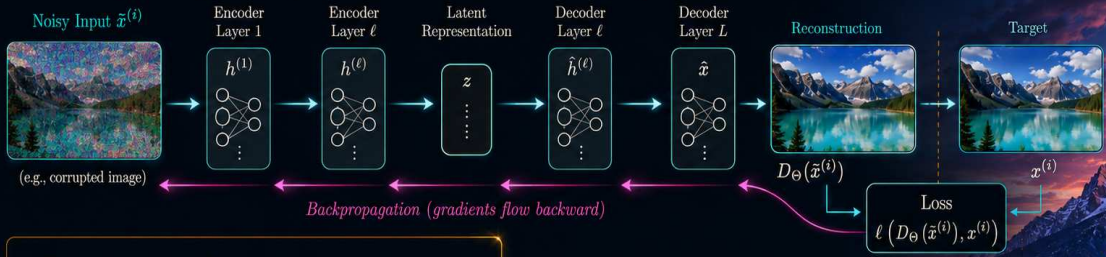
Noisy

Denoised

Target

Backpropagation and Weight Updates

gradients propagate from the reconstruction loss to every layer



1. $L_B(\Theta) = \frac{1}{|B|} \sum_{i \in B} \ell\left(D_\Theta\left(\tilde{x}^{(i)}\right), x^{(i)}\right)$
2. $\nabla_\Theta L_B = \left(\frac{\partial L_B}{\partial \theta}, \frac{\partial L_B}{\partial \phi}\right)$
3. $\Theta_{t+1} = \Theta_t - \eta \, \nabla_\Theta \, L_B(\Theta_t)$
4. $\frac{\partial L}{\partial W^{(\ell)}} = \delta^{(\ell)} \left(h^{(\ell-1)}\right)^T$
5. $\delta^{(\ell)} = \left(\left(W^{(\ell+1)}\right)^T \delta^{(\ell+1)}\right) \odot \sigma'\left(a^{(\ell)}\right)$

- The *forward pass* computes the reconstruction and its loss.
- The *chain rule* propagates error sensitivity backward layer by layer.
- *Gradient descent* or *Adam* updates the weights to reduce reconstruction error.
- *Mini-batches* make optimization efficient and statistically stable.

From Training to Inference

optimized autoencoders become reusable denoisers

1) Training Loop

Mini-batch
sample clean images x
add noise $\tilde{x} = x + n$

↓

Forward Pass
$\hat{x} = \mathcal{D}_\Theta(\tilde{x})$

↓

Loss (Reconstruction)
$L(\Theta) = \mathbb{E}\,\|x - \hat{x}\|_2^2$

↓

Update Parameters
$\Theta^* = \arg\min_\Theta L(\Theta)$

✦ After optimization, the same network denoises *previously unseen images.*

2) Inference (Denoising New Images)

Noisy Input $\tilde{x}$ → Denoised Output $\hat{x}$

$\tilde{x} = \mathcal{D}_{\Theta^*}(\tilde{x})$

Zoomed Patch

Denoising removes noise but may introduce slight smoothing (loss of fine texture).

Noisy Input → *Denoised Output*

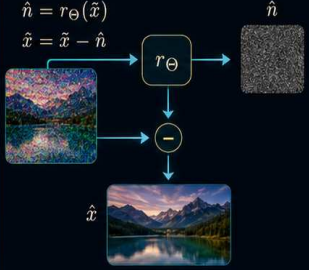
3) Residual Learning View

Noise as the target:

$\tilde{x} = x + n$

$\hat{n} = r_\Theta(\tilde{x})$

$\hat{x} = \tilde{x} - \hat{n}$



✦ Residual prediction often makes *learning easier* than direct clean-image prediction.

✦ Typical artifacts include slight *oversmoothing and loss of fine texture.*

✦ Good denoisers balance noise suppression with *detail preservation.*

U-Nets for Image Denoising

Multiscale encoder–decoder networks with skip connections and residual learning

Key Ideas

Multiscale Context

The encoder aggregates context at multiple scales to understand noise structures and semantics.

Skip Connections

Skip connections preserve fine details and sharp edges by fusing low-level features with high-level understanding.

Residual Learning

Predict the residual noise for easier learning and better generalization:

$\hat{r} = x - y, \quad \hat{y} = x - \hat{r}.$

End-to-End Optimization

Trained on noisy/clean image pairs with pixel-wise or perceptual losses for superior denoising performance.

