GNR 607
Programming Assignment Presentation

Team Members:

Aminur Hossain (ID: 24D1384)

Amartya Ray (ID: 24D1383)

Problem Statement

Problem No 22

Objective*:*

Given a multiband image of N bands, compute principal components and generate the approximate version of the input image by performing inverse principal component transform using 2, 3, ..., N-1 components.

Input Data*:*

- **Dataset:** 1 subset image taken from a Landsat-8 satellite full scene image
- **Bands:** 7 bands (for the Mumbai scene)

  LC08_L1TP_148047_20180423_20180502_01_T1.tar.gz

Project uploaded at: https://github.com/aminurds/SIP_Assignment_2024/tree/main

Procedure

Steps to Solve Problem

1.Load the Multiband Image:

- Import the Landsat-8 image with 7 bands for the Mumbai scene.

2.Compute Principal Components (PCA):

- Perform Principal Component Analysis (PCA) on the 7-band image to reduce dimensionality.

3.Reconstruction Using 2, 3, ..., N-1 Principal Components:

- Reconstruct the image using inverse PCA with 2, 3, 4, 5, and 6 components to generate approximate images.

4.Comparison:

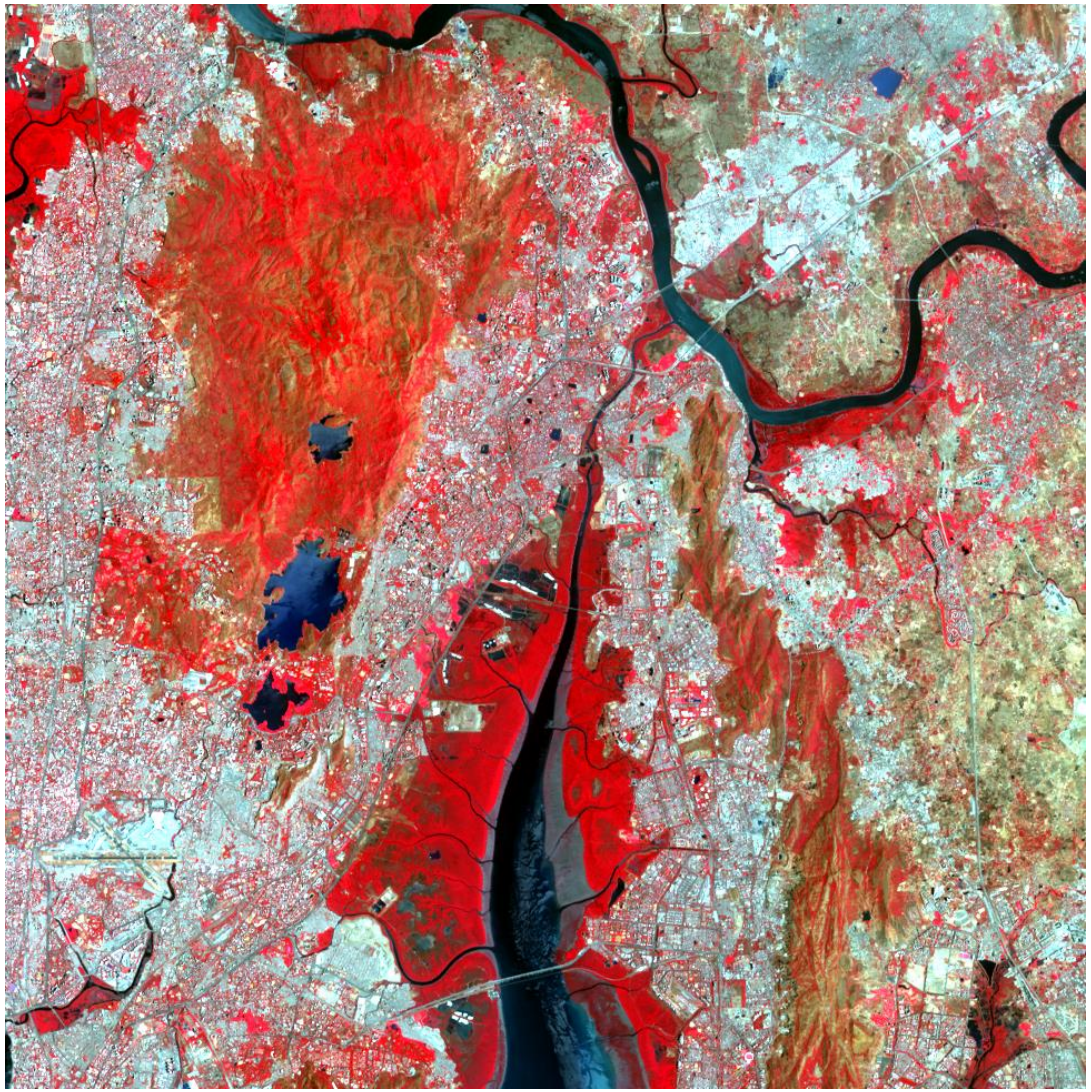
- Compare the quality of reconstructed images using various quality metrics such as PSNR and SSIM to evaluate how much detail is retained with fewer components.

5.Visualization:

- Display the original image alongside the reconstructed versions with different numbers of components for a visual comparison.

