Unit - 4: Image Compression
Part A: Theory

1. **Explain the need for image compression in multimedia applications. How does compression impact storage and transmission efficiency?**

Image compression is crucial in multimedia applications due to the large size of image files, which can impact storage requirements and data transmission rates. Compressing images reduces file sizes, making it easier to store, manage, and transfer images across networks.

Compression is necessary to handle the vast amount of data generated by high-resolution images in multimedia. Without it, storage and bandwidth would be insufficient, especially in high-traffic applications like video streaming and cloud storage. Compression helps to reduce storage space and minimizes the amount of data that needs to be transmitted, which improves both network speed and accessibility.

Image compression techniques focus on reducing **redundancy** within an image. By removing unnecessary data without significantly impacting visual quality, image compression algorithms reduce file sizes and enhance **storage efficiency**. For instance, a compressed image can reduce bandwidth requirements and storage demands by removing redundant or non-essential data.

Example Calculation:

If an uncompressed image is 5 MB and compression achieves a 50% reduction, the file size reduces to 2.5 MB. This reduction speeds up both storage and transmission, as smaller files are quicker to upload/download.

Using the **Compression Ratio (CR)** formula:

$$\text{CR} = \frac{\text{Original Size}}{\text{Compressed Size}}$$

For the example above,

$$\text{CR} = \frac{5}{2.5} = 2$$

Thus, the compression ratio is 2:1, indicating that the compressed file is half the size of the original.

image compression enhances the efficiency of multimedia applications by saving space and bandwidth, making it indispensable in modern digital communications.

---

2. **What is redundancy? Explain three types of Redundancy.**

Redundancy in image processing refers to unnecessary or repetitive information that can be removed to reduce image size without compromising essential visual content. Identifying and reducing redundancy is a core principle of image compression.

There are three primary types of redundancy:

1. **Coding Redundancy**: This occurs when some symbols are encoded with more bits than necessary. For example, an 8-bit code might be used for symbols that could be represented in fewer bits. Coding techniques like Huffman and arithmetic coding address this by assigning shorter codes to frequently occurring symbols, thus reducing file size.

2. **Inter-Pixel Redundancy**: Often, adjacent pixels in an image have similar values, leading to redundant information across pixels. Compression algorithms, such as predictive coding, leverage this by encoding differences between consecutive pixels rather than each pixel independently.

3. **Psycho-Visual Redundancy**: Certain details in images are not perceivable by the human eye. Lossy compression techniques, like JPEG, exploit this by removing minor details that don't significantly affect perceived image quality.

Example Table:

Redundancy Type	**Description**	**Example Compression Method**
Coding Redundancy	Redundant symbol encoding	Huffman Coding
Inter-Pixel Redundancy	Redundant adjacent pixel data	Predictive Coding
Psycho-Visual Redundancy	Details invisible to the human eye	JPEG (Lossy Compression)

Reducing these types of redundancies is crucial for achieving efficient image compression, allowing for smaller file sizes and faster data handling without perceptible quality loss.

---

3. **Define coding redundancy. Provide examples of how coding redundancy is used to reduce image file sizes.**

Coding redundancy refers to the use of extra bits for encoding information in images. Reducing coding redundancy is a key strategy in image compression, where efficient coding assigns shorter codes to frequently occurring symbols.

In image files, some pixel values or colours appear more frequently than others. If all values are encoded with the same number of bits, it leads to inefficiencies. By assigning shorter binary codes to more frequent symbols and longer codes to less frequent ones, coding redundancy is minimized.

Examples of Coding Redundancy in Image Compression:

1. Huffman Coding

• **Description:**

Huffman coding is a variable-length coding technique that assigns shorter codes to more frequent symbols and longer codes to less frequent ones.

By using fewer bits for common pixel values, Huffman coding reduces the total number of bits required to store an image.

• **Example:**

In a grayscale image with a dark background and occasional light areas, the dark pixels may represent 90% of the image. Instead of using 8 bits for each pixel, Huffman coding can use shorter codes for these common dark values, significantly reducing the file size.

2. Run-Length Encoding (RLE)

• **Description:**

Run-Length Encoding compresses data by replacing sequences (or “runs") of the same value with a single value and a count. This is especially

effective in images with large areas of uniform colour, such as logos or cartoons.

• Example: In an image with a long sequence of white pixels, RLE would store

this as "white, 100" instead of repeating "white" 100 times, resulting in a more

compact representation and a smaller file size.

