CSCI/MATH-485

Assignment-5

Snehitha Gorantla

04-15-2025

Singular Value Decomposition

Introduction

This report explores the application of Singular Value Decomposition (SVD) for grayscale image compression. The implemented approach divides the image into non-overlapping 8×8 blocks, applies SVD to each block, and reconstructs the image using only the top-k singular values for $k \in \{1, 2, \ldots, 8\}$. This technique allows us to analyze the trade-off between compression ratio and image quality.

Implementation Summary

1. **Image Preprocessing**: Load a grayscale image and ensure its dimensions are divisible by 8 by cropping if necessary.

```python
def load_and_preprocess_image(image_path):

    img = Image.open("C:/Users/snehi/Downloads/grayscale.png")

    width, height = img.size

    new_width = width - (width % 8)
    new_height = height - (height % 8)

    left = (width - new_width) // 2
    top = (height - new_height) // 2
    right = left + new_width
    bottom = top + new_height

    img = img.crop((left, top, right, bottom))
    return np.array(img)
```

2. **Block-wise SVD**: Divide the image into 8×8 blocks and apply SVD compression to each block:
    - For each block, compute the SVD: U, S, V^T
    - Retain only the top-k singular values and corresponding vectors
    - Reconstruct the block using these components
    - Combine all blocks to form the compressed image

```python
def compress_block(block, k):

    U, S, Vt = np.linalg.svd(block)
    U_k = U[:, :k]
    S_k = S[:k]
    Vt_k = Vt[:k, :]

    reconstructed = U_k @ np.diag(S_k) @ Vt_k
    return reconstructed
```

```python
def block_wise_svd(image, block_size=8, k=1):

    height, width = image.shape
    compressed_blocks = []

    for i in range(0, height, block_size):
        row_blocks = []
        for j in range(0, width, block_size):
            block = image[i:i+block_size, j:j+block_size]
            compressed_block = compress_block(block, k)
            row_blocks.append(compressed_block)
        compressed_blocks.append(np.hstack(row_blocks))

    reconstructed_image = np.vstack(compressed_blocks)
    return reconstructed_image
```

```python
def calculate_compression_ratio(k, block_size=8):

    original_size = block_size * block_size
    compressed_size = k * (block_size + block_size + 1)  # U: 8×k, Σ: k, Vᵀ: k×8
    return original_size / compressed_size
```

3. **Compression Analysis**: For each k value, calculate:
    - Compression ratio = Original data size / Compressed data size
    - Reconstruction error (Frobenius norm)
    - Peak Signal-to-Noise Ratio (PSNR)

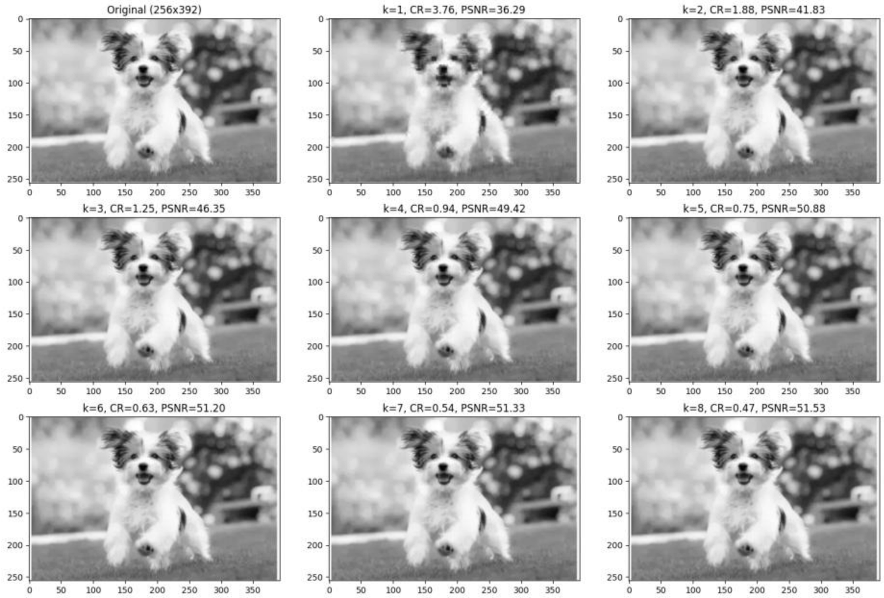
```python
def calculate_psnr(original, compressed):

    mse = mean_squared_error(original.flatten(), compressed.flatten())
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr
```