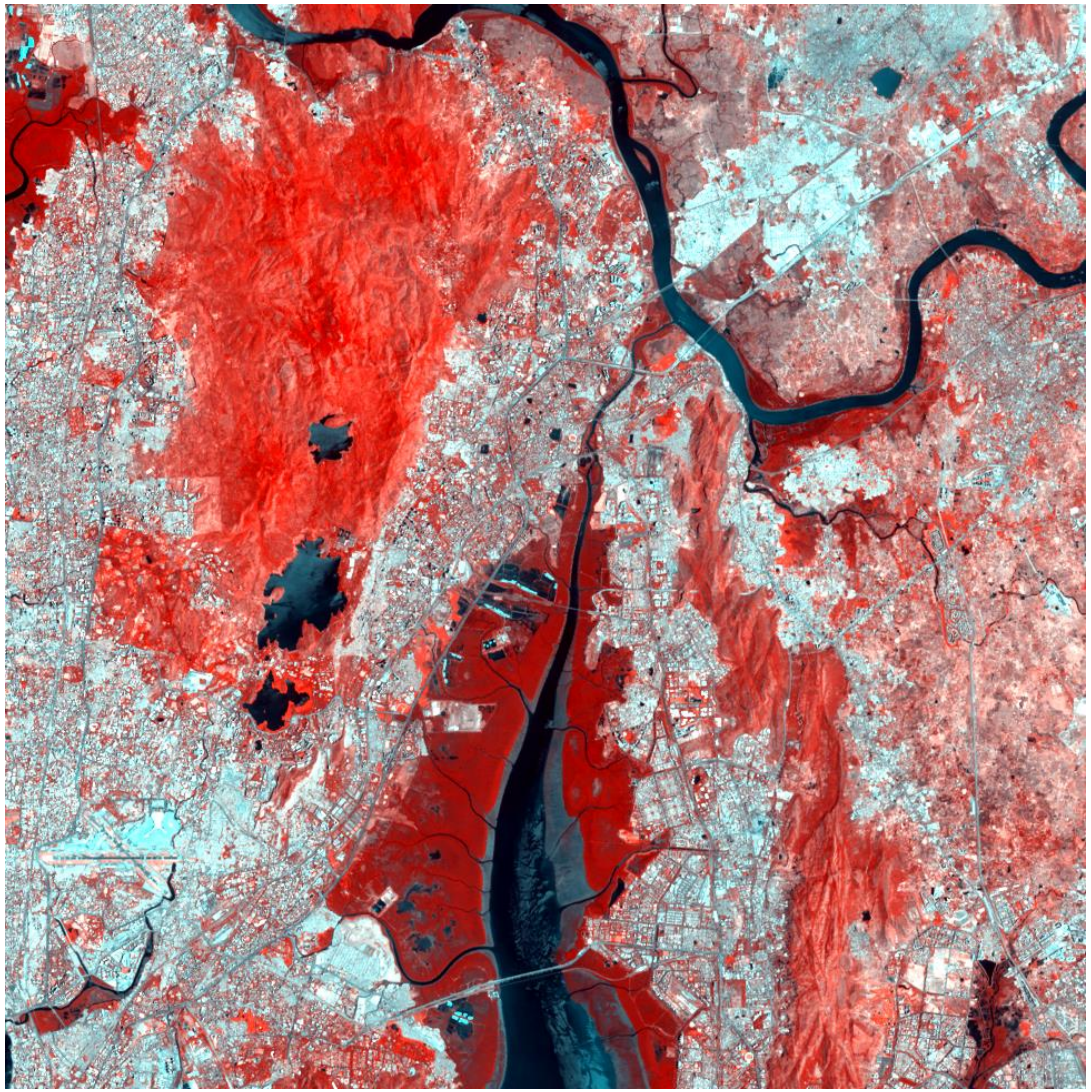
Input Images

Original RGB image

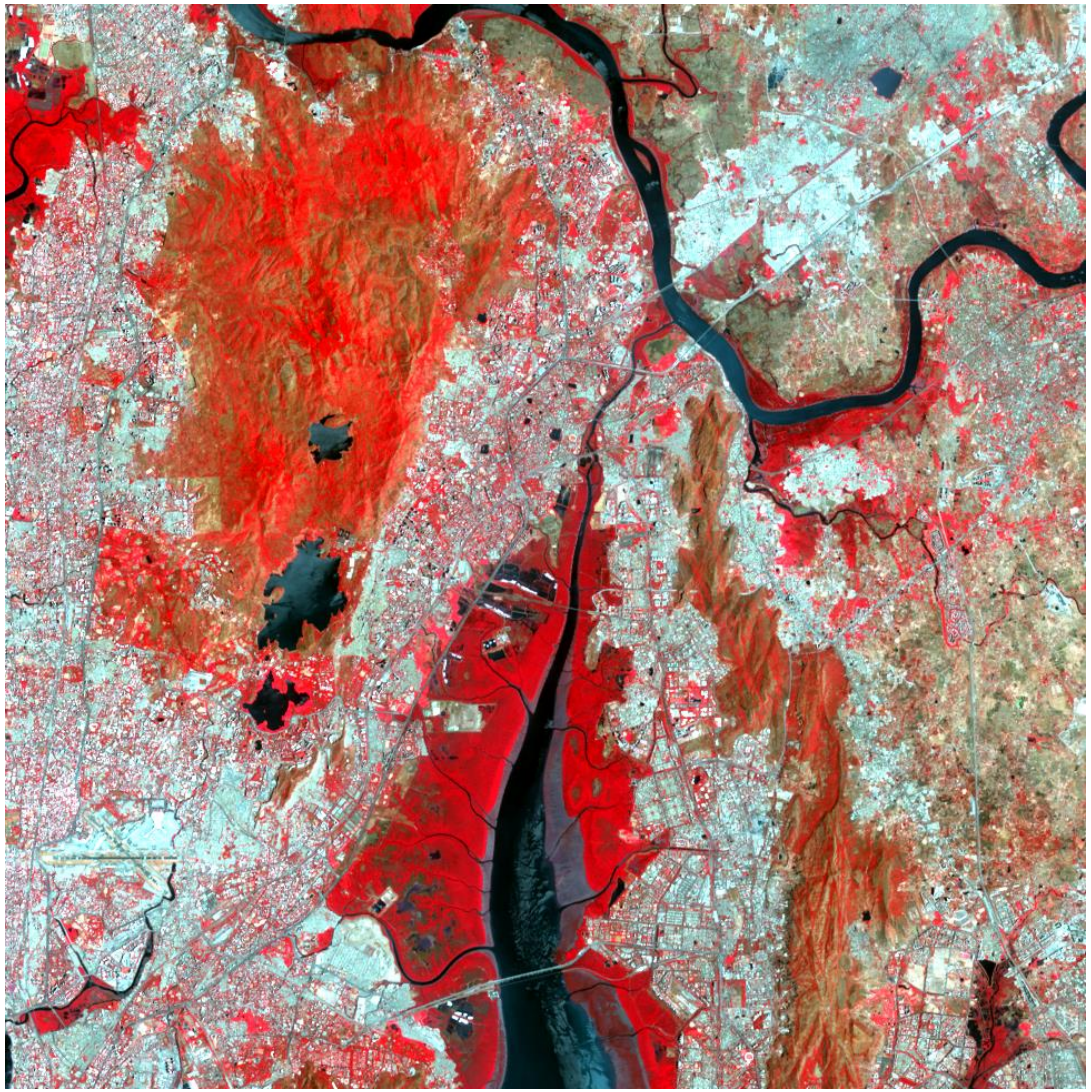


Original image in TCC

Reconstruction Results

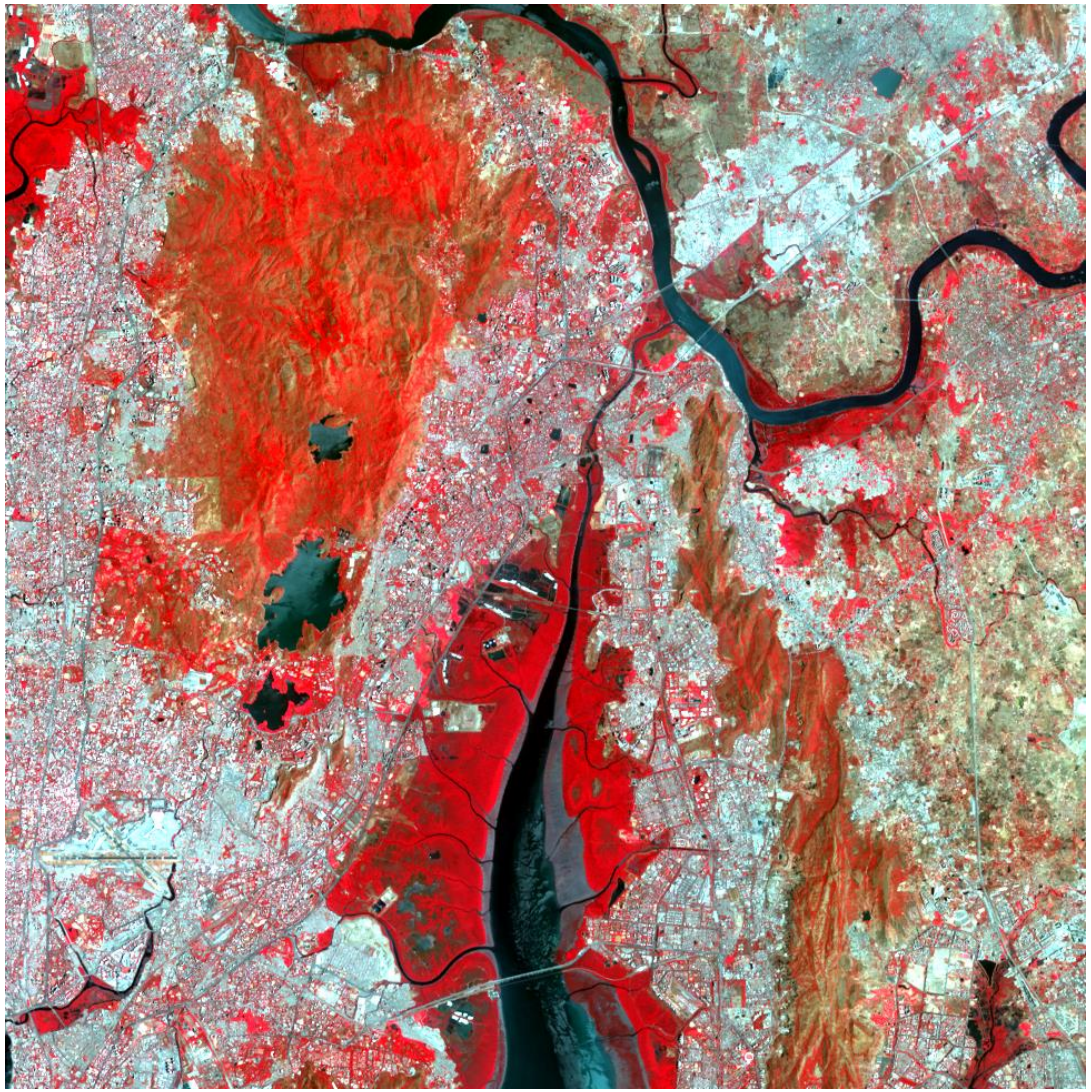


Reconstructed image with 1st 2 PCs; **PSNR – 42.37**

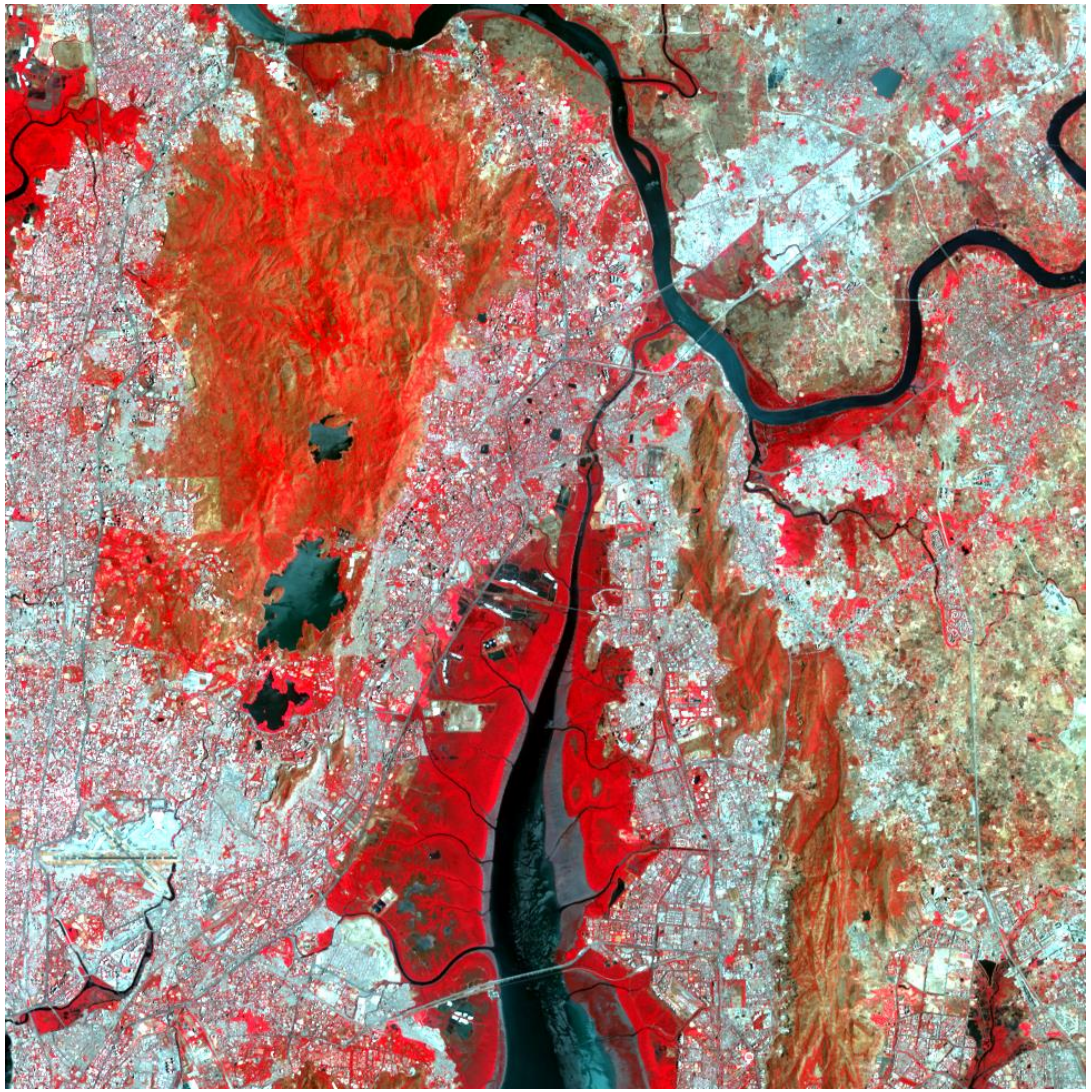


Reconstructed image with 1st 3 PCs; **PSNR – 53.18**

Reconstruction Results

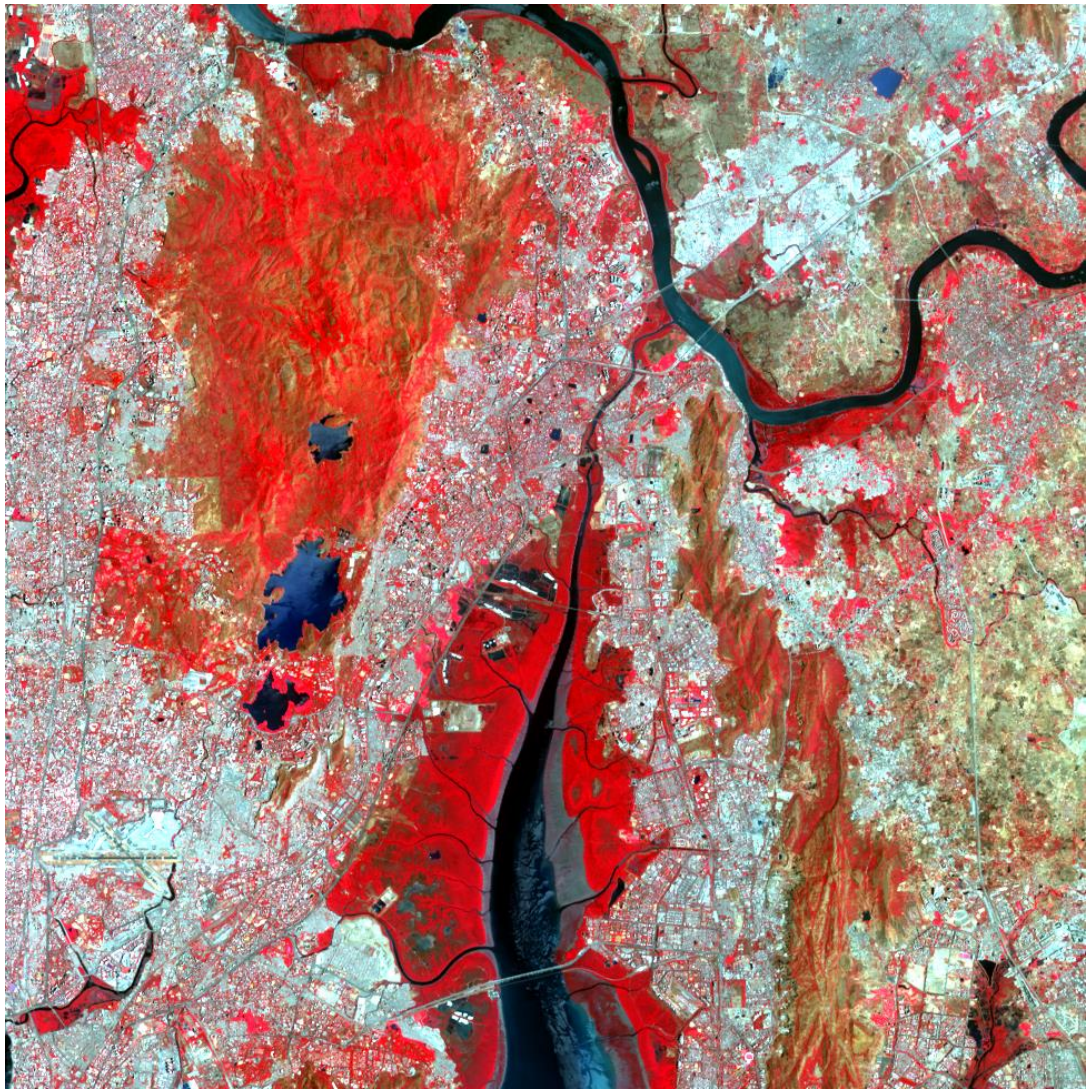


Reconstructed image with 1st 4 PCs; **PSNR – 56.42**