3. Arithmetic Coding

• **Description:**

Arithmetic coding encodes an entire message as a single number between 0 and 1, based on the probabilities of different symbols. This can

achieve higher compression ratios than Huffman coding by handling no integer bits per symbol.

• **Example:**

In an image with specific colour intensities that are more probable, arithmetic coding would assign a unique range to each intensity based on its

probability, storing the entire image data in a compact, encoded format.

Suppose we have an image with symbols **A, B, C, D,** appearing with frequencies 45%, 25%, 15%, and 15%, respectively. Using Huffman coding, **A** would receive the shortest code, reducing overall data usage. The **Compression Ratio** (CR) for Huffman coding is calculated as:

$$\text{CR} = \frac{\text{Original bits per symbol}}{\text{Average bits per symbol in Huffman}}$$

4. **Discuss inter-pixel redundancy and how it is exploited in image compression algorithms. Provide examples of common methods to reduce inter-pixel redundancy.**

Inter-pixel redundancy refers to the similarity between adjacent pixels in an image. This redundancy can be removed to reduce image size, as closely related pixel values can be efficiently represented.

In most images, adjacent pixels have similar intensity or colour values. Compression algorithms exploit inter-pixel redundancy by encoding only the differences between pixels rather than each pixel individually. This not only reduces the data needed for storage but also preserves image quality.

Example Methods:

1. **Predictive Coding**: Predictive coding uses previous pixel values to predict the current pixel, and only the difference between the actual and predicted value is encoded.

   If pixel **P** is predicted based on pixel **Q**, the encoded data is P−QP - QP−Q, which is typically a smaller value, leading to fewer bits.

2. **Run-Length Encoding (RLE)**: When there are long sequences of identical pixel values, RLE replaces these sequences with a single value and count.

   For instance, a sequence like **[255, 255, 255, 0, 0, 0]** becomes **(255,3), (0,3)**.

3. **Transform Coding**: Transform coding, particularly the Discrete Cosine Transform (DCT), converts spatial pixel values into frequency components.

   Low-frequency components often dominate, so high-frequency data (which changes rapidly between pixels) can be discarded in lossy formats.

Example Table:

Method	**Description**	**Data Reduction Approach**
Predictive Coding	Encodes pixel difference	Reduces repetitive encoding of similar values
Run-Length Encoding	Encodes sequences of identical pixels	Minimizes data for uniform areas
Transform Coding	Converts pixels to frequency domain	Reduces insignificant high frequencies

---

5. **Compare and contrast lossy and lossless image compression techniques. Provide examples of when each type of compression is more appropriate.**

Feature	Lossy Compression	Lossless Compression
Definition	Reduces file size by permanently eliminating some data.	Compresses data without losing any information.
Data Integrity	Some original data is lost; the reconstructed image may differ from the original.	All original data can be perfectly restored.
File Size	Generally, results in smaller file sizes, making it suitable for web use.	Results in larger file sizes compared to lossy compression, as no data is lost.
Image Quality	Quality may degrade significantly at high compression levels, leading to artifacts.	Maintains the original image quality, with no artifacts introduced.
Encoding Techniques	Common techniques include JPEG, WebP, and some forms of MP3 audio.	Common techniques include PNG, TIFF, and GIF.
Application Areas	Best suited for photographs, videos, and web images where smaller file size is crucial and minor quality loss is acceptable.	Ideal for images requiring high fidelity, such as medical imaging, technical drawings, and images with text.
Reusability	Not suitable for repeated editing, as quality can degrade with each save.	Allows for repeated editing and saving without quality loss.

6. **Explain Compression Ratio with an Example. What other metrics help in understanding the quality of the compression?**

Compression ratio is an essential metric for understanding the effectiveness of an image compression technique, but it doesn't fully convey image quality. Additional metrics like Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), Mean Squared Error (MSE), and Bitrate (or Bits per Pixel, BPP) help evaluate the impact of compression on image quality.

Compression Ratio: The **Compression Ratio (CR)** is defined as:

$$\textbf{CR} = \frac{\textbf{Original Size}}{\textbf{Compressed Size}}$$

A higher compression ratio indicates more significant size reduction. For example, if an image's original size is 12 MB and is compressed to 3 MB:

$$\textbf{CR} = \frac{\textbf{12 MB}}{\textbf{3 MB}} = \mathbf{4}$$

This implies a 4:1 reduction in file size.

