| **Name** | Adwait Purao |
| --- | --- |
| **UID no.** | 2021300101 |

| **Experiment 10** | |
| --- | --- |
| **AIM :** | Perform Image Compression and Decompression |
| **OBJECTIVE:** | * To reduce data redundancy in files (images and documents) during compression. * To conserve more hardware space and transmission bandwidth by reducing the file size through compression. * To enable faster and efficient data transfer without loss of quality by maintaining the quality of the file after decompression. * To perform compression and decompression of files without losing any data or compromising the quality of the original file. |
| **INTRODUCTION:** | This paper proposes using the Gzip algorithm for image and document compression, which combines Lz77 and Huffman techniques. Compression and decompression are vital in document management and communication systems. Image compression, particularly, is advantageous in digital image processing. The goal is to reduce redundancy in images and documents to efficiently store or transmit data. This approach conserves hardware space and bandwidth. In our proposed system, we achieve reduced data size without quality loss. |
| **BLOCK**  **DIAGRAM:** | Block Diagram |
| **IMPLEMENTATION:** | **from** PIL **import** Image **import** os **import** numpy **as** np **import** gzip  **def** **compress\_image**(input\_image\_path, compressed\_image\_path, quality=80):  original\_image = Image.open(input\_image\_path)  original\_size = os.path.getsize(input\_image\_path)   original\_image.save(compressed\_image\_path, 'JPEG', quality=quality, optimize=**True**)  compressed\_size = os.path.getsize(compressed\_image\_path)   **return** original\_size, compressed\_size  **def** **decompress\_image**(compressed\_image\_path, decompressed\_image\_path):  compressed\_image = Image.open(compressed\_image\_path)  compressed\_image.save(decompressed\_image\_path, 'JPEG', quality=100)  decompressed\_size = os.path.getsize(decompressed\_image\_path)   **return** decompressed\_size  **def** **calculate\_metrics**(original\_image\_path, compressed\_image\_path):  original\_image = Image.open(original\_image\_path).convert('RGB')  compressed\_image = Image.open(compressed\_image\_path).convert('RGB')   original\_image\_array = np.array(original\_image)  compressed\_image\_array = np.array(compressed\_image)   mse = np.mean((original\_image\_array - compressed\_image\_array) \*\* 2)  max\_pixel\_value = 255  psnr = 20 \* np.log10(max\_pixel\_value / np.sqrt(mse))   original\_image\_size = os.path.getsize(original\_image\_path)  compressed\_image\_size = os.path.getsize(compressed\_image\_path)  bpp\_original = (original\_image\_size \* 8) / (original\_image.size[0] \* original\_image.size[1])  bpp\_compressed = (compressed\_image\_size \* 8) / (compressed\_image.size[0] \* compressed\_image.size[1])   **return** mse, psnr, bpp\_original, bpp\_compressed  # Example usage input\_image\_path = 'B:\\Image\_Comp\_Decomp\\pulp\_fiction.jpg' compressed\_image\_path = 'compressed\_image.jpg' decompressed\_image\_path = 'decompressed\_image.jpg'  original\_size, compressed\_size = compress\_image(input\_image\_path, compressed\_image\_path) decompressed\_size = decompress\_image(compressed\_image\_path, decompressed\_image\_path)  print(f'Original image size: {original\_size} bytes') print(f'Compressed image size: {compressed\_size} bytes') print(f'Decompressed image size: {decompressed\_size} bytes')  mse, psnr, bpp\_original, bpp\_compressed = calculate\_metrics(input\_image\_path, compressed\_image\_path) print(f'MSE: {mse}') print(f'PSNR: {psnr} dB') print(f'BPP (Original): {bpp\_original}') print(f'BPP (Compressed): {bpp\_compressed}') |
| **OUTPUT:** | **Terminal:**    **Input Image:**  **Size :**    **Compressed Image:**  **Size :**      **Decompressed Image:**  **Size :** |
| **REFERENCE:** | K. Anand, M. Priyadharshini and K. Priyadharshini, "Compression And Decompression Of Files Without Loss Of Quality," 2023 International Conference on Networking and Communications (ICNWC), Chennai, India, 2023, pp. 1-6, doi: 10.1109/ICNWC57852.2023.10127236. keywords: {Image quality;Image coding;Communication systems;Digital images;Redundancy;Data compression;Bandwidth;Lz77;Gzip;Hybrid algorithm},  <https://ieeexplore.ieee.org/document/10127236> |
| **CONCLUSION:**  The proposed Gzip algorithm, integrating LZ77 and Huffman coding, efficiently compresses and decompresses images and documents without sacrificing quality. It reduces data redundancy, conserves space, boosts bandwidth, and maintains file integrity, offering a practical solution for compression and decompression needs. | |