Optimization for Computer Vision

Lecture II - Image Filtering

CentraleSupélec- Masters in Math and AI

December 10th, 2024

CentraleSupélec

Recap

Last lecture we talked about

- Geometric Primitives
- Geometric Image Formation
- Photometric Image Formation

This Lecture

Today we will discuss

- The short time Fourier transform and its relation to analyzing localized frequency (texture)
- The continuous wavelet transform
- Image Filtering

Images as functions

Today it will be useful to think about images not only as being represented by a 2D array of numbers on a computer but also as a 2D continuous function mapping from some domain Ω (usually a square) to the set $[0, 1]$ representing pixel intensity in grey-scale.

Reminder on the Fourier transform

The Fourier transform is defined for integrable functions f on $\mathbb{R}$ as

$$\mathcal{F}(f)(\xi) = \int_{\mathbb{R}} f(t) e^{-2\pi i t \xi} dt$$

It transforms a **signal** from the time (or space) domain into its frequency domain.

Fourier Identities

Convolutions in time

$$(f * g)(z) := \int_{\mathbb{R}} f(t)g(z-t)dt$$

become **pointwise products** in the frequency domain

$$\mathcal{F}(f * g)(\xi) = \mathcal{F}(f)(\xi)\mathcal{F}(g)(\xi)$$

Proof: change of variables inside the integral.

Fourier Identities

Translation in the time domain (denoted T_τ)

$$T_\tau f(t) = f(t-\tau)$$

becomes **modulation** in the frequency domain (denoted E_ω)

$$\mathcal{F}(T_\tau f)(\xi) = e^{-2\pi i \tau \xi}\mathcal{F}(f)(\xi) = E_{-\tau}\mathcal{F}(f)(\xi)$$

and vice-versa.
Proof: change of variables inside the integral.

Fourier Identities

A **dilation** with $a > 0$

$$D_a f(t) = \frac{1}{\sqrt{a}} f(t/a)$$

becomes a dilation in the other domain

$$\mathcal{F}(D_a f)(\xi) = \sqrt{a}\mathcal{F}(f)(a\xi) = D_{1/a}\mathcal{F}(f)(\xi)$$

Proof: change of variables inside the integral.

2D Fourier Transform

More useful for image processing is the 2D Fourier transform:

$$\mathcal{F}(f)(\xi_1, \xi_2) = \int_{\mathbb{R}^2} f(x, y) e^{-2\pi i \langle \mathbf{x}, \xi \rangle} dxdy$$

In the context of images, this tells us how the pixel intensities vary across both axes of the spatial domain.

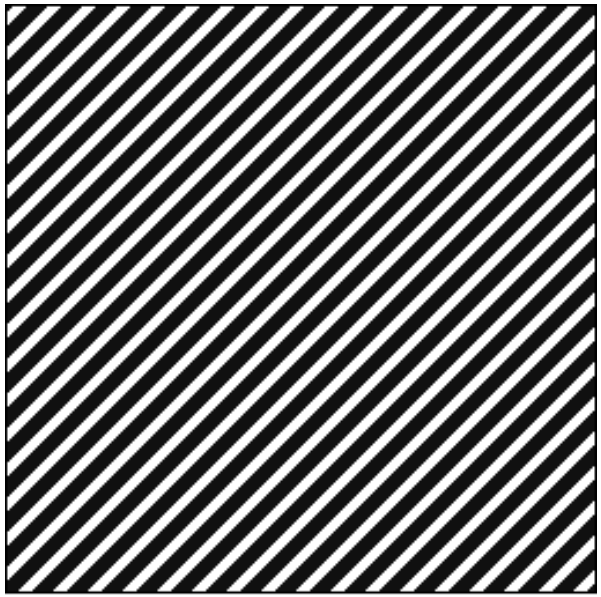
Uses in Image Processing

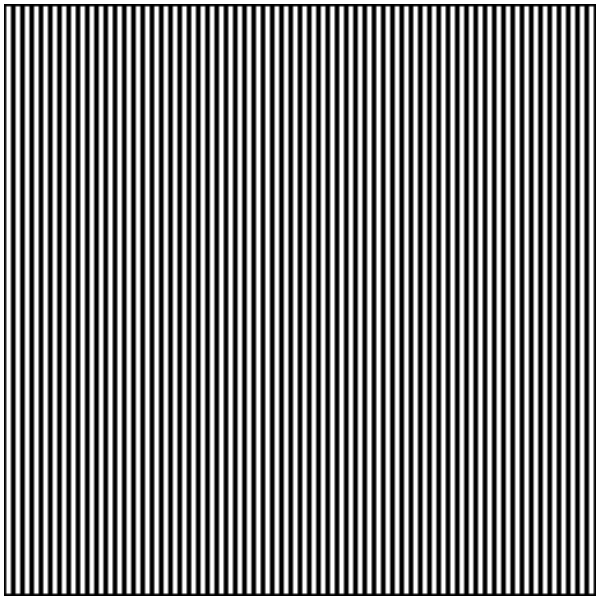
The applications of Fourier Transforms in computer vision are vast. We can filter out frequencies to remove noise or enhance features.

Low-pass filter → pixel intensities vary slowly → smooth an image.
High-pass filter → keep sharp changes in pixel intensity → sharpen image.

Shortcoming of the Fourier transform

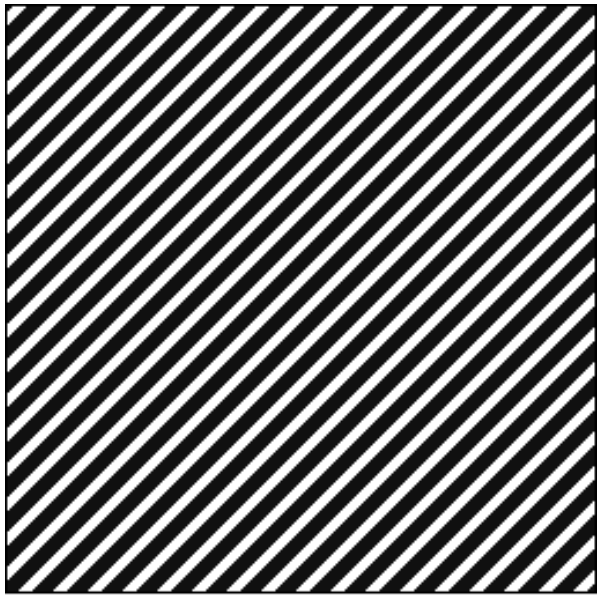
The Fourier transform gives us **global** frequency content of an image. This makes it useful for **stationary** signals/images.

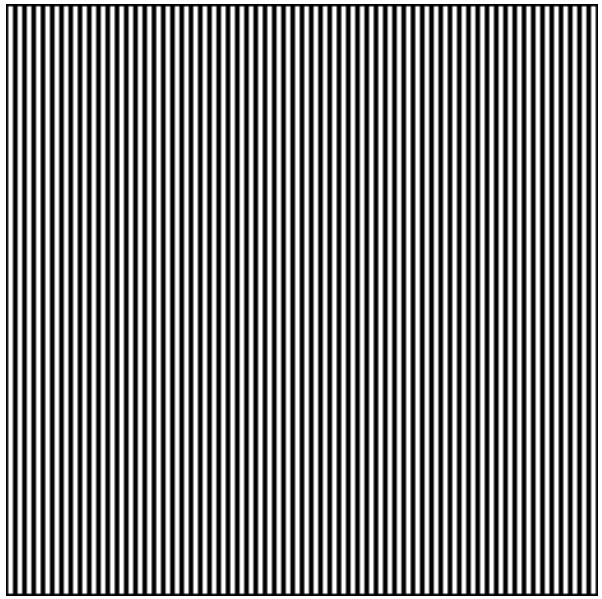




Shortcoming of the Fourier transform

The Fourier transform gives us **global** frequency content of an image. This makes it useful for **stationary** signals/images.





What is more useful for real images is to have information about **localized frequencies**.

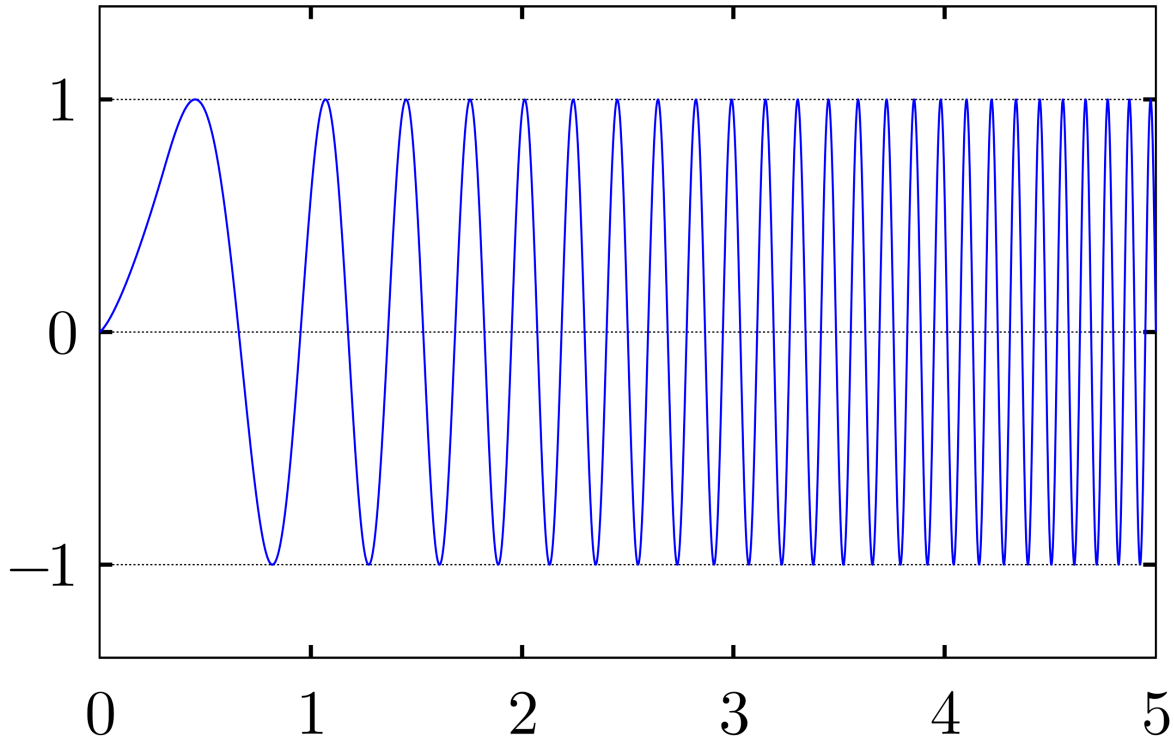
What is a Nonstationary Signal?

The classical example of a nonstationary signal is the linear chirp:

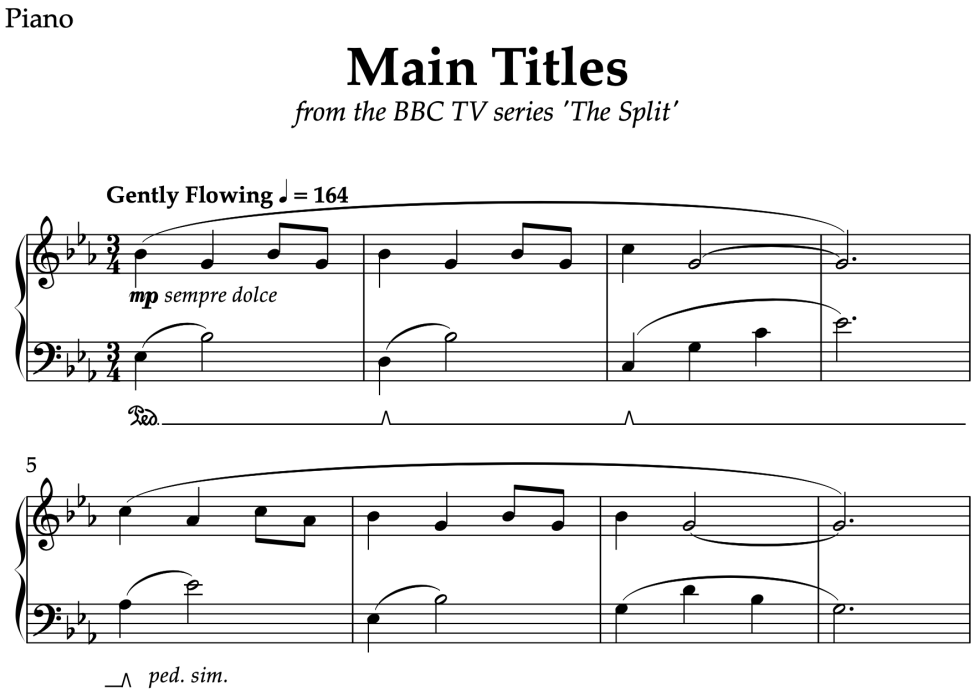
$$f(t) = \sin(c(t)t)$$

where $c(t) = at + b$ for $a, b \in \mathbb{R}$.