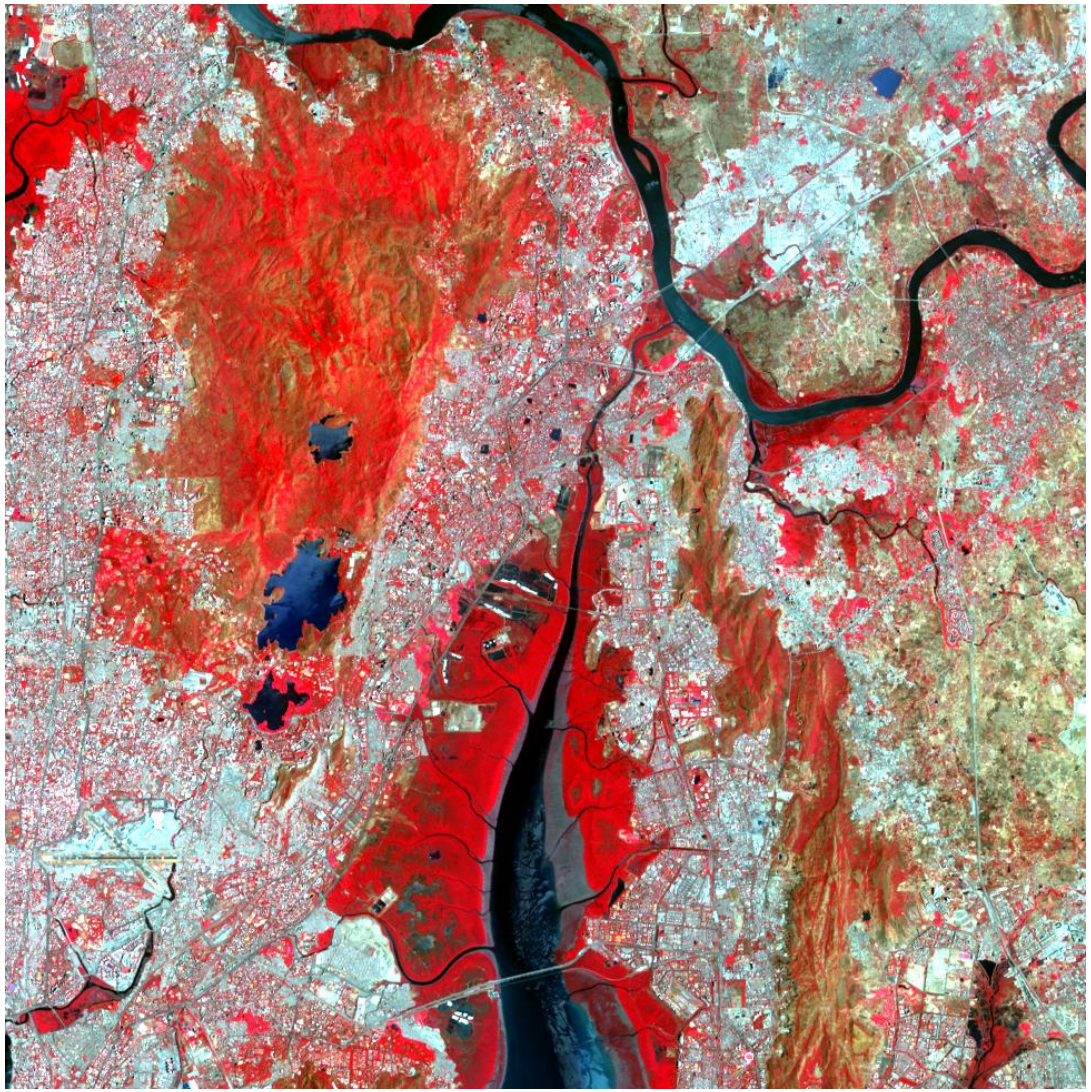


Reconstructed image with 1st 5 PCs; **PSNR – 64.12**

Reconstruction Results



Reconstructed image with 1st 6 PCs; **PSNR – 73.59**



Reconstructed image with all 7 PCs; original image

Statistical Results

Number of Components	Explained Variance (%)	PSNR Value	SSIM Value
2	92.65	42.27	0.97481
3	98.75	53.18	0.997816
4	99.65	56.42	0.999007
5	99.82	64.12	0.999828
6	99.95	73.59	0.999979

PSNR – Peak Signal to Noise Ratio
SSIM – Structural Similarity Index Measure

