Image Compression via Block-wise SVD

By: Akshay Aralikatti (012624106)

1. Analysis of Results

This project demonstrates how image compression can be achieved using **Singular Value Decomposition (SVD)** applied to non-overlapping 8×8 blocks of a grayscale image. The reconstruction quality and data reduction depend on the number of singular values (k) retained per block.

- When **k is low (e.g., 1 or 2)**, compression is very high, but the image becomes overly blurry and loses important detail.
- As **k increases**, more features, edges, and textures are preserved. At **k = 8**, all singular values are retained, resulting in a reconstruction that is visually identical to the original.
- The **compression ratio decreases** with higher k (i.e., we store more data), but **reconstruction error also decreases**, improving image quality.
- This block-wise SVD approach ensures local features are preserved and makes compression scalable for larger images.

The tradeoff between compression ratio and reconstruction quality is clearly visualized through the plotted graphs.

2. Implementation Summary

The workflow for this implementation is as follows:

1. **Preprocessing**:
    - The input image is converted to grayscale and resized to 256×256.
    - It is then cropped so both dimensions are divisible by 8, which is required for block-based processing.
2. **Block-wise SVD Compression**:
    - The image is divided into 8×8 blocks.
    - For each block, SVD is applied, and only the top k singular values are retained.
    - The block is then reconstructed using this reduced representation.
    - This step is repeated for different values of k (1 through 8).
3. **Evaluation**:
    - For each k, two metrics are calculated:

- **Compression Ratio** = 64 / (k × (8 + 8 + 1))
- **Reconstruction Error** = Frobenius norm between original and reconstructed image

4. **Visualization**:
    - The original and selected compressed images (k = 1, 3, 5, 8) are displayed side-by-side.
    - Two graphs are plotted:
    - Compression Ratio vs. k
    - Reconstruction Error vs. k

3. Code Snippets:

a. Preprocessing the image:

```
def preprocess_image(image_path):
    img = Image.open(image_path).convert('L')
    width, height = img.size
    new_width = width - (width % 8)
    new_height = height - (height % 8)
    img = img.crop((0, 0, new_width, new_height))
    img = img.resize((256, 256))
    return np.array(img)
```

b. Compressing a single 8x8 block

```
def compress_block(block, k):
    U, S, V = np.linalg.svd(block)
    return U[:, :k] @ np.diag(S[:k]) @ V[:k, :]
```

c. Compressing the Entire Image and analyzing:

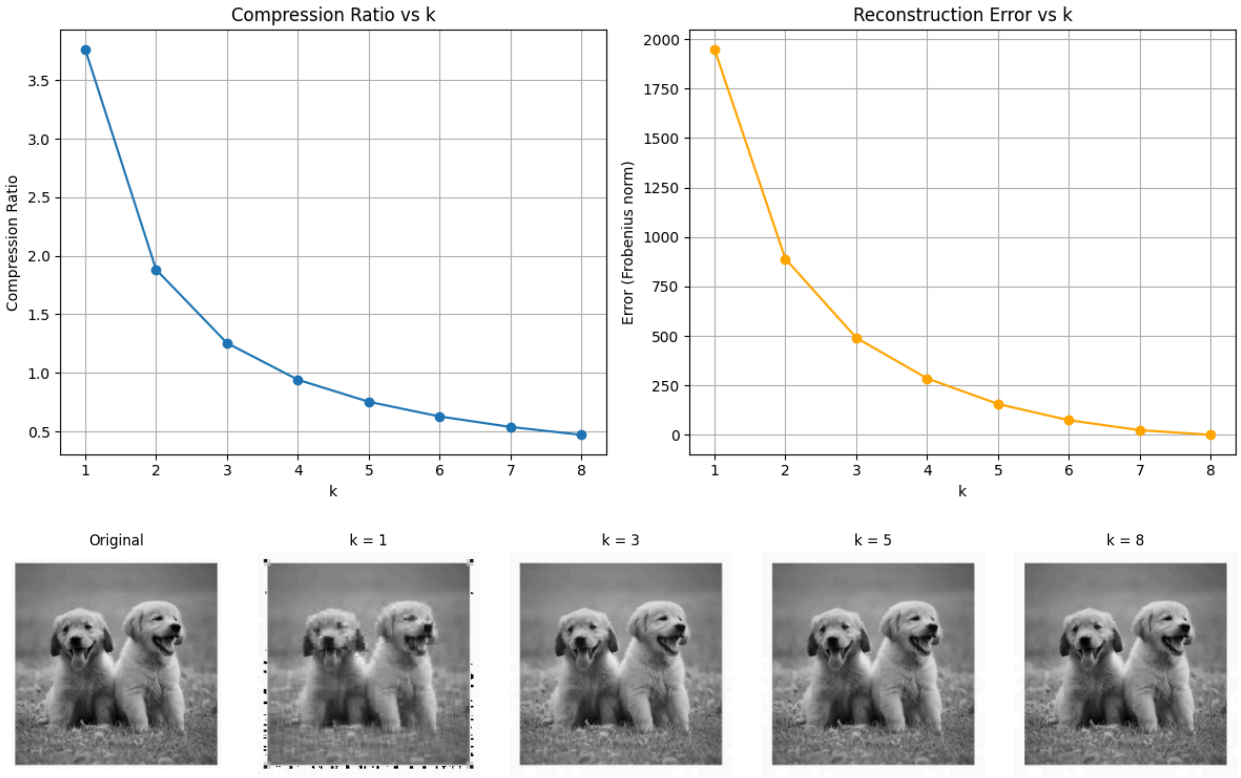
```
def compress_image(image, k_values):
    h, w = image.shape
    block_size = 8
    compressed_images, compression_ratios, reconstruction_errors = [],
[], []
```

```
    for k in k_values:
        compressed_image = np.zeros_like(image, dtype=np.float64)
        for i in range(0, h, block_size):
            for j in range(0, w, block_size):
                block = image[i:i+block_size, j:j+block_size]
                compressed_block = compress_block(block, k)
                compressed_image[i:i+block_size, j:j+block_size] =
compressed_block

        compressed_images.append(compressed_image)

        # Compression ratio and error
        original_vals = block_size * block_size
        compressed_vals = k * (block_size + block_size + 1)
        compression_ratios.append(original_vals / compressed_vals)
        reconstruction_errors.append(np.linalg.norm(image -
compressed_image))

    return compressed_images, compression_ratios, reconstruction_errors
```

d. Visualizing Results:

```
def plot_results(k_values, compression_ratios, errors):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(k_values, compression_ratios, marker='o')
    plt.title('Compression Ratio vs k')
    plt.xlabel('k')
    plt.ylabel('Compression Ratio')
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(k_values, errors, marker='o', color='orange')
    plt.title('Reconstruction Error vs k')
    plt.xlabel('k')
    plt.ylabel('Error (Frobenius norm)')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

4. Visualisation:



5. Conclusion

This block-wise SVD compression project demonstrates how mathematical decomposition techniques can be used effectively for lossy image compression. By tuning k, we can strike a balance between file size and image clarity, making it highly practical for image storage and transmission applications.