Digital Image Processing

# 1. Sampling and Quantization

Sampling and Quantization are fundamental concepts in image processing. They refer to the processes of converting continuous images into discrete representations suitable for digital systems.

Sampling and quantization are essential processes in converting a continuous-tone image into its digital form so it can be stored, processed, and analyzed by digital systems. These processes play a crucial role in digital image processing and influence both the image quality and its file size.

**Sampling** refers to measuring the intensity (gray level or color) of an image at regular intervals over its spatial extent. Imagine an analog image as a continuously varying 2D surface of intensity values. When we perform sampling, we are selecting certain points—called *pixels*—at fixed spatial locations. The density of this selection is called the **sampling rate**, and it determines how many pixels the image will have. A higher sampling rate captures more detail and typically results in a sharper image. Inadequate sampling can lead to aliasing, where important image features are lost or misrepresented.

Once the spatial sampling is complete, each sampled point still contains a continuous value of intensity, which needs to be mapped to a discrete digital level to be represented in a computer. This process is called **quantization**. In quantization, each intensity value is assigned to one of a fixed number of digital levels. For example, an 8-bit image has 256 levels (0–255). This step introduces **quantization error**, which can result in loss of image detail or banding in areas of gradual intensity change.

Together, sampling and quantization define the resolution and fidelity of the digital image. Sampling affects the spatial resolution, while quantization impacts the grayscale or color resolution.

**Non-Uniform Sampling and Quantization**

While uniform sampling and quantization use equally spaced intervals and levels respectively, **non-uniform sampling** involves varying the distances between sample points depending on the content. For instance, more samples may be taken in regions of the image with high detail, and fewer samples where the content is smooth. This is common in techniques like foveated imaging, where the center is sampled densely while the periphery is sampled sparsely—mimicking the human eye.

**Non-uniform quantization**, on the other hand, uses non-equally spaced intensity levels. This is useful when certain ranges of intensity occur more frequently than others, allowing more quantization levels (finer precision) in that range. A practical example is µ-law and A-law companding in audio and image compression, where small values are represented with higher precision than large ones.

These techniques help in **compression**, reduce the data required to represent an image, and sometimes improve perceptual quality by focusing resources where they are most needed.

**Diagram Description:**

Analog Image --> Sampling --> Discrete Grid of Pixels --> Quantization --> Digital Pixel Values

Sampling determines where to measure. Quantization determines what those measurements mean. Both are crucial for digital imaging systems such as cameras, medical scanners, and satellite image processors.

# 2. Non-Uniform Sampling and Quantization

Non-uniform sampling and quantization are techniques where the sampling rate or the precision of quantization is not constant. In certain applications, this technique may be applied to improve the efficiency of image compression or other processing tasks.

Image enhancement is a critical task in image processing whose primary objective is to improve the visual appearance of an image or to convert the image to a form better suited for analysis by human eyes or machine vision systems. It is often the first step in many image processing applications such as medical imaging, surveillance, and industrial inspection, where clarity and visual quality are paramount.

**Objective of Image Enhancement**

The main goal of image enhancement is to bring out image details that are obscured or to highlight features of interest in an image. Unlike image restoration, which attempts to reconstruct or recover an image that has been degraded, enhancement is more subjective and context-dependent. What may be considered an enhancement in one application might be undesirable in another. For instance, increasing contrast may help in highlighting objects in one image but might exaggerate noise in another.

Enhancement can involve multiple operations, such as sharpening, contrast stretching, brightness adjustment, noise suppression, and edge enhancement. The technique used depends on the specific application and the characteristics of the input image. In some cases, enhancements are also used to improve the performance of further automated image analysis processes such as segmentation and classification.

**Categories of Image Enhancement**

Image enhancement techniques can be broadly classified into two major categories based on the domain in which they operate:

**1. Spatial Domain Techniques**

Spatial domain methods operate directly on the image pixels. These techniques involve manipulating the values of individual pixels or the values of pixels in the neighborhood of each pixel. Some common spatial domain enhancement techniques include:

* **Contrast Stretching**: Improves contrast by expanding the range of intensity values.
* **Histogram Equalization**: Redistributes intensity values to flatten the histogram and enhance contrast.
* **Smoothing Filters**: Used to reduce noise and minor variations (e.g., mean filter, Gaussian blur).
* **Sharpening Filters**: Highlight edges and fine details by enhancing intensity transitions (e.g., Laplacian filter).

Spatial filters can be linear or nonlinear, and they are usually implemented via convolution using a kernel or mask applied over the image matrix.

**2. Frequency Domain Techniques**

In frequency domain techniques, the image is first transformed using a mathematical transform such as the Fourier Transform. The resulting frequency representation is then modified to achieve the desired enhancement before being transformed back into the spatial domain.