Additional Metrics:

1. Peak Signal-to-Noise Ratio (PSNR):

PSNR measures the ratio between the maximum possible pixel value and the distortion introduced by compression, giving an objective assessment of image quality. A higher PSNR indicates better quality.

Formula:

$$\textbf{PSNR} = \mathbf{10} \cdot \log_{10}\left(\frac{\textbf{MAX}^2}{\textbf{MSE}}\right)$$

- **MAX:** The maximum pixel value, usually 255 for 8-bit images.

- **MSE:** Mean Squared Error between the original and compressed images.

Example Calculation:

For an 8-bit grayscale image with MAX = 255 and an MSE of 20:

$$\textbf{PSNR} = 10 \cdot \log_{10}\left(\frac{255^2}{20}\right) \approx 34.15\ \textbf{dB}$$

Higher PSNR values (typically above 30 dB) suggest acceptable quality.

2. Mean Squared Error (MSE):

MSE quantifies the average squared difference between the original and compressed images, providing a measure of the error.

Formula:

$$\textbf{MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[I(i,j) - K(i,j)]^2$$

- M, N: Dimensions of the image.

- $I(i,j)$ and $K(i,j)$: Pixel values of the original and compressed images at position (i,j).

A lower MSE indicates less error and better compression quality.

3. Structural Similarity Index (SSIM):

SSIM is a perceptual metric that evaluates image quality based on changes in luminance, contrast, and structure, providing a score between 0 and 1, with 1 indicating perfect similarity.

Formula:

$$\textbf{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

-$\mu x, \mu y$ Mean intensities of images x and y

- σx2, σy2: Variances.

- $\boldsymbol{\sigma xy}$: Covariance.

- $\boldsymbol{C1}, \boldsymbol{C2}$: Constants to stabilize the division.

Example:

An SSIM of 0.95 suggests the compressed image maintains high structural similarity to the original.

4. Bitrate (or Bits per Pixel - BPP):

BPP indicates the amount of data used per pixel in the compressed image and is defined as:

$$\textbf{BPP} = \frac{\textbf{Total bits in the compressed image}}{\textbf{Total number of pixels}}$$

Example:

For an image with 1,000,000 pixels and a compressed file size of 2,000,000 bits:

$$\textbf{BPP} = \frac{\boldsymbol{2,000,000}}{\boldsymbol{1,000,000}} = \textbf{2 bits per pixel}$$

Lower BPP values indicate higher compression but may reduce image quality.

Table:

Metric	Definition	Formula / Description	Interpretation
Compression Ratio	Ratio of original to compressed size	$\textbf{CR} = \frac{\textbf{Original Size}}{\textbf{Compressed Size}}$	Higher is better for storage efficiency
PSNR	Peak signal-to-noise ratio	$\textbf{PSNR} = 10 \cdot \log_{10}\left(\frac{\textbf{MAX}^2}{\textbf{MSE}}\right)$	Higher values indicate better quality
MSE	Mean squared error	$\textbf{MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[I(i,j) - K(i,j)]^2$	Lower values indicate less distortion
SSIM	Structural similarity index	$\textbf{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$	Closer to 1 means high similarity
BPP	Bits per pixel	$\textbf{BPP} = \frac{\textbf{Total bits in the compressed image}}{\textbf{Total number of pixels}}$	Lower indicates more compression

These metrics together provide a comprehensive understanding of the quality and efficiency of image compression.

7. **Identify Pros and Cons of the following algorithms**

- **Huffman coding,**
- **Arithmetic coding,**
- **LZW coding,**
- **Transform coding,**
- **Run-length coding.**

Algorithm	Pros	Cons	Detailed Example
Huffman Coding	1. Optimal for known symbol frequencies: Provides minimal redundancy.	1. Not adaptive: Requires prior knowledge of symbol probabilities.	ZIP File Compression: Compresses text by encoding common characters with shorter bit sequences.
	2. Simple to implement: Widely used in standard data compression.	2. Inefficient for small datasets: Code length may not be compact for short messages.	Image Compression: Assigns shorter binary codes to frequently occurring pixel values in images.
	3. Efficient decoding: Fast and straightforward decoding using binary trees.	3. Preprocessing needed: Requires creating the coding tree before compression.	PNG Images: Uses Huffman coding as part of the DEFLATE algorithm for efficient lossless compression.
Arithmetic Coding	1. Higher compression efficiency: Achieves better compression	1. Computationally intensive: More complex to implement,	H.264 Video Codec: Compresses video data more efficiently than Huffman