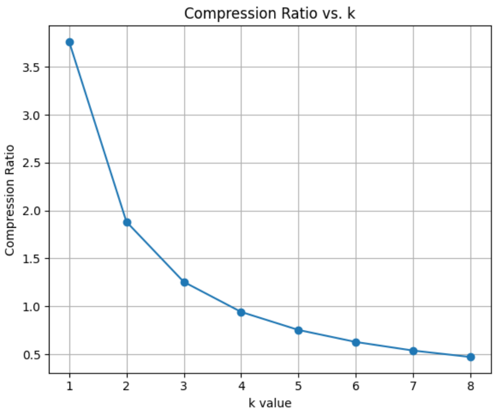
```python
def calculate_frobenius_norm(original, compressed):

    return np.linalg.norm(original - compressed)
```

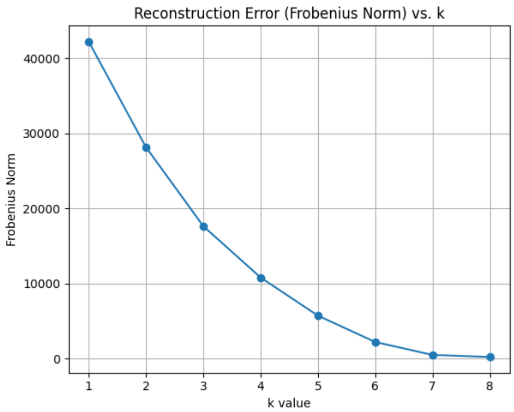
4. **Visualization**: Generate visual comparisons of the original and reconstructed images, along with plots of compression metrics versus k.

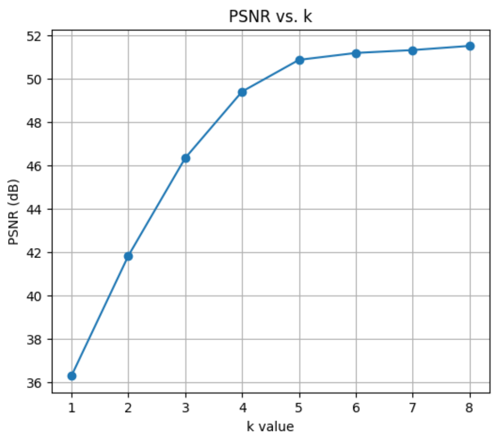
Output

```
k       Compression Ratio       Frobenius Norm  PSNR (dB)
1       3.7647          42225.23                36.29
2       1.8824          28160.26                41.83
3       1.2549          17645.24                46.35
4       0.9412          10804.94                49.42
5       0.7529          5726.75         50.88
6       0.6275          2218.03         51.20
7       0.5378          493.03          51.33
8       0.4706          214.29          51.53
```









Analysis of Results

1. Compression Ratio Trend:
    - The compression ratio decreases as k increases, following a hyperbolic pattern
    - At k=4 and beyond, the compression ratio falls below 1.0, indicating that we're using more storage than the original image
    - This crossover point is critical for determining the practical utility of SVD compression
2. Reconstruction Error:
    - The Frobenius norm decreases rapidly as k increases
    - The most significant error reduction occurs between k=1 and k=4
    - The error continues to decrease at a slower rate beyond k=4
3. Image Quality (PSNR):
    - PSNR increases steadily with k, showing a logarithmic pattern
    - The most significant quality improvements occur in the range k=1 to k=4
    - Beyond k=5, the improvement in PSNR becomes minimal