Akademia Górniczo-Hutnicza

Wydział Informatyki

Algorytmy macierzowe

Ćwiczenie 3

Hierarchiczna kompresja macierzy

Jakub Szewczyk i Albert Arnautov

Kraków, 18 listopada 2025

Spis treści

1 Wstęp **3**

2 Pseudokod **3**
2.1 Algorytm tworzenia drzewa kompresji macierzy 3
2.2 Algorytm kompresji liścia drzewa macierzy kompresji 3
2.3 Algorytm rekonstrukcji macierzy 4

3 Ważne fragmenty kodu **4**
3.1 Algorytm kompresji macierzy 4

4 Wyniki kompresji wybranej bitmapy dla różnych parametrów **6**
4.1 Kompresja dla $r = 1$ i $\epsilon = \sigma_1$ 7
4.2 Kompresja dla $r = 1$ i $\epsilon = \sigma_{2^k}$ 8
4.3 Kompresja dla $r = 1$ i $\epsilon = \sigma_{2^k/2}$ 9
4.4 Kompresja dla $r = 4$ i $\epsilon = \sigma_1$ 10
4.5 Kompresja dla $r = 4$ i $\epsilon = \sigma_{2^k}$ 11
4.6 Kompresja dla $r = 4$ i $\epsilon = \sigma_{2^k/2}$ 12
4.7 Kompresja dla $r = 4$ i $\epsilon = \sigma_{80}$ 13

1 Wstęp

Celem ćwiczenia było zaimplementowanie metody hierarchicznej kompresji macierzy

2 Pseudokod

2.1 Algorytm tworzenia drzewa kompresji macierzy

Algorytm 1 Algorytm tworzenia drzewa kompresji macierzy

procedure CREATETREE($A, t_{\min}, t_{\max}, s_{\min}, s_{\max}, r, \epsilon$)
  $A_{\text{part}} \leftarrow A[t_{\min} : t_{\max} + 1, s_{\min} : s_{\max} + 1]$
  $[U, D, V] \leftarrow \text{SVD}(A_{\text{part}})$
  $D_\epsilon \leftarrow \{d \in D \mid d > \epsilon\}$
  if $|D_\epsilon| \leq r$ **then**
    return MatrixLeaf($A_{\text{part}}, (t_{\min}, t_{\max}, s_{\min}, s_{\max}), U, D, V, r$)
  else
    $t_{\text{mid}} \leftarrow \lfloor (t_{\min} + t_{\max})/2 \rfloor$
    $s_{\text{mid}} \leftarrow \lfloor (s_{\min} + s_{\max})/2 \rfloor$
    node $\leftarrow$ MatrixNode()
    node.children[0] $\leftarrow$ CreateTree($A, t_{\min}, t_{\text{mid}}, s_{\min}, s_{\text{mid}}, r, \epsilon$)
    node.children[1] $\leftarrow$ CreateTree($A, t_{\min}, t_{\text{mid}}, s_{\text{mid}} + 1, s_{\max}, r, \epsilon$)
    node.children[2] $\leftarrow$ CreateTree($A, t_{\text{mid}} + 1, t_{\max}, s_{\min}, s_{\text{mid}}, r, \epsilon$)
    node.children[3] $\leftarrow$ CreateTree($A, t_{\text{mid}} + 1, t_{\max}, s_{\text{mid}} + 1, s_{\max}, r, \epsilon$)
    return node
  end if
end procedure

2.2 Algorytm kompresji liścia drzewa macierzy kompresji

Algorytm 2 Algorytm kompresji liścia drzewa macierzy kompresji

procedure MATRIXLEAF(A, size$, U, D, V, r$)
  leaf $\leftarrow$ new leaf node
  leaf.size $\leftarrow$ size ▷ $(t_{\min}, t_{\max}, s_{\min}, s_{\max})$
  if $\|A\| < \epsilon$ **then** ▷ Check if matrix is approximately zero
    leaf.rank $\leftarrow 0$
    leaf.is_zero $\leftarrow$ True
    return leaf
  end if
  leaf.is_zero $\leftarrow$ False
  leaf.rank $\leftarrow r$
  leaf.singular_vals $\leftarrow D[1 : r]$
  leaf.U $\leftarrow U[:, 1 : r]$
  leaf.V $\leftarrow V[1 : r, :]$
  return leaf
end procedure