The frequency is changing linearly over time.



Sheet Music



If we think about an audio signal or a song, the Fourier transform tells us what notes are present, in what proportions, throughout the entirety of the song. It does not tell us **when** or **where** these frequencies are occurring in time (or space)!

Short Time Fourier Transform (STFT)

This motivates to consider the **Short Time Fourier Transform** or **STFT**. Given some window function ψ (e.g., $\psi(t) = \mathbb{1}_{[-1,1]}(t)$ or $\psi(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(t)^2}$), we define

$$\begin{aligned}\mathcal{F}^{\psi}(f)(\tau,\xi) &= \int_{\mathbb{R}} f(t)\psi(t-\tau)e^{-2\pi i t\xi}dt \\ &= \langle f, E_\xi T_\tau \psi\rangle_{L^2}\end{aligned}$$

This tells us about localized frequencies through the window ψ.

Discrete STFT

Assume that our signal is a vector in $\mathbb{R}^L$ which has been sampled **regularly** in time/space. The discrete STFT is defined by sampling:

$$\{\mathcal{F}^{\psi}(f)(\tau_n, \xi_m)\}_{n\in\mathbb{Z}, m\in\mathbb{Z}}$$

Discrete STFT

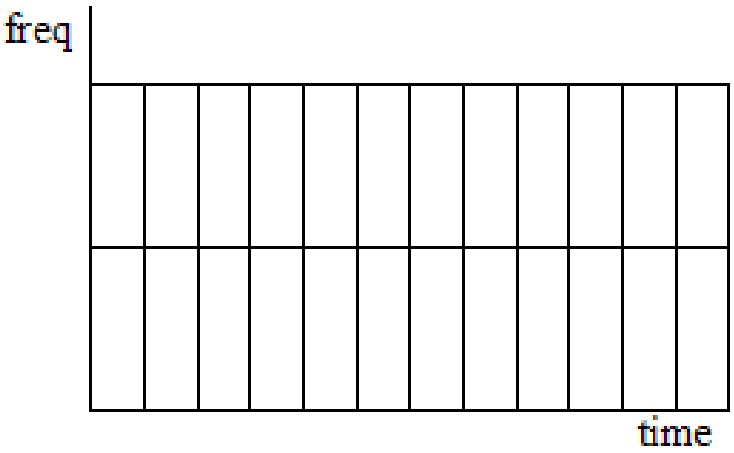
Assume that our signal is a vector in $\mathbb{R}^L$ which has been sampled **regularly** in time/space. The discrete STFT is defined by sampling:

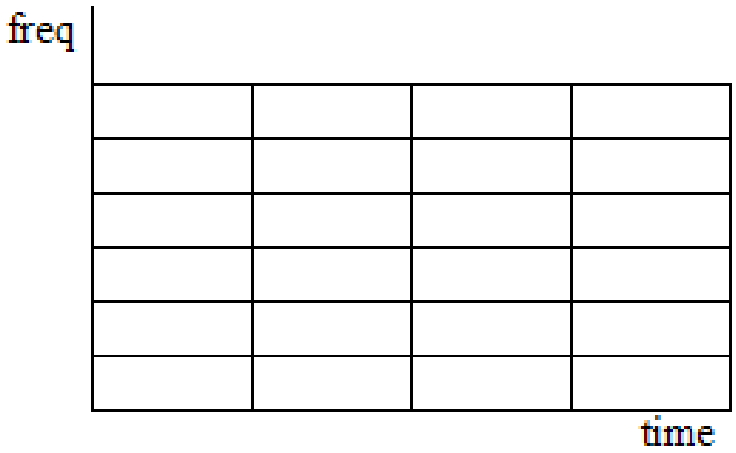
$$\{\mathcal{F}^{\psi}(f)(\tau_n, \xi_m)\}_{n\in\mathbb{Z}, m\in\mathbb{Z}}$$

These functions are called the atoms of the STFT or sometimes a dictionary.

Tiling of the TF-Plane

Due to the **Gabor-Heisenberg uncertainty principle**, we cannot have a window which gives us both high resolution in time and frequency

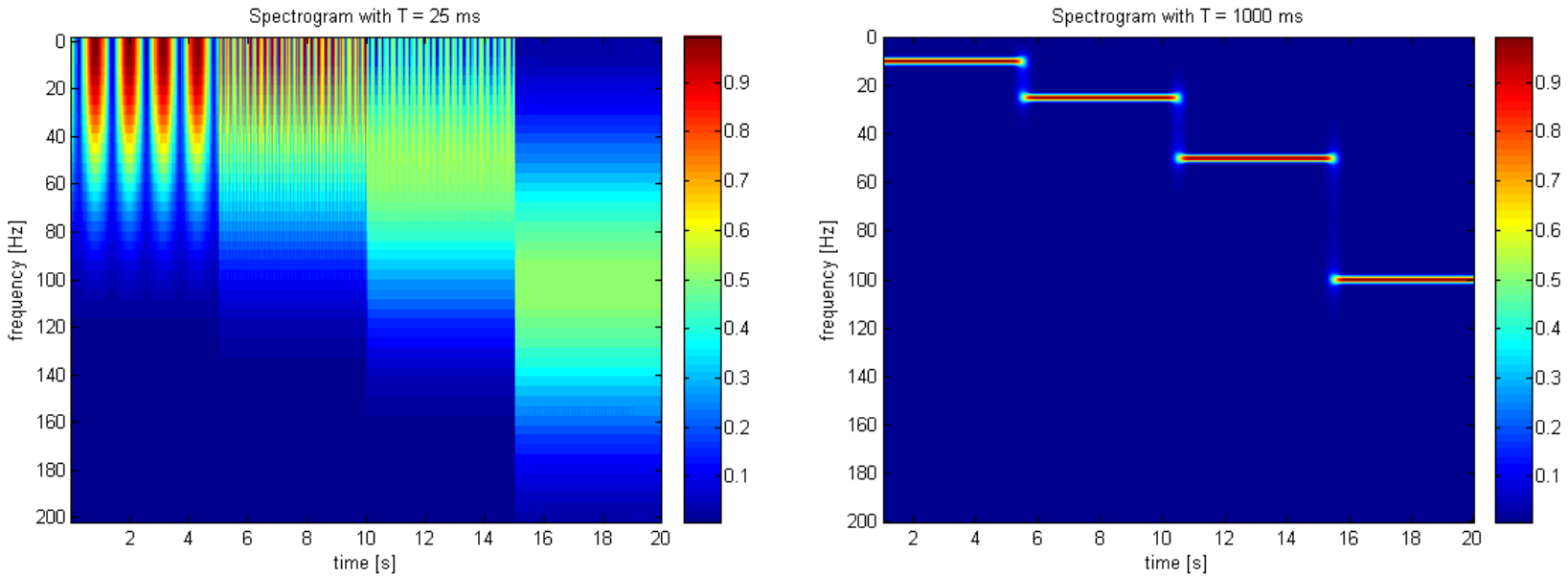




The best simultaneous accuracy is achieved by a Gaussian window and this transform is called the **Gabor Transform**.

Example STFT

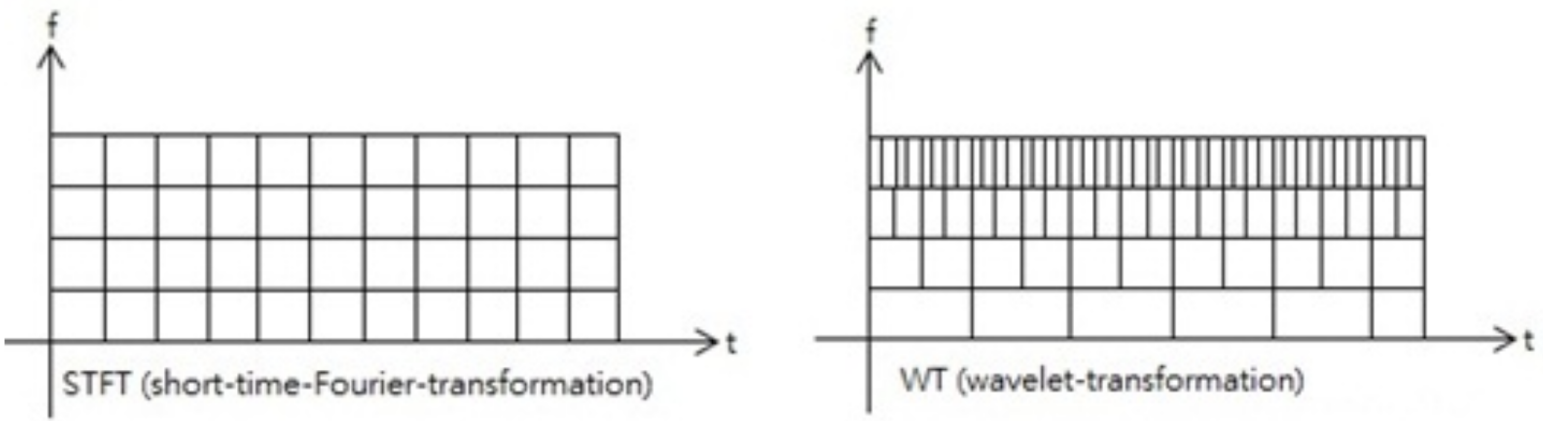
Here we take a uniform window of two different lengths:



The signal is a piecewise combination of four sinusoids of different frequencies.
Left side has good precision in time but bad precision in frequency.
Right has good precision in frequency but bad precision in time.

Motivation for Wavelet Transform

When we constructed the STFT we focused on the transformations T and E and we didn't care a lot about D. The wavelet transform takes a different approach and constructs a time-frequency representation using T and D in the time domain (resp. E and D in frequency).



Continuous Wavelet Transform

Just as we defined the STFT using translations and modulation, we do the same for the **Continuous Wavelet Transform (CWT)** using translations and dilations:

$$\begin{aligned} X_\psi(f)(\alpha,\tau) &:= \frac{1}{\sqrt{\alpha}} \int_{\mathbb{R}} f(t)\bar{\psi}\left(\frac{t-\tau}{\alpha}\right) \\ &= \langle f, D_\alpha T_\tau \psi \rangle_{L^2} \end{aligned}$$

We call ψ the **mother wavelet** which generates the **daughter wavelets** $D_\alpha T_\tau \psi$.

Wavelet Vocabulary

$$X_\psi(f)(\alpha,\tau) := \frac{1}{\sqrt{\alpha}} \int_{\mathbb{R}} f(t)\bar{\psi}\left(\frac{t-\tau}{\alpha}\right)$$

- We say $X_\psi(f)(\alpha,\tau)$ is the wavelet coefficient of the function $f(t)$.
- ψ is also called the analyzing wavelet.
- $a > 0$ is the scale parameter.
- $\tau \in \mathbb{R}$ is the position parameter.

CWT in the Fourier domain

Taking the Fourier transform of the CWT gives the following:

$$\mathcal{F}(X_\psi(f)(\alpha,\tau))(\xi) = \sqrt{\alpha}\mathcal{F}(f)(\xi)\overline{\mathcal{F}\psi}(a\xi)$$

CWT in the Fourier domain

Taking the Fourier transform of the CWT gives the following:

$$\mathcal{F}(X_\psi(f)(\alpha,\tau))(\xi) = \sqrt{\alpha}\mathcal{F}(f)(\xi)\overline{\mathcal{F}\psi}(a\xi)$$

This means, as α varies, we use the same filter in the Fourier domain $\overline{\mathcal{F}\psi}$, with various dilations, but with the same pattern.

How to Compute?

For fixed scale parameter α, we can interpret the CWT as a convolution with the daughter wavelet:

$$X_\psi(f)(\alpha,\tau) := \frac{1}{\sqrt{\alpha}} \int_{\mathbb{R}} f(t)\bar{\psi}\left(\frac{t-\tau}{\alpha}\right)$$

Using the Fourier identities, we can compute this by taking the Fourier transform, multiplying pointwise, and taking the inverse Fourier transform.