Algorithm	Pros	Cons	Detailed Example
	than Huffman for larger symbol sets.	especially for large data.	coding, saving bandwidth.
	2. Adaptive to real-time data: Works well for streaming and dynamic data.	2. Historical licensing concerns: Limited use due to past patent issues.	JPEG2000: Utilizes arithmetic coding for better compression rates compared to standard JPEG.
	3. Fractional bit encoding: Can encode with non-integer bits for optimal efficiency.	3. Complex decoding: Requires advanced algorithms for practical use.	File Archiving: Applied in some advanced data compressors for high-efficiency results.
LZW Coding	1. Dictionary-based compression: Efficiently compresses data with repeated patterns.	1. Ineffective for highly diverse data: Performs poorly if data has few repetitions.	GIF Image Format: Encodes simple images by reducing repetitive patterns, widely used in web graphics.
	2. No need to transmit dictionary: Reduces overhead as the decoder rebuilds it.	2. Memory usage: High memory requirements for large dictionaries.	UNIX Compress Tool: Utilizes LZW for file compression, effectively reducing file sizes with repeated content.
	3. Easy implementation: Popular in file compression utilities.	3. Suboptimal for small files: Dictionary creation can	TIFF Format: Used in LZW-compressed TIFF files for high-

Algorithm	Pros	Cons	Detailed Example
		consume initial space.	quality image storage.
Transform Coding	1. Effective for lossy compression: Reduces data by transforming it into frequency components.	1. High computational complexity: Requires substantial processing power.	JPEG Image Compression: Uses DCT to transform and discard high-frequency components, saving space.
	2. Removes psycho-visual redundancies: Discards details less visible to the human eye.	2. Potential for artifacts: Block-based compression can result in visible artifacts.	Audio Compression: Applied in MP3 encoding, where transform coding reduces less perceptible frequencies.
	3. Better compression of important data: Focuses on low-frequency elements.	3. Lossy nature: Can lead to quality degradation.	HEVC (H.265): Uses transform coding to compress video by separating low and high-frequency content.
Run-length Coding	1. Simple and efficient: Compresses sequences of repeated data with minimal processing.	1. Not suitable for complex data: Increases file size when data lacks repetition.	Fax Transmission: Encodes long sequences of identical pixels (e.g., black or white) for efficient transmission.

Algorithm	Pros	Cons	Detailed Example
	2. Low computational cost: Easy to implement and decode in real-time.	2. Limited to specific cases: Works best with data that has long runs of the same value.	Bitmap Compression: Reduces size for simple images with large areas of uniform colour.
	3. Real-time use: Ideal for simple on-the-fly compression.	3. Ineffective for detailed images: Does not handle diverse pixel values well.	RLE in PCX Format: Used in early computer graphics formats for efficient compression of uniform areas.

8. **Perform Huffman coding on a given set of pixel values. Show the step-by-step process and calculate the compression ratio achieved.**

Define Pixel Values and Their Frequencies Suppose we have the following pixel values and their frequencies:

Pixel Value	Frequency
A	45
B	13
C	12
D	16
E	9
F	5

Build the Huffman Tree

1.Sort Pixel Frequencies:

- Start by arranging pixel values by their frequencies in ascending order:
- Initial order: F (5), E (9), C (12), B (13), D (16), A (45)

2.Combine the Two Lowest-Frequency Nodes:

- Combine F (5) and E (9) to form a new node with a combined frequency of 14.
- Updated order: C (12), B (13), (F+E:14), D (16), A (45)

3.Repeat Combinations:

- Combine C (12) and B (13) to form a new node with a frequency of 25.
- Updated order: (F+E:14), D (16), (C+B:25), A (45)
- Combine (F+E:14) and D (16) to form a node with a frequency of 30.
- Updated order: (C+B:25), (F+E+D:30), A (45)

- Combine (C+B:25) and (F+E+D:30) to create a node with a frequency of 55.
- Final combination: A (45) + (C+B+F+E+D:55) = 100

Huffman Tree Structure:

$$
\begin{array}{c}
100 \\
/ \quad \backslash \\
A \quad 55 \\
/ \quad \backslash \\
25 \qquad 30 \\
/\backslash \qquad / \quad \backslash \\
C \quad B \quad F+E \quad D
\end{array}
$$