These techniques are particularly useful for:

* Enhancing edges and fine details using high-pass filters.
* Smoothing images using low-pass filters.
* Suppressing periodic noise or improving visual features based on their frequency content.

One powerful feature of frequency domain methods is the ability to design filters that target specific spatial frequencies, making them effective in enhancing textures or removing specific types of artifacts.

**Choosing Between the Two**

The choice between spatial and frequency domain enhancement depends on the nature of the image and the goal of the enhancement. Spatial domain methods are more intuitive and easier to implement, but frequency domain techniques can be more effective in some contexts, particularly when periodic patterns or noise are involved.

In conclusion, image enhancement serves the dual purpose of improving the visual appearance of images and facilitating subsequent processing tasks. By using either spatial or frequency domain techniques—or a combination of both—enhancement processes can make subtle image features more prominent, leading to better interpretation and analysis.

# 3. Objective of Image Enhancement Technique

The main objective of image enhancement techniques is to improve the visibility or interpretability of an image for human perception or further processing tasks. This can involve various methods such as contrast stretching, filtering, etc.

## Categories of Image Enhancement

There are two main categories of image enhancement techniques:  
1. Spatial Domain Methods: These operate directly on pixels of the image.  
2. Frequency Domain Methods: These involve transforming the image to the frequency domain for enhancement.

In image processing, **gradient** and **Laplacian** are fundamental operators used primarily for edge detection and image enhancement. These mathematical tools originate from calculus and are adapted to detect intensity variations in images—especially at boundaries where significant changes in pixel values occur. Enhancing edges helps in better visualization of objects, especially in applications like medical imaging, document scanning, and machine vision systems.

**Gradient: Definition and Role**

The **gradient** of an image is a vector that represents both the **magnitude** and **direction** of the greatest rate of intensity change at a pixel. It is essentially used to detect edges, which are locations in an image where the brightness changes sharply.

In 2D, the gradient of a digital image f(x,y)f(x, y)f(x,y) is defined as:

∇f=[∂f∂x,∂f∂y]\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]∇f=[∂x∂f​,∂y∂f​]

Where:

* ∂f∂x\frac{\partial f}{\partial x}∂x∂f​ is the rate of change of intensity in the horizontal direction.
* ∂f∂y\frac{\partial f}{\partial y}∂y∂f​ is the rate of change of intensity in the vertical direction.

The **magnitude** of the gradient is given by:

∣∇f∣=(∂f∂x)2+(∂f∂y)2|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}∣∇f∣=(∂x∂f​)2+(∂y∂f​)2​

This gradient magnitude is high at the edges and low in smooth regions, making it useful for highlighting boundaries.

In digital implementation, the derivatives are approximated using convolution masks (kernels). Common gradient operators include:

* **Sobel operator**
* **Prewitt operator**
* **Roberts cross operator**

These masks approximate the partial derivatives using discrete differences and are applied over a neighborhood (usually 3×3) around each pixel.

**Laplacian: Definition and Role**

The **Laplacian** is a second-order derivative operator that measures the rate at which the first derivative (gradient) changes. It detects areas where intensity changes rapidly and is commonly used to highlight fine details like lines, points, and edges.

Mathematically, the Laplacian is given by:

∇2f=∂2f∂x2+∂2f∂y2\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}∇2f=∂x2∂2f​+∂y2∂2f​

In discrete form, one commonly used Laplacian mask is:

[ 0 -1 0 ]

[-1 4 -1]

[ 0 -1 0 ]

Or with diagonals:

[-1 -1 -1]

[-1 8 -1]

[-1 -1 -1]

The Laplacian is **isotropic**, meaning it responds equally to changes in all directions. Unlike gradient operators, it does not provide edge direction, only location.

**Roles in Image Enhancement**

Both gradient and Laplacian are used in **sharpening**, a key enhancement technique. Here’s how:

* **Gradient-based methods** emphasize edges by identifying and amplifying areas with high first-order derivatives.
* **Laplacian-based methods** highlight regions with rapid intensity changes by enhancing second-order discontinuities.

Often, the result of a Laplacian filter is added back to the original image to enhance fine details. This process is called **Laplacian sharpening**:

g(x,y)=f(x,y)+c⋅∇2f(x,y)g(x, y) = f(x, y) + c \cdot \nabla^2 f(x, y)g(x,y)=f(x,y)+c⋅∇2f(x,y)

Where ccc is a scaling constant.