Code Snippets

```python
def read_multiband_image(file_path):
    with rasterio.open(file_path) as src:
        bands = src.read()
        meta = src.meta
    return bands, meta

def write_image(file_path, image, meta):
    with rasterio.open(file_path, 'w', **meta) as dest:
        dest.write(image)
```

```python
def normalize_data(bands):
    num_bands = len(bands)
    height = len(bands[0])
    width = len(bands[0][0])
    # Initialize arrays for the normalized image, mean, and std
    normalized_image = np.empty_like(bands, dtype=np.float64)
    mean = np.zeros((num_bands, 1), dtype=np.float64)
    std = np.zeros((num_bands, 1), dtype=np.float64)
    # Flatten the image bands and calculate mean and std manually
    for band_index in range(num_bands):
        band = bands[band_index]
        # Calculate mean manually
        sum_band = 0.0
        for i in range(height):
            for j in range(width):
                sum_band += band[i][j]
        mean[band_index] = sum_band / (height * width)
        # Calculate standard deviation manually
        sum_squared_diff = 0.0
        for i in range(height):
            for j in range(width):
                sum_squared_diff += (band[i][j] - mean[band_index]) ** 2
        std[band_index] = (sum_squared_diff / (height * width)) ** 0.5
        # Normalize the band
        normalized_image[band_index] = ((band - mean[band_index]) / std[band_index])
    return normalized_image, mean, std
```

