IMAGE MATCHING + IMAGE COMPRESSION

Group Members:

1)Debesh Maheshwari (B21AI012)
2)Dev Agarwal (B21CS023)
3)Anurag Kumar Bharti (B21CS012)

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

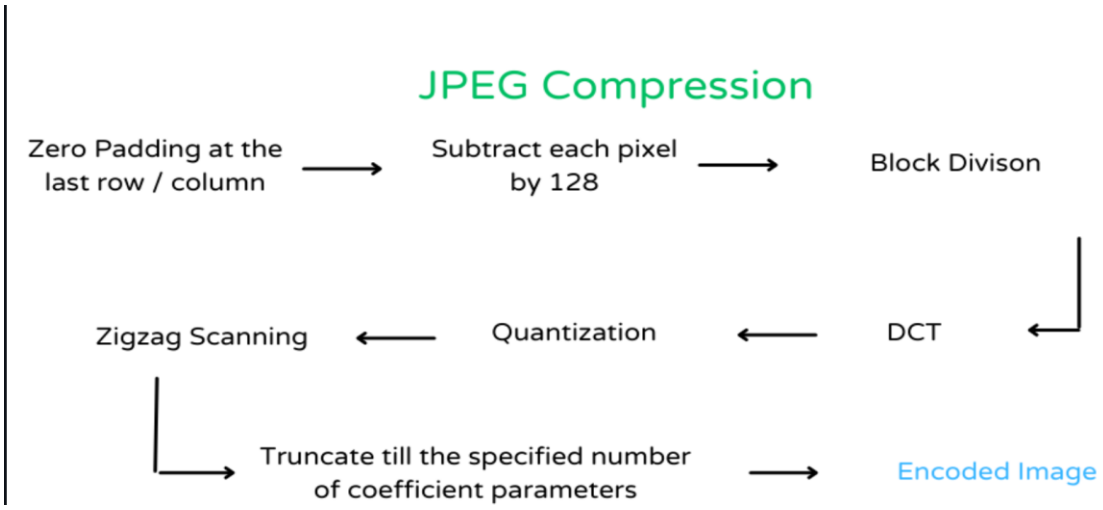
2) Image compression using BTC

3) Colour Quantization

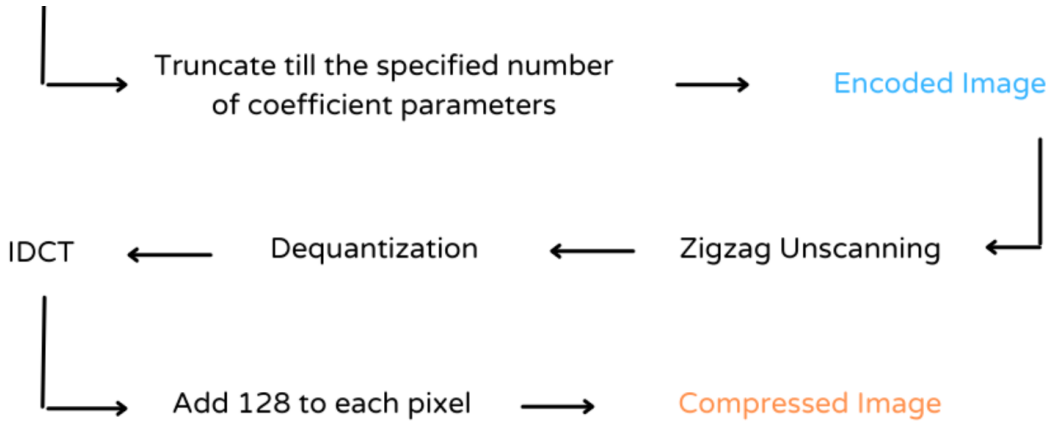
JPEG compression

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.

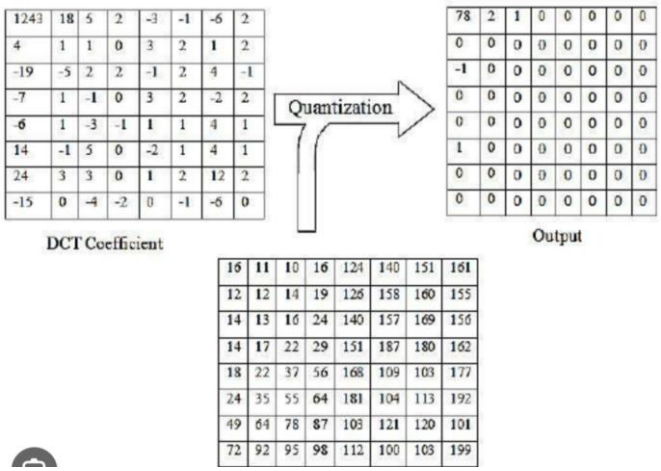
16	11	10	16	124	140	151	161
12	12	14	19	126	158	160	155
14	13	16	24	140	157	169	156
14	17	22	29	151	187	180	162
18	22	37	56	168	109	103	177
24	35	55	64	181	104	113	192
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	199

Quantization table

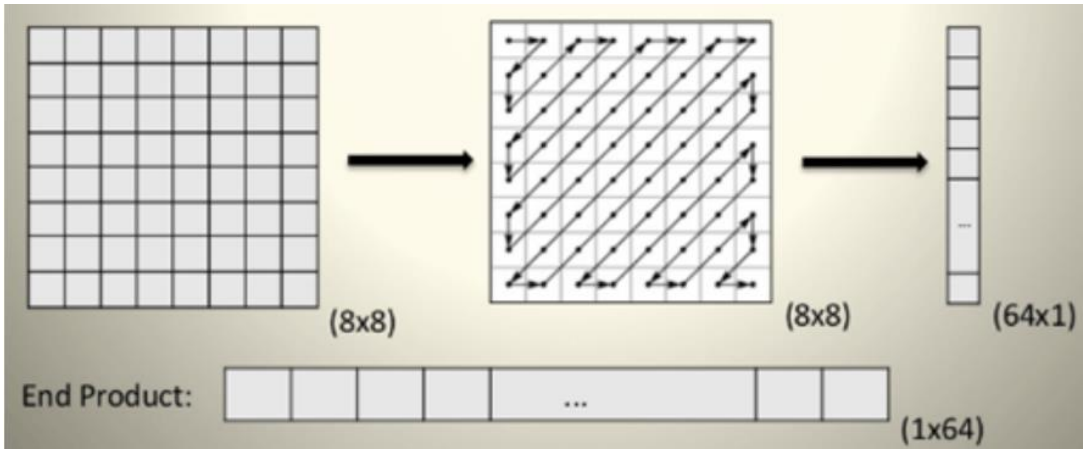
2. **Block-based Processing**: JPEG compression divides the image into blocks, typically 8x8 pixels. Each block undergoes a series of operations, including DCT, quantization, and zigzag scanning.
3. **Discrete Cosine Transform (DCT)**: The **cv.dct()** function is applied to each block to transform the 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



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**).

