

Algorytmy macierzowe

Laboratorium nr. 3

Hierarchiczna kompresja macierzy

Antoni Kucharski
Joachim Grys

22.11.2024

1 Wstęp

Celem laboratorium było zaimplementowanie algorytmu hierarchicznej kompresji macierzy. Algorytm ten polega na podziale macierzy na mniejsze fragmenty, które są następnie kompresowane. W przypadku, gdy dany fragment nie spełnia warunków kompresji, jest on dzielony na mniejsze fragmenty, które są kompresowane osobno. W ten sposób uzyskujemy hierarchiczną strukturę macierzy, która pozwala na efektywną kompresję danych. Sama kompresja podmacierzy polega na zastosowaniu algorytmu SVD (Singular Value Decomposition) do znalezienia osobliwych wartości macierzy.

Następnie należało wykonać praktyczną implementację algorytmu do przetestowania kompresji obrazu z różnymi parametrami wejściowymi i porównać wyniki.

2 Warunek dopuszczalności kompresji

Warunek dopuszczalności sprawdza, czy dany fragment macierzy może zostać skompresowany, czy należy wykonać rekurencyjny podział na cztery mniejsze podmacierze. Przykładowe warunki dopuszczalności to:

- maksymalny rozmiar podmacierzy
- maksymalna głębokość rekursji
- maksymalna liczba wartości osobliwych r
- maksymalna wartość osobliwa większa od zadanego progu ϵ

W implementacji wykorzystano dwa ostatnie warunki.

3 Implementacja

3.1 Pseudokod

Algorithm 1 Hierarchical Matrix Compression

```
1: function COMPRESSMATRIX(matrix, r,  $\epsilon$ , rmin, rmax, cmin, cmax)
2:   [U,  $\Sigma$ , V]  $\leftarrow$  randomized_svd(matrix, r)
3:   if rmax - rmin  $\leqslant$  r or  $\Sigma[r - 1] \leqslant \epsilon$  then
4:     return [U,  $\Sigma$ , V]
5:   else
6:     rows  $\leftarrow$  [rmin,  $\frac{r_{\min}+r_{\max}}{2}$ , rmax]
7:     cols  $\leftarrow$  [cmin,  $\frac{c_{\min}+c_{\max}}{2}$ , cmax]
8:     submatrices  $\leftarrow$   $\emptyset$ 
9:     for i  $\leftarrow 0$  to 1 do
10:      for j  $\leftarrow 0$  to 1 do
11:        submatrix  $\leftarrow$  matrix[rows[i] : rows[i + 1], cols[j] : cols[j + 1]]
12:        [Usub,  $\Sigma_{\text{sub}}$ , Vsub]  $\leftarrow$  CompressMatrix(submatrix, r,  $\epsilon$ , rows[i], rows[i + 1], cols[j], cols[j + 1])
13:        submatrices  $\leftarrow$  submatrices  $\cup$  {[Usub,  $\Sigma_{\text{sub}}$ , Vsub]}
14:      end for
15:    end for
16:    return submatrices
17:  end if
18: end function
```

3.2 Klasa CompressTree

```
1  class CompressTree:
2      def __init__(self, matrix, row_min, row_max, col_min, col_max):
3          self.matrix = matrix
4          self.row_min = row_min
5          self.row_max = row_max
6          self.col_min = col_min
7          self.col_max = col_max
8
9          self.leaf = False
10         self.childs = [[None, None], [None, None]]
11
12     def make_leaf(self, U, Sigma, V):
13         self.leaf = True
14         self.u = U
15         self.s = Sigma
16         self.v = V
```

Główna klasą w implementacji algorytmu jest klasa CompressTree. Funkcjonalność kompresji i dekompresji realizują poniższe metody:

3.3 Metoda create_tree

```
1  def create_tree(self, r, epsylon):
2      U, Sigma, V = randomized_svd(self.matrix[self.row_min:self.row_max, self.col_min:
3          → self.col_max], n_components=r)
4      if self.row_max == self.row_min + r:
5          self.make_leaf(U, Sigma, V)
6      elif Sigma[r - 1] <= epsylon:
7          self.make_leaf(U, Sigma, V)
8      else:
9          rows = [self.row_min, (self.row_min + self.row_max)//2, self.row_max]
10         cols = [self.col_min, (self.col_min + self.col_max)//2, self.col_max]
11         for i in range(2):
12             for j in range(2):
13                 self.childs[i][j] = CompressTree(self.matrix, rows[i], rows[i+1],
14                     → cols[j], cols[j+1])
15                 self.childs[i][j].create_tree(r, epsylon)
```