Properties of the CWT

There are three main properties of the CWT:

1. It is linear:

$$f(t) = \rho_1 f_1(t) + \rho_2 f_2(t)$$
$$\Longrightarrow X_\psi(f)(\alpha,\tau) = \rho_1 X_\psi(f_1)(\alpha,\tau) + \rho_2 X_\psi(f_2)(\alpha,\tau)$$

2. It is covariant under translation:

$$f_0(t) = T_\gamma f(t) \Longrightarrow X_\psi(f_0)(\alpha,\tau) = X_\psi(f)(\alpha,\tau-\gamma)$$

3. It is covariant under dilation:

$$f_s(t) = f(st) \Longrightarrow X_\psi(f_s)(\alpha,\tau) = \frac{1}{\sqrt{s}} X_\psi(f)(s\alpha,s\tau)$$

Inverse CWT

Grossman and Morlet (1984) showed that the CWT is invertible with the formula:

$$f(t) = \frac{1}{C_\psi} \int_0^{+\infty} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\alpha}} X_\psi(f)(\alpha, \tau) \psi\left(\frac{t-\tau}{\alpha}\right) \frac{d\alpha d\tau}{a^2}$$

whenever the constant C_ψ is finite:

$$C_\psi = \int_{-\infty}^{+\infty} \frac{\|\mathcal{F}\psi(\nu)\|^2}{|\nu|} d\nu$$

is finite. This is the so-called **admissibility** condition of the mother wavelet ψ. It means also that $\mathcal{F}\psi(0) = 0$, i.e., the average of the mother wavelet ψ must be 0.

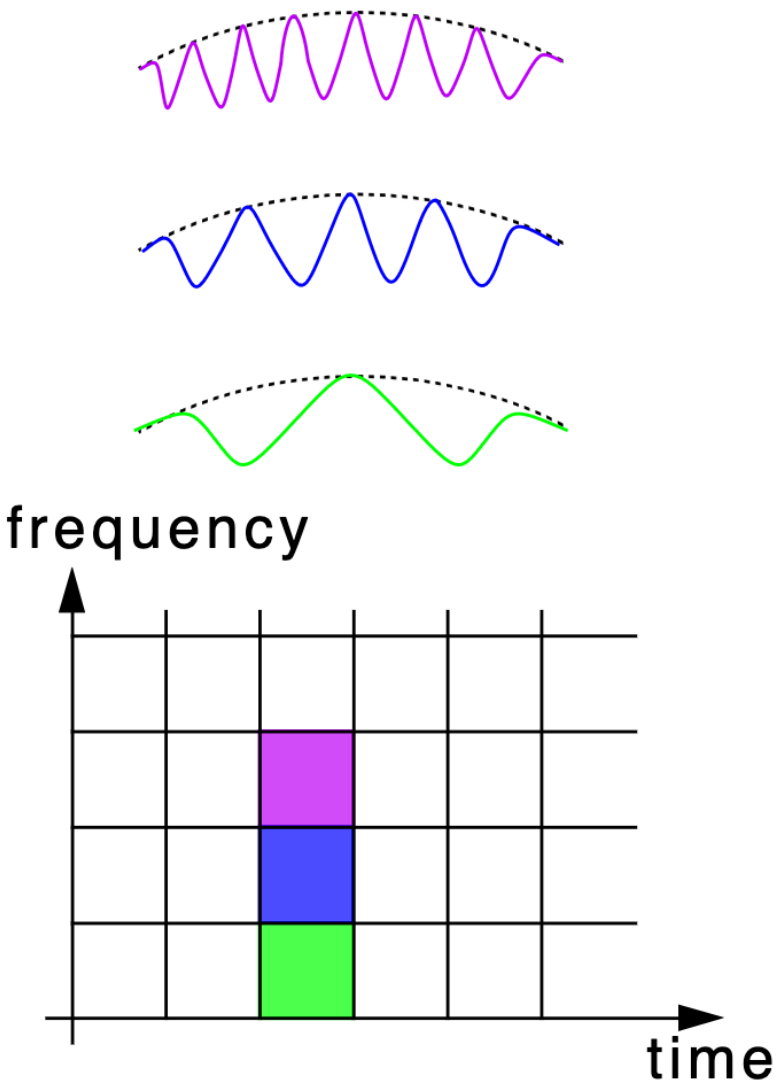
Discrete Wavelet Transform

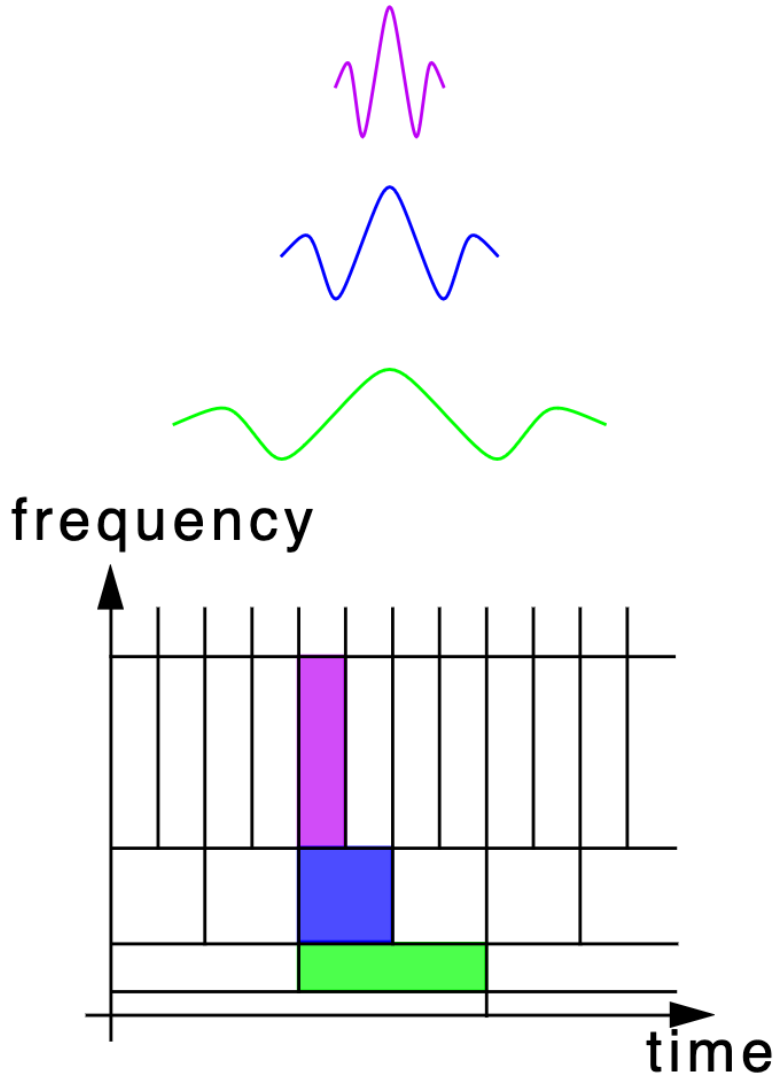
We discretize the scales and times to form the **discrete wavelet transform**

$$\{X_\psi(f)(\alpha_n, \tau_m)\}_{n\in\mathbb{Z}, m\in\mathbb{Z}}$$

We usually take a dyadic discretization of α and τ, e.g., $\alpha_n = 2^n$. For a 2D signal (image) we typically compute for rows and columns separately (tensor product) or using more sophisticated wavelets (curvelets, ridgelets, etc).

Time-Frequency Comparison





Orthogonal Wavelet Bases

The discrete family of wavelets generated by the mother wavelet can either be orthogonal or not. The classical example of orthogonal wavelets is given by the Haar wavelet basis:

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \leqslant t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leqslant t < 1 \\ 0 & \text{otherwise} \end{cases}$$

Used by Alfred Haar as early as 1910!

Difficulties Associated to Wavelet Transform

- Choosing the mother wavelet (orthogonality, vanishing moments)
- Boundary effects (padding, no neighboring points)
- Computational complexity
- Obstructions in higher dimensions

Sampling Operator

How can we represent the sampling operator?
Intuitively, we can write

$$s_T(x) = \begin{cases} 1 & x = nT \quad n \in \mathbb{N} \\ 0 & x \neq nT \quad n \in \mathbb{N} \end{cases}$$

and describe our discrete signal as

$$f[x] = f(xT)s_T(xT).$$

However, this is not so useful for Fourier analysis - the function has a null Fourier transform regardless of the f we take. We can instead write

$$\text{Ш}_T(x) = \sum_{i \in \mathbb{Z}} \delta(x - iT)$$

using the Dirac delta. The function Ш_T is called an **impulse train** or a **Dirac comb** with period T.

Sampling Operator

Using the duality between pointwise multiplication and convolution via the Fourier transform, we can deduce the Fourier transform of the sampled image:

$$\mathcal{F}(\text{LIII}_T\, f)(\xi) = \mathcal{F}(\text{IIII}_T) * \mathcal{F}(f)(\xi) = \frac{1}{T}\,\text{LIII}_{\frac{1}{T}} * \mathcal{F}(f)(\xi)$$

So sampling in time/space corresponds to convolution with a Dirac comb in frequency $\Longrightarrow$ periodized function of ξ.

Aliasing

Sampling Theorem (Shannon-Nyquist):

- If a signal's Fourier transform is supported on $[-B, B]$
- Then we must sample with period $T \leqslant \frac{1}{2B}$ to avoid aliasing
- Or equivalently, sampling frequency must be $f_s \geqslant 2B$

Aliasing

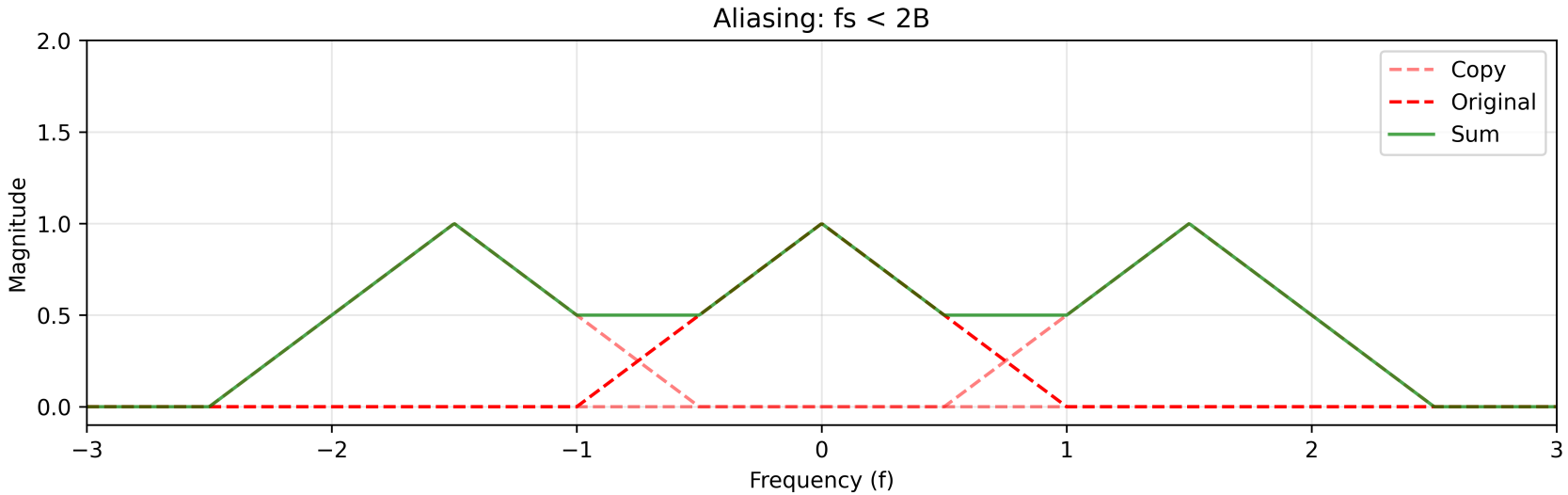
Sampling Theorem (Shannon-Nyquist):

