IMAGE COMPRESSION

Introduction:

Image compression is a fundamental technique used in digital image processing to reduce the size of an image file while preserving its visual quality to a reasonable extent. The primary goal of image compression is to minimize the amount of data required to represent an image, thereby saving storage space and reducing transmission time over networks. It plays a crucial role in various applications such as digital photography, video streaming, medical imaging, satellite imagery, and more.

Types of Image Compression

Image compression techniques can be broadly categorized into two main types:

1. **Lossless Compression:** Lossless compression algorithms preserve all original image data when compressing and decompressing images. They ensure exact reconstruction of the original image without any loss of information. Lossless compression is suitable for scenarios where image fidelity is critical, such as medical imaging or text documents.
2. **Lossy Compression:** Lossy compression algorithms achieve higher compression ratios by selectively discarding less important image information during compression. While this results in some loss of image quality, the human visual system may tolerate such losses to a certain extent without significant perceptual degradation. Lossy compression is widely used in applications like digital photography, web images, and video compression.

Methods:

1) JPEG compression

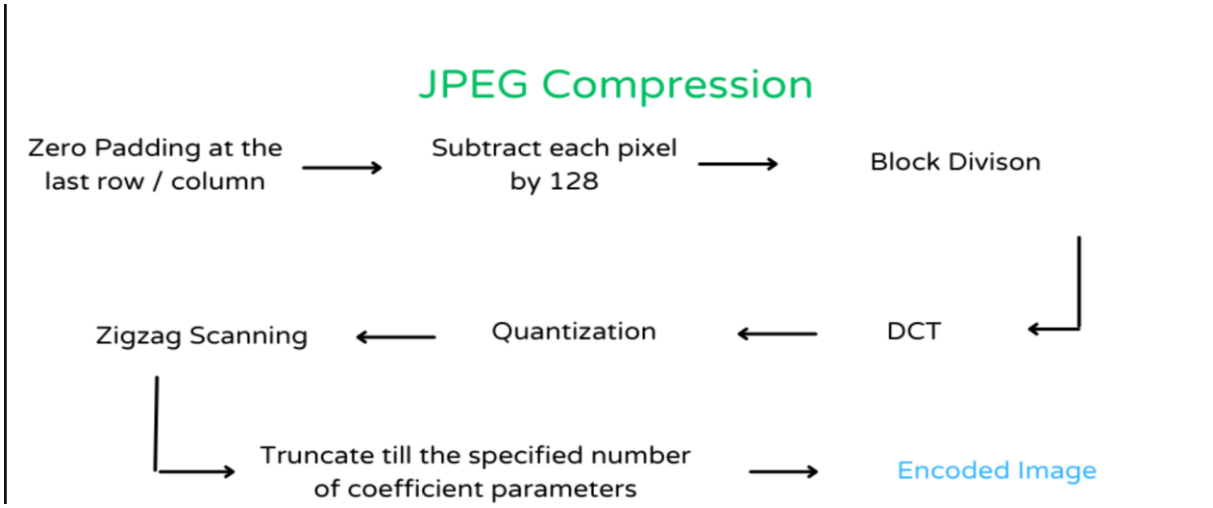
2) Image compression using BTC

3) Colour Quantization

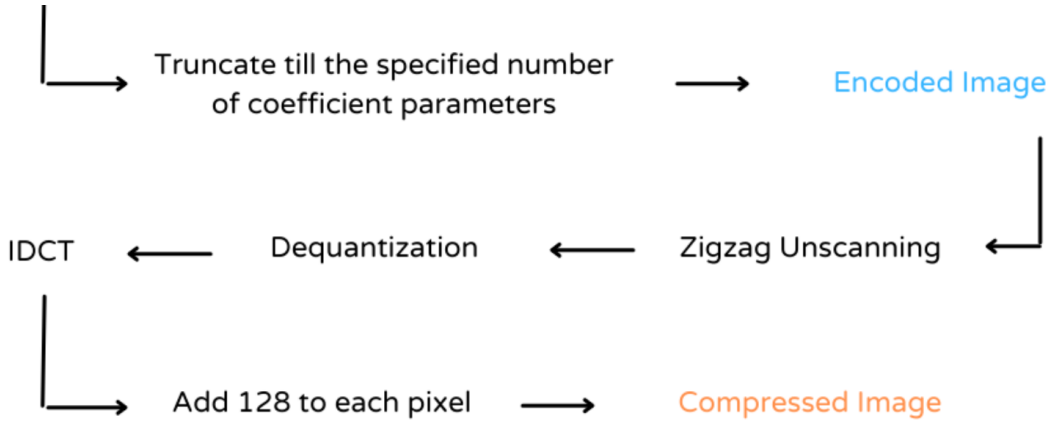
JPEG compression