Residual Denoising

$\hat{r} = \mathcal{N}(x)$

$\hat{y} = x - \hat{r}$

Training Objective

$$\mathcal{L} = \lambda_1 \|\hat{y} - y\|_1 + \lambda_2 \|\hat{y} - y\|_2^2 + \lambda_3 \mathcal{L}_{\text{perc}}$$

with ground-truth clean image y.

$\mathcal{L}_{\text{perc}}$: perceptual loss (e.g., VGG features)

Dimensions

Input: $x \in \mathbb{R}^{H\times W\times C}$

Output: $\hat{y} \in \mathbb{R}^{H\times W\times C}$ or $\hat{r} \in \mathbb{R}^{H\times W\times C}$

Typically $C = 1$ (grayscale) or 3 (RGB)

U-Net Architecture: Encoder, Decoder, and Skip Connections

Multiscale fully convolutional denoisers

Input (Noisy)

$x \in \mathbb{R}^{H \times W \times 3}$

Output (Denoised)

$\hat{y} \in \mathbb{R}^{H \times W \times 3}$

Contracting Path (Encoder) | *Bottleneck* | *Expanding Path (Decoder)*

$H \times W$ — 64

$\frac{H}{2} \times \frac{W}{2}$ — 128

$\frac{H}{4} \times \frac{W}{4}$ — 256

$\frac{H}{8} \times \frac{W}{8}$ — 512

1024

512 — 256 — 128 — 64

Key Ideas

Encoder extracts context

Progressively aggregates features at multiple scales while reducing spatial resolution.

Bottleneck builds global representation

Captures wide receptive field information for robust denoising.

Decoder reconstructs detail

Progressively upsamples and refines features to recover high-resolution details.

Skip connections preserve detail

Bridge matching scales to retain edges and fine textures that might be lost during downsampling.

Core Equations

Convolutional update (per layer l):

$$z_{l+1} = \phi\left(W_l * z_l + b_l\right)$$

Skip connection (copy feature):

$$s_l = z_l$$

Network output:

$$\hat{y} = f_\theta(x)$$

Legend

- 3×3 Conv + BN + ReLU
- 2×2 Max Pooling (stride 2)
- 2×2 Up-Convolution (stride 2)
- Concatenate (Skip Connection)

Convolutional Block (per scale)

BN → ReLU

(Applied twice at each scale)

Resolution & Channels

Spatial size: $H \times W \rightarrow \frac{H}{2} \times \frac{W}{2} \rightarrow \frac{H}{4} \times \frac{W}{4} \rightarrow \frac{H}{8} \times \frac{W}{8}$

Channels: $64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64$

Skip Connections

Features from encoder are concatenated with decoder features at the same scale.

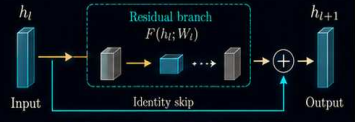
These connections help the network preserve edges, textures, and fine-grained details.

Residual Networks Inside U-Nets

Local residual blocks and global residual learning for denoising

1) Residual Block (local)

Learn a residual mapping $F(h_l; W_l)$

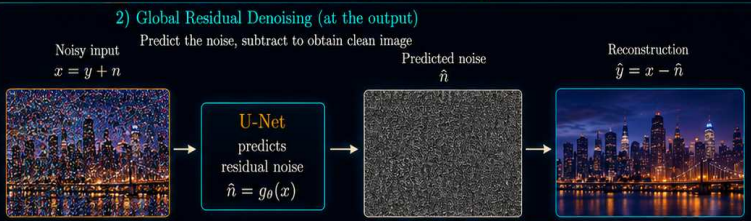


$$h_{l+1} = h_l + F(h_l; W_l)$$

- If $F(h_l; W_l) = 0$, the block becomes identity.
- Eases optimization and preserves information.

2) Global Residual Denoising (at the output)

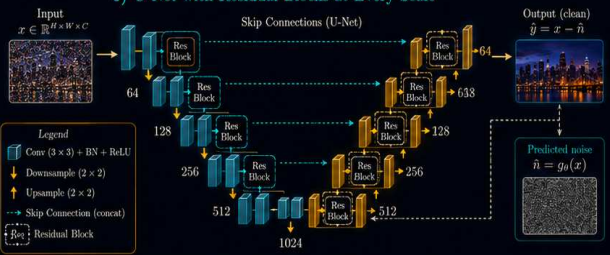
Predict the noise, subtract to obtain clean image



$$\hat{n} = g_\theta(x), \qquad \hat{y} = x - \hat{n}$$

Trained with ground-truth clean image y.
Loss: $\mathcal{L} = \lambda_1 \|\hat{n} - n\|_1 + \lambda_2 \|\hat{y} - y\|_1$

3) U-Net with Residual Blocks at Every Scale



4) Skip Connections (U-Net) vs Residual Connections (ResNet)

Skip Connections in U-Nets	Residual Connections in ResNets
Bridge encoder and decoder across scales.	Operate within each block at the same scale.
Preserve spatial details and high-frequency information.	Learn residual mappings near identity.
Enable gradient flow across long distances in the network.	Easier optimization and deeper networks.