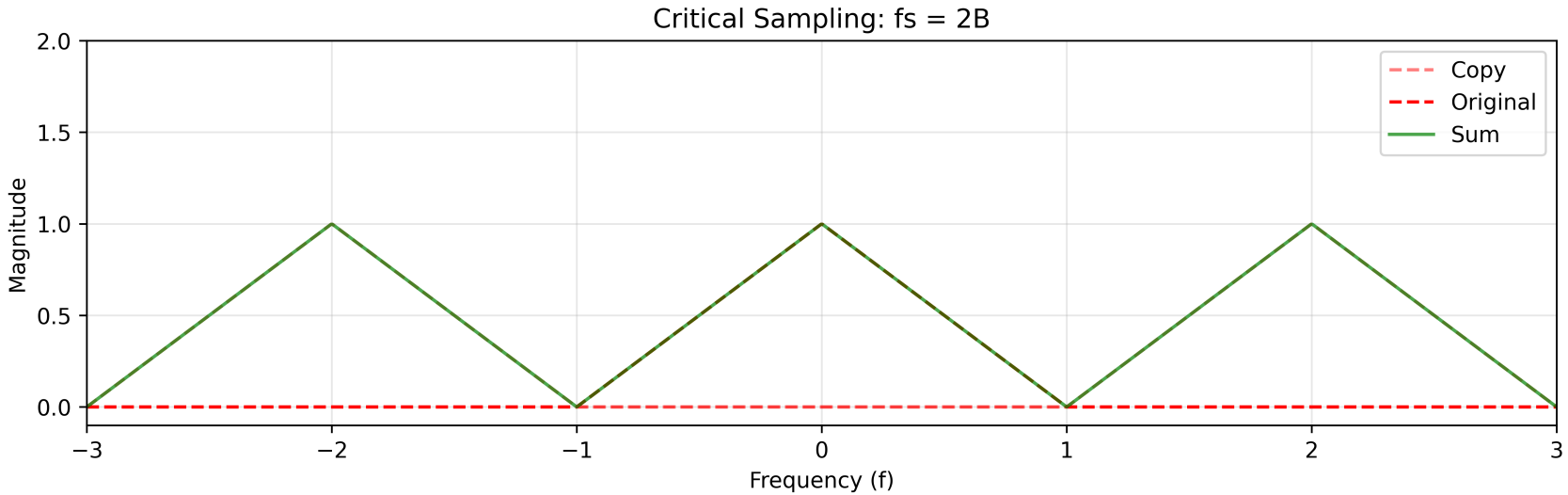
- If a signal's Fourier transform is supported on $[-B, B]$
- Then we must sample with period $T \leqslant \frac{1}{2B}$ to avoid aliasing
- Or equivalently, sampling frequency must be $f_s \geqslant 2B$

Why?

- Sampling in time creates periodic copies in frequency
- Period of copies = sampling frequency
- If period too small, copies overlap (aliasing)

Aliasing: Visualization





Clean: fs > 2B

Reconstruction and Anti-aliasing

Perfect Reconstruction:

- Multiply frequency domain by ideal low-pass filter (brick wall)
- Equivalent to convolution with sinc function in time domain
- $\text{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$

Reconstruction and Anti-aliasing

Perfect Reconstruction:

- Multiply frequency domain by ideal low-pass filter (brick wall)
- Equivalent to convolution with sinc function in time domain
- $\text{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$

In Practice:

- Use smooth filters (e.g., Gaussian) instead of brick wall
- Anti-alias before sampling to prevent overlap

Point Operators

We will study **operators** on images which are functions that take one or more input images and produces an output image.

$$g(\mathbf{x}) = h(f(\mathbf{x}))$$

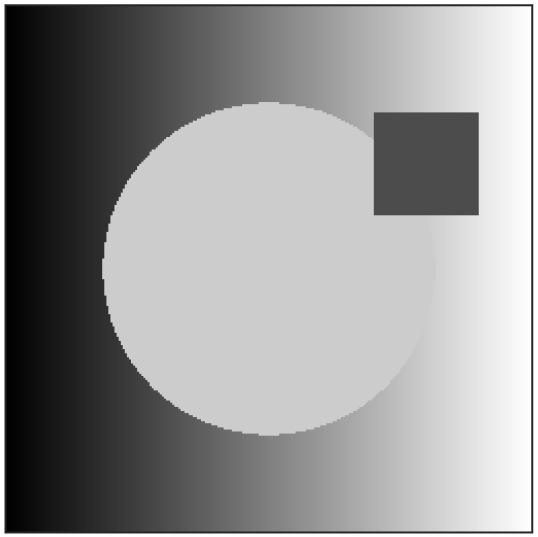
Here $\mathbf{x}$ is the domain of the image and $f(\mathbf{x})$ is the image itself. The simplest kinds of operators are **point operators** where each pixel in the output image depends only on the corresponding pixel value in the input image.

$$g(\mathbf{x}) = cf(\mathbf{x}) + b$$

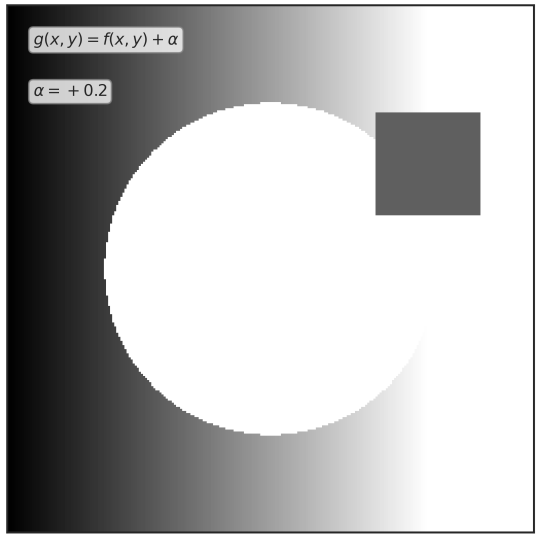
Linear point operators correspond to uniform brightness and contrast transformations.

Point Operators

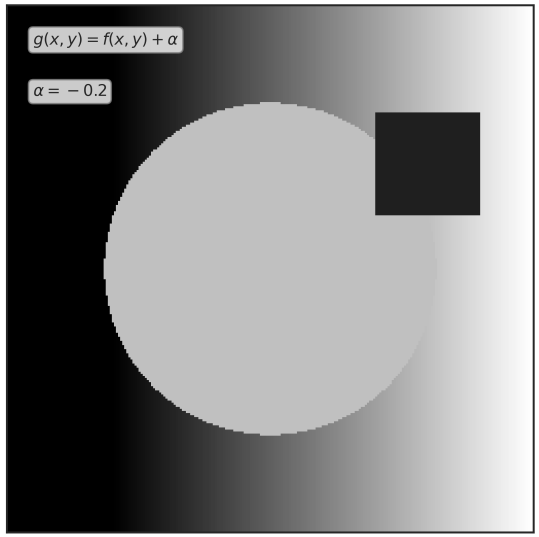
Original Image



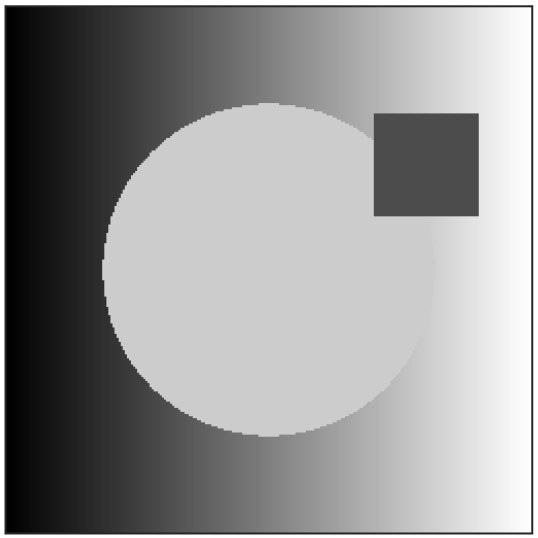
Increased Brightness



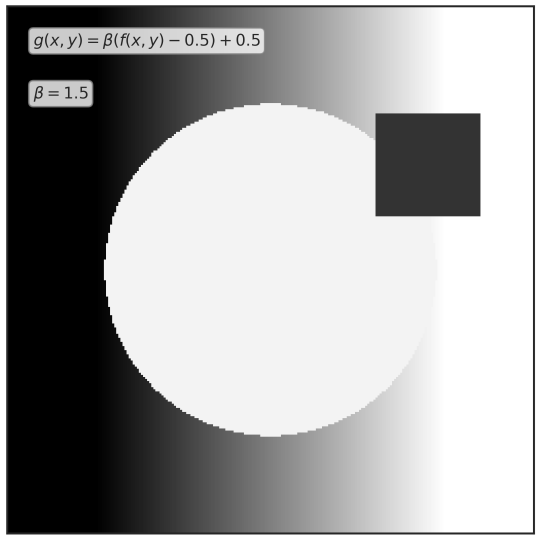
Decreased Brightness



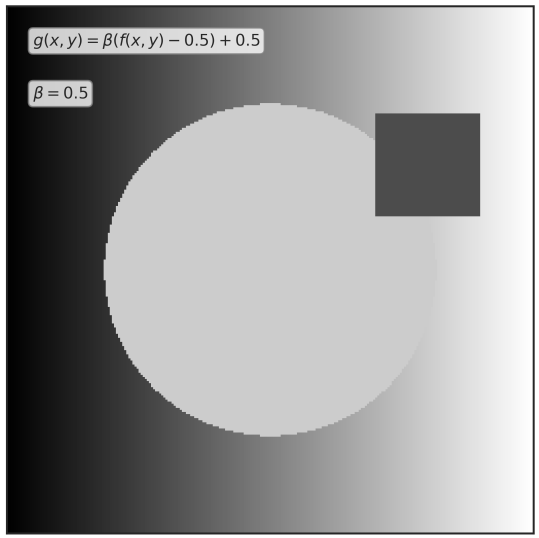
Original Image



Increased Contrast



Decreased Contrast



Neighborhood Operators

Transformations that use a neighborhood of an input pixel to determine an output pixel's value are called **neighborhood operators**. These can be used to **filter** an image - to softly blur it, to sharpen details, etc.

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

$=$

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Linear Filters

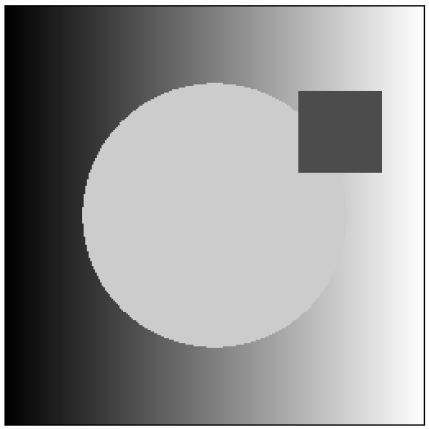
The most widely used type of neighborhood operator is the **linear filter**: each output pixel's value is a weighted sum of pixels values within some neighborhood $\mathcal{N}$ of the input pixel:

$$g(x,y) = \sum_{k,l} f(x-k, y-l)h(k,l) = \sum_{k,l} f(x,y)h(x-k, y-l)$$

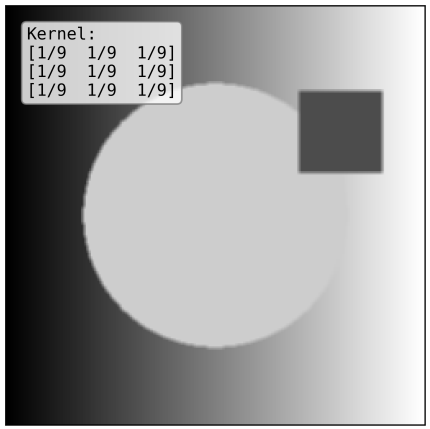
We call h the **kernel** or **mask** of the operator/filter. Also sometimes called the **impulse response function**.