JPEG (Joint Photographic Experts Group) is a widely used image compression standard designed for efficiently compressing digital images while maintaining acceptable visual quality. It is particularly suited for compressing photographic images and continuous-tone images, making it one of the most common formats for storing and transmitting images on the web and in digital media applications.

The JPEG encoder in this project takes a color/grayscale image as input and applies the following steps:



The JPEG decoder reverses the above steps to reconstruct the original image from the compressed data.



1. **Quantization Matrix**: The JPEG compression starts with defining a quantization matrix. This matrix is used during quantization to reduce the precision of the frequency coefficients obtained after applying the Discrete Cosine Transform (DCT) to image blocks.

16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

Quantization table

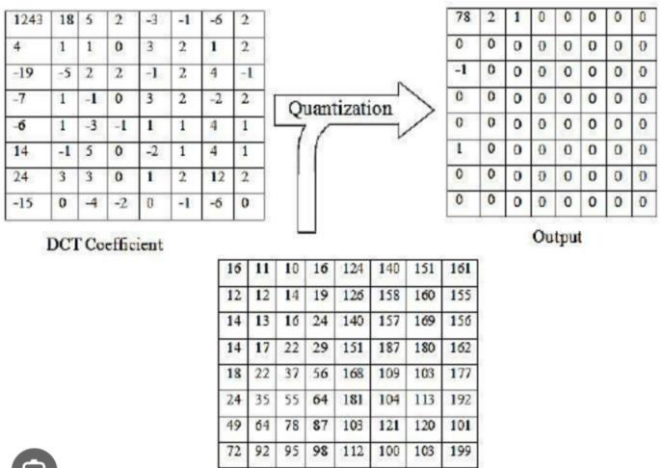
2. **Block-based Processing**: JPEG compression divides the image into blocks, typically 8x8 pixels. Each block undergoes a series of operations, including DCT, quantization, and zigzag scanning.

3. **Discrete Cosine Transform (DCT)**: The **cv.dct()** function is applied to each block to transform the spatial domain data into the frequency domain. This transformation separates high-frequency and low-frequency components.

4. **Quantization**: The transformed coefficients are then quantized by dividing them with the quantization matrix. This step introduces loss by reducing the precision of coefficients based on their perceptual importance.