Assign Binary Codes to Each Pixel Value Assign 0 to the left branch and 1 to the right branch:

Pixel Value	Huffman Code
A	0
B	110
C	111
D	101
E	1001
F	1000

Calculate Total Bit Usage with Huffman Coding

- Calculate the total bits required:

- Total Bit Length $= (5 \times 1000) + (9 \times 1001) + (12 \times 111) + (13 \times 110) + (16 \times 101) + (45 \times 0)$
- $= 25 + 81 + 132 + 156 + 160 + 0 = 554\ bits$

- **Compression Ratio Formula**:

$$\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}}$$

Example Calculation:

If the original size was 800 *bits, then*:

$$\text{Compression Ratio} = \frac{800}{554} \approx 1.44$$

Table of Codes:

Pixel Value	Frequency	Huffman Code
A	45	0
B	13	110
C	12	111
D	16	101
E	9	1001
F	5	1000

---

9. **Explain the concept of arithmetic coding and how it differs from Huffman coding. Why is arithmetic coding considered more efficient in some cases?**

Aspect	Arithmetic Coding	Huffman Coding
Encoding Process	Encodes the entire message as a fractional number in the range [0, 1), iteratively narrowing the interval.	Encodes each symbol with a variable-length binary code based on symbol frequency.
Symbol Representation	Represents a sequence of symbols as a single continuous fractional value.	Represents each symbol as a fixed-length or variable-length binary code.
Efficiency	More efficient for skewed distributions (non-uniform or fractional probabilities).	Less efficient for skewed data; works best for uniform or near-uniform distributions.
Memory Usage	Requires more memory for precision in storing fractional values during encoding and decoding.	Requires less memory, as it uses a binary tree structure.
Complexity	Computationally complex due to fractional arithmetic and continuous interval updates.	Simpler to implement, involving tree construction and binary code assignment.
Compression Ratio	Can achieve better compression ratios, especially for data with highly uneven symbol frequencies.	Provides good compression but may be less efficient for long-tailed distributions.
Code Length	Can use fractional bits for encoding, providing a more compact representation.	Uses integer-based code lengths, typically in bits, which can lead to less compact encoding.
Handling Large Alphabets	More efficient with large symbol sets or continuous data, as it doesn't rely on a binary tree structure.	Can become inefficient with large alphabets because the binary tree grows in size with the number of symbols.

Arithmetic coding is considered more efficient in some cases because:

- It encodes data with **fractional bit precision**, closely matching the entropy, unlike Huffman coding's integer bit lengths.

- **Higher compression efficiency** is achieved, especially for data with uneven symbol probabilities.

- It handles **large symbol sets** and **non-uniform distributions** effectively.

- **Adaptive capabilities** allow real-time updating of probabilities, improving efficiency for changing data patterns.

- Represents entire messages as a single number, minimizing **overhead** associated with individual code boundaries in Huffman coding.

10.Provide an example of LZW coding on a simple sequence of image pixel values.

LZW (Lempel-Ziv-Welch) coding is a dictionary-based compression algorithm that dynamically builds a dictionary based on input data patterns. This method is effective for compressing data with repetitive sequences.

Example Sequence of Pixel Values

Consider the simple image pixel sequence:

A B A B A B A B

Step-by-Step LZW Coding Process

1. Initialize the Dictionary:

- Start with a dictionary containing each unique symbol in the sequence.
- Assign an initial index to each symbol:
  - A = 0
  - B = 1

Initial dictionary:

Symbol	Index
A	0
B	1

2. Encoding Process:

- Scan the sequence and build the dictionary by adding new patterns incrementally.

- Output the index for the longest current phrase found in the dictionary, then add the new phrase (current phrase + next symbol) to the dictionary.

Encoding Steps:

Step	Current Phrase	Next Symbol	Dictionary Output	New Dictionary Entry	Index
1	A	B	0	AB	2
2	B	A	1	BA	3
3	A	B	0	Already exists	-
4	AB	A	2	ABA	4
5	A	B	0	Already exists	-
6	AB	(End)	2	-	-

Final Encoded Output: The LZW encoded sequence for the input A B A B A B A B is: 0, 1, 0, 2, 0, 2.

3. Final Dictionary:

Symbol	Index
A	0
B	1
AB	2
BA	3
ABA	4

4. Decoding the Output:

To reconstruct the original sequence from the encoded output 0, 1, 0, 2, 0, 2, use the dictionary to map each index and retrieve the original data pattern.