2.3 Algorytm rekonstrukcji macierzy

Algorytm 3 Algorytm rekonstrukcji macierzy

procedure ReconstructMatrix(tree, shape)
    $m, n \leftarrow$ shape
    result $\leftarrow$ ZerosMatrix(m, n)
    ReconstructNode(tree, result)
    return result
end procedure
procedure ReconstructNode(node, result)
    if node is instance of MatrixLeaf **then**
        $(t_{\min}, t_{\max}, s_{\min}, s_{\max}) \leftarrow$ node.size
        if node.rank > 0 **and not** node.is_zero **then**
            $A_{\text{recon}} \leftarrow$ node.U $\times$ diag(node.singular_vals) $\times$ node.V
            result$[t_{\min} : t_{\max} + 1, s_{\min} : s_{\max} + 1] \leftarrow A_{\text{recon}}$
        end if
    else
        for child $\in$ node.children **do**
            ReconstructNode(child, result)
        end for
    end if
end procedure

3 Ważne fragmenty kodu

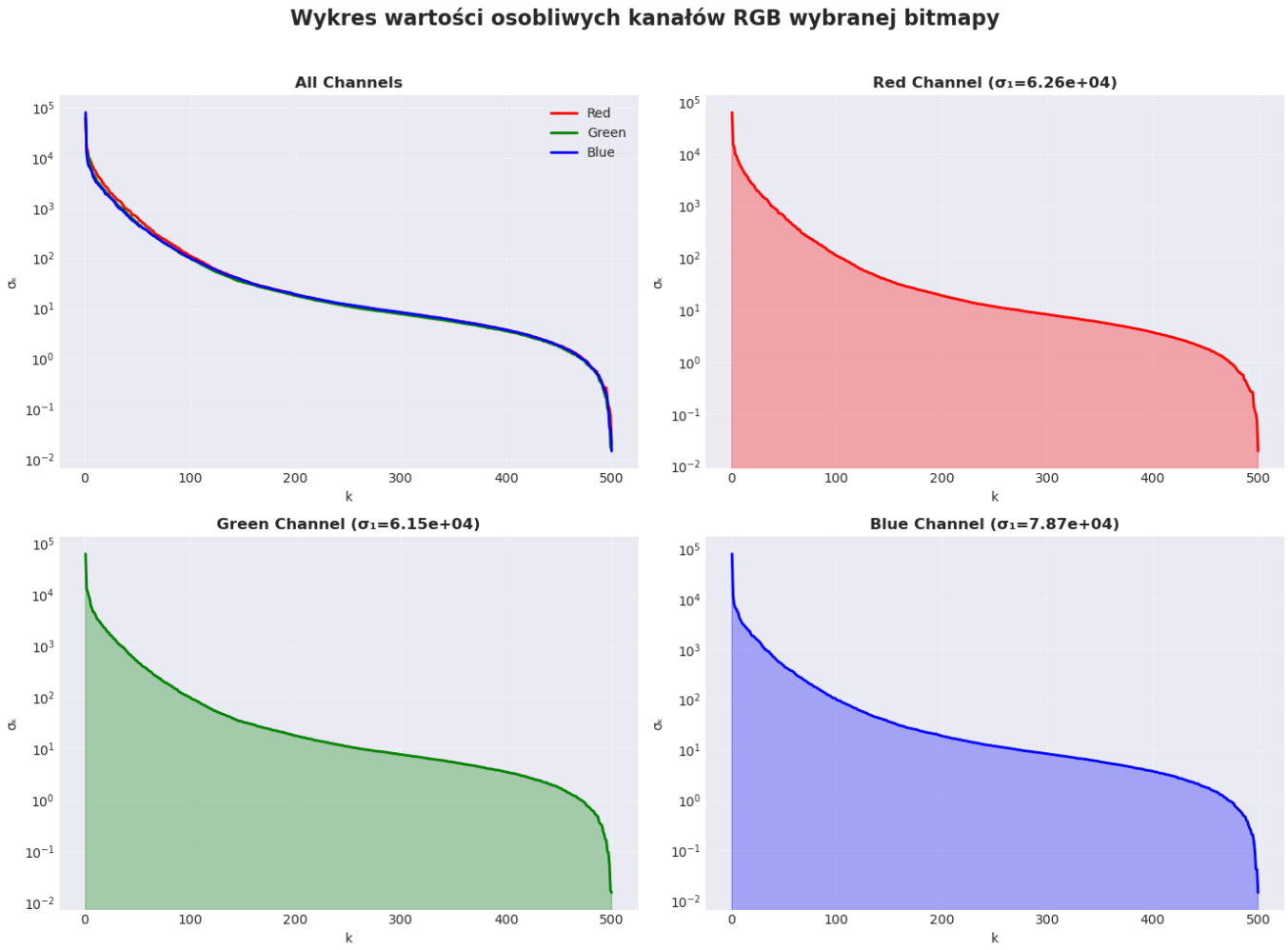
3.1 Algorytm kompresji macierzy

```python
class MatrixLeaf:
    def __init__(self, A, size, U, D, V, r):
        self.size = size
        if np.all(np.abs(A) < eps):
            self.rank = 0
            self.is_zero = True
            return
        self.is_zero = False

        rows, cols = A.shape
        max_possible_rank = min(rows, cols)
        available_singular_values = len(D)
        self.rank = min(r, max_possible_rank, available_singular_values)

        self.singular_vals = D[:self.rank]
        self.U = U[:, :self.rank]
        self.V = V[:self.rank, :]


class MatrixNode:
    def __init__(self):