**Comparison and Use**

| **Feature** | **Gradient** | **Laplacian** |
| --- | --- | --- |
| Order | First derivative | Second derivative |
| Output | Edge direction and magnitude | Edge location only |
| Sensitivity | Sensitive to noise, but less than Laplacian | Highly sensitive to noise |
| Use | Edge detection, directionality | Edge enhancement, sharpening |

In summary, both gradient and Laplacian operators are powerful tools for enhancing images by emphasizing important structural information such as boundaries and fine textures. While gradient highlights edges with direction, Laplacian boosts sharp transitions. They are often used together for robust enhancement and preprocessing before segmentation or recognition tasks.

# 4. Gradient and Laplacian

The Gradient and Laplacian are important mathematical operators used in edge detection and image enhancement.  
  
Gradient: It measures the rate of change of pixel intensities in an image. The gradient is used for edge detection by highlighting areas of rapid intensity change.  
Laplacian: A second-order derivative operator used to detect areas of rapid intensity change and can be used for edge sharpening.

# 5. Controlling Over-Segmentation Problem

Over-segmentation occurs when an image is divided into too many segments, leading to loss of important structure and overcomplicated representations. This can be controlled by optimizing segmentation parameters, applying post-processing steps, or using advanced segmentation algorithms.

**Image segmentation** is the process of dividing an image into meaningful parts, often to isolate objects or regions for further analysis. While segmentation is a critical step in image analysis and computer vision, one common issue that arises—especially in algorithms like watershed, edge detection, or superpixel generation—is **over-segmentation**.

**What is Over-Segmentation?**

Over-segmentation occurs when an image is divided into too many small, irrelevant, or fragmented regions. Instead of cleanly separating meaningful objects, the algorithm produces excessive segments, making the image hard to analyze and increasing computational complexity. This typically happens when:

* There’s noise in the image.
* There are subtle intensity variations.
* The segmentation method is too sensitive.

For example, a simple object like a coin may be divided into dozens of tiny segments because of surface texture, reflections, or shadows.

**Causes of Over-Segmentation**

1. **High Sensitivity to Noise**: Some algorithms treat even the slightest change in pixel intensity as a boundary.
2. **Poor Preprocessing**: Without adequate smoothing or denoising, minute variations can create false edges.
3. **Fine Parameter Tuning**: Thresholds and parameters in edge detection or region-growing methods can be too restrictive.
4. **Lack of Global Context**: Algorithms may focus on local gradients or textures without understanding larger object structures.

**Strategies to Control Over-Segmentation**

To combat over-segmentation, various preprocessing, postprocessing, and algorithmic refinement methods are used. Here are the most effective ones:

**1. Preprocessing: Denoising and Smoothing**

Before applying segmentation, images should be cleaned of noise using smoothing filters:

* **Gaussian blur**: Reduces high-frequency noise.
* **Median filter**: Especially effective in preserving edges while removing salt-and-pepper noise.
* **Bilateral filter**: Smooths images without losing edge information.

These reduce tiny variations that might lead to false segment boundaries.

**2. Marker-Controlled Watershed Segmentation**

Watershed segmentation is a powerful but often over-segmenting technique. To avoid this:

* Use **markers** to indicate foreground and background objects manually or automatically.
* Apply **distance transforms** to define object centers as foreground markers.
* This approach suppresses irrelevant boundaries and focuses segmentation only where it matters.

**3. Region Merging After Initial Segmentation**

Once small segments are formed:

* Merge adjacent segments based on color, texture, or intensity similarity.
* Use statistical measures (e.g., mean and standard deviation) to decide whether segments should be combined.
* Graph-based methods like **Felzenszwalb’s segmentation** also use similarity metrics to decide when to merge.

**4. Superpixel-Based Methods**

Superpixels group pixels into small coherent regions:

* Use algorithms like SLIC (Simple Linear Iterative Clustering).
* These segments are more meaningful and reduce fragmentation.
* They can be merged further based on context or application needs.

**5. Machine Learning and Deep Learning Methods**

Recent advances allow the use of AI to overcome over-segmentation:

* CNNs and transformers learn context-aware features that guide precise segmentation.
* Models like U-Net can produce accurate and compact segmentations by learning from annotated datasets.

**6. Threshold Optimization**

For intensity-based methods:

* Avoid fixed global thresholds.
* Use **adaptive thresholding** or **Otsu’s method** to dynamically determine cutoffs based on image statistics.

**7. Morphological Operations**

Postprocessing techniques like:

* **Dilation and erosion**
* **Closing and opening**

Help merge fragmented regions and eliminate thin boundary lines, leading to more coherent segments.