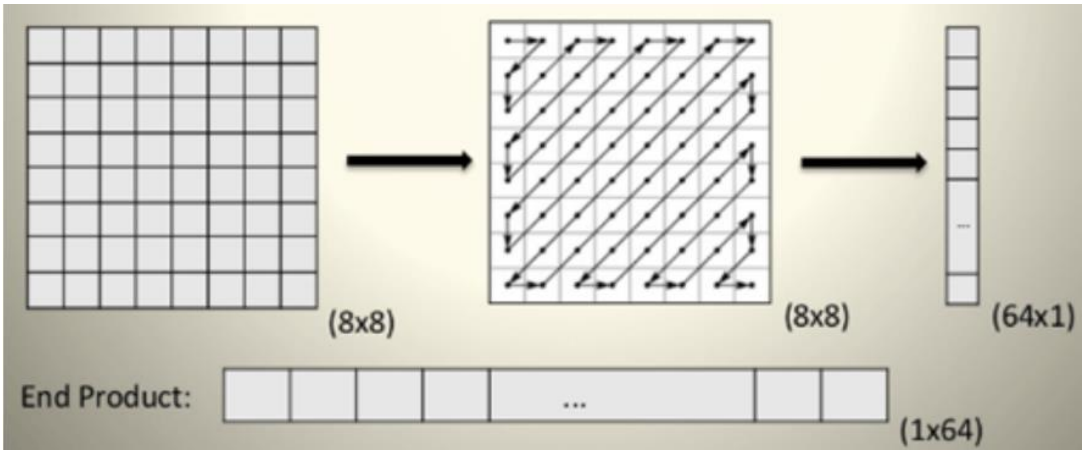
$$X'(u,v) = Round\left(\frac{X(u,v)}{Q(u,v)}\right)$$

X(u,v): original DCT coefficient
X'(u,v): DCT coefficient after quantization
Q(u,v): quantization value



16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

5. **Zigzag Scanning**: After quantization, the coefficients are rearranged using zigzag scanning (**zigzag_scan()** function). This scanning method organizes the coefficients in a linear sequence, which facilitates run-length encoding and entropy coding.

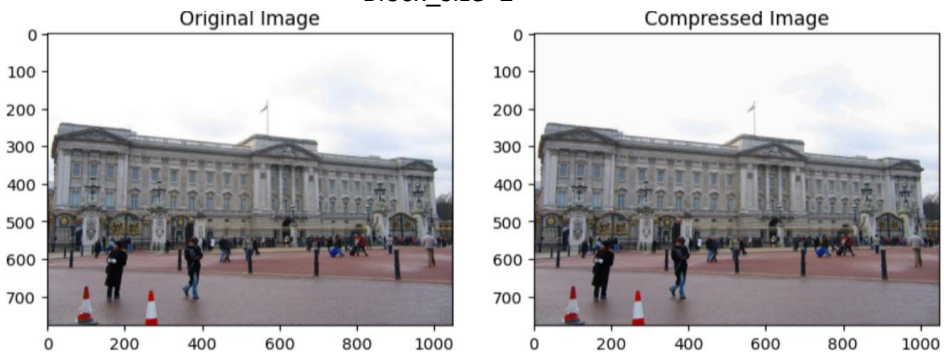


6. **JPEG Encoding**: The **grayscale_jpeg_encoder()** function performs the entire encoding process for grayscale images. It includes DCT, quantization, zigzag scanning, and retaining a specified number of coefficients (**num_coefficients**).
7. **JPEG Decoding**: The **grayscale_jpeg_decoder()** function reverses the encoding process. It involves unscanning the zigzag pattern, dequantizing the coefficients, and applying the Inverse DCT (**cv.idct()**) to reconstruct the compressed image.
8. **PSNR Calculation**: The **calculate_psnr()** function computes the Peak Signal-to-Noise Ratio (PSNR) between the original and compressed images. PSNR is a metric used to assess the quality of the reconstructed image compared to the original.
9. **Compression Ratio**: The compression ratio is calculated as the ratio of the number of bits required to represent the original image to the number of bits after compression. It indicates how much the image has been compressed.

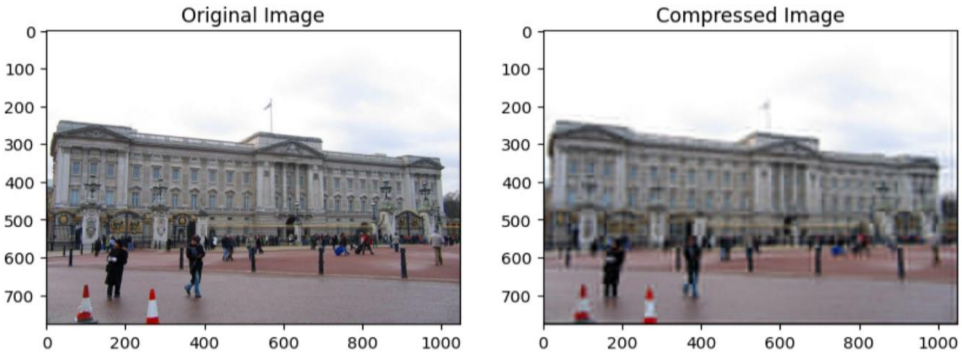
Observations:

1. **PSNR vs. Compression Ratio Trade-off**:
    - Increasing the number of coefficients generally leads to higher PSNR values, indicating better image quality. However, this comes at the cost of a lower compression ratio.
    - Lower PSNR values may correspond to visible artifacts or loss of quality in the compressed image, especially as the compression ratio increases.
2. **Effect of Block Size**:
    - The block size (**block_size**) used for processing impacts both the compression efficiency and the level of detail preserved in the compressed image. Smaller block sizes can preserve finer details but may lead to larger file sizes due to less efficient compression.

Block_size=2



Block_size=32



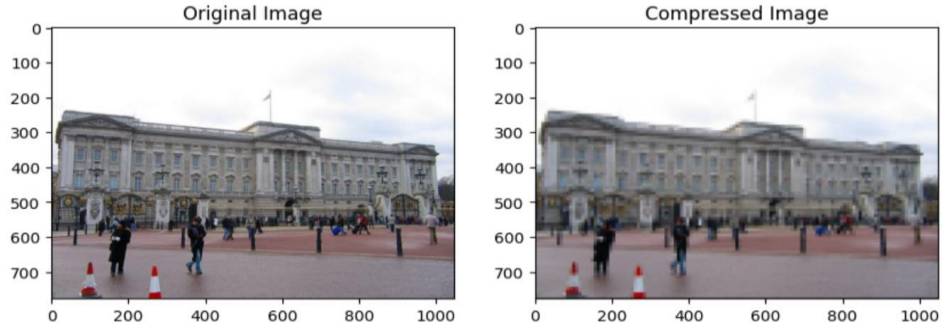
3. **Color vs. Grayscale Compression**:

    - The code supports both color and grayscale image compression. Color images typically require processing each color channel separately, affecting both compression performance and visual fidelity.

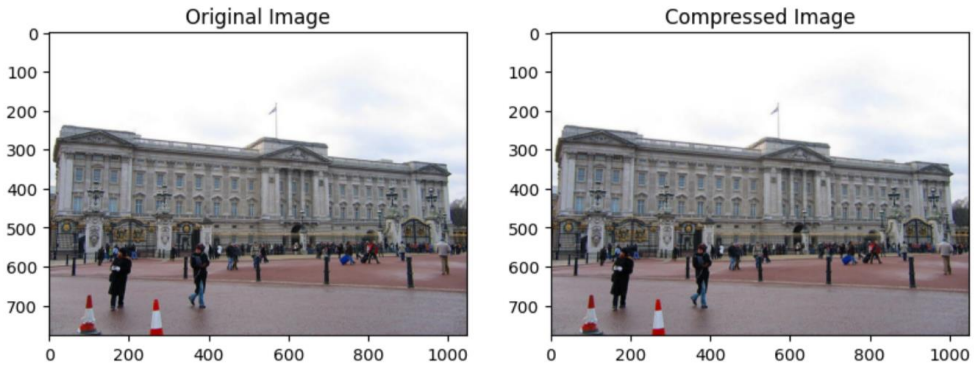
4. **Effect of Number of coefficients**:

    - The **number of coefficients** used for processing impacts both the compression efficiency and the level of detail preserved in the compressed image. Smaller **number of coefficients** gives smaller psnr.