        self.children = []
```

```python
def CreateTree(A, t_min, t_max, s_min, s_max, r, epsilon):
    A_part = A[t_min:t_max+1, s_min:s_max+1]
    U, D, V = svd(A_part)
    node: MatrixNode
    D_epsilon = D[np.where(D > epsilon)]
    if len(D_epsilon) <= r:
        node = MatrixLeaf(A_part, (t_min, t_max, s_min, s_max), U, D, V, r)
    else:
        node = MatrixNode()
        t_newmax = (t_min + t_max) // 2
        s_newmax = (s_min + s_max) // 2
        node.children.append(CreateTree(A, t_min, t_newmax, s_min, s_newmax, r, epsilon))
        node.children.append(CreateTree(A, t_min, t_newmax, s_newmax + 1, s_max, r, epsilon))
        node.children.append(CreateTree(A, t_newmax + 1, t_max, s_min, s_newmax, r, epsilon))
        node.children.append(CreateTree(A, t_newmax + 1, t_max, s_newmax + 1, s_max, r, epsilon))

    return node

def CompressMatrix(A, r, epsilon):
    return CreateTree(A, 0, A.shape[0] - 1, 0, A.shape[1] - 1, r, epsilon)

def ReconstructMatrix(tree_node, shape):
    result = np.zeros(shape)

    def reconstruct_node(node, result):
        if isinstance(node, MatrixLeaf):
            t_min, t_max, s_min, s_max = node.size
            if node.rank > 0 and not node.is_zero:
                leaf_recon = node.U @ np.diag(node.singular_vals) @ node.V
                result[t_min:t_max+1, s_min:s_max+1] = leaf_recon
        else:
            for child in node.children:
                reconstruct_node(child, result)

    reconstruct_node(tree_node, result)
    return result
```

