Data Compression via Singular Value Decomposition (SVD) (Project Code "C").
You may begin by selecting one or more test images (grayscale or color) and representing them as matrices. And then you will apply SVD to decompose the image matrix into singular values and vectors, then implement a compression scheme by truncating the smaller singular values and reconstructing the image from the reduced-rank approximation. The project should analyze how the number of singular values retained affects image quality, compression ratio, and storage requirements, using quantitative error metrics (e.g., Frobenius norm error) and qualitative visual inspection. Extensions may include color image compression (applying SVD to each channel separately), comparisons with other basic compression methods, or discussions of trade-offs between efficiency and fidelity.
Note in this project you need to handwrite the SVD algorithm, meaning that you cannot import any existing SVD packages.

How effectively can SVD-based compression reduce image storage requirements while maintaining acceptable visual quality, and what are the quantitative and qualitative trade-offs between compression ratio and image fidelity?

This project seeks to implement a custom SVD algorithm from scratch and develop a compression scheme that systematically evaluates the relationship between the number of retained singular values and resulting image quality, compression efficiency, and storage requirements.

Quantitative Findings:

- How does error decrease as k increases?
    - As you add more singular values (increase k), your reconstruction error **decreases monotonically** (always gets better, never worse).
- At what k value does PSNR reach acceptable levels (>30 dB)?
    - For typical images, k = 50-100 often achieves PSNR > 30 dB
    -
- What's the "knee point" where adding more singular values gives diminishing returns?
    - This is where **most of the important information has been captured**. After this point, you're adding a lot of data for small quality improvements.

Compression Trade-offs:

- What compression ratio can you achieve while maintaining good quality?
    - Low k (5-20): High compression, Poor quality
    - Medium k (50-100): Moderate compression, Good quality, Sweet spot!
    - High k (200+): Low compression, Excellent quality

- How does storage requirement scale with k?
    - Storage(k) = k × (m + n + 1)
    - Compression Ratio = k(m + n + 1) / (m × n)
    - If you double k, you double the storage requirement.

Qualitative Observations:

- Which image features appear first (low k)?
    - **Large-scale structure** - overall shape and brightness distribution
- Which details are lost at low k values?
    - **Fine details** and sharp edges, textures, gradients
- Are edges preserved better than textures?
    - **Edges** are seen at lower K values (~30) than textures (~100)

Optimal k Selection:

- Recommend optimal k for different use cases (high quality vs. high compression)
    - **Optimal k depends on context.**
    - **Thumbnail previews** (k = 10-20): Quick loading, rough preview.
    - **Web images** (k = 50-75): Balance of quality and loading speed
    - **Archival/Professional** (k = 100-150):Preserve quality, compression is secondary
    - **Diminishing Returns** (k > 100)**:** Each additional value helps less