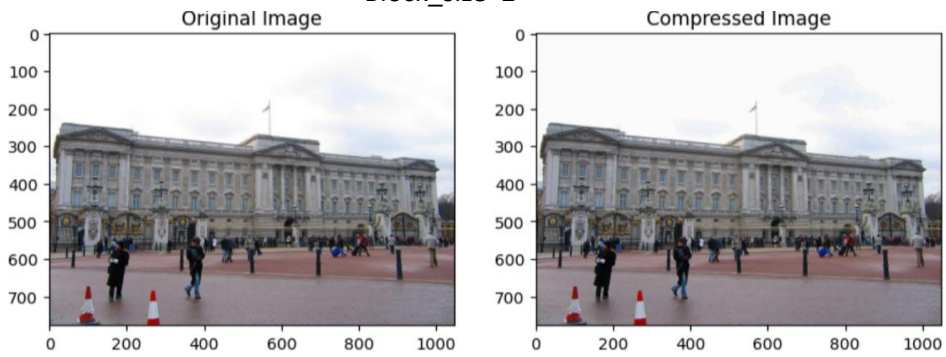
7. **JPEG Decoding**: The **grayscale_jpeg_decoder()** function reverses the encoding process. It involves unscanning the zigzag pattern, dequantizing the coefficients, and applying the Inverse DCT (**cv.idct()**) to reconstruct the compressed image.

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.

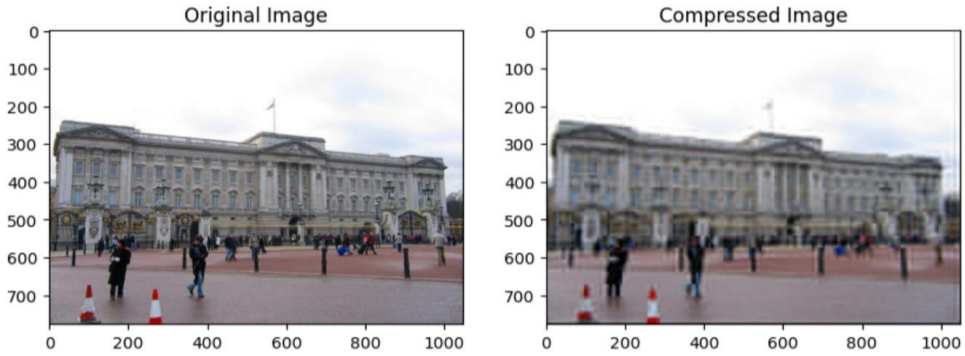
Observations:

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



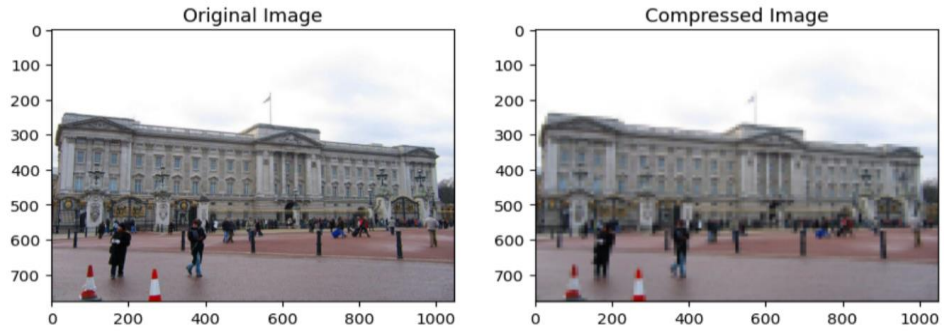
3. **Color vs. Grayscale Compression**:

    - 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.

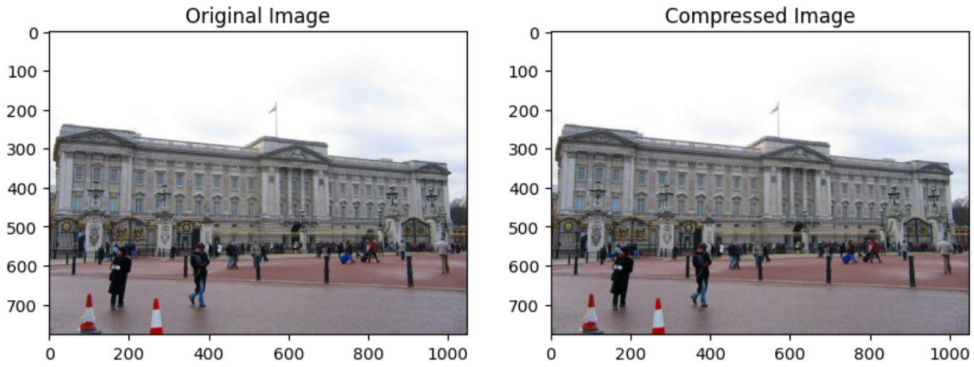
4. **Effect of Number of coefficients**:

    - 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



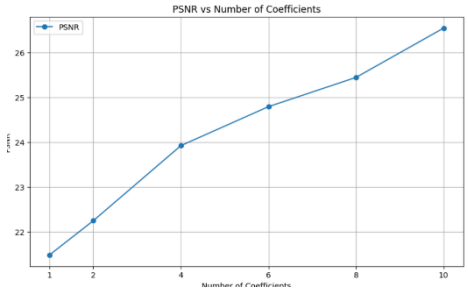
Coefficients=32

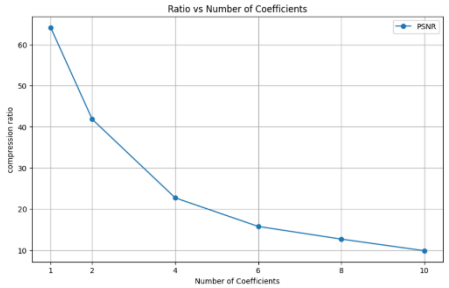


Results:

PSNR and Compression Ratio Variation:

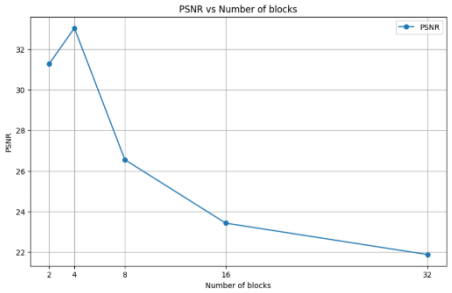
- The code calculates the PSNR (Peak Signal-to-Noise Ratio) and compression ratio for different configurations of the JPEG compression process, including varying the number of coefficients (**num_coefficients**).
- The PSNR reflects the quality of the compressed image compared to the original, while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





Size of original image: 334.66 KB

compre
age: 56.



