

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

```
1 class Node:
2     def __init__(self):
3         self.rank = 0
4         self.size = (0, 0)
5         self.singular_values = []
6         self.U = None
7         self.V = None
8         self.children = []
9         self.channels = []
10
11     def __repr__(self):
12         return f"Node(rank={self.rank}, size={self.size},\
13             children={len(self.children)})"
```

Rysunek 1: Struktura drzewa macierzy.

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

Rysunek 2: Częściowe SVD.

```

1  def compress_matrix(A, tmin, tmax, smin, smax, U, D, V, r):
2      node = Node()
3      if np.all(A[tmin:tmax, smin:smax] == 0):
4          node.rank = 0
5          node.size = (tmax - tmin, smax - smin)
6          return node
7
8      sigma = np.diag(D)
9      node.rank = r
10     node.size = (tmax - tmin, smax - smin)
11     node.singular_values = sigma[:r].tolist()
12     node.U = U[:, :r].tolist()
13     node.V = (D[:r, :r] @ V[:r, :]).tolist()
14     return node

```

Rysunek 3: Kompresja macierzy.

```

1  def create_tree(A, tmin, tmax, smin, smax, r, epsilon):
2      if tmax - tmin <= 1 and smax - smin <= 1:
3          node = Node()
4          node.rank = 1
5          node.size = (1, 1)
6          node.singular_values = [A[tmin, smin]]
7          return node
8
9      U, D, V = truncated_SVD(A[tmin:tmax, smin:smax], r + 1)
10
11     if (D.shape[0] > r and D[r, r] < epsilon):
12         new_r = sum(np.diag(D)[:r] > epsilon)
13         return compress_matrix(A, tmin, tmax, smin, smax, U, D, V, new_r)
14
15     node = Node()
16     tmid = (tmin + tmax) // 2
17     smid = (smin + smax) // 2
18
19     node.children.append(create_tree(A, tmin, tmid, smin, smid, r, epsilon))
20     node.children.append(create_tree(A, tmin, tmid, smid, smax, r, epsilon))
21     node.children.append(create_tree(A, tmid, tmax, smin, smid, r, epsilon))
22     node.children.append(create_tree(A, tmid, tmax, smid, smax, r, epsilon))
23     return node

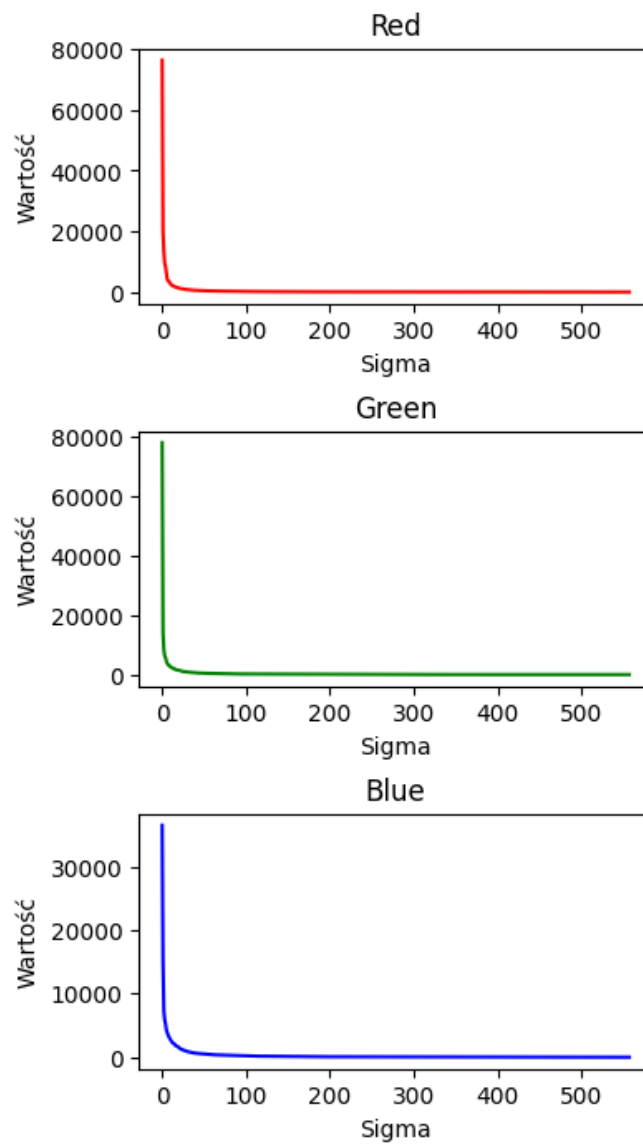
```

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.