Coefficients=1



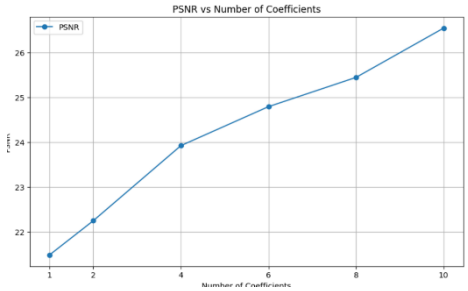
Coefficients=32

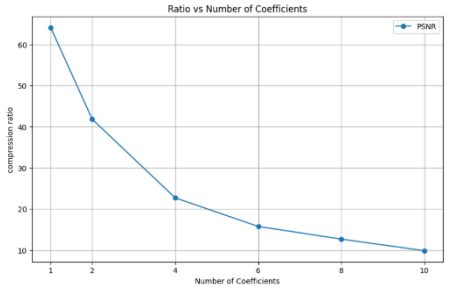


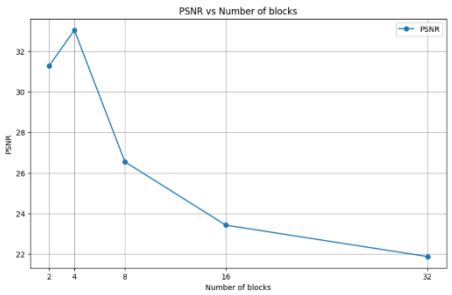
Results:

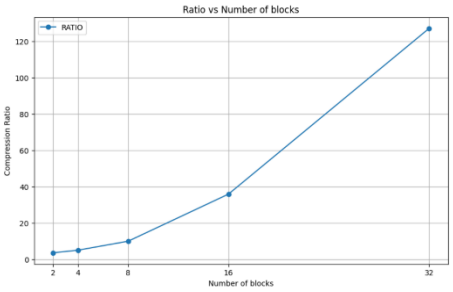
PSNR and Compression Ratio Variation:

- The code calculates the PSNR (Peak Signal-to-Noise Ratio) and compression ratio for different configurations of the JPEG compression process, including varying the number of coefficients (**num_coefficients**).
- The PSNR reflects the quality of the compressed image compared to the original, while the compression ratio indicates the level of data reduction achieved.







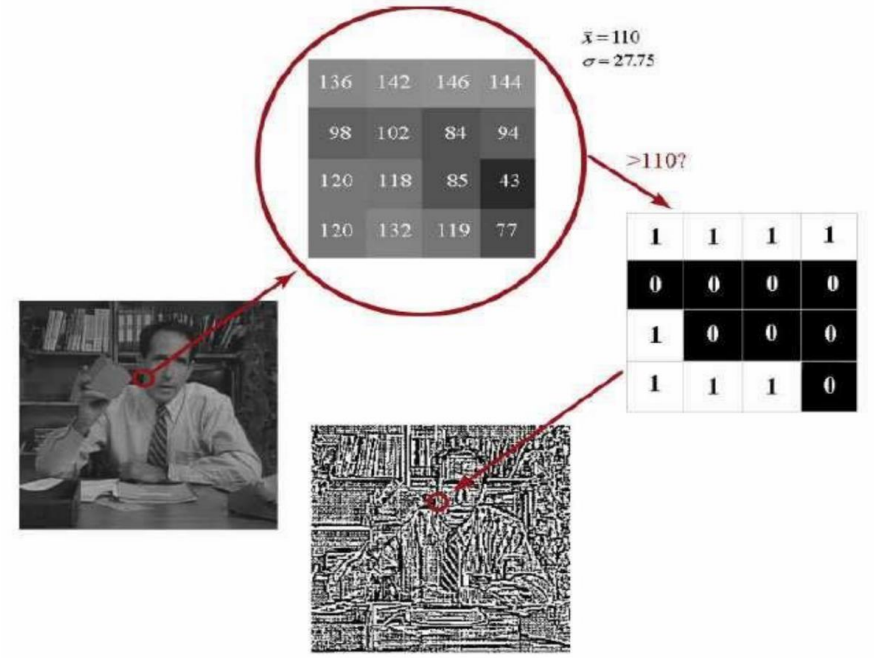


Size of original image: 334.66 KB

Size of compressed image: 56.86 KB

Image compression using BTC:

Block Truncation Coding (BTC) is a simple yet effective technique for image compression, particularly suited for binary and grayscale images. Unlike some other compression methods that focus on transforming the image data or using complex algorithms, BTC operates by dividing the image into blocks and quantizing pixel values within each block. This approach makes BTC computationally efficient and suitable for real-time applications.