11. What is transform coding? Explain how it helps in compressing image data by reducing redundancies in the frequency domain.

Transform Coding: Transform coding is a key technique in image and signal compression. It converts image data from the spatial domain (pixel representation) to the frequency domain. The primary goal is to represent image data in a way that concentrates energy into a few significant coefficients, enabling more efficient data compression.

1. Basic Concept of Transform Coding: Transform coding applies mathematical transforms to image data to reduce redundancies. It leverages the idea that in many images, pixel values are correlated, leading to redundant information. By converting image data into the frequency domain, these redundancies can be isolated and effectively compressed.

2. Mathematical Representation: The transformation can be represented as:

$$Y = T \cdot X \cdot T^T$$

where:

- Y is the transformed matrix in the frequency domain.
- X is the original image matrix in the spatial domain.
- T is the transformation matrix (e.g., Discrete Cosine Transform matrix).

3. Common Transforms Used:

- **Discrete Cosine Transform (DCT):** Most widely used in image compression (e.g., JPEG).
- **Fourier Transform (DFT):** Suited for complex frequency analysis.
- **Wavelet Transform:** Used in JPEG 2000 for multi-resolution analysis.

4. Example of DCT in Image Compression: To illustrate how transform coding works, consider a small 4×44 \times 44×4 image block:

$$X = \begin{bmatrix} 52 & 55 & 61 & 66 \\ 70 & 61 & 64 & 73 \\ 63 & 59 & 66 & 90 \\ 71 & 62 & 68 & 85 \end{bmatrix}$$

Applying the DCT: The DCT matrix T for a 4×4 block is applied to X to yield the frequency matrix Y:

$$Y = T \cdot X \cdot T^T$$

The resulting Y matrix has high-frequency components concentrated in the bottom right, which can be quantized more aggressively.

5. How Transform Coding Reduces Redundancies: Transform coding reduces spatial redundancies by isolating the high-frequency (detail) and low-frequency (basic structure) components:

- Low-Frequency Coefficients: Represent the essential image structure and have higher values.
- High-Frequency Coefficients: Represent finer details and typically have lower values that can be quantized to zero.

This concentration of energy in fewer coefficients means that data storage can be reduced by only storing significant coefficients, while smaller coefficients are often discarded or heavily quantized.

6. Quantization Process: The transformed coefficients are divided by a quantization matrix to simplify storage. For example:

$$Q = \frac{Y}{Q_{\text{matrix}}}$$

A higher quantization factor results in more compression but can also introduce artifacts.

7. Reconstruction: The decompressed image is reconstructed by performing the inverse transform:

$$X' = T^T \cdot Y' \cdot T$$

8. Example Table of DCT Coefficients:

Coefficient Position	Value (Low Compression)	Value (High Compression)
Top Left (DC)	120	120
Bottom Right (High Freq.)	5	0

9. Visual Representation (Chart):

- Chart of Coefficients:
    - The chart shows higher values in the top-left corner representing low-frequency components.
    - The bottom-right corner, representing high-frequency components, has minimal or zero values after quantization.

10. Applications:

- JPEG Compression: Uses 8x8 DCT blocks to achieve lossy compression, balancing quality and compression rate.
- Video Compression: Formats like MPEG and H.264 use transform coding to handle motion-based redundancy.

11. Formula for Compression Metrics: Compression ratio is defined as:

$$\text{Compression Ratio} = \frac{\text{Original Size}}{\text{Compressed Size}}$$

Metrics for Quality Evaluation:

- Mean Squared Error (MSE): Measures the average squared difference between original and reconstructed images.

$$\text{MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[X(i,j) - X'(i,j)]^2$$

- Peak Signal-to-Noise Ratio (PSNR):

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

where MAX is the maximum possible pixel value of the image.

12. Significance of Sub-Image Size Selection and Blocking in Image Compression

1. Importance of Sub-Image Size Selection:

- The choice of sub-image size impacts both *compression efficiency* and *image quality*.
- Smaller blocks capture localized details better but may increase computation.
- Larger blocks reduce computational load but can struggle with high-frequency variations.

2. Blocking Process and Transformation:

- Image XXX is divided into smaller sub-images Bij:

$$X = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} B_{ij}$$

- The DCT is commonly applied to each block for transformation:

$$Y = T \cdot B_{ij} \cdot T^{T}$$

3. Application Example: JPEG Compression:

- JPEG divides images into 8x8 blocks.
- Each block is transformed using DCT and quantized to reduce data storage.
- Quantization localizes errors within blocks, reducing visible artifacts.