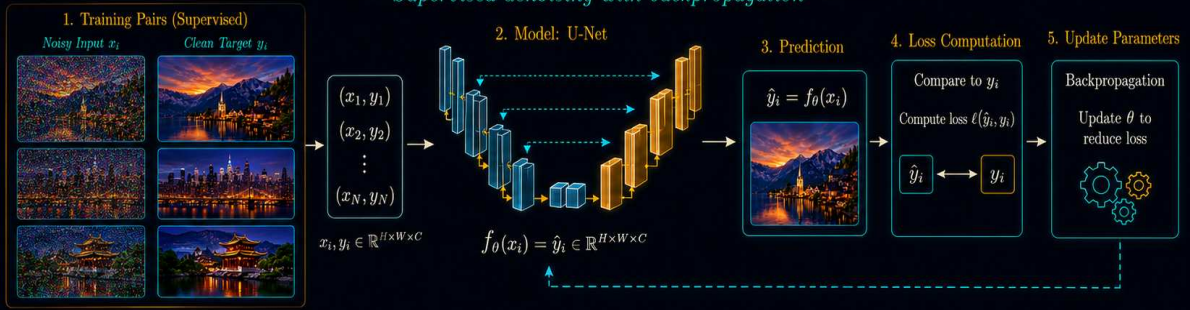
Why Residual Learning Helps

- **Eases Optimization** — Learns residuals near zero, making training stable.
- **Preserves Identity** — Carries forward important signal unaltered.
- **Improves Gradient Flow** — Mitigates vanishing gradients in deep architectures.

U-Nets leverage skip connections across scales and residual blocks within each level, while global residual learning predicts and removes noise.

TRAINING OBJECTIVE AND OPTIMIZATION

Supervised denoising with backpropagation



Objective: Minimize Expected Loss

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \ell\left(f_\theta(x_i), y_i\right)$$

N: number of training pairs

Common Loss Functions

MSE (L2) $\quad \ell_{\text{MSE}} = \frac{1}{HWC} \|\hat{y} - y\|_2^2$

MAE (L1) $\quad \ell_{\text{MAE}} = \frac{1}{HWC} \|\hat{y} - y\|_1$

PSNR (Quality) $\quad \text{PSNR} = 10 \log_{10}\left(\frac{MAX^2}{MSE}\right)$

H, W: height and width, C: channels,
MAX: maximum pixel value (e.g., 1 or 255)

Optional Perceptual Term

$$\ell = \lambda_1 \ell_{\text{MSE}} + \lambda_2 \ell_{\text{MAE}} + \lambda_3 \ell_{\text{perc}}$$

ℓ_{perc}: perceptual loss using features $\phi(\cdot)$ from a fixed pretrained network (e.g., VGG).

$\lambda_1, \lambda_2, \lambda_3 \geq 0$: trade-off weights

How Backpropagation Updates θ

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$$

η: learning rate
θ: all trainable parameters

Training Notes

- Optimizers: Adam or SGD
- Mini-batches for efficiency
- Learning rate: tune or schedule
- Monitor validation PSNR to track generalization

What the loss means: The loss measures the discrepancy between the predicted clean image $\hat{y}$ and the ground-truth y. Lower loss $\Rightarrow$ closer reconstructions and higher perceptual quality.

Goal: Learn parameters θ that map noisy inputs x to clean outputs y by minimizing the expected loss over the training distribution.

U-Net Denoising Results: Grayscale

Edge preservation, texture recovery, and artifact analysis

1) Clean Reference (y)
Ground-truth clean image

2) Noisy Input (x)
$\sigma = 25$ (Gaussian noise)

3) U-Net Denoised Output ($\hat{y}$)
$\hat{y} = f_\theta(x)$

4) Predicted Residual / Noise ($\hat{n}$)
$\hat{n} = x - \hat{y}$

Quantitative Metrics

	MSE (↓)	MAE (↓)	PSNR (↑)
Noisy Input (x)	6.250e-02	1.782e-01	20.18 dB
U-Net Output ($\hat{y}$)	1.385e-03	2.872e-02	32.76 dB

Model Equations

$$\hat{y} = f_\theta(x) \quad \textit{(denoised output)}$$

$$\hat{n} = x - \hat{y} \quad \textit{(predicted residual / noise)}$$

Typical Artifacts

Slight oversmoothing in flat regions
May reduce subtle gradients and shading.

