III zestaw zadań - Hierarchiczna kompresja macierzy

Kacper Kozubowski, Mateusz Podmokły
III rok Informatyka WI

27 listopad 2024

1 Treść zadania

Proszę wybrać kolorową bitmapę, np 500 x 500, zamienić ją na 3 macierze RGB (wartości z przedziału $[0, 255]$) i zaimplementować:

1. Rekurencyjną kompresję macierzy z wykorzystaniem częściowego SVD
2. Rysowanie skompresowanej macierzy
3. Rysowanie skompresowanej bitmapy

2 Specyfikacja użytego środowiska

Specyfikacja:

- Środowisko: Jupyter Notebook,
- Język programowania: Python,
- System operacyjny: Microsoft Windows 11,
- Architektura systemu: x64.

3 Działanie algorytmów

3.1 Wykorzystane biblioteki

W realizacji rozwiązania wykorzystane zostały następujące biblioteki:

```
1  import numpy as np
2  import cv2
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches
```

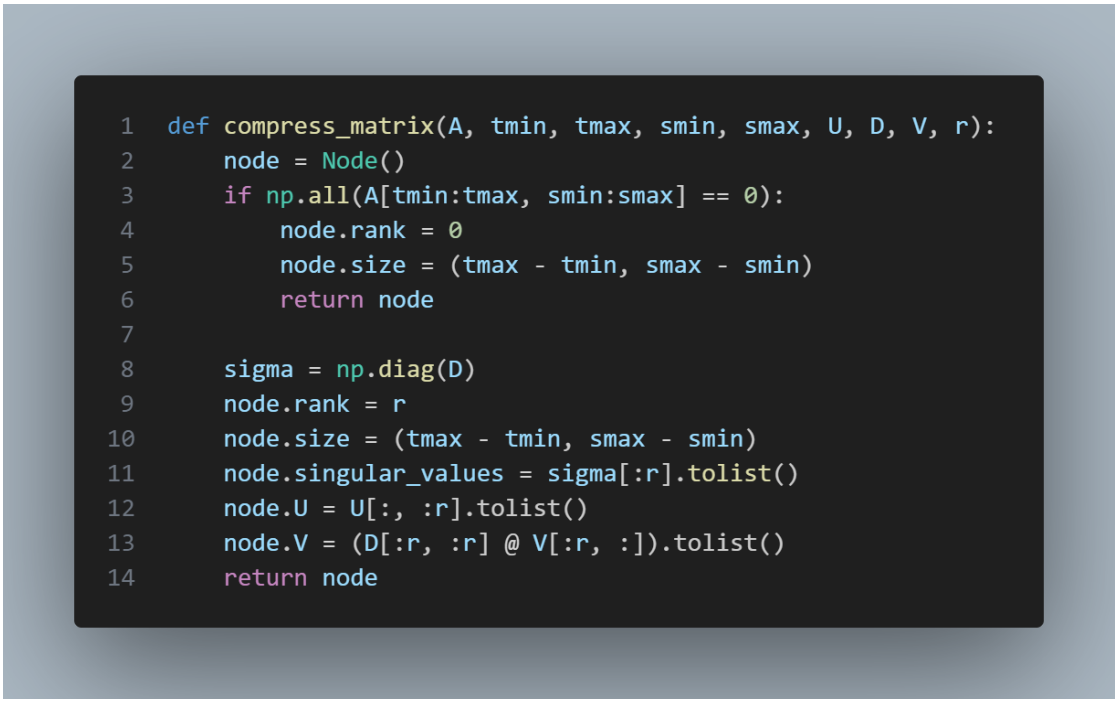
3.2 Kod funkcji

```python
class Node:
    def __init__(self):
        self.rank = 0
        self.size = (0, 0)
        self.singular_values = []
        self.U = None
        self.V = None
        self.children = []
        self.channels = []

    def __repr__(self):
        return f"Node(rank={self.rank}, size={self.size},\
            children={len(self.children)})"
```

Rysunek 1: Struktura drzewa macierzy.

```python
def truncated_SVD(A, rank):
    U, S, Vt = np.linalg.svd(A, full_matrices=False)
    return U[:, :rank], np.diag(S[:rank]), Vt[:rank, :]
```

Rysunek 2: Częściowe SVD.



```python
def compress_matrix(A, tmin, tmax, smin, smax, U, D, V, r):
    node = Node()
    if np.all(A[tmin:tmax, smin:smax] == 0):
        node.rank = 0
        node.size = (tmax - tmin, smax - smin)
        return node

    sigma = np.diag(D)
    node.rank = r
    node.size = (tmax - tmin, smax - smin)
    node.singular_values = sigma[:r].tolist()
    node.U = U[:, :r].tolist()
    node.V = (D[:r, :r] @ V[:r, :]).tolist()
    return node
```

Rysunek 3: Kompresja macierzy.