Size of
ssed im
86 KB

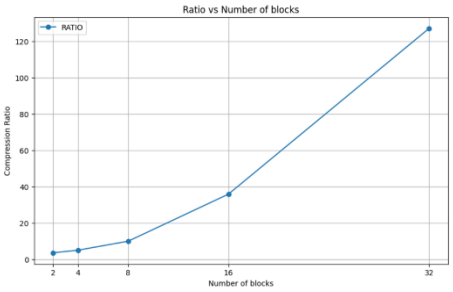
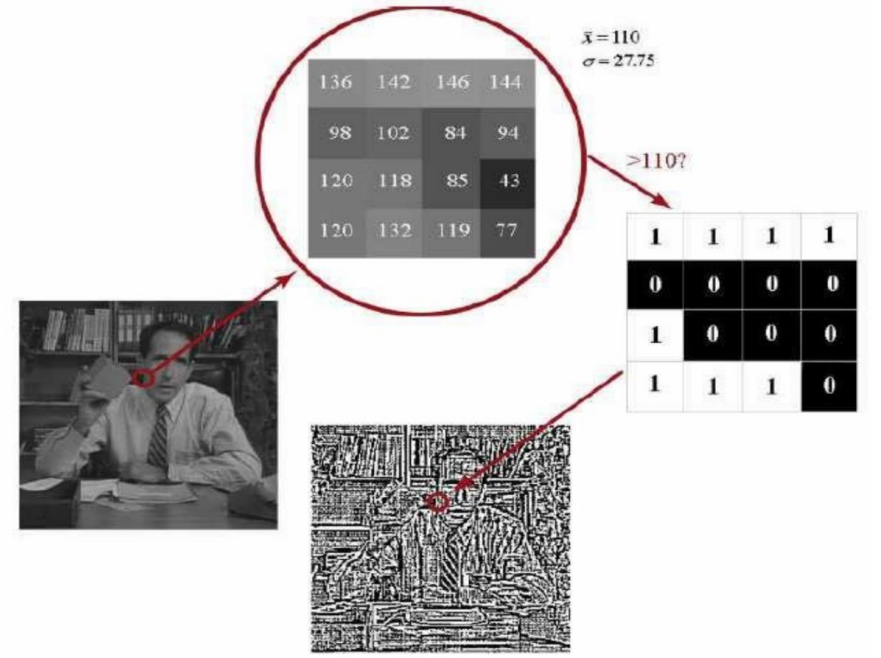


Image compression using BTC:

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**).

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

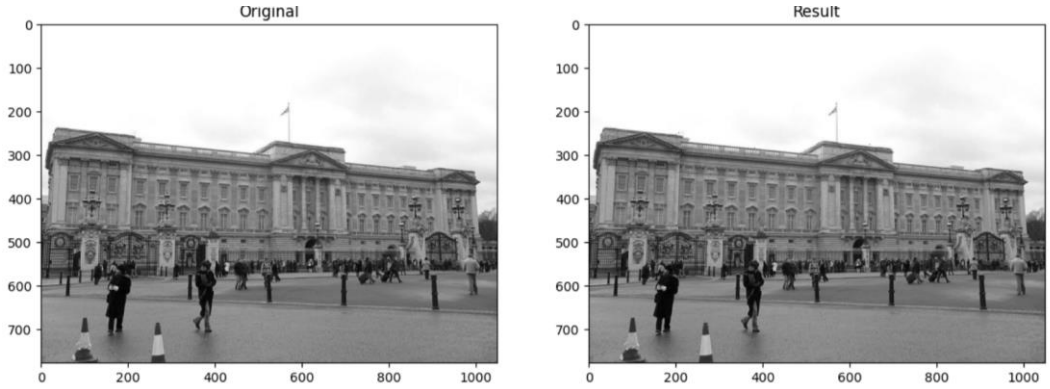
This process creates a binary bitmap indicating which pixels were quantized to **b**. During decompression, the quantized image and bitmap are used to reconstruct the compressed image. BTC's simplicity and efficiency make it suitable for applications where moderate compression ratios are acceptable,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

Observations:

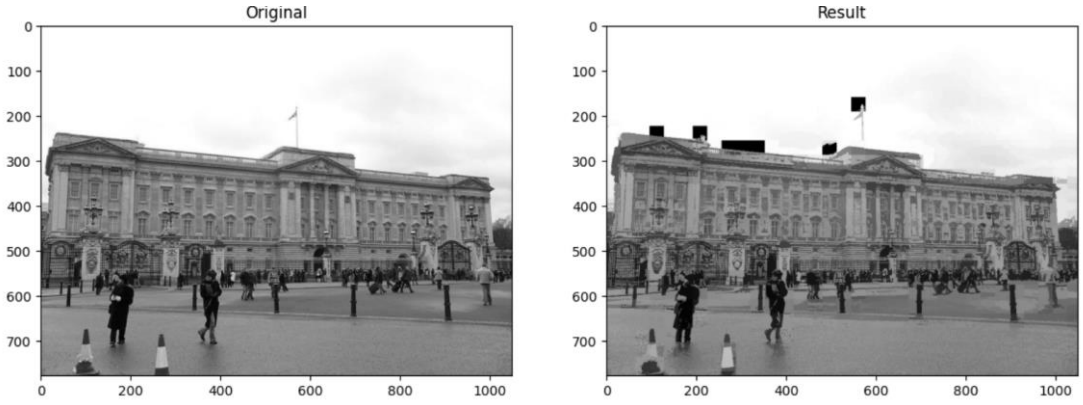
- The BTC algorithm operates by dividing the input image into blocks and then computing statistical parameters for each block to determine quantization bounds.