Loss of micro-texture
Fine repetitive textures can be suppressed.

Occasional blur near high-contrast edges
Edge transitions may appear slightly softened.

U-Net Denoising Results: Color Images and Zoomed Artifacts

Residual learning improves detail, but reconstruction artifacts remain

Clean RGB Image y | Noisy RGB Image $x = y + n$ | U-Net Denoised Image $\hat{y} = x - \hat{n}$ | Residual / Error Map $x - \hat{y}$

$$x = y + n$$
$$\hat{n} = g_\theta(x)$$
$$\hat{y} = x - \hat{n}$$
$$x, y, \hat{y} \in \mathbb{R}^{H \times W \times 3}$$
$$n, \hat{n} \in \mathbb{R}^{H \times W \times 3}$$

1 Recovered edges
2 2 Color boundaries
3 3 Remaining artifacts

Zoomed Crops: Details and Artifacts

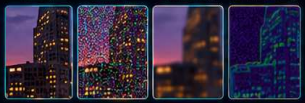
1 Recovered edges (building details)

Clean y | Noisy x | Denoised $\hat{y}$ | Residual $x - \hat{y}$

Edges and fine structures are well recovered, with mild haloing around high-contrast boundaries.

2 Preserved color boundaries (building / sky)

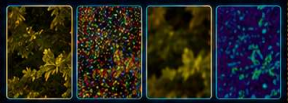
Clean y | Noisy x | Denoised $\hat{y}$ | Residual $x - \hat{y}$



Color transitions are largely preserved, with slight smoothing near sharp boundaries.

3 *Remaining artifacts (foliage / texture)*

Clean y | Noisy x | Denoised $\hat{y}$ | Residual $x - \hat{y}$



Texture is somewhat smoothed toward the mean; faint checkerboard/upsampling artifacts can remain.

Quantitative Comparison (RGB)

Method	MSE ↓	MAE ↓	PSNR ↑
Noisy (x)	0.01874	0.08962	17.27 dB
U-Net Denoised ($\hat{y}$)	0.00231	0.03018	26.37 dB
Improvement	8.1×	3.0×	+9.10 dB

Residual Learning

Predicting noise instead of the clean image enables better detail recovery and faster convergence.

What Improves

Edges, structures, and color boundaries are well recovered, with high perceptual fidelity.

What Remains

Some smoothing toward the mean, mild haloing, and faint checkerboard artifacts in textured regions.

Takeaway

U-Nets deliver strong denoising performance, but perfect reconstruction is limited by capacity, training data, and upsampling.

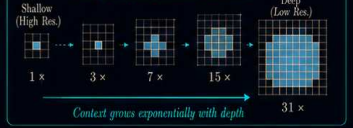
Why U-Nets Work for Denoising

Multiscale context, feature reuss, and efficient fully convolutional inference

1) Multiscale Receptive Fields

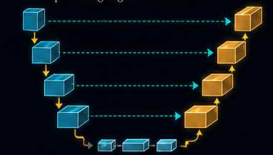
Capture both global context and fine details across scales.

Receptive Field Growth (conceptual)



2) Skip Connections (Feature Reuse)

Fuse high-level semantics with low-level details, preserving edges and textures.



Feature Reuse in Action

Preserve what matters. Reuse what was learned.

3) Residual Learning

Learn the noise (residual) instead of the signal, simplifying optimization.

Denoising mapping:

$$\hat{y} = f_\theta(x)$$

Residual formulation:

$$\hat{y} = x - g_\theta(x)$$

Learn easier target (noise) → faster convergence

identity mapping when $g_\theta(x) \approx 0$

Noisy Input x

Denoised Output $\hat{y}$

Efficiency: Fully Convolutional Inference

Fully Convolutional	*Patch-Free Prediction*	*Mixed Precision*	*Tiled Processing*
Same network works on any image size.	Predict full images without patch stitching artifacts.	FP16 Lower memory and bandwidth, faster inference.	Process large images in tiles with overlap.

Efficient, scalable, and deployment-friendly.