The Block Truncation Coding (BTC) algorithm is an image compression technique that operates on blocks of pixels within an image. Initially, the algorithm divides the input image into non-overlapping blocks of a specified size (e.g., 8x8 pixels). For each block, BTC computes statistical parameters such as the mean and variance, which are used to determine the lower and upper bounds for quantization. Pixels above the mean are set to an upper value (**b**), while pixels below or equal to the mean are set to a lower value (**a**).

$$a = m_1 - \sigma\sqrt{\frac{q}{k-q}}$$
$$b = m_1 + \sigma\sqrt{\frac{k-q}{q}}.$$

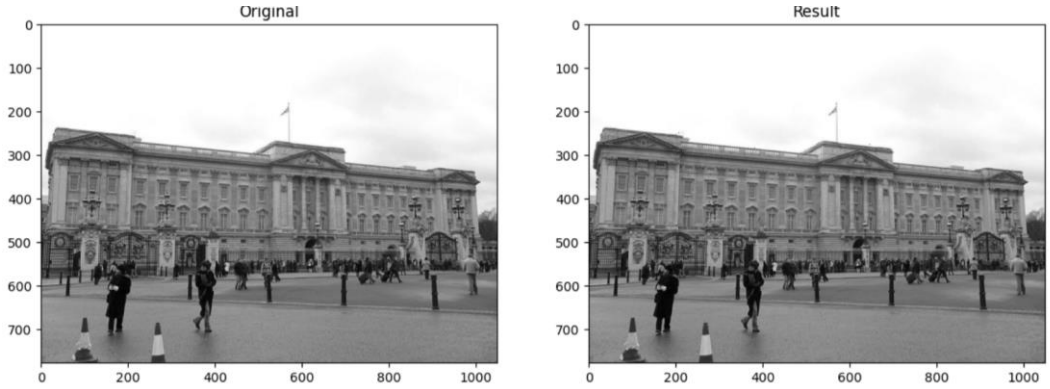
This process creates a binary bitmap indicating which pixels were quantized to **b**. During decompression, the quantized image and bitmap are used to reconstruct the compressed image. BTC's simplicity and efficiency make it suitable for applications where moderate compression ratios are acceptable, especially in scenarios with limited computational resources or where simplicity is favored over high compression efficiency.

Observations:

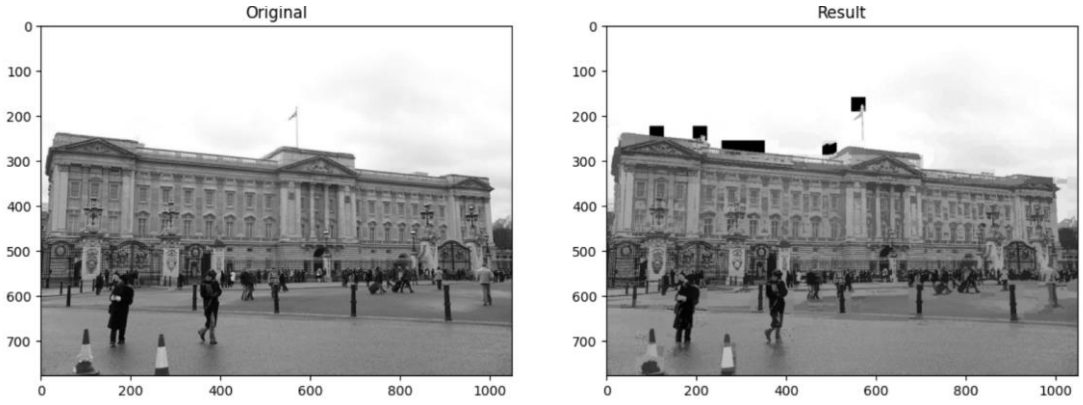
- The BTC algorithm operates by dividing the input image into blocks and then computing statistical parameters for each block to determine quantization bounds.