- 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



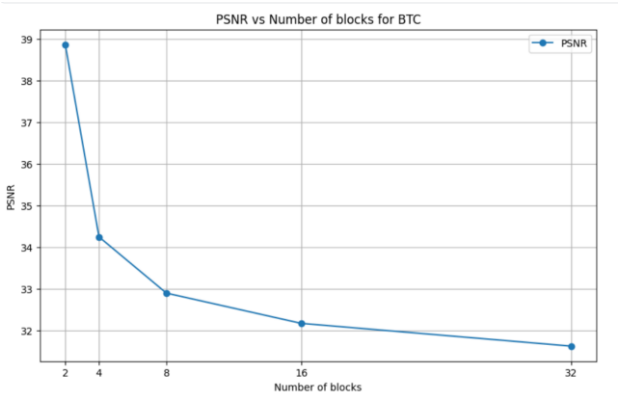
Block_size=32



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.

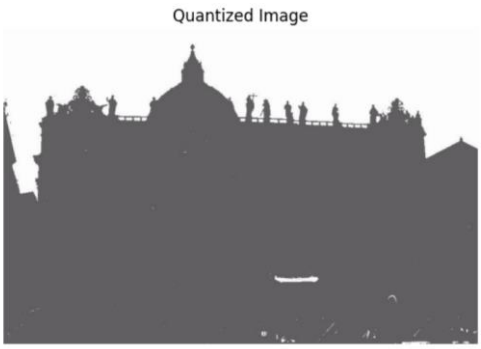
Cluster Centers: Random initial cluster centers are chosen within the image, representing dominant colors. These centers are iteratively adjusted to better match pixel clusters.

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

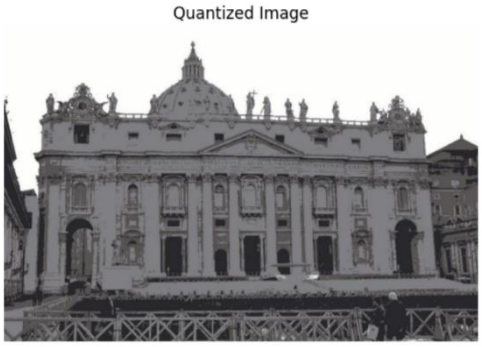
Quantized Image: The output image exhibits reduced color diversity compared to the original, showcasing the impact of color quantization.

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.

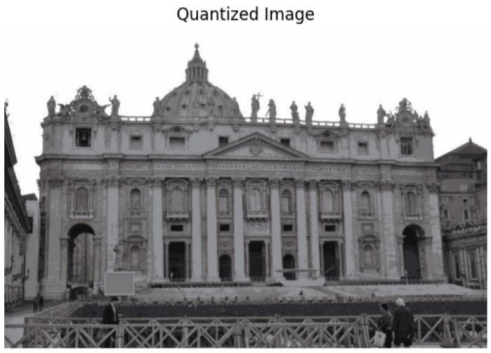
The images obtained for different number of clusters are:



K=2



K=4

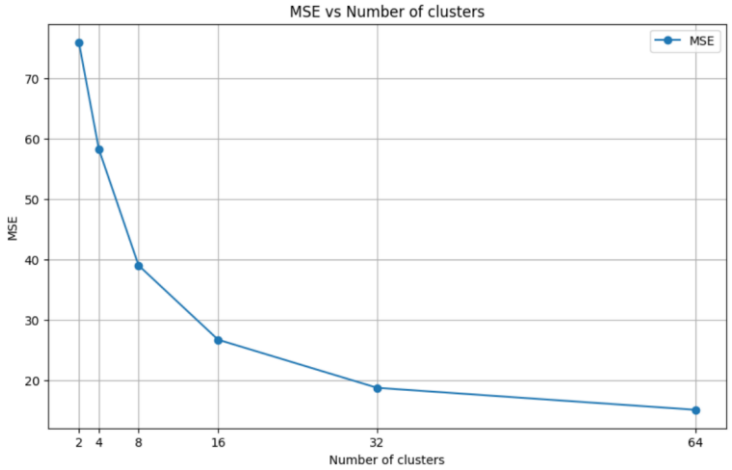


K=8

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.

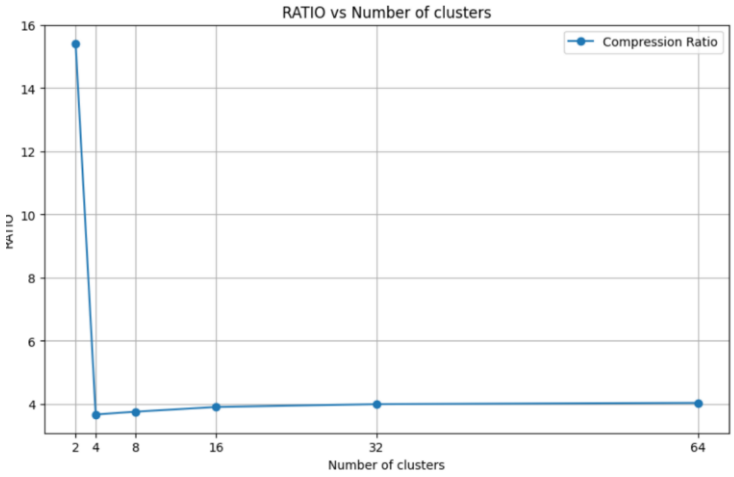


IMAGE MATCHING

Introduction:

Image matching is a fundamental task in computer vision that involves finding correspondences between different images or parts of the same image. It plays a crucial role in various applications such as object recognition, image retrieval, 3D reconstruction, and augmented reality. The goal of image matching is to establish relationships between images based on their visual content, enabling machines to understand and interpret visual information.

Types of Image Matching:

1)Feature-Based Matching:

- **Keypoint Detection:** Feature-based matching algorithms detect distinctive points or keypoints in images, such as corners or blobs, that are invariant to transformations like rotation, scaling, and illumination changes.
- **Descriptor Extraction:** Once keypoints are detected, descriptors are computed to represent the local appearance around each keypoint. Common descriptors include SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), and ORB (Oriented FAST and Rotated BRIEF).
- **Matching:** Matching involves finding correspondences between keypoints in different images based on their descriptors. This is typically done using techniques like nearest neighbor search and geometric verification.

2)Geometric Matching:

- **RANSAC (Random Sample Consensus):** Feature-based matching algorithms detect distinctive points or keypoints in images, such as corners or blobs, that are invariant to transformations like rotation, scaling, and illumination changes.