Linear Filters

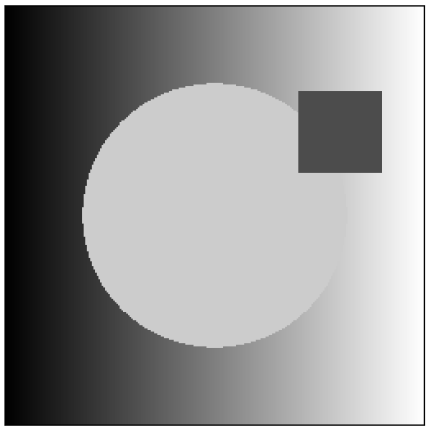
Original Image



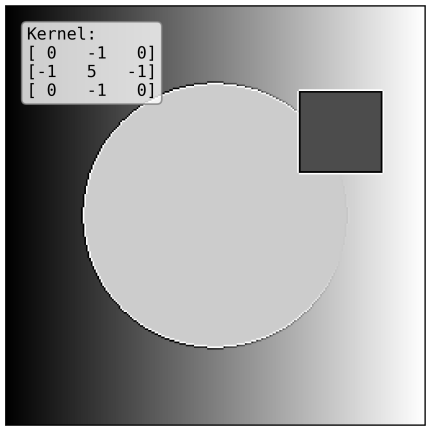
3x3 Mean Blur



Original Image



3x3 Sharpening



Convolution is a Linear Operator

Convolution can be written as a matrix-vector product by first vectorizing the input and then forming a sparse matrix **H** that allows one to write

$$\mathbf{g} = \mathbf{H}\mathbf{f}$$

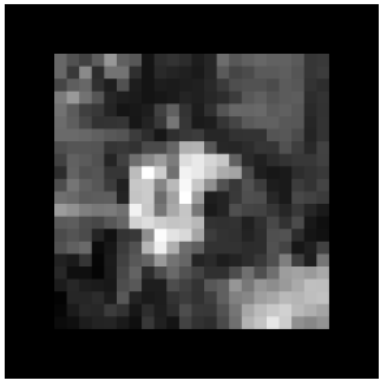
$$\begin{array}{|c|c|c|c|c|} \hline 72 & 88 & 62 & 52 & 37 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1/4 & 1/2 & 1/4 \\ \hline \end{array} \Leftrightarrow \frac{1}{4}\begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix} \begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

Padding Convolutions

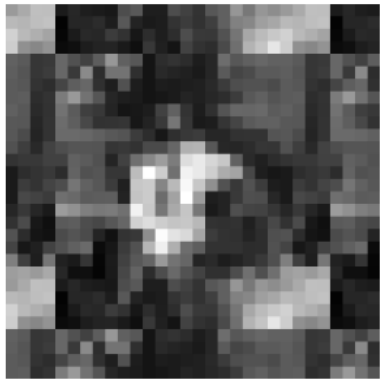
Convolution does not preserve the dimensionality of the input (except in trivial case). Therefore, it is necessary to **pad** input images with additional data $\Longrightarrow$ boundary effects.

- **zero**: set all pixels outside the boundaries to 0
- **constant**: set all pixels outside the boundaries to some constant c
- **clamp**: repeat edge pixels indefinitely
- **wrap**: wrap around as if on torus
- **mirror**: reflect pixels across boundary

Padding Convolutions

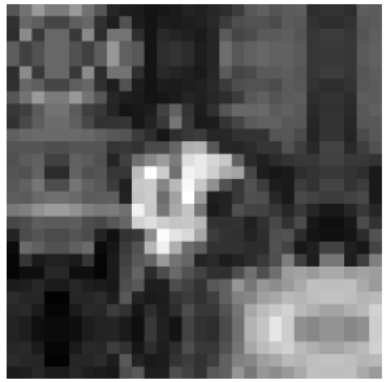


zero

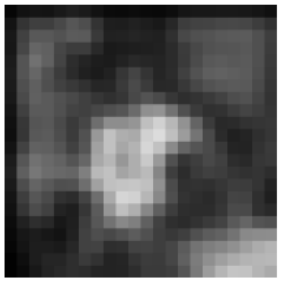


wrap

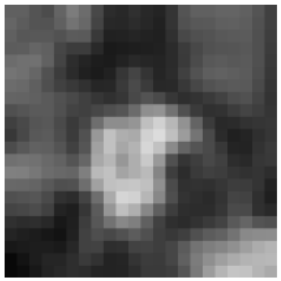
clamp



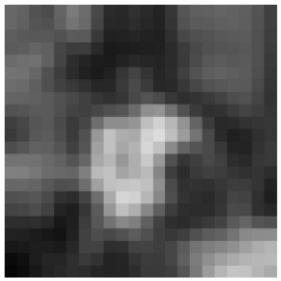
mirror



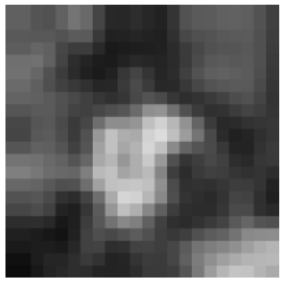
blurred zero



normalized zero



blurred clamp



blurred mirror

High Pass and Low Pass Filters

Two fundamental types of filters in image processing:

- **Low Pass Filter:** Preserves low frequency components while attenuating high frequencies
  - Smooths out rapid variations in intensity
  - Reduces noise and blurs edges
  - Example kernel: $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

High Pass and Low Pass Filters

Two fundamental types of filters in image processing:

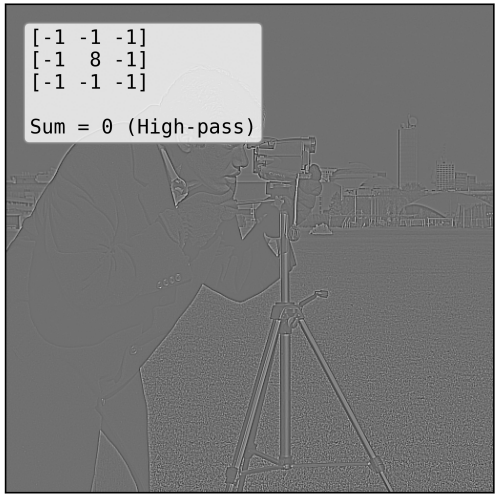
- **Low Pass Filter:** Preserves low frequency components while attenuating high frequencies
  - Smooths out rapid variations in intensity
  - Reduces noise and blurs edges
  - Example kernel: $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- **High Pass Filter:** Preserves high frequency components while attenuating low frequencies
  - Enhances rapid variations in intensity
  - Sharpens edges and fine details
  - Example kernel: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Band-pass Filters

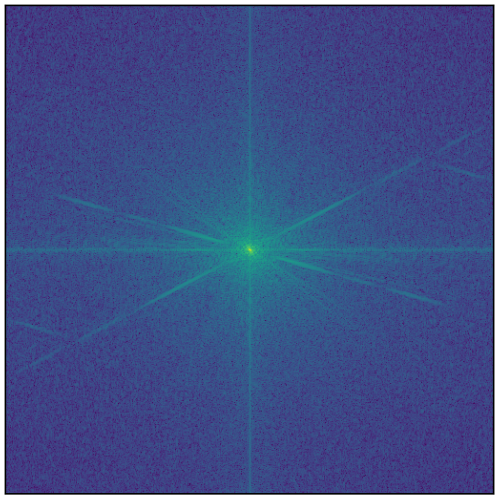
Original Image

Low-Pass Filtered

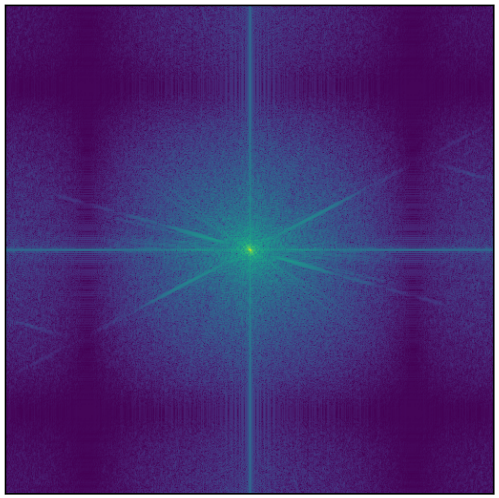
High-Pass Filtered



Original Magnitude Spectrum



Low-Pass Magnitude Spectrum



High-Pass Magnitude Spectrum

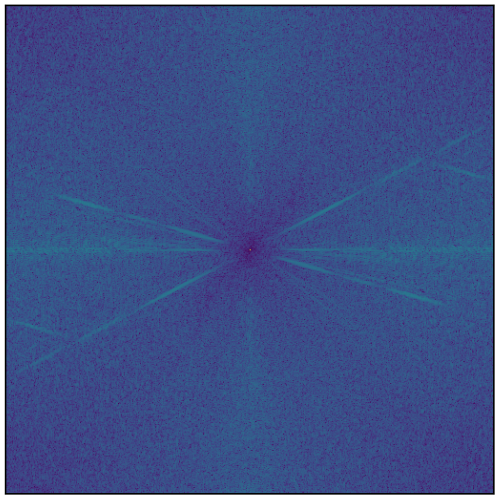


Image Gradient

The **gradient** of an image f is a vector field that points in the direction of steepest increase in intensity: $\nabla f = [\frac{\partial f}{\partial x} \ \frac{\partial f}{\partial y}]^T$
For discrete images, we approximate these partial derivatives using finite differences:

$$\frac{\partial f}{\partial x} \approx f(x+1, y) - f(x, y)$$

$$\frac{\partial f}{\partial y} \approx f(x, y+1) - f(x, y)$$

The **magnitude** of the gradient

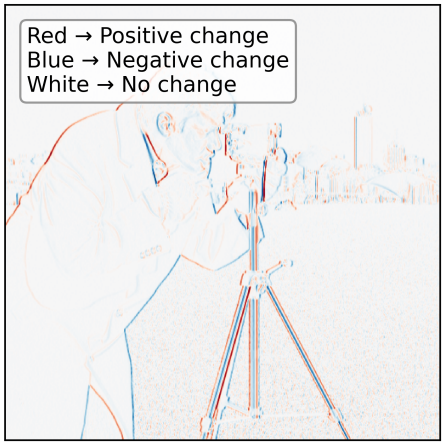
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

gives us a measure of how the image intensity is changing.

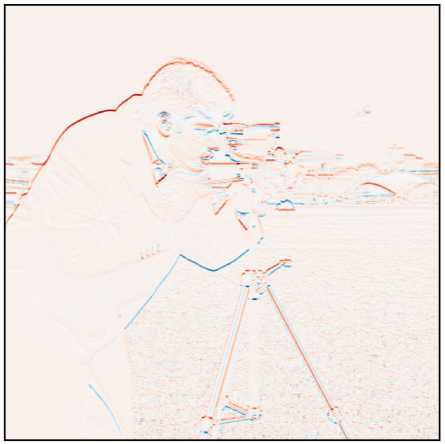
Image Gradient

Original Image

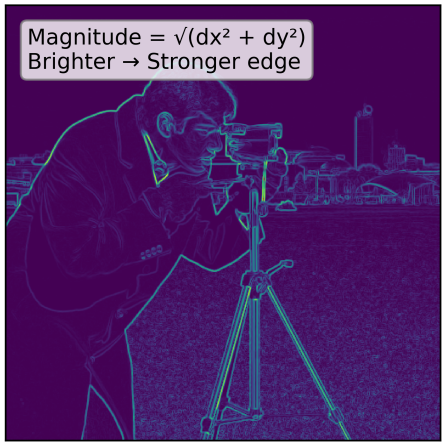
Horizontal Gradient



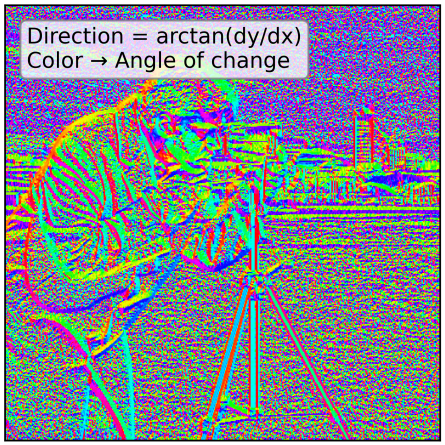
Vertical Gradient



Gradient Magnitude



Gradient Direction



Strong Edges

Separable Filters

A 2D filter $\mathbf{K}$ is called **separable** if it corresponds to applying successively convolution with a horizontal kernel $\mathbf{h}$ and a vertical kernel $\mathbf{v}$,

$$\mathbf{K} = \mathbf{v}\mathbf{h}^T$$

This takes $2K$ operations rather than K^2.
How can we determine if a given filter $\mathbf{K}$ is separable? Use SVD: $\mathbf{K}$ is separable iff $\mathbf{K}$ has only one nonzero singular value $\implies \sqrt{\sigma_0}\mathbf{u}_0$ and $\sqrt{\sigma_0}\mathbf{v}_0$ are the vertical and horizontal filters.