Why It's Efficient (per convolution layer)

$$\text{Cost} \approx O\left(HW\,k^2\,C_{in}C_{out}\right)$$

All operations are convolutions → highly optimized on modern HW.

At a Glance: What U-Nets Deliver

- ✓ Global *context* from deep layers
- ✓ Precise *details* via *skip connections*
- ✓ Stable training with *residual learning*
- ✓ Scalable, *efficient*, and *patch-free* inference
- ✓ Superior *denoising* across diverse noise types

Better Quality, Faster Inference, Lower Cost

Takeaway: *U-Nets combine global context with local detail, making them powerful denoisers and a natural bridge toward residual architectures.*

Example: Periodic Replication

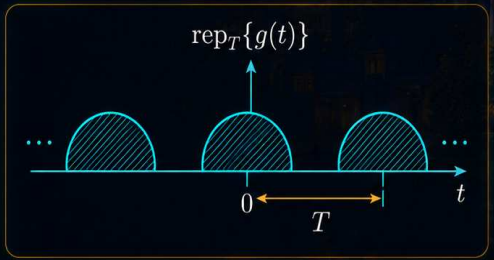
✧ We know:

$$\sum_{n=-\infty}^{\infty} \delta(t-n) \;\leftrightarrow\; \frac{1}{T}\sum_{m} \delta\left(u-\frac{m}{T}\right)$$

✧ Now consider:

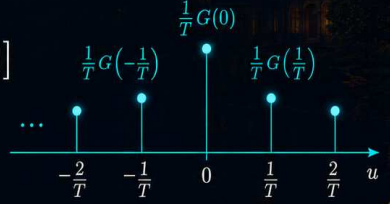
$$\text{rep}_T\{g(t)\} \;=\; \sum_{m} g(t-mT)$$

$$\text{rep}_T\{g(t)\} \;=\; g(t) * \sum_{m} \delta(t-mT)$$

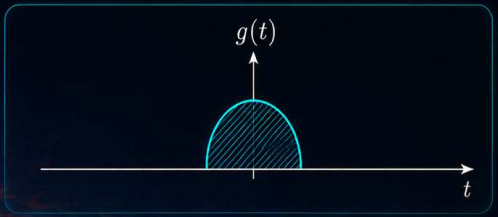


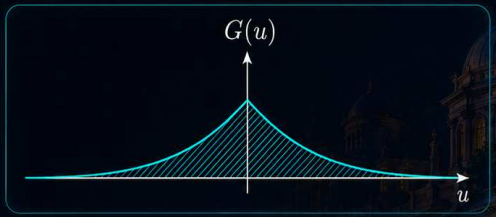
Fourier Transform of the Repeated Signal

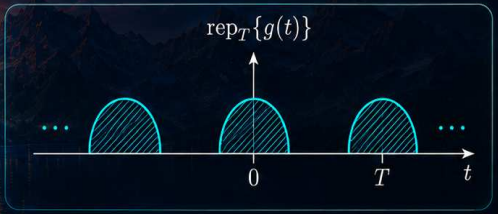
$$
\begin{aligned}
F\{\mathrm{rep}_T(g(t))\} &= F\Big\{g(t) * \sum_m \delta(t - mT)\Big\} \\
&= G(u)\frac{1}{T}\sum_m \delta\Big(u - \frac{m}{T}\Big) \\
&= \frac{1}{T}\sum_m G\Big(\frac{m}{T}\Big)\delta\Big(u - \frac{m}{T}\Big) \\
&= \frac{1}{T}\,\mathrm{comb}_{\frac{1}{T}}[G(u)]
\end{aligned}
$$

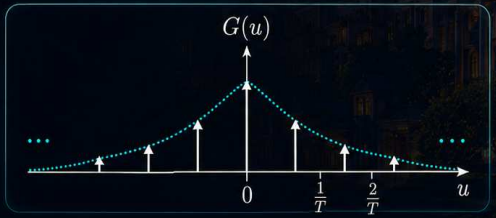


Periodic Replication and Spectral Sampling







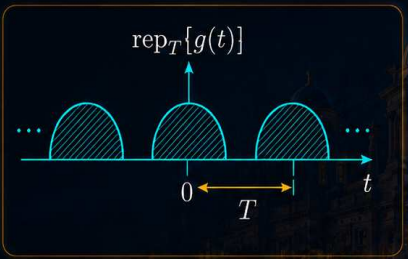


$$\mathcal{F}\{\mathrm{rep}_T\{g(t)\}\} = \frac{1}{T}\sum_{m=-\infty}^{\infty} G\left(\frac{m}{T}\right)\delta\left(u-\frac{m}{T}\right)$$