**Conclusion**

Over-segmentation is a significant challenge in image segmentation, but it can be controlled using a combination of preprocessing (to reduce noise), intelligent algorithms (to focus segmentation), and postprocessing (to merge fragments). Choosing the right technique depends on the application—whether it's medical imaging, object tracking, or scene analysis. The ideal approach balances detail preservation with meaningful object isolation, ensuring both visual clarity and computational efficiency.

# 6. Prewitt Kernel for Edge Detection

**Edge detection** is a crucial step in image processing and computer vision, enabling the identification of object boundaries and features. Among the many operators developed for this purpose, the **Prewitt operator** is a classic and widely used method due to its simplicity and efficiency.

**Introduction to Prewitt Operator**

The **Prewitt operator** is a **gradient-based edge detection method** that approximates the first derivative of image intensity. It uses discrete convolution kernels to compute an approximation of the gradient in horizontal and vertical directions.

The goal is to detect edges by identifying areas in an image where the intensity changes rapidly—typically at the boundaries between different objects or regions.

**1. Prewitt Kernels for Horizontal and Vertical Edge Detection**

The Prewitt operator consists of two 3×3 convolution masks, one for detecting **horizontal edges** and one for **vertical edges**:

**Horizontal Prewitt Kernel (detects vertical edges):**

Gx=[−101−101−101]G\_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}Gx​=​−1−1−1​000​111​​

This kernel emphasizes differences in pixel intensity across the **x-axis** (horizontal direction). When convolved with an image, it responds strongly to **vertical edges**—places where there is a significant change in intensity from left to right.

**Vertical Prewitt Kernel (detects horizontal edges):**

Gy=[111000−1−1−1]G\_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}Gy​=​10−1​10−1​10−1​​

This kernel detects intensity changes along the **y-axis** (vertical direction). It responds strongly to **horizontal edges**—regions with strong changes from top to bottom.

**2. Detecting Diagonal Edges with Prewitt-like Kernels**

Although the original Prewitt operator is designed for horizontal and vertical edges, diagonal edges can be detected by creating **custom kernels** using similar logic.

**Diagonal Prewitt Kernel (45° direction):**

Gd1=[011−101−1−10]G\_{d1} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}Gd1​=​0−1−1​10−1​110​​

**Diagonal Prewitt Kernel (135° direction):**

Gd2=[11010−10−1−1]G\_{d2} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}Gd2​=​110​10−1​0−1−1​​

These diagonal kernels are not part of the standard Prewitt operator but are adapted for applications that require more directional sensitivity.

**3. Gradient Magnitude and Edge Strength**

Once the horizontal and vertical gradients are computed (GxG\_xGx​ and GyG\_yGy​), the **gradient magnitude** is calculated using:

∣G∣=Gx2+Gy2|G| = \sqrt{G\_x^2 + G\_y^2}∣G∣=Gx2​+Gy2​​

Or for computational simplicity:

∣G∣=∣Gx∣+∣Gy∣|G| = |G\_x| + |G\_y|∣G∣=∣Gx​∣+∣Gy​∣

The resulting image highlights the edges with strong gradient changes, and edge strength corresponds to the gradient magnitude.

**4. Applications and Advantages of Prewitt Operator**

**Applications:**

* Detecting object boundaries in images.
* Segmenting images based on texture and structure.
* Used in medical imaging, document analysis, and object tracking.

**Advantages:**

* Simple and fast computation.
* Effective in detecting horizontal and vertical edges.
* Smoother results than Sobel in some scenarios since the weights are uniform (no emphasis on the center row or column).

**5. Comparison with Other Operators**

| **Operator** | **Weighting** | **Edge Directions** | **Noise Sensitivity** |
| --- | --- | --- | --- |
| Prewitt | Uniform weights | Vertical, Horizontal | Moderate |
| Sobel | Weighted center | Vertical, Horizontal | Lower than Prewitt |
| Roberts | Diagonal (2×2) | 45°, 135° | Highly sensitive |

**Summary**

The **Prewitt operator** provides a straightforward method for edge detection using convolution kernels to estimate image gradients. While it natively supports horizontal and vertical edges, it can be extended to diagonal directions by designing suitable kernels. Its balance of simplicity and performance makes it a go-to method in many classical image processing tasks.

# 7. Minimum Mean Square Error Filtering for Image Restoration

Minimum Mean Square Error (MMSE) Filtering is used in image restoration techniques to minimize the error between the restored image and the original image by reducing noise or distortions.

**Image restoration** aims to recover an original image that has been degraded by various types of noise or distortion. One of the most sophisticated and statistically optimal approaches for restoration is the **Minimum Mean Square Error (MMSE) filtering** technique.