3)Deep Learning-Based Matching:

- **Siamese Networks:** Siamese networks are a type of neural network architecture used for learning similarity between two inputs. In image matching, Siamese networks can learn to compare pairs of images and determine their similarity or dissimilarity.
- **Convolutional Neural Network(CNNs):** NNs have shown remarkable success in various computer vision tasks, including image matching. They can learn hierarchical representations of images, capturing both low-level and high-level features for matching.

4)Graph-Based Matching:

- **Graph-Based Matching:** Graph-based matching approaches represent images as graphs, with nodes representing keypoints or image patches and edges representing relationships between them. Matching involves finding correspondences between nodes in different graphs.

Methods to find Descriptors:

1. **Sift**
   The SIFT (Scale-Invariant Feature Transform) algorithm is a computer vision technique used for feature detection and description. It detects distinctive key points or features in an image that are robust to changes in scale, rotation, and affine transformations. SIFT(scale invariant feature transform) works by identifying key points based on their local intensity extrema and computing descriptors that capture the local image information around those key points.

2. **N-Sift**
   NG-SIFT (Normalized -gardient SIFT) is a variant of the Scale-Invariant Feature Transform (SIFT) algorithm used for extracting keypoints and descriptors from images. Unlike traditional SIFT, which relies on gradient information, NG-SIFT utilizes non-gravitational gradients, computed from magnitude and angle information obtained through methods like Sobel operators. This approach allows NG-SIFT to be robust to image rotations and translations, making it suitable for various computer vision tasks such as object recognition, image stitching, and feature matching.

3. **PCA-Sift**
   PCA-SIFT is an adaptation of the SIFT algorithm, aiming to overcome computational challenges while maintaining feature quality. It utilizes Principal Component Analysis (PCA) to reduce feature dimensionality without compromising discriminative power. This enhances computational efficiency without sacrificing accuracy, making it suitable for large-scale and real-time applications in computer vision and image processing.

4. **A-Sift**
   Affine SIFT (A-SIFT) extends the traditional SIFT algorithm to handle affine transformations like shearing, scaling, and skewing, in addition to rotation and translation invariance. These transformations commonly occur in real-world scenarios, especially in object recognition and scene understanding tasks. By detecting affine-covariant keypoints and computing affine-invariant descriptors, A-SIFT enables more accurate feature matching across a wider range of geometric transformations. This enhancement is crucial for robust feature-based image processing, particularly in scenarios where objects undergo non-rigid deformations or perspective changes.

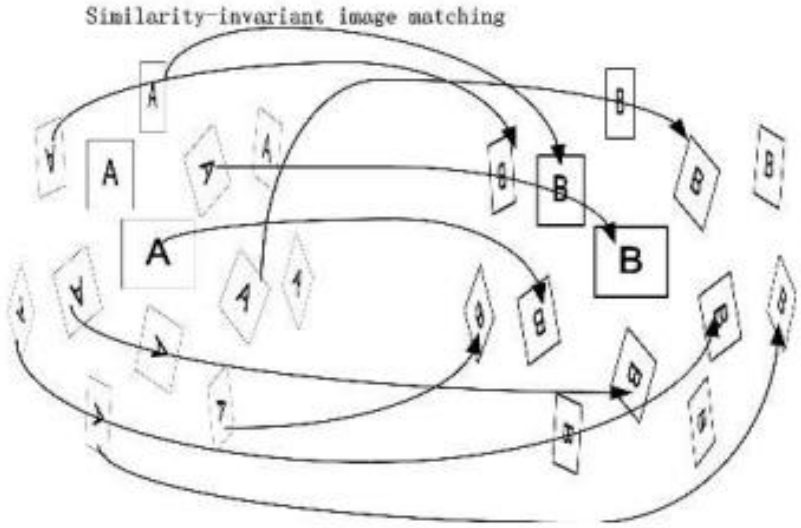


Figure 5: ASIFT algorithm overview.

5. **Mops**
   MOPS (Multi-Scale Oriented Patches) is a feature extraction technique used in computer vision for describing local image patches. It divides image patches into sub-regions and computes histograms of gradient orientations within each sub-region. These histograms capture both the spatial information and the orientation of gradients, allowing MOPS to represent each patch effectively. By concatenating histograms from multiple scales, MOPS creates feature descriptors that are robust to scale variations and preserve both fine and coarse details in the image. This technique is particularly useful for feature-based matching tasks, where it provides efficient and accurate representation of local image structures.

Methods to find good matches:

1. **Flann**
   FLANN is a library that provides efficient implementations of approximate nearest neighbor search algorithms. In image matching, FLANN is often used to speed up the matching process of feature descriptors by quickly finding approximate nearest neighbors, especially in high-dimensional spaces like those of SIFT or SURF descriptors.
2. **Ransac**
   RANSAC is an iterative algorithm used to estimate parameters of a mathematical model from a set of observed data points that may contain outliers. In image matching, RANSAC is commonly employed to estimate geometric transformation parameters (e.g., homography) between two images by iteratively fitting models to subsets of correspondences and selecting the model with the best consensus set.
3. **Gale Shapely**
   Also known as the stable marriage algorithm, Gale-Shapley is used to solve the problem of matching two sets of elements based on their preferences. In image matching, the Gale-Shapley algorithm can be adapted to establish correspondences between keypoints detected

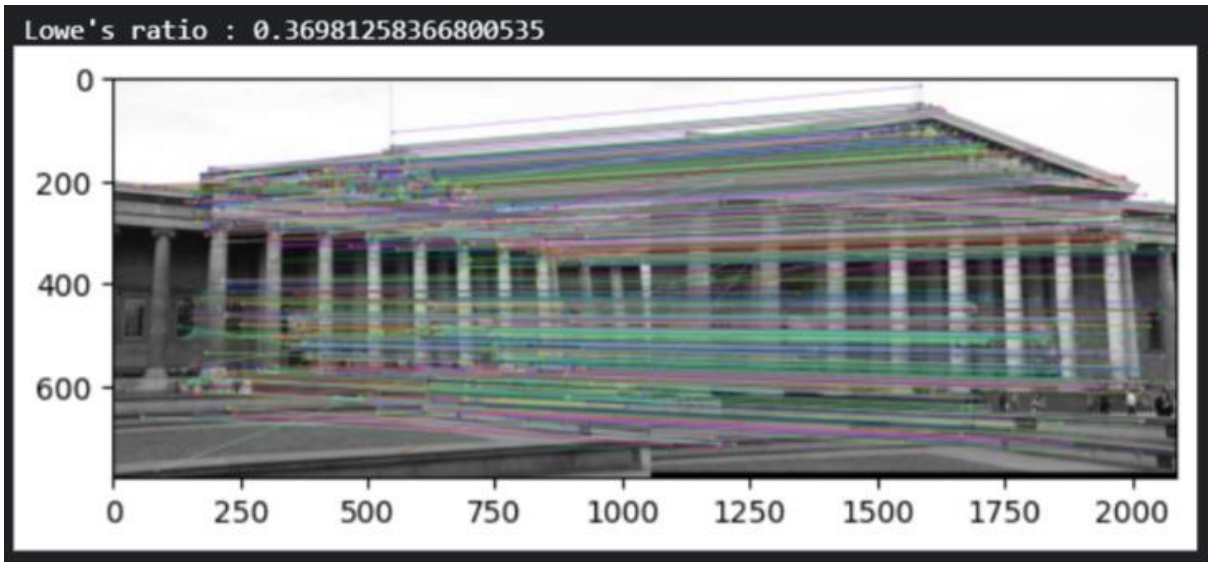
in different images. Each keypoint expresses preferences for keypoints in the opposite image based on similarity measures, and matches are iteratively formed to ensure stability and optimality.