- The computed lower and upper bounds (**a** and **b**) are used to quantize pixels within each block, resulting in a binary bitmap indicating which pixels were quantized to the upper bound **b**.
- The compression performance and visual quality of the compressed image are influenced by the block size chosen for BTC. Smaller block sizes may preserve more detail but can lead to larger file sizes and less compression. Larger block sizes can achieve higher compression ratios but may lose fine details.

Block_size=2



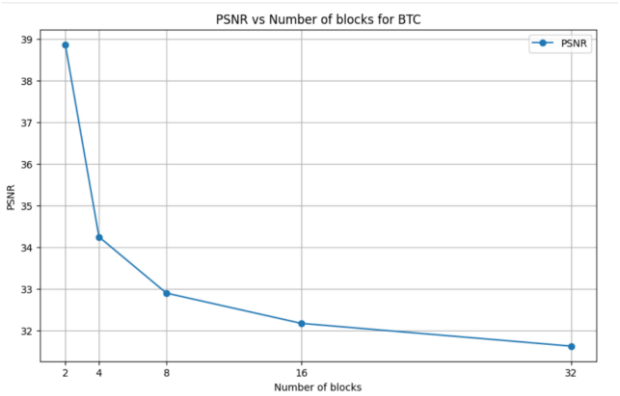
Block_size=32



Results:

- After applying BTC compression to the input image using an 8x8 block size, the resulting binary bitmap (**BTC_Bitmap.jpg**) and the reconstructed compressed image (**BTC_Result.jpg**) are obtained.
- The binary bitmap visually represents which pixels were quantized to the upper value **b** during the compression process.
- The reconstructed compressed image shows the visual quality of the compressed result compared to the original image.

- The PSNR (Peak Signal-to-Noise Ratio) metric can be used to quantitatively evaluate the quality of the reconstructed image compared to the original, with higher PSNR values indicating better preservation of image fidelity during compression.



NOTE: Size of original image: 477.09 KB
Size of generated bitmap: 99.71 KB
Size of compressed image: 275.49 KB

Colour quantization

Introduction:

Color quantization is a fundamental technique in digital image processing that involves reducing the number of distinct colors used in an image while preserving its visual quality as much as possible. olor quantization works by grouping similar colors together and representing them with a smaller set of colors from a predefined color palette or color space. This reduction in color complexity can lead to smaller file sizes and more efficient image representation without significantly compromising the perceptual quality of the image to the human eye.

The color quantization algorithm implemented in the provided code utilizes the K-means clustering technique. It begins by initializing variables and loading an image into a NumPy array. The algorithm then randomly selects initial cluster centers (representing colors) within the image. It iteratively assigns each pixel in the image to the closest cluster center based on RGB distance and updates the cluster centers by computing the mean RGB values of their assigned pixels. This assignment-update process repeats until convergence or a specified number of iterations. After convergence, each pixel's RGB values are replaced with those of its assigned cluster center, effectively reducing the image's color palette to the specified number of clusters (K)

Observations:

Color Reduction: The algorithm effectively reduces the image's color palette to a specified value (K=32), simplifying the colors and aiding in image compression.