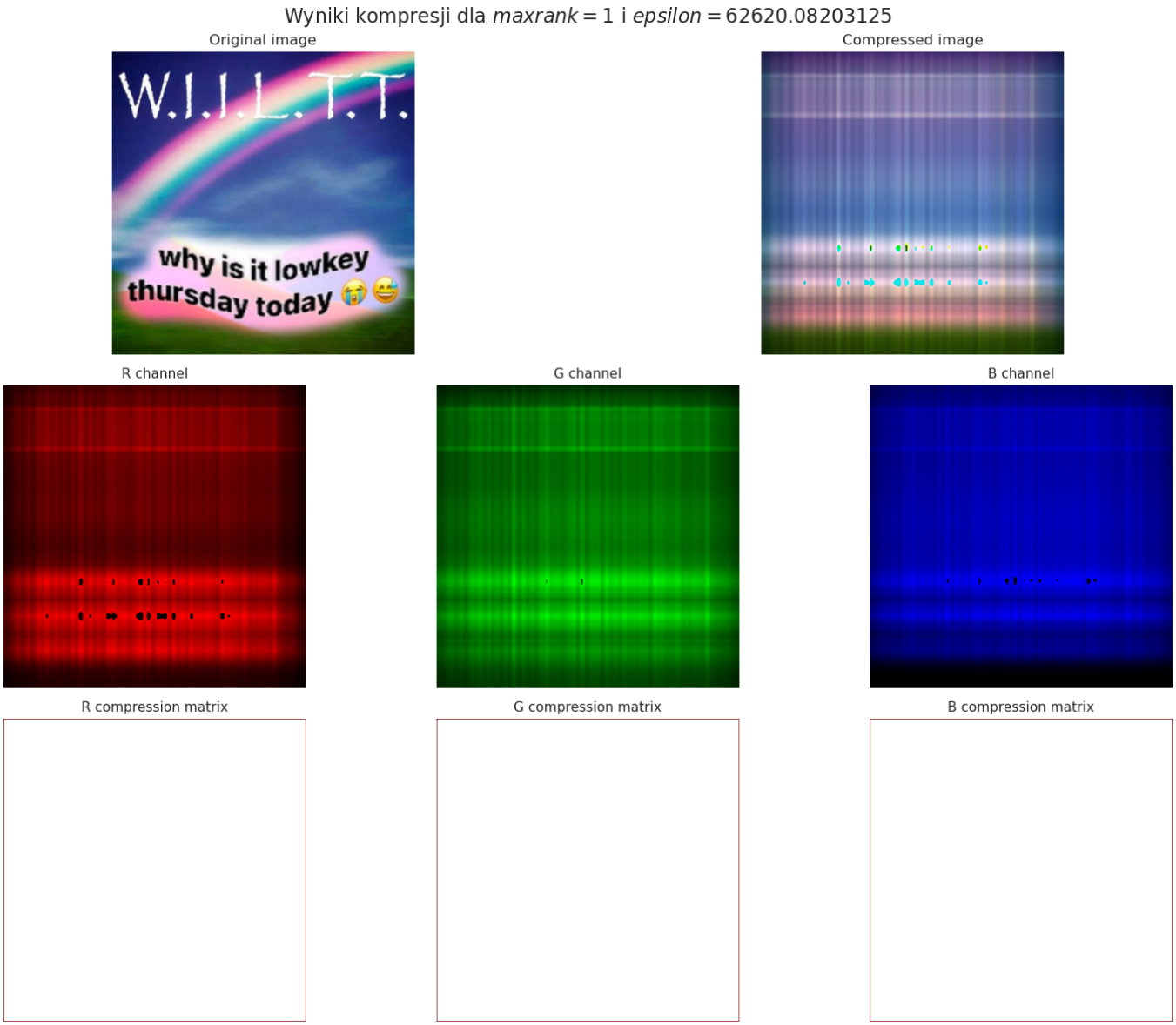
4 Wyniki kompresji wybranej bitmapy dla różnych parametrów

Rysunek 1: Wybrana bitmapa



Do wyboru epsilon używamy wartości osobliwych kanału czerwonego.