**Introduction to MMSE Filtering**

The **MMSE filter** is a statistical estimation technique used to estimate the original image by minimizing the **mean square error (MSE)** between the actual image and the restored image. In simple terms, it attempts to find a restored image that is as close as possible to the original, uncorrupted image.

MMSE is especially useful when the degradation process and the noise characteristics are known or can be modeled.

**1. Image Degradation Model**

Let’s begin by understanding the image degradation model, which can be represented as:

g(x,y)=h(x,y)∗f(x,y)+η(x,y)g(x, y) = h(x, y) \* f(x, y) + \eta(x, y)g(x,y)=h(x,y)∗f(x,y)+η(x,y)

Where:

* g(x,y)g(x, y)g(x,y): observed (degraded) image
* h(x,y)h(x, y)h(x,y): degradation function (blur, motion, etc.)
* f(x,y)f(x, y)f(x,y): original image
* η(x,y)\eta(x, y)η(x,y): additive noise
* ∗\*∗: convolution operation

The goal is to estimate f(x,y)f(x, y)f(x,y) from g(x,y)g(x, y)g(x,y).

**2. MMSE Estimation in Frequency Domain**

In the frequency domain, the above degradation model becomes:

G(u,v)=H(u,v)F(u,v)+N(u,v)G(u, v) = H(u, v)F(u, v) + N(u, v)G(u,v)=H(u,v)F(u,v)+N(u,v)

Where:

* G(u,v)G(u, v)G(u,v): Fourier transform of the degraded image
* H(u,v)H(u, v)H(u,v): degradation function in frequency domain
* F(u,v)F(u, v)F(u,v): original image in frequency domain
* N(u,v)N(u, v)N(u,v): noise in frequency domain

The **MMSE filter** aims to estimate F(u,v)F(u, v)F(u,v) as:

F^(u,v)=H∗(u,v)∣H(u,v)∣2+Sη(u,v)Sf(u,v)⋅G(u,v)\hat{F}(u, v) = \frac{H^\*(u, v)}{|H(u, v)|^2 + \frac{S\_{\eta}(u, v)}{S\_f(u, v)}} \cdot G(u, v)F^(u,v)=∣H(u,v)∣2+Sf​(u,v)Sη​(u,v)​H∗(u,v)​⋅G(u,v)

Where:

* H∗(u,v)H^\*(u, v)H∗(u,v): complex conjugate of degradation function
* Sη(u,v)S\_{\eta}(u, v)Sη​(u,v): power spectral density (PSD) of the noise
* Sf(u,v)S\_f(u, v)Sf​(u,v): power spectral density of the original image

**3. Interpretation of the MMSE Formula**

* The term Sη(u,v)Sf(u,v)\frac{S\_{\eta}(u, v)}{S\_f(u, v)}Sf​(u,v)Sη​(u,v)​ acts as a **regularization factor** that suppresses noise based on the signal-to-noise ratio.
* When noise is high, this term dominates and reduces the contribution of high-frequency components, which often carry noise.
* When noise is low or the image has strong components, the filter behaves more like an **inverse filter**.

**4. Comparison with Wiener Filter**

The MMSE filter is very closely related to the **Wiener filter**. In fact, Wiener filtering is a practical implementation of MMSE filtering under certain assumptions. The Wiener filter is given by:

F^(u,v)=H∗(u,v)∣H(u,v)∣2+K⋅G(u,v)\hat{F}(u, v) = \frac{H^\*(u, v)}{|H(u, v)|^2 + K} \cdot G(u, v)F^(u,v)=∣H(u,v)∣2+KH∗(u,v)​⋅G(u,v)

Where K=Sη(u,v)Sf(u,v)K = \frac{S\_{\eta}(u, v)}{S\_f(u, v)}K=Sf​(u,v)Sη​(u,v)​ is assumed to be constant.

Hence, **MMSE is a generalization** of Wiener filtering that allows KKK to vary with frequency.

**5. Advantages of MMSE Filtering**

* **Optimality**: Minimizes mean square error—one of the best criteria for restoration.
* **Noise Adaptation**: Automatically adjusts based on the signal and noise characteristics.
* **Flexible**: Works for both white and colored noise.
* **Theoretical foundation**: Built on solid statistical estimation theory.

**6. Limitations of MMSE Filtering**

* **Requires Prior Knowledge**: Needs PSDs of the signal and noise, which may not be readily available.
* **Computational Complexity**: More intensive than simpler methods like inverse or averaging filters.
* **Frequency Assumptions**: Assumes the degradation and noise are stationary processes.