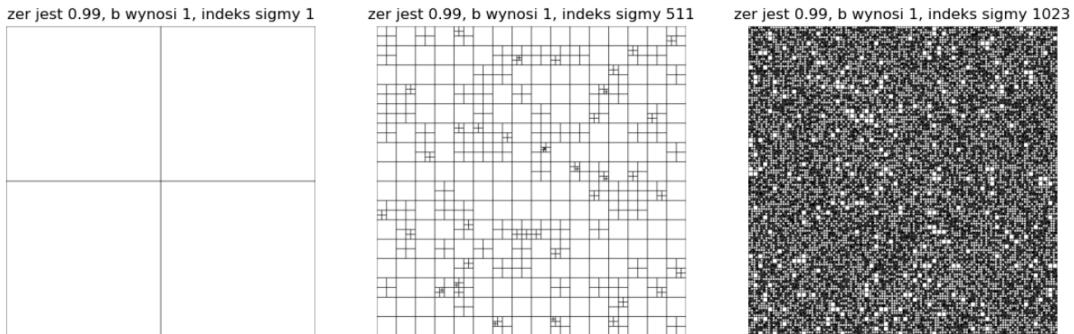
3.4 Metoda decompress

```
1  def decompress(self, dest_matrix):
2      if self.leaf:
3          r = len(self.s)
4          sigma = np.zeros((r,r))
5          np.fill_diagonal(sigma, self.s)
6          dest_matrix[self.row_min:self.row_max, self.col_min: self.col_max] = self.u @
7              → sigma @ self.v
8
9      else:
10         for i in range(2):
11             for j in range(2):
12                 self.childs[i][j].decompress(dest_matrix)
```

4 Rysowanie drzewa kompresji

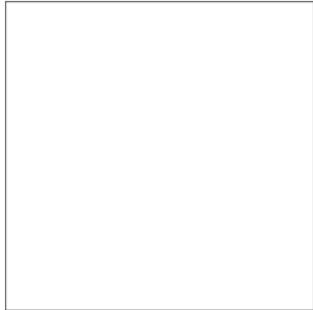
```
1 def draw_tree(root, axis=plt, title=''):
2     image = np.ones(root.matrix.shape)*255
3
4     Q = deque()
5     Q.append(root)
6     while Q:
7         v = Q.pop()
8         if v.leaf:
9             r = len(v.s)
10            gray = 125
11            image[v.row_min:v.row_max, v.col_min:v.col_min+r] = gray*np.ones((v.row_max
12                           - v.row_min, r))
13            image[v.row_min:v.row_min + r, v.col_min:v.col_max] = gray*np.ones((r ,
14                           v.col_max - v.col_min))
15            image[v.row_min, v.col_min:v.col_max] = np.zeros((1,v.col_max - v.col_min))
16            image[v.row_max-1, v.col_min:v.col_max] = np.zeros((1,v.col_max -
17                           v.col_min))
18            image[v.row_min:v.row_max,v.col_min] = np.zeros(v.row_max-v.row_min)
19            image[v.row_min:v.row_max,v.col_max-1] = np.zeros(v.row_max-v.row_min)
20
21     else:
22         for i in range(2):
23             for j in range(2):
24                 Q.append(v.childs[i][j])
25
26     axis.imshow(image,cmap = "gray", vmin=0, vmax=255)
```

5 Testy

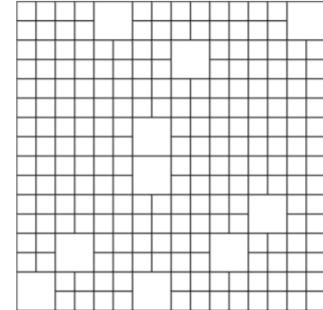


Rysunek 1: Test 1

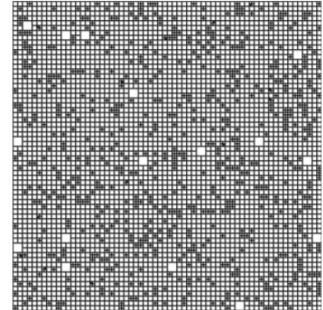
zer jest 0.99, b wynosi 4, indeks sigmy 1



zer jest 0.99, b wynosi 4, indeks sigmy 511

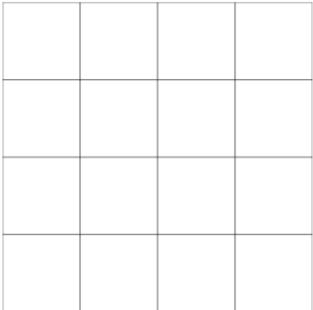


zer jest 0.99, b wynosi 4, indeks sigmy 1023

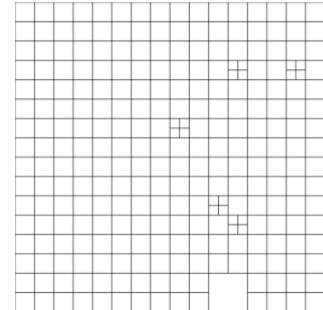


Rysunek 2: Test 2

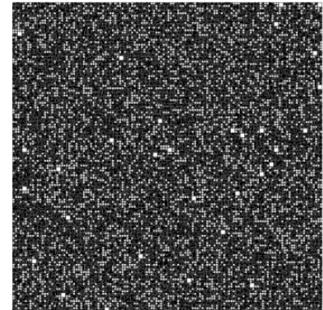
zer jest 0.98, b wynosi 1, indeks sigmy 1