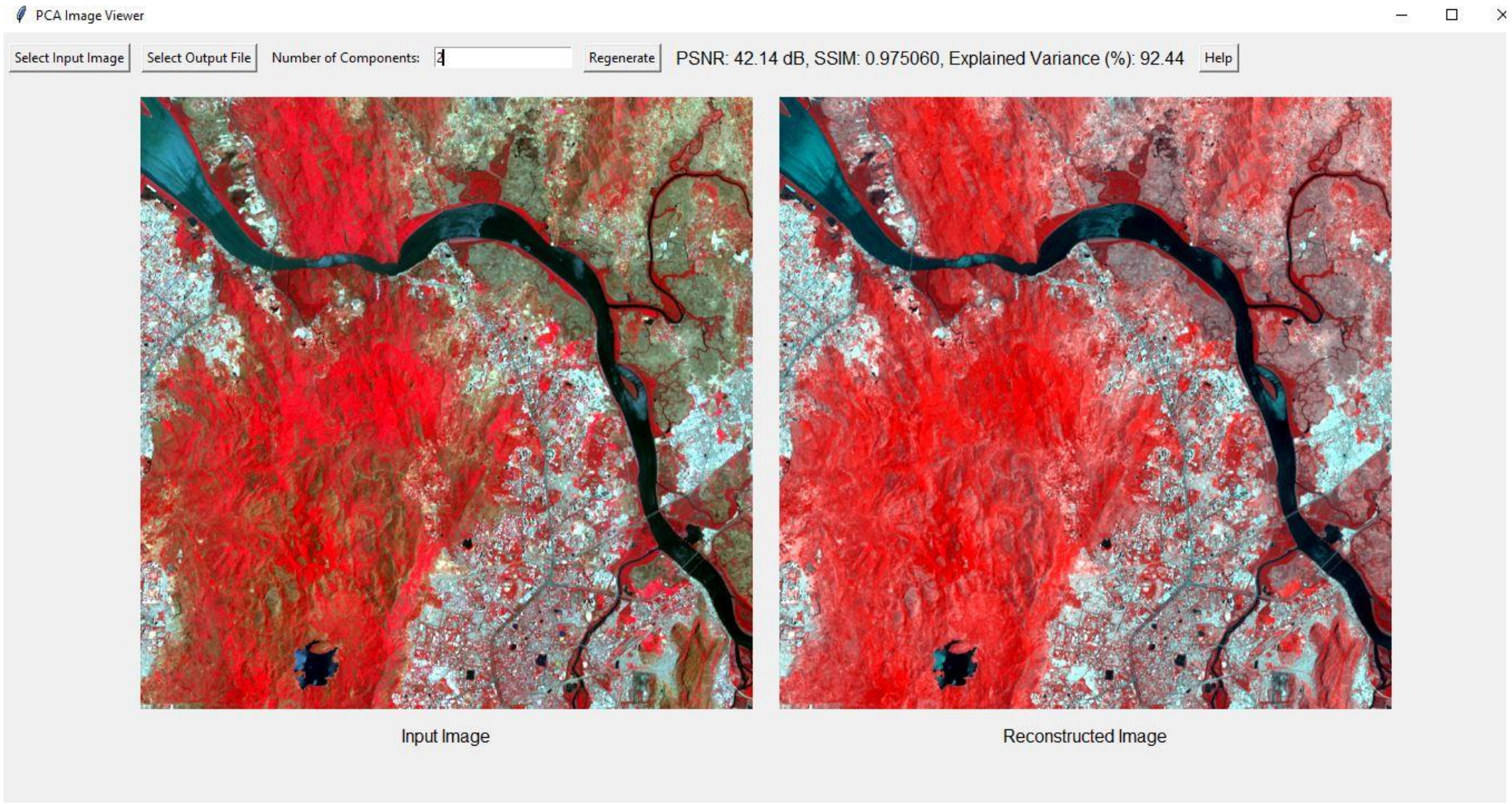
```python
def compute_band_covariance(flattened_bands, i, j, N, normalization_factor):
    b1 = flattened_bands[i]
    b2 = flattened_bands[j]
    cov = sum(b1[k] * b2[k] for k in range(N))
    return i, j, cov * normalization_factor

def compute_covariance_matrix(image):
    num_bands = len(image)
    height = len(image[0])
    width = len(image[0][0])
    N = height * width
    normalization_factor = 1 / (N - 1)
    flattened_bands = [[pixel for row in band for pixel in row] for band in image]
    covariance_matrix = [[0 for _ in range(num_bands)] for _ in range(num_bands)]
    # Use a ThreadPoolExecutor to compute band pairs in parallel
    with ThreadPoolExecutor() as executor:
        futures = []
        for i in range(num_bands):
            for j in range(i, num_bands):
                futures.append(executor.submit(compute_band_covariance, flattened_bands, i, j, N, normalization_factor))
        for future in futures:
            i, j, value = future.result()
            covariance_matrix[i][j] = value
            if i != j:
                covariance_matrix[j][i] = value
    return covariance_matrix
```