**7. Applications of MMSE Filtering**

* **Satellite and astronomical image restoration**
* **Medical image processing** (e.g., removing noise from MRI scans)
* **Blur correction** in surveillance footage
* **Restoration of scanned documents** with fading or background noise

**Conclusion**

The **Minimum Mean Square Error (MMSE) filter** is a powerful and optimal technique for image restoration, especially when noise and degradation models are well-understood. By minimizing the average squared difference between the original and estimated images, it offers a statistically sound way to recover high-quality images from corrupted observations. While it may be computationally demanding and require prior knowledge of signal and noise statistics, its performance in terms of quality restoration makes it a preferred method in high-accuracy applications.

# 8. Mean and Geometric Mean Filter for Image Restoration

Mean Filter: The mean filter is a linear filter that averages the pixel values in a neighborhood, reducing noise.  
Geometric Mean Filter: This filter replaces a pixel with the geometric mean of its neighborhood, often used for noise reduction in images.

In the domain of **image restoration**, various filters are used to suppress noise and recover the original image details. Among them, **mean filtering** techniques are fundamental due to their simplicity and effectiveness, especially against certain types of noise like Gaussian or uniform noise.

In this explanation, we’ll focus on:

* **Arithmetic Mean Filter**
* **Geometric Mean Filter**

Both are smoothing filters but differ in how they compute the average and how they respond to noise and image features.

**🟠 1. Arithmetic Mean Filter**

**Definition:**

The **Arithmetic Mean Filter** replaces the value of a pixel with the **average (arithmetic mean)** of the gray levels in the neighborhood defined by a window (usually a square matrix like 3×3, 5×5, etc.).

**Mathematical Formula:**

f^(x,y)=1mn∑(s,t)∈Sxyg(s,t)\hat{f}(x, y) = \frac{1}{mn} \sum\_{(s,t) \in S\_{xy}} g(s,t)f^​(x,y)=mn1​(s,t)∈Sxy​∑​g(s,t)

Where:

* f^(x,y)\hat{f}(x, y)f^​(x,y) is the restored pixel value.
* SxyS\_{xy}Sxy​ is the neighborhood window centered at pixel (x, y).
* g(s,t)g(s,t)g(s,t) is the intensity value at point (s,t) in the degraded image.
* m×nm \times nm×n is the size of the window.

**Working:**

* It smooths the image by reducing intensity variations between neighboring pixels.
* Effective in removing Gaussian and uniform noise.
* However, it **blurs** sharp edges and fine details.

**Example:**

For a 3×3 window with pixel values:

[120122124123125126121124123]\begin{bmatrix} 120 & 122 & 124 \\ 123 & 125 & 126 \\ 121 & 124 & 123 \end{bmatrix}​120123121​122125124​124126123​​

Arithmetic Mean = 120+122+124+123+125+126+121+124+1239=11089≈123.11\frac{120 + 122 + 124 + 123 + 125 + 126 + 121 + 124 + 123}{9} = \frac{1108}{9} \approx 123.119120+122+124+123+125+126+121+124+123​=91108​≈123.11

The center pixel is replaced with 123.

**🟢 2. Geometric Mean Filter**

**Definition:**

The **Geometric Mean Filter** is another neighborhood averaging method where the pixel value is replaced by the **geometric mean** of the neighborhood values.

**Mathematical Formula:**

f^(x,y)=[∏(s,t)∈Sxyg(s,t)]1mn\hat{f}(x, y) = \left[ \prod\_{(s,t) \in S\_{xy}} g(s,t) \right]^{\frac{1}{mn}}f^​(x,y)=​(s,t)∈Sxy​∏​g(s,t)​mn1​

Where:

* ∏\prod∏ denotes the product of all pixel values in the window.
* All values g(s,t)g(s,t)g(s,t) must be strictly positive.

**Working:**

* It **preserves edge features** better than arithmetic mean.
* Reduces noise while retaining more detail.
* Especially effective against **multiplicative noise** like speckle noise.
* Less sensitive to outliers than arithmetic mean.

**Example:**

Using the same values from above:

Geometric Mean=120⋅122⋅124⋅123⋅125⋅126⋅121⋅124⋅1239\text{Geometric Mean} = \sqrt[9]{120 \cdot 122 \cdot 124 \cdot 123 \cdot 125 \cdot 126 \cdot 121 \cdot 124 \cdot 123}Geometric Mean=9120⋅122⋅124⋅123⋅125⋅126⋅121⋅124⋅123​

This would result in a pixel value slightly lower than the arithmetic mean (since geometric mean is always ≤ arithmetic mean).