Separable Filters

The discrete Gaussian filter with standard deviation σ is defined as:

$$G_\sigma(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}} = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2}{2\sigma^2}} e^{-\frac{j^2}{2\sigma^2}}$$

It can be written as a product of two 1D Gaussian filters:

$$G_\sigma(i,j) = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{j^2}{2\sigma^2}}\right)$$

Making it separable into vertical and horizontal components:

$$G_\sigma = \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2}{2\sigma^2}}\right]_i \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{j^2}{2\sigma^2}}\right]_j$$

Separable Filters

The Sobel filter is a discrete differentiation operator that computes an approximation of the gradient of an image intensity function. It consists of two 3×3 kernels, one for detecting horizontal changes ($\mathbf{S}_x$) and one for vertical changes ($\mathbf{S}_y$):

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

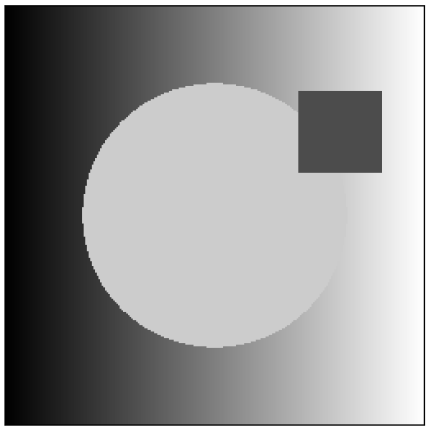
The Sobel filter's components can be decomposed into separable filters:

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

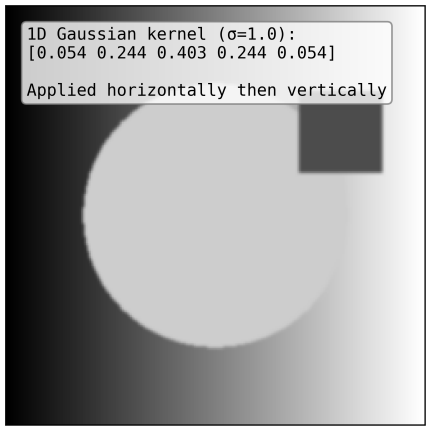
$$\mathbf{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Separable Filters

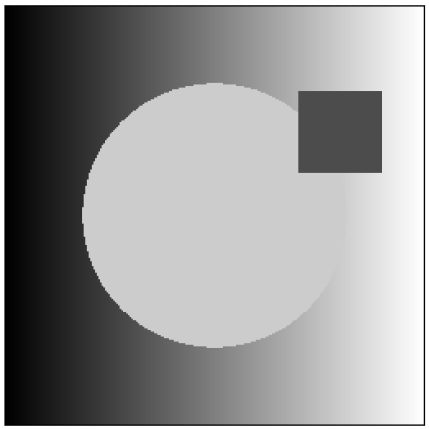
Original Image



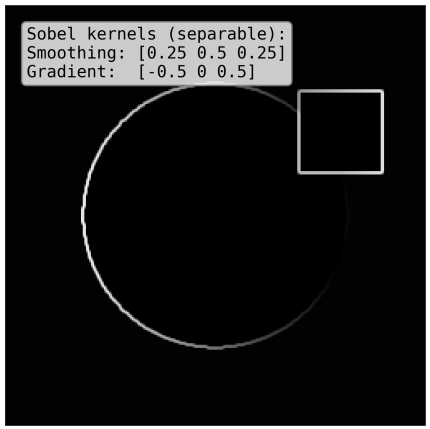
Gaussian Blur (Separable)



Original Image



Sobel Edge Detection (Separable)



Gaussian Pyramid and Aliasing

The **Gaussian pyramid** represents an image at multiple scales:

- Start with original image
- Blur with Gaussian filter (low-pass)
- Downsample by factor of 2
- Repeat process on downsampled image

Gaussian Pyramid and Aliasing

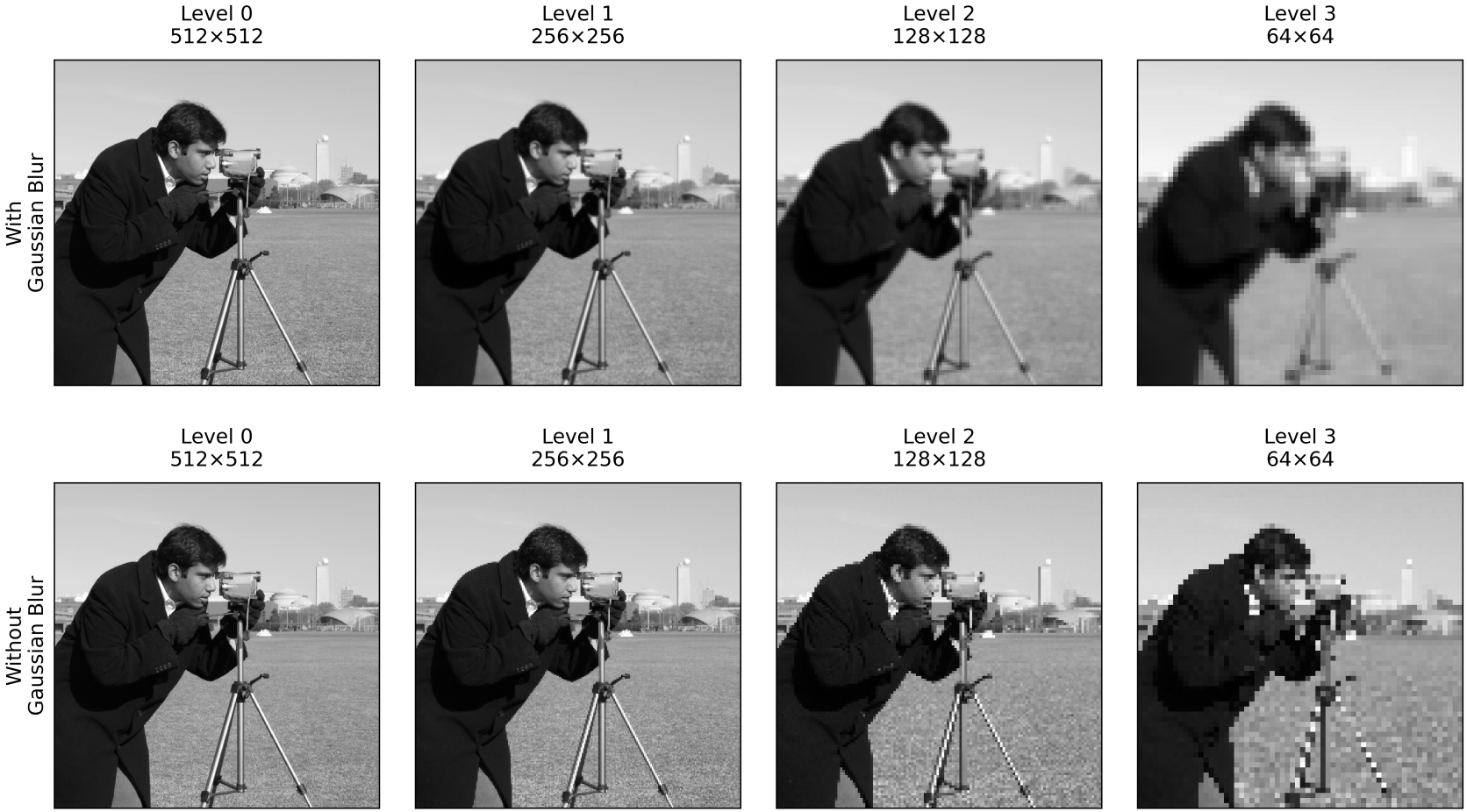
The **Gaussian pyramid** represents an image at multiple scales:

- Start with original image
- Blur with Gaussian filter (low-pass)
- Downsample by factor of 2
- Repeat process on downsampled image

Why Gaussian blur before downsampling?

- Downsampling can cause aliasing artifacts
- Nyquist theorem: must remove frequencies above $\frac{1}{2}$ of new sampling rate
- Gaussian blur acts as anti-aliasing filter
- Without blur: high frequencies "fold back" as aliases

Gaussian Pyramid: Example



Top row: Proper Gaussian pyramid with anti-aliasing
Bottom row: Aliasing artifacts from direct downsampling

Laplacian Pyramid

The **Laplacian pyramid** stores difference images between Gaussian pyramid levels:

- Start with Gaussian pyramid $G_0, G_1, ..., G_n$
- For each level i (except last):
  - Upsample G_{i+1} to size of G_i
  - Blur upsampled image (same Gaussian kernel)
  - Subtract from G_i to get Laplacian level L_i

Laplacian Pyramid

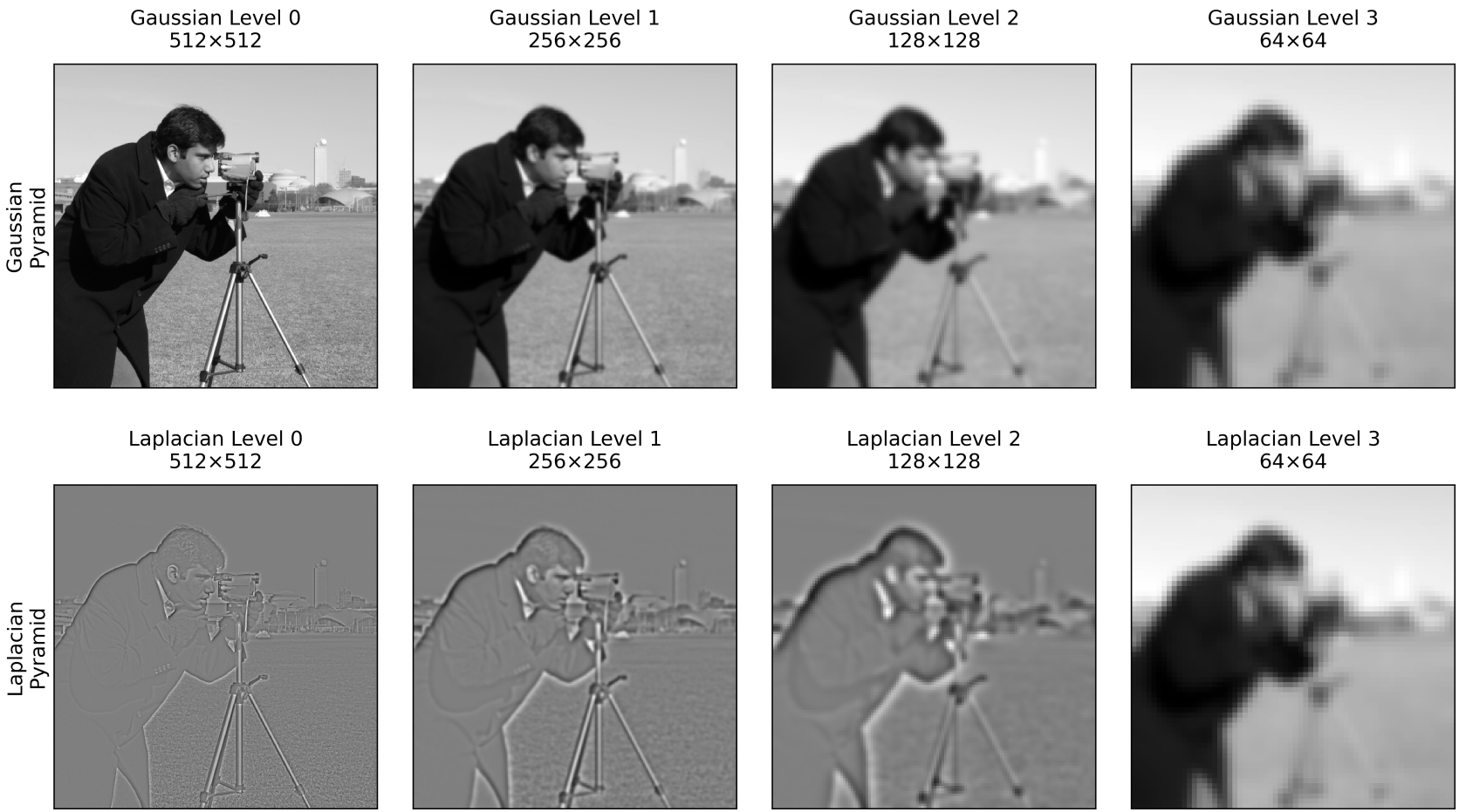
The **Laplacian pyramid** stores difference images between Gaussian pyramid levels:

- Start with Gaussian pyramid $G_0, G_1, ..., G_n$
- For each level i (except last):
  - Upsample G_{i+1} to size of G_i
  - Blur upsampled image (same Gaussian kernel)
  - Subtract from G_i to get Laplacian level L_i