4. **PSR-EM**
   PSR-EM (Patch Similarity Ratio and Expectation Maximization using Gaussian Mixture Models) combines patch similarity ratio with the Expectation Maximization (EM) algorithm and Gaussian Mixture Models (GMMs) for improved image matching. Initially, GMMs are fitted to the feature descriptors of both images, and the means are extracted. These means serve as the initialization for fitting the descriptors again, followed by computing the log likelihood of descriptors in one image using the GMM fitted in the other image. Matches are determined based on the closest log likelihood values, establishing correspondence between features in the images. This iterative process refines matching, enhancing accuracy and robustness in image feature correspondence.
5. **BF matcher**
   The Brute-Force Matcher is a straightforward method for feature matching that computes the distances between all pairs of descriptors and selects the nearest neighbors. In image matching, the BF Matcher compares each feature descriptor in one image with all descriptors in another image, making it computationally expensive but exhaustive in finding matches.

IMPLEMENTATION OF IMAGE MATCHING ALGORITHMS

1)SIFT + FLANN

Observations



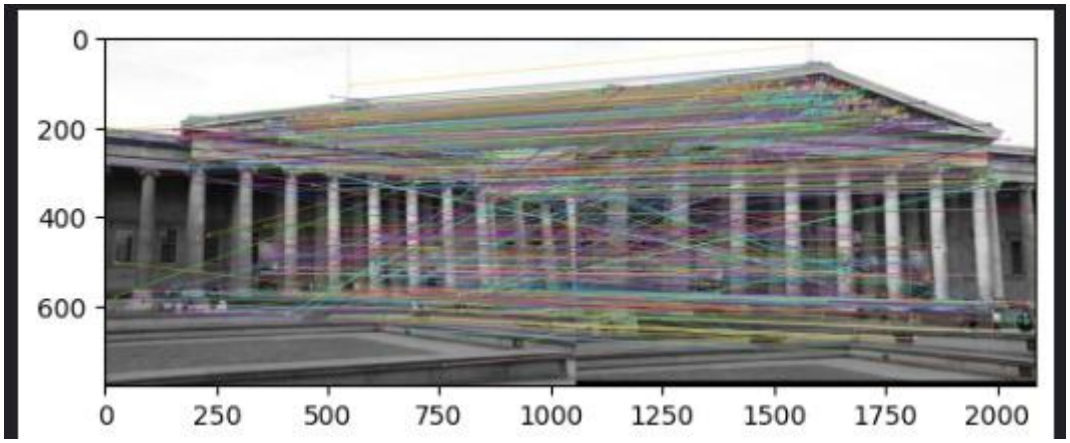
Lowe's ratio = 0.3698…

Combining SIFT feature extraction with FLANN-based matching generally yields favorable results in terms of efficiency and robustness. However, occasional false matches between descriptors can

occur due to factors like ambiguous local features, noise, parameter sensitivity, and challenging geometric transformations. Ransac is applied is such cases to remove the outliers. Although we are getting lowe's ratio 0.36 but after implementing the algorithm we get 100 % good matches.

2)SIFT + GALE SHAPELY

Observations



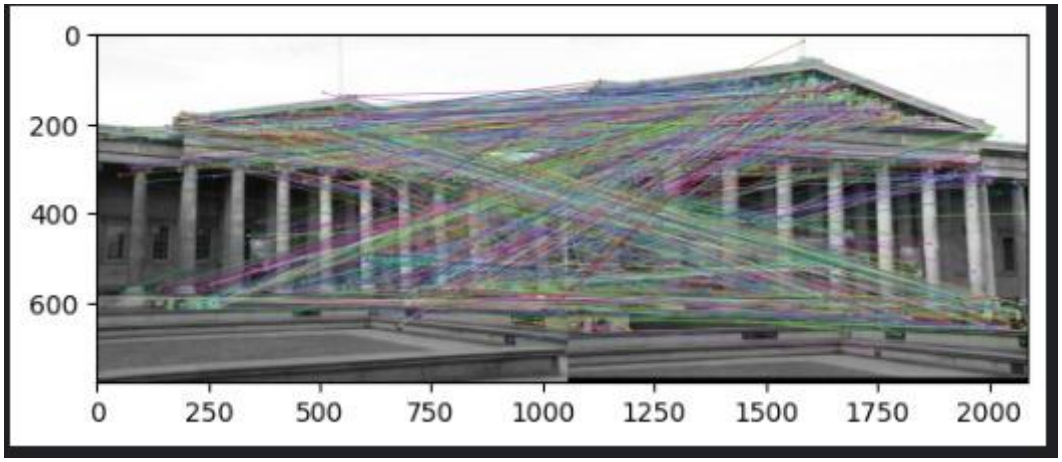
The method combines SIFT descriptors with the Gale Shapley algorithm for feature matching, offering efficiency, stability, and robustness in establishing correspondences between keypoints in images. SIFT descriptors provide robustness to various transformations, while the Gale Shapley algorithm ensures stable matches by iteratively proposing and refining matches based on descriptor preferences. This approach addresses ambiguities and inconsistencies in feature matching, making it suitable for diverse computer vision applications. Overall, the integration of SIFT with Gale Shapley-based stable matching provides an effective and reliable solution for feature-based image analysis.