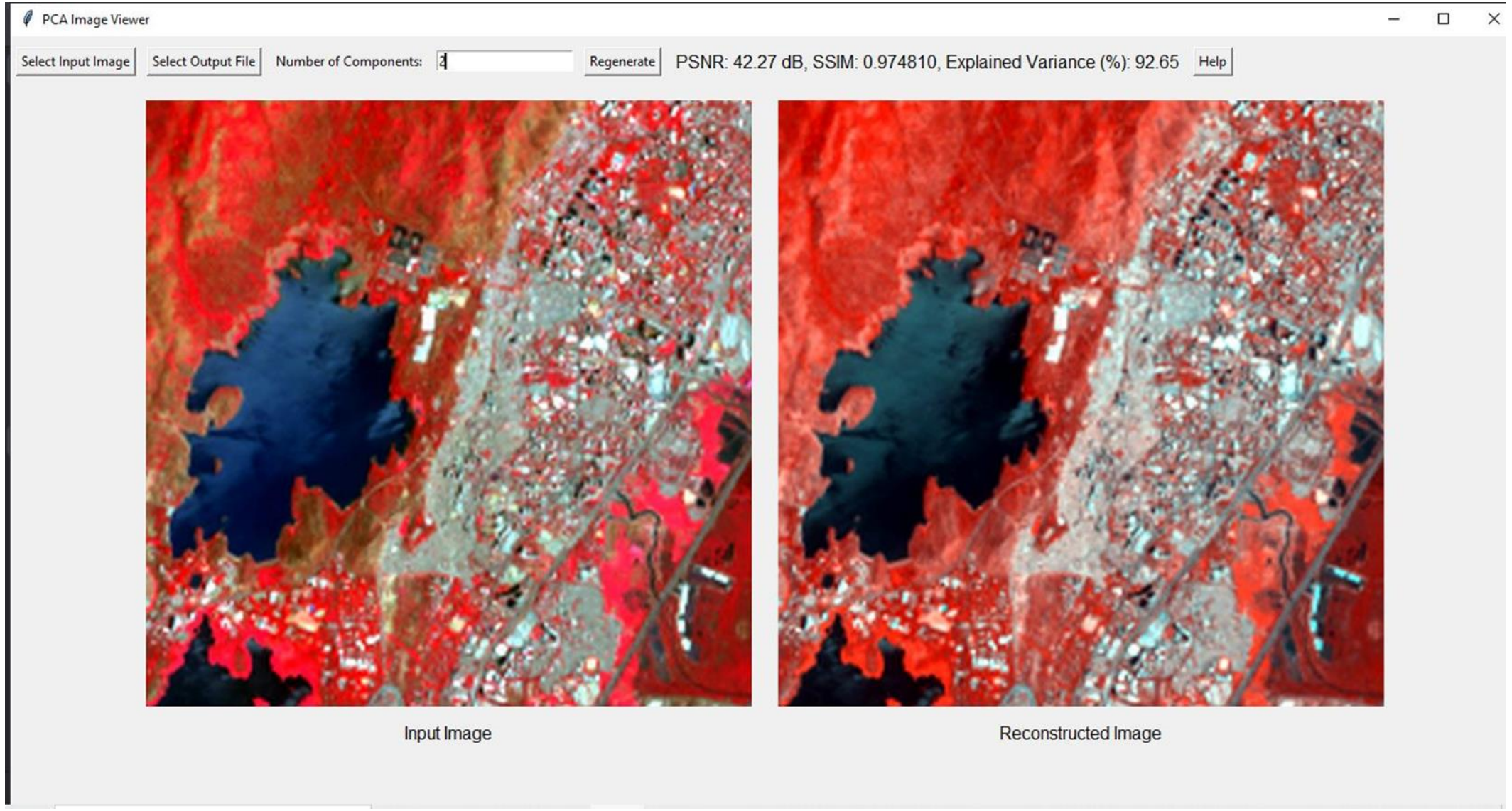
```python
def perform_pca(bands):
    # Compute covariance matrix
    covariance_matrix = compute_covariance_matrix(bands)
    # Compute eigenvalues and eigenvectors
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
    # Sort eigenvalues and eigenvectors in descending order
    idx = sort_indices_desc(eigenvalues)
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]
    num_bands = len(bands)
    height = len(bands[0])
    width = len(bands[0][0])
    # Project the data onto the eigenvectors
    principal_components = np.zeros_like(bands)
    for i in range(num_bands):
        ev = eigenvectors[:, i]
        pc = np.zeros_like(bands[0])
        for j in range(num_bands):
            pc = pc + bands[j] * ev[j]
        principal_components[i] = pc
    # Example matrix principal_components (7x10x15)
    n_bands = len(principal_components)
    height = len(principal_components[0])
    width = len(principal_components[0][0])
    # Initialize the new matrix with the desired shape (150x7)
    reshaped_principal_components = np.zeros((height * width, n_bands))
    # Manually reshape the matrix
    for i in range(n_bands):  # Loop through the first dimension (7)
        for j in range(height):  # Loop through the second dimension (10)
            for k in range(width):  # Loop through the third dimension (15)
                # Calculate the new row index for the reshaped matrix
                new_index = j * width + k
                reshaped_principal_components[new_index, i] = principal_components[i, j, k]
    return eigenvectors, reshaped_principal_components, (num_bands, height, width), eigenvalues
```

```python
def inverse_pca_and_reconstruct(eigenvectors, principal_components, shape, num_components, mean, std):
    # Select the top num_components eigenvectors
    selected_eigenvectors = eigenvectors[:, :num_components]
    # Select the corresponding principal components
    selected_components = principal_components[:, :num_components]
    # Reconstruct the image
    reconstructed = np.dot(selected_components, selected_eigenvectors.T)
    reconstructed = reconstructed.T
    # De-normalize the reconstructed data
    num_bands, height, width = shape
    reconstructed = reconstructed.reshape(num_bands, height * width)
    reconstructed = (reconstructed * std) + mean
    reconstructed = reconstructed.reshape(num_bands, height, width)
    selected_components_reshaped = selected_components.reshape(height, width, num_components)
    # print(selected_components_reshaped.shape)
    return reconstructed, selected_components_reshaped
```

GUI Screenshots



GUI Screenshots



GUI Screenshots Diff

Thank You