Properties:

- Each level captures details at specific scale
- Perfect reconstruction: can recover original from pyramid
- Band-pass decomposition: each level = specific frequency band

Laplacian Pyramid: Example



Wavelet Pyramid

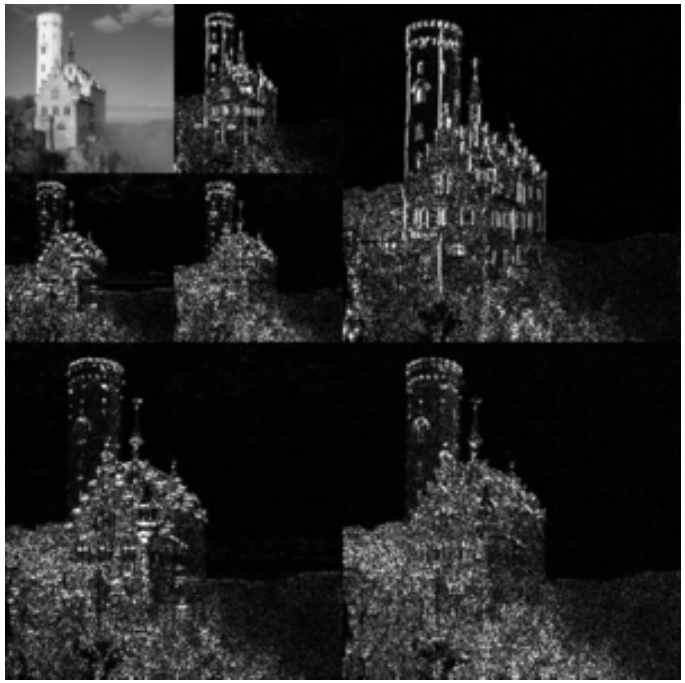
The **wavelet pyramid** decomposes an image into oriented edge responses:

- Similar to Laplacian pyramid but with orientation selectivity
- Each level split into oriented subbands (usually horizontal, vertical, and diagonal)
- Preserves both scale and orientation information

Wavelet Pyramid

The **wavelet pyramid** decomposes an image into oriented edge responses:

- Similar to Laplacian pyramid but with orientation selectivity
- Each level split into oriented subbands (usually horizontal, vertical, and diagonal)
- Preserves both scale and orientation information



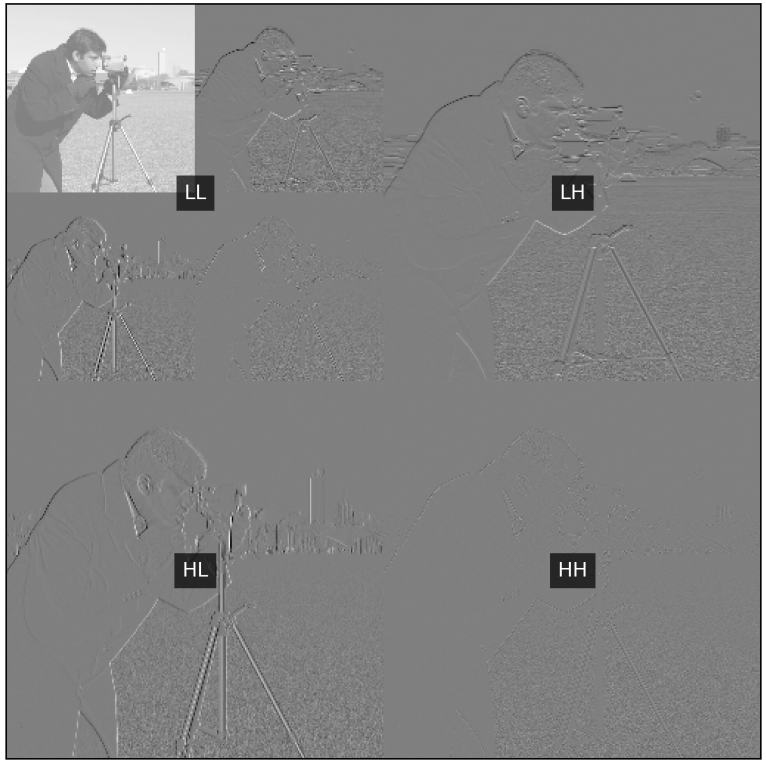
Key ideas:

- Uses wavelet filters at each scale
- Perfect reconstruction possible (like Laplacian)

Wavelet Pyramid: Example

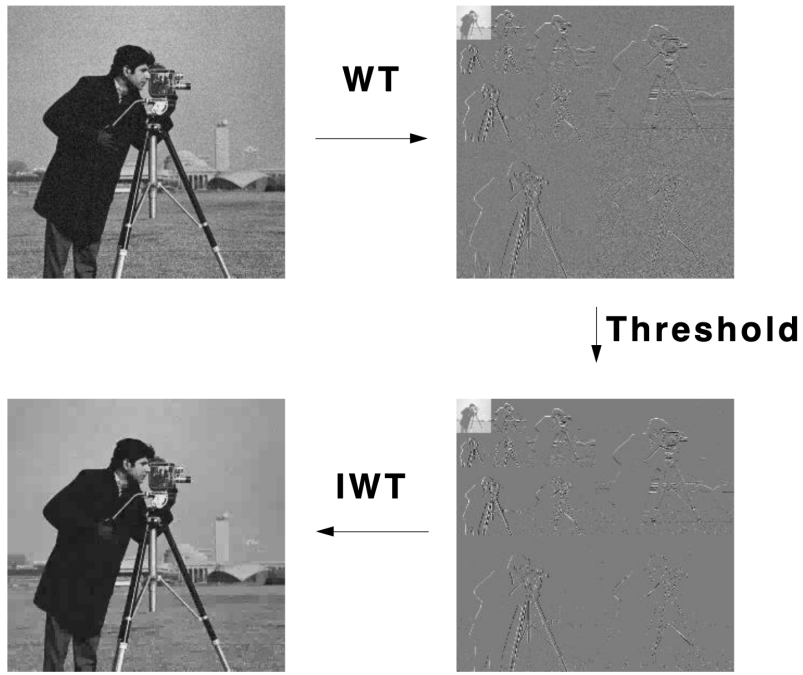
Original Image

Wavelet Decomposition
(Traditional Layout)



Denoising with Wavelets - Sketch of Idea

- Dominant features are represented by large wavelet coefficients.
- The noise should affect all wavelet coefficients uniformly.
- (soft) Thresholding small coefficients to 0 maintains the image but kills the noise.



Edge/Line Detection

The derivative in the direction θ is given by

$$f_\theta = \cos(\theta) f_x + \sin(\theta) f_y$$

using the image gradient. This is called a **steerable** derivative because we can "steer" the derivative filter to any orientation θ.

Key properties:

- The steerable derivative achieves maximum response when θ is perpendicular to an edge
- The magnitude of response indicates edge strength
- We can find dominant edge orientation by maximizing $|f_\theta|$ over θ

Naive edge detector:

1. Compute f_x and f_y using (Sobel) gradient filters
2. For each pixel, find $\theta_{max} = \arg\max_\theta |f_\theta|$
3. Threshold $|f_{\theta_{max}}|$ to identify edge pixels

Line Detection with Laplacian

The **Laplacian filter** is a second-order derivative filter defined as:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

It is often approximated by the kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Responds strongly to lines and edges
- Zero-crossings indicate edge locations
- Often combined with Gaussian smoothing (LoG filter)

Histogram of Oriented Gradients (HOG)

HOG is a feature descriptor that captures local shape information:

- Divides image into small cells (e.g., 8×8 pixels)
- Computes histogram of gradient directions in each cell
- Normalizes histograms across larger blocks (e.g., 2×2 cells)

Key properties:

- Invariant to local illumination changes
- Captures edge orientation patterns
- Widely used in object detection

Canny Edge Detection

The Canny edge detector is a multi-stage algorithm:

- **Smoothing:** Apply Gaussian filter to reduce noise
- **Gradients:** Compute magnitude and direction using Sobel
- **Non-maximum suppression:** Thin edges by keeping only local maxima along gradient direction
- **Double thresholding:** Classify pixels as strong/weak using two thresholds
- **Edge tracking:** Connect weak edges to strong edges through hysteresis

Canny Edge Detection: Step 1 - Gaussian Smoothing

Mathematical operation:

- Given input image f, compute:
- $f_\sigma = f * G_\sigma$ where G_σ is Gaussian kernel:
- $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

Canny Edge Detection: Step 1 - Gaussian Smoothing

Mathematical operation:

- Given input image f, compute:
- $f_\sigma = f * G_\sigma$ where G_σ is Gaussian kernel:
- $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$

Properties:

- Bandlimited signal in frequency domain
- Suppresses frequencies above $\approx \frac{1}{\sigma}$

Canny Edge Detection: Step 2 - Gradient Computation

Sobel operators:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradient computation:

- $\frac{\partial f_\sigma}{\partial x} = f_\sigma * S_x$
- $\frac{\partial f_\sigma}{\partial y} = f_\sigma * S_y$
- Magnitude: $|\nabla f_\sigma| = \sqrt{(\frac{\partial f_\sigma}{\partial x})^2 + (\frac{\partial f_\sigma}{\partial y})^2}$

Canny Edge Detection: Step 3 - Non-maximum Suppression

Mathematical formulation:

- Given gradient magnitude $M = |\nabla f_\sigma|$ and direction θ
- Quantize θ to nearest 45° angle: θ_q
- For each pixel (x, y):
- Let $(x_1, y_1), (x_2, y_2)$ be neighbors in direction $\theta_q^\perp$
- Output $N(x, y) =$

$$\begin{cases} M(x,y) & \text{if } M(x,y) > M(x_1,y_1) \text{ and } M(x,y) > M(x_2,y_2) \\ 0 & \text{otherwise} \end{cases}$$

Canny Edge Detection: Step 4 - Double Thresholding

Mathematical operation: Given non-maximum suppressed image N, high threshold t_h, low threshold t_l:

$$T(x,y) = \begin{cases} 2 & \text{if } N(x,y) > t_h \text{ (strong edge)} \\ 1 & \text{if } t_l \leqslant N(x,y) \leqslant t_h \text{ (weak edge)} \\ 0 & \text{if } N(x,y) < t_l \text{ (non-edge)} \end{cases}$$

Canny Edge Detection: Step 5 - Edge Tracking by Hysteresis

Mathematical formulation:

- Let $S = \{(x, y) : T(x, y) = 2\}$ be strong edge pixels
- Let $W = \{(x, y) : T(x, y) = 1\}$ be weak edge pixels
- For each $(x, y) \in W$:
- Let $N_8(x, y)$ be 8-connected neighbors
- Final edge map:

$$E(x, y) = \begin{cases} 1 & \text{if } T(x, y) = 2 \\ 1 & \text{if } T(x, y) = 1 \text{ and } \exists (u, v) \in N_8(x, y) : E(u, v) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Canny Edge Detection: Example

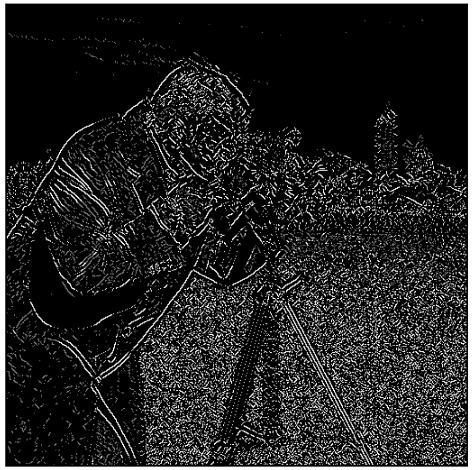
Original Image

Gaussian Smoothing

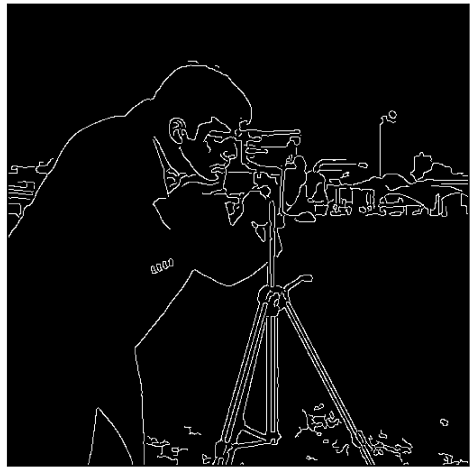
Gradient Magnitude

Non-max Suppression

Double Threshold



Final Edges



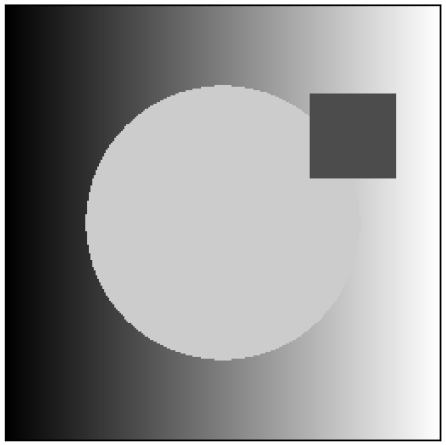
Nonlinear Filters: Median Filter

Median Filter:

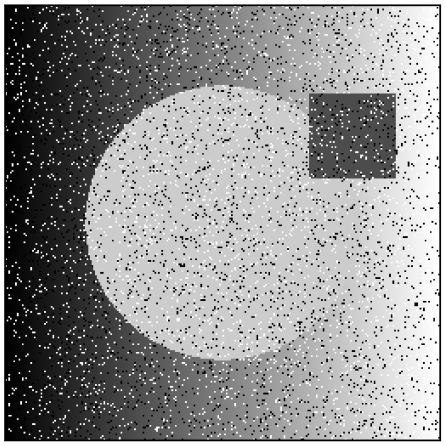
- For each pixel, take a neighborhood around it
- Replace pixel value with median of neighborhood
- More robust to noise than linear filters, better edge preservation

Nonlinear Filters: Median Filter

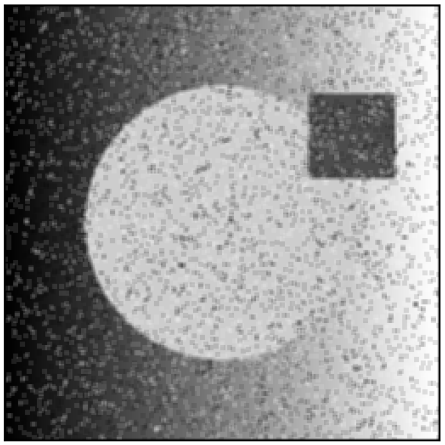
Original



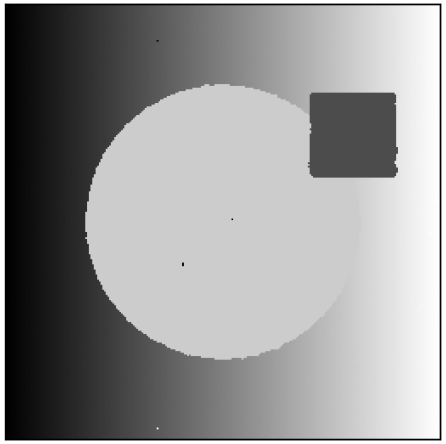
With Salt & Pepper Noise



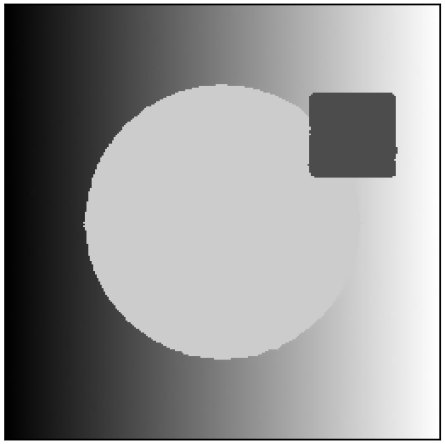
Mean Filter (3×3)



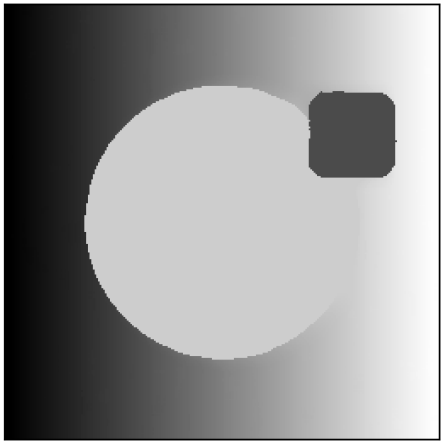
Median Filter (3×3)



Median Filter (5×5)



Median Filter (15×15)

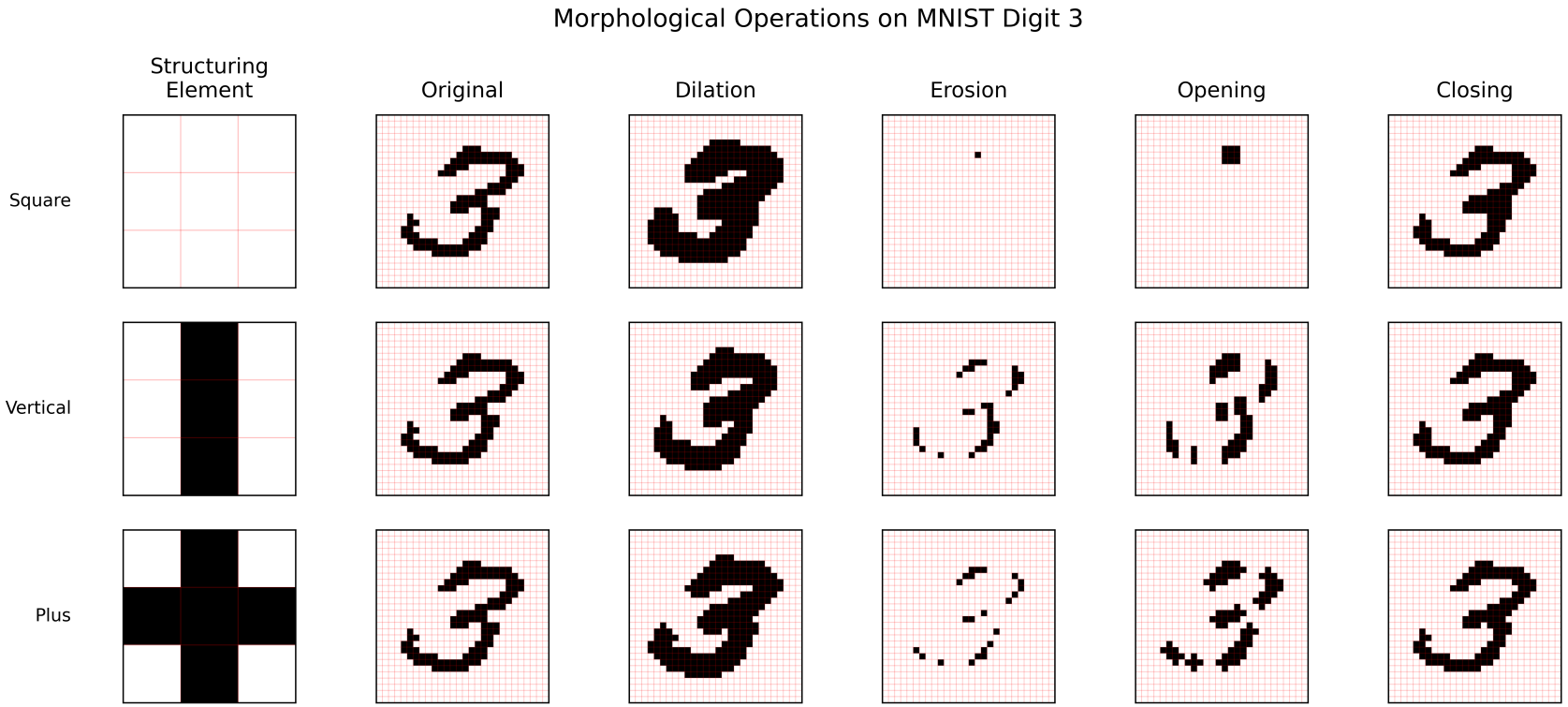


Nonlinear Filters: Morphological Operators

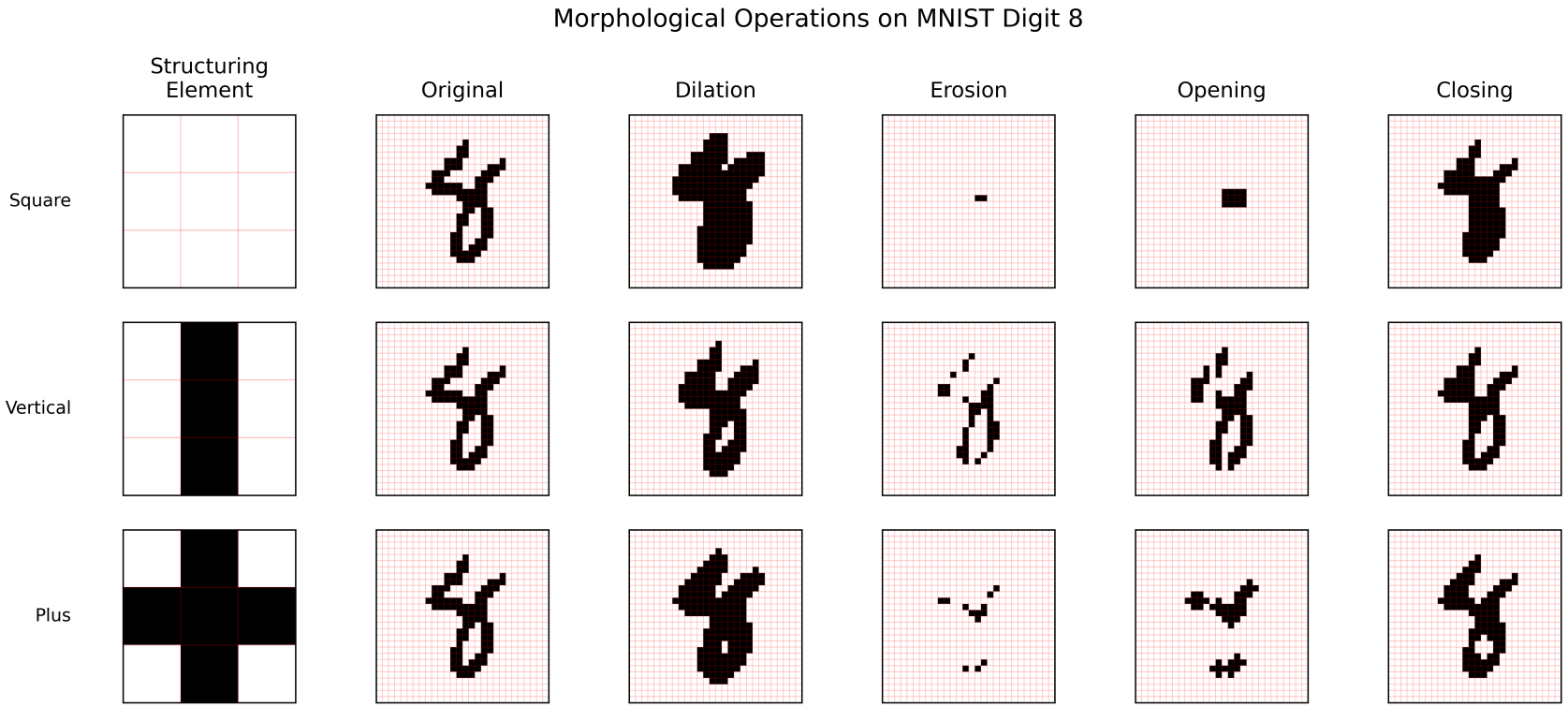
For **binary** images we can define a special set of operators called **morphological** operators. These are operators which are typically nonlinear and defined using lattice operations (max, min) and composition. Let A be a binary image and B be a **structuring element** which dictates the neighborhood shape.

- **Dilation:** $(A \oplus B)(x) = \max_{b \in B} A(x - b)$
- **Erosion:** $(A \ominus B)(x) = \min_{b \in B} A(x + b)$
- **Opening:** $A \circ B = (A \ominus B) \oplus B$
- **Closing:** $A \bullet B = (A \oplus B) \ominus B$

Nonlinear Filters: Morphological Operators



Nonlinear Filters: Morphological Operators



Conclusion

- Filtering is essential in image processing and computer vision
- We can understand filters and construct filters using harmonic analysis
- Useful for feature extraction - edges, lines, geometric structure

References

- "Sparse Image and Signal Processing - Wavelets, Curvelets, Morphological Diversity" by Starck, Murtagh, and Fadili