3)SIFT + PSR

Observations

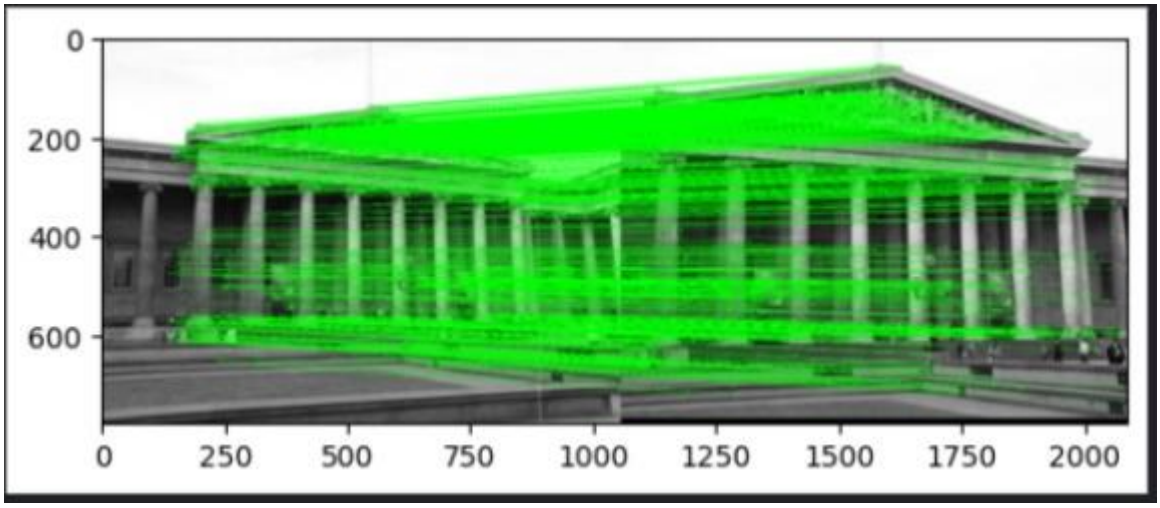
The method combines PSR-EM (Patch Similarity Ratio and Expectation Maximization) with SIFT descriptors for feature matching. It begins by initializing Gaussian Mixture Models (GMMs) with the descriptors from each image and computes likelihoods of descriptors under each other's GMMs. Matches are determined based on the closest likelihoods, ensuring robust correspondence estimation. This iterative process continues until matches are found for all descriptors.

In this case we can observe the overlapping between the good matches which can be improved using RANSAC.



4)A-SIFT + FLANN

Observations



ASIFT (Affine SIFT) with FLANN integration provides a robust and efficient solution for feature-based image matching. By simulating affine transformations, ASIFT expands the search space for feature detection, enhancing robustness against geometric distortions. Key features include:

1)Affine Transformation Simulation: ASIFT generates multiple image variants through affine transformations, improving robustness to tilt and rotation.

2)Feature Detection and Description: SIFT feature detection and description are applied to each transformed image, extracting keypoints and descriptors.

3)Efficient Matching with FLANN: FLANN accelerates feature matching by approximating nearest neighbors, ensuring fast and accurate correspondence estimation.

4)Robustness to Affine Distortions: ASIFT's simulation of affine transformations enhances robustness to geometric variations, enabling reliable feature matching.

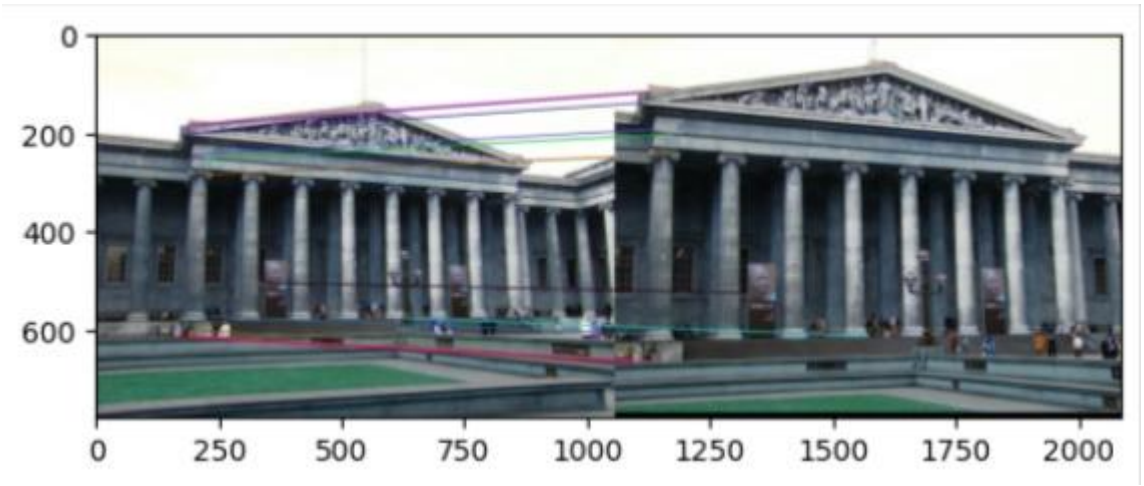
5)Homography Estimation: RANSAC-based homography estimation refines correspondences, filtering outliers and improving alignment accuracy.

Asift provides the similar results like the sift flann algorithm.