4.1 Kompresja dla $r = 1$ i $\epsilon = \sigma_1$



Otrzymane compression ratio: 249.75

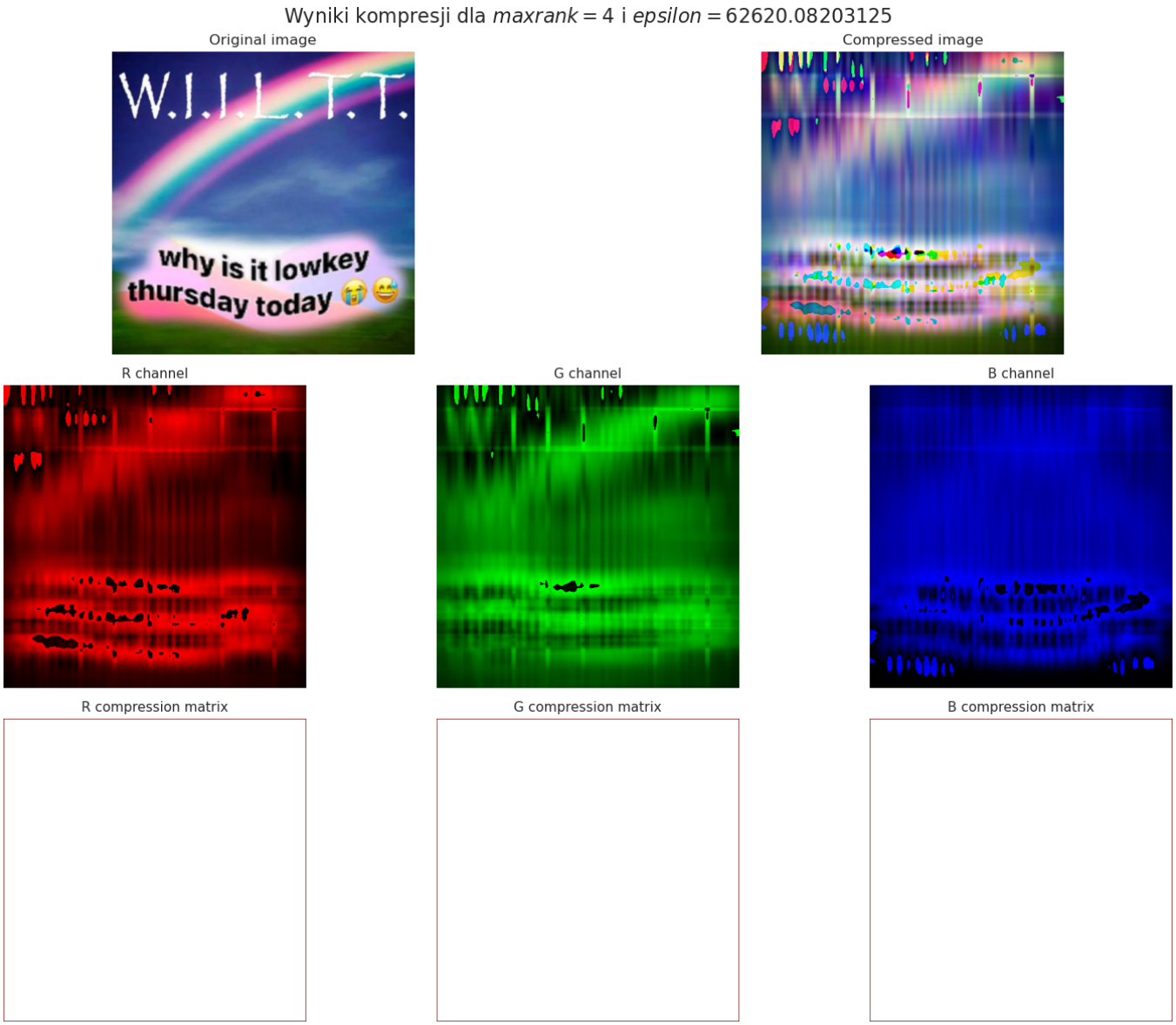
4.2 Kompresja dla $r = 1$ i $\epsilon = \sigma_{2^k}$