**🔵 Comparison of Arithmetic vs Geometric Mean Filters**

| **Feature** | **Arithmetic Mean Filter** | **Geometric Mean Filter** |
| --- | --- | --- |
| Type of Noise Handled | Gaussian, uniform | Multiplicative (e.g., speckle) |
| Edge Preservation | Poor | Better |
| Smoothing Strength | High (causes blur) | Moderate |
| Outlier Sensitivity | High | Low |
| Computation Complexity | Low | Moderate |

**🟣 Applications**

* **Arithmetic Mean**:
  + Basic smoothing operations
  + Reducing low-level random noise
  + Used in preprocessing pipelines
* **Geometric Mean**:
  + Medical imaging (speckle noise)
  + Remote sensing
  + Synthetic aperture radar (SAR) image restoration

**🔻 Limitations**

* **Arithmetic Mean Filter**:
  + Can overly smooth the image and remove important texture details.
  + Poor at handling salt-and-pepper noise.
* **Geometric Mean Filter**:
  + Computationally heavier due to multiplication and root extraction.
  + Requires all pixel values to be positive and non-zero.

**Conclusion**

Both **mean and geometric mean filters** are powerful tools in image restoration, each with its own strengths and weaknesses. While the arithmetic mean is simple and effective for general smoothing, the geometric mean is preferred when noise is multiplicative and preserving detail is critical. Choosing the right filter depends on the nature of the noise and the restoration objective.

# 9. Huffman Coding Procedure for Image Compression

**Huffman coding** is a **lossless data compression technique** used in image processing and other data encoding systems. It is a type of **entropy encoding** algorithm that is optimal when symbols have **non-uniform** probabilities.

This method reduces the average code length used to represent each symbol by assigning **shorter codes to more frequent symbols** and **longer codes to less frequent symbols**.

**🔶 Why Use Huffman Coding in Images?**

In digital images, certain pixel intensity values (like 0, 255, etc.) might occur more frequently than others. Huffman coding compresses such images by replacing frequently occurring pixel values with **short binary codes**, effectively reducing the storage space.

**🔷 Steps in Huffman Coding**

Let’s go through the standard **procedure** for generating Huffman codes for a given set of symbols and their frequencies.

**🔹 Step 1: List Symbols with Frequencies**

Let’s consider the following symbols and their frequencies:

| **Symbol** | **Frequency** |
| --- | --- |
| A | 5 |
| B | 9 |
| C | 12 |
| D | 13 |
| E | 16 |
| F | 45 |

**🔹 Step 2: Build a Min-Heap Tree (Huffman Tree)**

Create a **leaf node** for each symbol and insert them into a priority queue (min-heap), ordered by frequency.

**While there is more than one node** in the heap:

Remove two nodes with the **lowest frequencies**.

Create a **new internal node** with frequency equal to the sum of these two nodes.

Set the two nodes as **children** of the new node.

Insert the new node back into the heap.

The final node in the heap is the **root** of the Huffman Tree.

**🔹 Step 3: Assign Binary Codes**

Traverse the tree from the root:

Assign 0 for left edge, 1 for right edge.

Concatenate edge values along the path to each symbol.

**🔷 Huffman Tree Construction for Our Example**

Let's construct the tree step-by-step:

Initial nodes:  
A(5), B(9), C(12), D(13), E(16), F(45)

Combine A(5) + B(9) = Node1(14)  
Remaining: Node1(14), C(12), D(13), E(16), F(45)

Combine C(12) + D(13) = Node2(25)  
Remaining: Node1(14), Node2(25), E(16), F(45)

Combine Node1(14) + E(16) = Node3(30)  
Remaining: Node2(25), Node3(30), F(45)

Combine Node2(25) + Node3(30) = Node4(55)  
Remaining: Node4(55), F(45)

Combine F(45) + Node4(55) = Root(100)

**🔷 Huffman Tree Diagram (Sketch Format)**

css

CopyEdit

[100]

/ \

[45] [55]

/ \

[25] [30]

/ \ / \

[12] [13] [14] [16]

/ \

[5] [9]

**🔹 Step 4: Assign Binary Codes**

Now we assign codes starting from the root:

F: 0

Node4 (right): 1

Node2 (25): 10

C: 100

D: 101

Node3 (30): 11

Node1 (14): 110

A: 1100

B: 1101

E: 111

**🔷 Final Huffman Codes:**

| **Symbol** | **Frequency** | **Huffman Code** |
| --- | --- | --- |
| F | 45 | 0 |
| C | 12 | 100 |
| D | 13 | 101 |
| A | 5 | 1100 |
| B | 9 | 1101 |
| E | 16 | 111 |

**🔷 Advantages of Huffman Coding**

**Lossless Compression**: No data is lost.

**Optimal for Known Probabilities**: Provides the most efficient encoding.

**Widely Used**: Used in JPEG, PNG, MP3, and other formats.

**🔷 Limitations**

**Requires Full Data Knowledge**: Frequencies must be known beforehand.

**Not Ideal for Small Files**: Overhead of tree construction may dominate.

**No Adaptive Learning**: Doesn’t adapt unless modified for streaming data.

**🔷 Applications in Image Processing**

**JPEG Compression**: After DCT, Huffman coding compresses the quantized coefficients.

**Binary Image Compression**: Works well where pixel patterns repeat.

**Lossless Image Formats**: PNG uses a variant of Huffman coding.

# 10. Block Diagram of General Image Compression System

Image compression is a critical process in digital image processing, especially when transmitting or storing large image files efficiently. Compression reduces the number of bits required to represent an image without significant degradation of visual quality.