4. Impact on Compression Efficiency:

- Smaller blocks allow adaptive compression by applying different levels of compression based on content.
- Adaptive compression helps reduce spatial redundancy and increase overall compression efficiency.

5. Image Quality Considerations:

- Block Artifacts: Large blocks can cause visible grid lines post-decompression, especially at low bitrates.
- Detail Preservation: Smaller blocks maintain high-frequency elements like edges but may fail to represent uniform areas efficiently.
- The balance between block size and quality must be carefully managed to minimize artifacts.

6. Examples of Different Block Sizes:

Block Size	Compression Efficiency	Image Quality Impact	Artifact Risk
4x4	High	Good for edges, risk of noise	Low
8x8	Standard (JPEG)	Balanced	Moderate
16x16	Lower	Retains smooth areas	High

7. Use in Advanced Techniques:

- Variable Block Size: Used in H.265/HEVC, ranging from 4x4 to 64x64, allowing for better quality and compression.

- Sub-Block Transformations: Reduces artifacts by processing complex image areas more effectively.

8. Example Metric Representation:

- Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}[X(i,j) - X'(i,j)]^2$$

- Structural Similarity Index (SSIM):
-

$$\text{SSIM}(X, X') = \frac{(2\mu_X\mu_{X'} + C_1)(2\sigma_{XX'} + C_2)}{(\mu_X^2 + \mu_{X'}^2 + C_1)(\sigma_X^2 + \sigma_{X'}^2 + C_2)}$$

These metrics help evaluate how sub-image size impacts perceived quality.

13. Explain the process of implementing Discrete Cosine Transform (DCT) using Fast Fourier Transform (FFT). Why is DCT preferred in image compression?

1. Introduction to DCT and Its Role in Image Compression:

- The Discrete Cosine Transform (DCT) is a technique that converts an image from the spatial domain to the frequency domain, allowing for energy compaction and easier compression.
- By transforming an image into its frequency components, DCT helps in identifying and retaining significant image details while discarding less important information.

2. Mathematical Representation of DCT:

The 2D DCT for an $N \times N$ image block $X(i,j)$ is defined as:

$$\boldsymbol{Y(u,v) = \alpha(u)\alpha(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X(i,j) \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2N}\right]}$$

where:

$$\boldsymbol{\alpha(u) = f(x) = \begin{cases} \frac{1}{\sqrt{N}}, & if\ u = 0 \\ \sqrt{\frac{2}{N}}, & if\ u > 0 \end{cases}}$$

3. Implementing DCT Using FFT:

- While DCT can be computed directly, using the Fast Fourier Transform (FFT) can significantly speed up the process due to the efficiency of FFT algorithms.

- The DCT can be derived from the FFT by leveraging the following relationship:
    - Mirror the input signal to make it even, transforming the DCT into an FFT computation.
    - Perform FFT on this even signal.
    - Extract the real part of the result to obtain the DCT coefficients.

4. Step-by-Step Process:

1. **Mirror the Input Image Block**:
    - Extend the original block $\boldsymbol{X(i,j)}$ to create an even symmetric block suitable for FFT.
2. **Apply FFT**:
    - Compute the FFT on the mirrored block, which involves complex multiplications but operates in $\boldsymbol{O(N\ log\ N)}$ time.
3. **Extract DCT Coefficients**:
    - Use only the real part of the FFT output to represent DCT coefficients, as DCT inherently deals with real-valued outputs.

5. Example Calculation: Given an 8x8 image block:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x0,0} & \boldsymbol{x0,1} & \cdots & \boldsymbol{x0,7} \\ \boldsymbol{x1,0} & \boldsymbol{x1,1} & \ldots & \boldsymbol{x1,7} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x7,0} & \boldsymbol{x7,1} & \ldots & \boldsymbol{x7,7} \end{bmatrix}$$

- Extend $\boldsymbol{X}$ by creating a symmetric matrix.
- Apply FFT and retain real components to compute DCT values.