5)MOPS + RANSAC

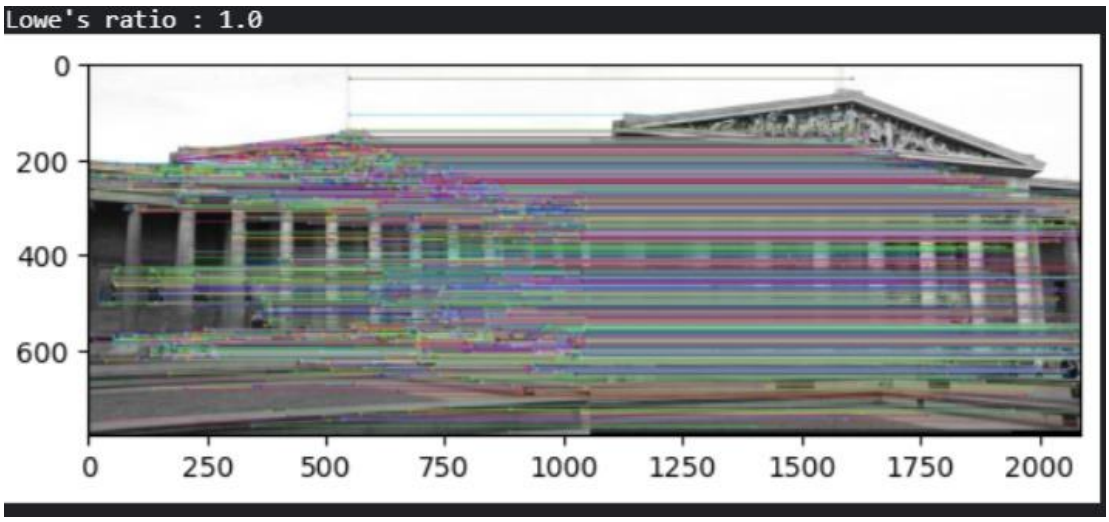
Observations

The method combines Multi-Scale Oriented Patches (MOPS) for feature description with RANSAC for robust correspondence estimation in image matching. It begins by detecting keypoints using the Harris Corner Detector, followed by computing feature descriptors using MOPS. Feature matching is performed between keypoints in the two images, and RANSAC is employed to prune outlier matches and estimate geometric transformations. This iterative process refines feature matches, enhancing accuracy and reliability. The final pruned match set reflects robust correspondences between keypoints, facilitating accurate image alignment and registration.



6)N-SHIFT + FLANN

Observations



NG-SIFT (Normalized Gradient SIFT) integrates gradient normalization with SIFT feature detection and FLANN-based matching for robust and efficient image matching. By normalizing gradients, the method enhances resilience to illumination variations, ensuring consistent feature representation across images. Keypoints and descriptors computed using NG-SIFT are invariant to changes in lighting conditions, improving the reliability of feature matching. FLANN facilitates fast and accurate correspondence estimation between keypoints, enabling efficient image analysis tasks such as object recognition and image retrieval. Lowe's ratio is 1 in this case that means we are getting 100% good matches.

****DEEP LEARNING BASED IMAGE MATCHING****

Observation:

DL-based matching using Light Glue presents a comprehensive approach to feature extraction and matching in computer vision tasks. Key observations include:

1)Feature Representation Learning: Light Glue utilizes deep learning to directly learn feature representations from image data, capturing rich and discriminative features.

2)End-to-End Matching Pipeline: The method offers an end-to-end pipeline for feature extraction and matching, eliminating the need for handcrafted descriptors and separate algorithms.

3)Adaptability: DL-based matching is adaptable to various domains and datasets, leveraging diverse training data to learn generalizable feature representations.

4)Large-Scale Performance: Light Glue demonstrates promising performance in large-scale matching tasks, efficiently handling extensive datasets with millions of images.

Inbuilt LIghtGlue is used for DL based approach which is producing better results as compared to other methods as compared to other methods stated above.

IMAGE STITCHING

The Image_Stitching class provides functionality for stitching images together to create panoramic views. Here's a summary of its key features and methods:

1)Feature Extraction and Matching: Utilizes the SIFT algorithm to extract and match keypoints between two images.

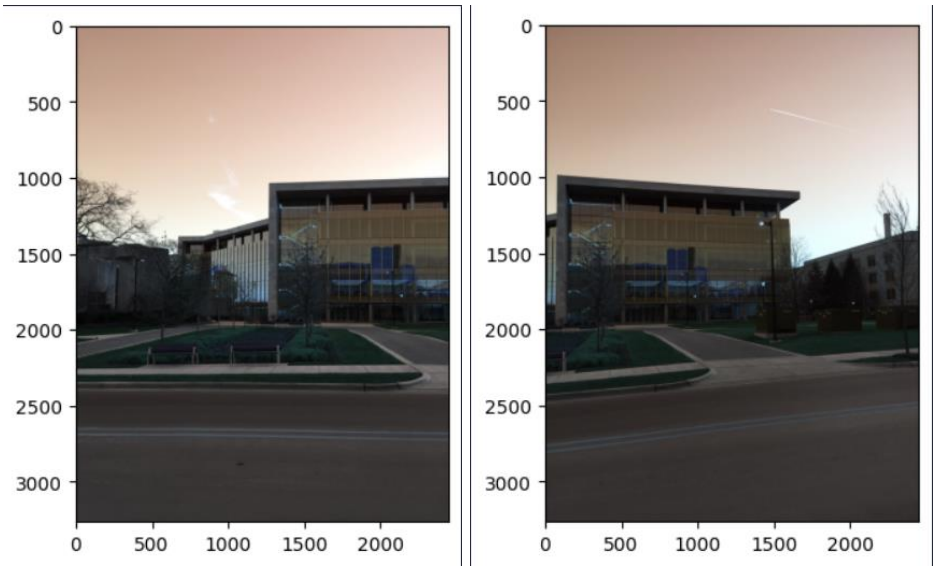
2)Homography Estimation: Estimates the homography matrix using the RANSAC algorithm to align the images properly.

3)Mask Generation: Generates masks to blend the images seamlessly, ensuring gradual transitions between them.

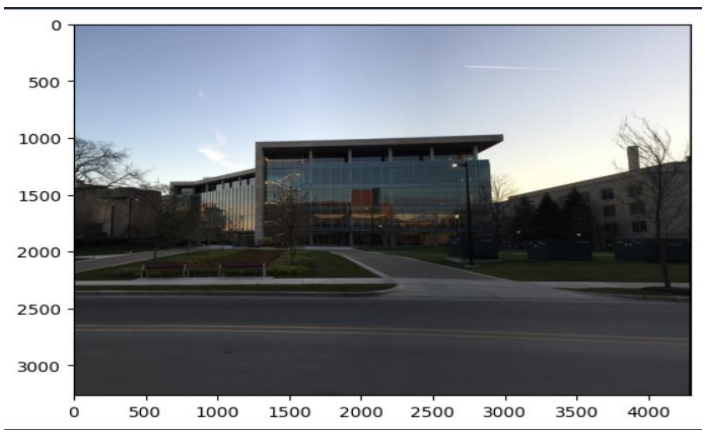
4)Blending: Blends the images together using the estimated homography and masks to create a smooth final panorama.

5)Parameter Tuning: Allows for the adjustment of parameters like matching ratio, minimum matches, and smoothing window size to optimize stitching results.

Image 1 Image2



Stitched Image



References:

1) https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

2) https://www.ipol.im/pub/art/2011/my-asift/article_lr.pdf

3) https://inst.eecs.berkeley.edu/~cs194-26/fa14/Papers/MOPS.pdf

4) https://github.com/deepanshut041/feature-detection/tree/master/brief

5) https://link.springer.com/article/10.1007/s10489-024-05330-3#Sec12