Fourier-Series Interpretation

✧ Hence:

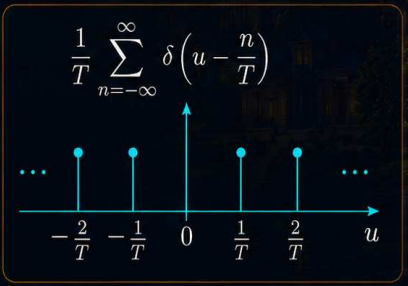
$$\text{rep}_T[g(t)] = g(t) * \frac{1}{T} \sum_n e^{j2\pi n u_0 t}$$



✧ Taking Fourier Transforms:

$$F\{\text{rep}_T[g(t)]\} = G(u) \cdot \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta\left(u - \frac{n}{T}\right)$$

$$= \frac{1}{T} \sum_n G(u)\, \delta\left(u - \frac{n}{T}\right)$$



Final Form of the Replicated-Signal Spectrum

✦ Hence:

$$\text{rep}_T[g(t)] = g(t) * \frac{1}{T}\sum_n e^{j2\pi n u_0 t}$$

✦ Taking Fourier Transforms:

$$\begin{aligned} F\{\text{rep}_T[g(t)]\} &= G(u)\cdot\frac{1}{T}\sum_{n=-\infty}^{\infty}\delta\left(u-\frac{n}{T}\right) \\ &= \frac{1}{T}\sum_n G(u)\delta\left(u-\frac{n}{T}\right) \\ &= \frac{1}{T}\sum_{n=-\infty}^{\infty} G\left(\frac{n}{T}\right)\delta\left(u-\frac{n}{T}\right) \end{aligned}$$

$$= \frac{1}{T}\,comb_{\frac{1}{T}}[G(u)]$$

Replicated-Signal Spectrum

A line spectrum with spacing $\frac{1}{T}$, with weights $\frac{1}{T}G\left(\frac{n}{T}\right)$ at $u=\frac{n}{T}$.

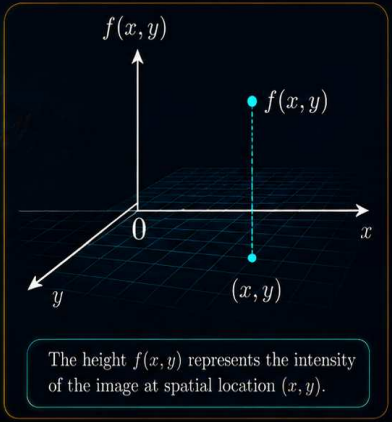
2-Dimensional Systems

In 2 dimensions, the input and output of a system are functions of two independent variables. In image processing, the variables are the spatial coordinates (x, y) and the value of the function is the intensity of the image at that point.

- A 2-D system is described by a function of two variables.

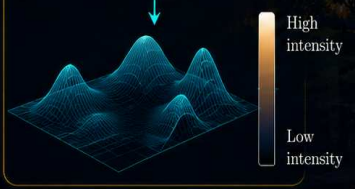
$$f : \mathbb{R}^2 \to \mathbb{R}$$

- For images, $f(x, y)$ gives the intensity (brightness) at the spatial location (x, y).
- Here, x and y are spatial coordinates (horizontal and vertical positions).



The height $f(x, y)$ represents the intensity of the image at spatial location (x, y).

Example: Image Intensity Function



- Thus, a 2-D system maps each point in the plane (x, y) to a value $f(x, y)$ — the image intensity at that point.

Impulse Response and 2-D Convolution

✧ Impulse response characterizes a linear shift-invariant 2D system ✧

$$\delta(x,y) \longrightarrow \boxed{\text{System}} \longrightarrow h(x,y)$$

$$\ell(x,y) \longrightarrow \boxed{\text{System}} \longrightarrow g(x,y)$$

✧ Input–output relation

$$\ell(x,y) \rightarrow \boxed{\text{System}} \rightarrow g(x,y) = h(x,y) * \ell(x,y)$$

✧ 2-D convolution

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x-\sigma, y-\beta)\ell(\sigma,\beta)\, d\sigma\, d\beta$$

Input image

$\ell(x,y)$

$*$

Kernel $h(x,y)$

1	1	1
1	−8	1
1	1	1

$=$

Output image $g(x,y)$

$g(x,y)$