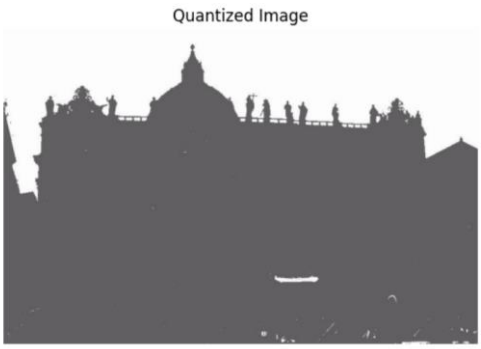
Cluster Centers: Random initial cluster centers are chosen within the image, representing dominant colors. These centers are iteratively adjusted to better match pixel clusters.

Convergence: The algorithm iterates until convergence or a set number of iterations, stabilizing cluster centers and pixel assignments.

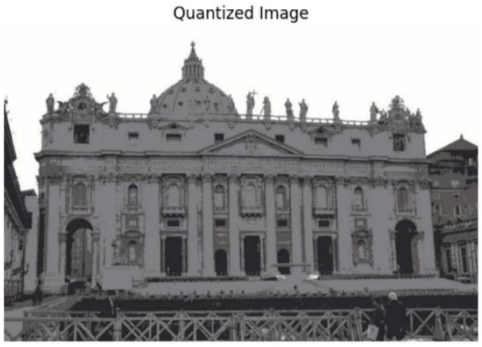
Quantized Image: The output image exhibits reduced color diversity compared to the original, showcasing the impact of color quantization.

Quality vs. Compression: There's a trade-off between image quality and file size based on the chosen value of K. Higher K values offer better color representation but may increase file size.

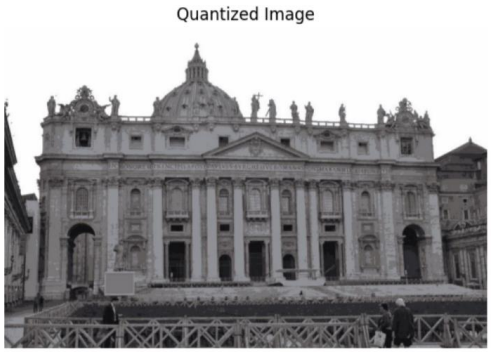
The images obtained for different number of clusters are:



K=2



K=4

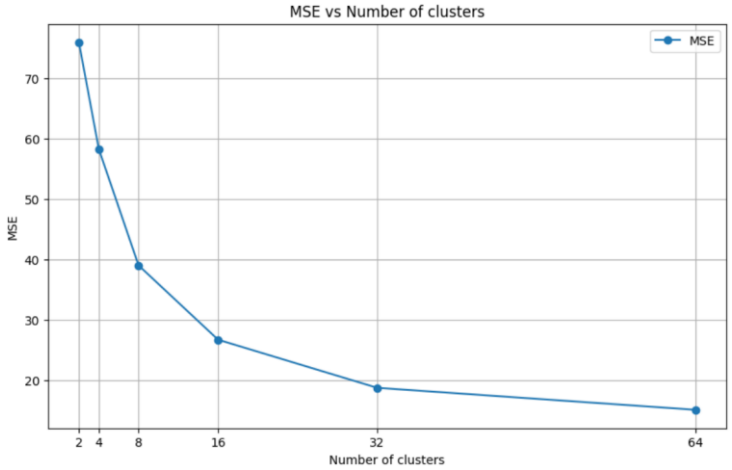


K=8

K=64

Results:

1. **MSE vs Number of Clusters**:
    - The plot shows how the Mean Squared Error (MSE) changes with the number of clusters used in the color quantization process.
    - As the number of clusters increases, the MSE tends to decrease. This is expected because more clusters allow for a closer representation of the original colors, resulting in lower quantization error.



2. **Compression Ratio vs Number of Clusters**:

   - The plot illustrates the relationship between the Compression Ratio and the number of clusters used in color quantization.
   - Typically, as the number of clusters increases, the compression ratio decreases. This is because more clusters imply more information to store color details, leading to a larger file size and lower compression ratio.
   - However, there might be nuances based on how the compression algorithm handles color information and the specific image dataset used.