6. Why DCT is Preferred in Image Compression:

- **Energy Compaction**: DCT packs most of the significant information into a few low-frequency components. This property enables efficient compression by quantizing and encoding only the most important coefficients.
- **Low Blocking Artifacts**: Compared to other transforms like the Discrete Fourier Transform (DFT), DCT's ability to handle image discontinuities minimizes blocking artifacts when compressing images, such as in JPEG.
- **Reduced Complexity**: With optimized algorithms and integration with FFT, DCT can be computed quickly, making it ideal for real-time applications.
- **Performance Metrics**:
    - **Compression Ratio (CR)**:

$$\text{CR} = \frac{\text{Original Size}}{\text{Compressed Size}}$$

- **Peak Signal-to-Noise Ratio (PSNR)**:

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

7. Example of DCT Efficiency:

Block Size	Energy Retention (%)	Typical Usage
8x8	~95%	JPEG, video compression

8. Table of DCT and FFT Comparison:

Feature	**DCT**	**FFT**
Output Type	Real-valued coefficients	Complex-valued coefficients
Application in Images	Compression (JPEG, MPEG)	Less common in image compression
Computation Speed	Moderate	$Fast\ (\boldsymbol{O}(\boldsymbol{Nlog\ N})$

---

14. Describe how run-length coding is used in image compression, particularly for images with large areas of uniform colour. Provide an example to illustrate your explanation.

1. Overview of Run-Length Coding (RLC): Run-Length Coding (RLC) is a lossless compression method that efficiently encodes consecutive repeating values by storing the value and the count of repetitions. This reduces storage for images with uniform areas, such as simple graphics or documents.

2. How RLC Works: RLC compresses data by replacing consecutive pixel sequences with a single value followed by the number of times it repeats.

4. **Example Illustration**: Given a pixel sequence:

$$\{\mathbf{200}, \mathbf{200}, \mathbf{200}, \mathbf{150}, \mathbf{150}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}\}$$

RLC compresses this to:

(200,3), (150,2), (0,4) (200, 3), (150, 2), (0, 4) (200,3), (150,2), (0,4)

4. Detailed Explanation in Steps:

Step	Current Pixel	Run Count	Output
1	200	1 → 3	(200, 3)
2	150	1 → 2	(150, 2)
3	0	1 → 4	(0, 4)

5. **Application on 2D Image**: Consider a simple 2D image matrix:

$$\begin{bmatrix} 100 & 100 & 100 & 50 & 50 \\ 100 & 100 & 50 & 50 & 50 \\ 75 & 75 & 75 & 75 & 75 \\ 0 & 0 & 255 & 255 & 255 \end{bmatrix}$$

RLC Encoding:

- Row 1: (100,3), (50,2) (100, 3), (50, 2) (100,3), (50,2)
- Row 2: (100,2), (50,3) (100, 2), (50, 3) (100,2), (50,3)
- Row 3: (75,5) (75, 5) (75,5)
- Row 4: (0,2), (255,3) (0, 2), (255, 3) (0,2), (255,3)

6. **Efficiency Analysis**:

- **Compression Gain**: For images with uniform colours, RLC greatly reduces the number of stored elements. A line with 20 identical pixels $X = \{255, \dots, 255\}$ becomes (255,20) (255, 20) (255,20), significantly decreasing data size.
-

7. Advantages and Disadvantages:

Advantages	**Disadvantages**
Simple and quick implementation	Ineffective for images with variable data
Excellent for images with uniform areas	Limited compression for complex images
Lossless—preserves original data	Can increase data size in non-uniform areas

8. Example: Consider a 1D line:

$$\{\mathbf{128}, \mathbf{128}, \mathbf{128}, \mathbf{255}, \mathbf{255}, \mathbf{255}, \mathbf{255}, \mathbf{64}, \mathbf{64}\}$$

Encoded output:

(128,3), (255,4), (64,2) (128, 3), (255, 4), (64, 2) (128,3), (255,4), (64,2)

9.Decoding Process: To decode, each pair $(\boldsymbol{xi}, \boldsymbol{ni})$ is expanded to $\boldsymbol{xi}$ repeated $\boldsymbol{ni}$ times:

$$\{(\mathbf{128}, \mathbf{3}) \rightarrow \mathbf{128}, \mathbf{128}, \mathbf{128}\}$$

10. Use in Image Types:

- **Optimal for**: Simple graphics, scanned documents, cartoon images.
- **Less Effective for**: Photographic images or highly detailed scenes.

Table:

Aspect	**Details**
Compression Type	Lossless
Best Application	Uniform, large-colour-area images
Encoding Output Format	$(\boldsymbol{Value}, \boldsymbol{Count})$
Example Efficiency	Reduces data size in homogeneous areas