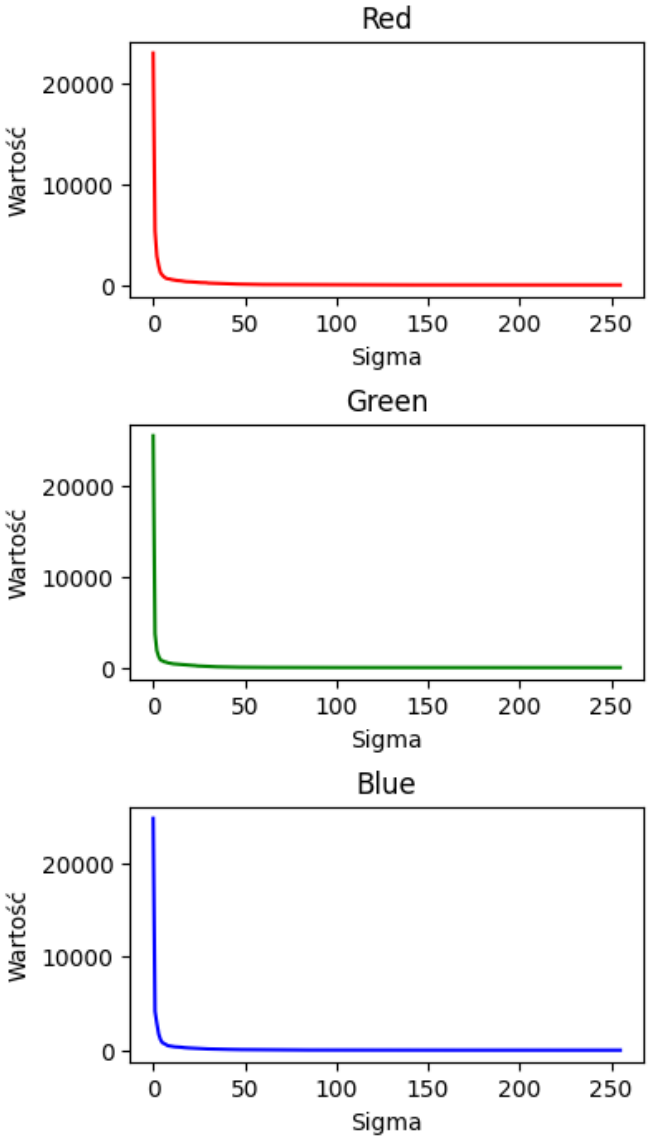
```python
def create_tree(A, tmin, tmax, smin, smax, r, epsilon):
    if tmax - tmin <= 1 and smax - smin <= 1:
        node = Node()
        node.rank = 1
        node.size = (1, 1)
        node.singular_values = [A[tmin, smin]]
        return node

    U, D, V = truncated_SVD(A[tmin:tmax, smin:smax], r + 1)

    if (D.shape[0] > r and D[r, r] < epsilon):
        new_r = sum(np.diag(D)[:r] > epsilon)
        return compress_matrix(A, tmin, tmax, smin, smax, U, D, V, new_r)

    node = Node()
    tmid = (tmin + tmax) // 2
    smid = (smin + smax) // 2

    node.children.append(create_tree(A, tmin, tmid, smin, smid, r, epsilon))
    node.children.append(create_tree(A, tmin, tmid, smid, smax, r, epsilon))
    node.children.append(create_tree(A, tmid, tmax, smin, smid, r, epsilon))
    node.children.append(create_tree(A, tmid, tmax, smid, smax, r, epsilon))
    return node
```

Rysunek 4: Tworzenie drzewa kompresji.

4 Wyniki algorytmów

4.1 Singular Value Decomposition (SVD)

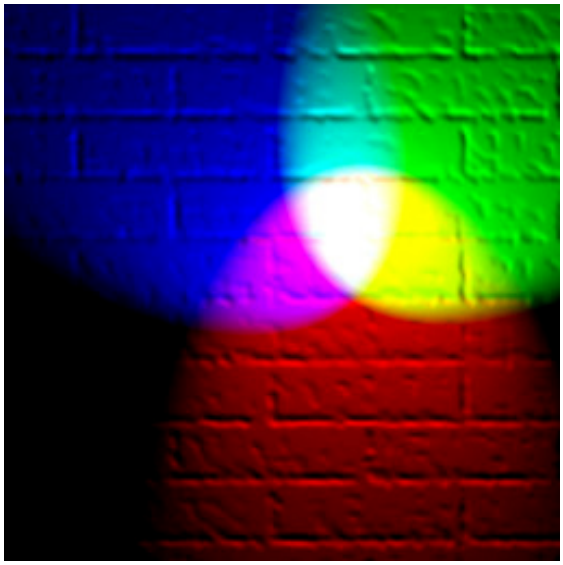
Wartości osobliwe dla każdego kanału RGB przykładowej bitmapy uzyskane z SVD.



Rysunek 5: Wartości osobliwe kanałów RGB.

4.2 Przykładowa bitmapa testowa

Nieskompresowana bitmapa w całości



Rysunek 6: Obraz testowy.

Rysunek 7: Podział na kanały RGB.