Otrzymane compression ratio: 0.4570

4.3 Kompresja dla $r = 1$ i $\epsilon = \sigma_{2^k/2}$

Otrzymane compression ratio: 2.546

4.4 Kompresja dla $r = 4$ i $\epsilon = \sigma_1$



Otrzymane compression ratio: 62.437

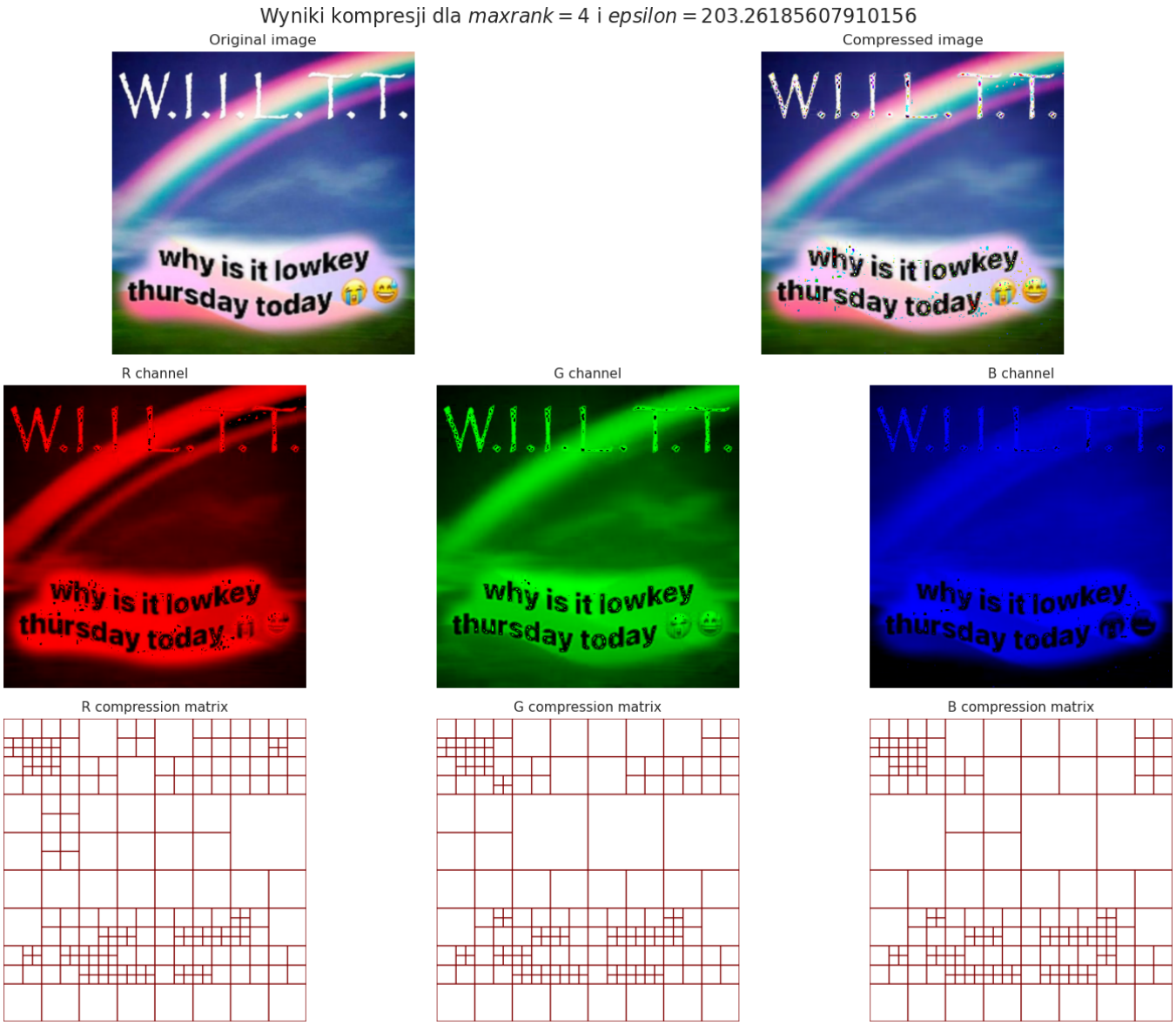
4.5 Kompresja dla $r = 4$ i $\epsilon = \sigma_{2^k}$

Otrzymane compression ratio: 0.4689

4.6 Kompresja dla $r = 4$ i $\epsilon = \sigma_{2^k/2}$

Otrzymane compression ratio: 1.8357

4.7 Kompresja dla $r = 4$ i $\epsilon = \sigma_{80}$



Otrzymane compression ratio: 5.405