**🔶 Objective of Image Compression**

* Reduce storage space.
* Reduce transmission bandwidth.
* Maintain image quality (depending on whether it's **lossy** or **lossless**).

**🔷 Types of Compression**

* **Lossless Compression**: No information is lost. Perfect for medical, legal, and scientific images (e.g., PNG).
* **Lossy Compression**: Some data is lost, but the image remains visually acceptable (e.g., JPEG, WebP).

**🔷 General Block Diagram of an Image Compression System**

Here is a simplified sketch of the **function block diagram**:

pgsql

CopyEdit

+--------------+ +------------------+ +-------------+

| Input Image | --> | Transformation | --> | Quantization|

+--------------+ +------------------+ +-------------+

|

v

+------------------+ +-------------------+

| Encoding (e.g., | --> | Compressed Image |

| Huffman/Arithmetic)| | Output |

+------------------+ +-------------------+

**🔶 Functional Blocks Explained**

**1. Input Image**

* The raw image is a 2D matrix of pixel intensity values.
* Can be grayscale or RGB.

**2. Transformation**

* Converts the image into a **frequency domain** or another domain that is more compressible.
* Common Transformations:
  + **DCT (Discrete Cosine Transform)** – used in JPEG.
  + **Wavelet Transform** – used in JPEG2000.
  + **Fourier Transform** – theoretical basis.
* Purpose:
  + Energy compaction: most important info moves to fewer coefficients.
  + Removes redundancy between neighboring pixels.

**3. Quantization**

* Converts transformed coefficients to a reduced precision.
* This is where **lossy compression** mainly occurs.
* Rounds off small insignificant values to zero (which saves space).
* **Scalar quantization** or **vector quantization** methods are used.

**Example** (Quantization Table in JPEG):

| **Coefficient** | **Original** | **Quantized** |
| --- | --- | --- |
| DCT(0,0) | 120 | 12 |
| DCT(1,1) | 8 | 1 |
| DCT(3,4) | 2 | 0 |

**4. Encoding**

* Encodes quantized values to bitstream efficiently.
* Common Techniques:
  + **Huffman Coding**
  + **Arithmetic Coding**
  + **Run-Length Encoding (RLE)**
* Objective:
  + Represent frequent values with shorter codes.
  + Reduce bit redundancy further.

**5. Compressed Image Output**

* Final output is a compact binary file.
* Can be stored or transmitted.
* May have formats like .jpg, .jpeg, .png, .webp.

**🔷 Decoder Side (Reconstruction Block Diagram)**

The reverse process happens at the receiver's end:

pgsql

CopyEdit

+--------------------+ +------------------+ +----------------+

| Compressed Image | --> | Decoding | --> | Dequantization|

+--------------------+ +------------------+ +----------------+

|

v

+-------------------+ +-----------------+

| Inverse Transform | --> | Reconstructed |

| (IDCT, IWT, etc) | | Image |

+-------------------+ +-----------------+

**🔶 Key Considerations**

* **Compression Ratio**:

Compression Ratio=Original SizeCompressed Size\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}}Compression Ratio=Compressed SizeOriginal Size​

A higher ratio indicates better compression.

* **Trade-Off**:
  + More compression → lower quality (lossy).
  + Less compression → higher quality (lossless).
* **Performance Metrics**:
  + **PSNR (Peak Signal-to-Noise Ratio)** – for image quality.
  + **MSE (Mean Squared Error)** – for distortion.

**🔶 Applications**

* **Web Development** – faster image loading (JPEG, WebP).
* **Satellite Imaging** – high-resolution images, compressed for transmission.
* **Medical Imaging** – requires lossless formats (DICOM, PNG).
* **Digital Cameras** – use JPEG for space-saving.

**🔚 Conclusion**

The general image compression system is a sophisticated combination of transformation, quantization, and encoding techniques. The purpose is to make image storage and transmission more efficient while balancing visual quality and size. Modern image formats like JPEG are perfect examples of how this system is implemented in real-world applications.