zer jest 0.98, b wynosi 1, indeks sigmy 511

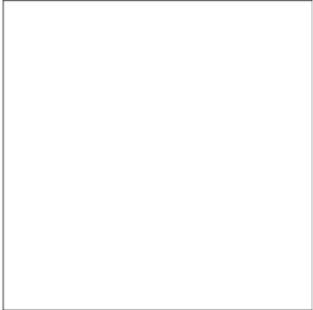


zer jest 0.98, b wynosi 1, indeks sigmy 1023

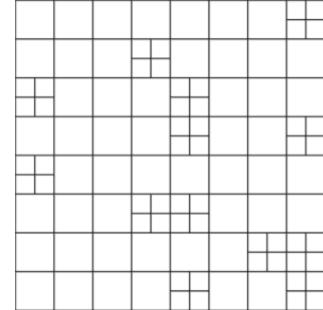


Rysunek 3: Test 3

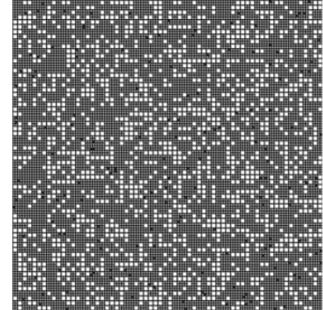
zer jest 0.98, b wynosi 4, indeks sigmy 1



zer jest 0.98, b wynosi 4, indeks sigmy 511

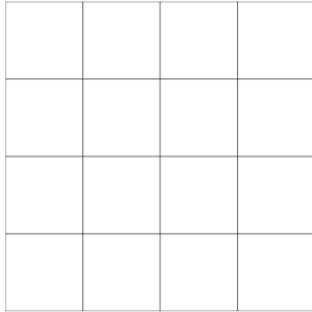


zer jest 0.98, b wynosi 4, indeks sigmy 1023

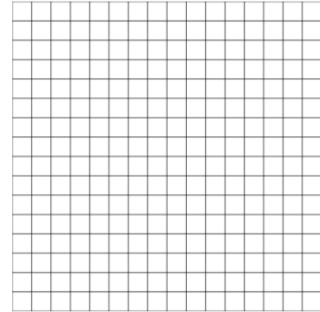


Rysunek 4: Test 4

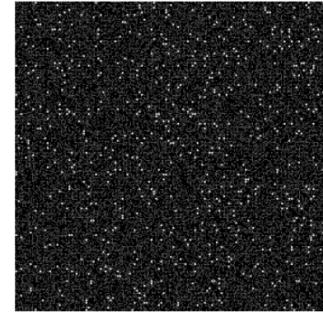
zer jest 0.95, b wynosi 1, indeks sigmy 1



zer jest 0.95, b wynosi 1, indeks sigmy 511

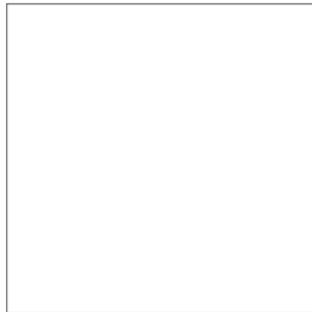


zer jest 0.95, b wynosi 1, indeks sigmy 1023

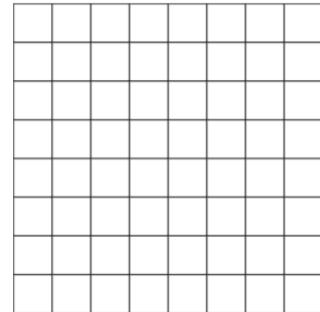


Rysunek 5: Test 5

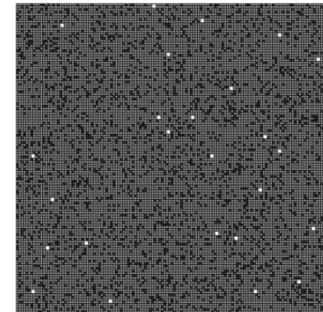
zer jest 0.95, b wynosi 4, indeks sigmy 1



zer jest 0.95, b wynosi 4, indeks sigmy 511

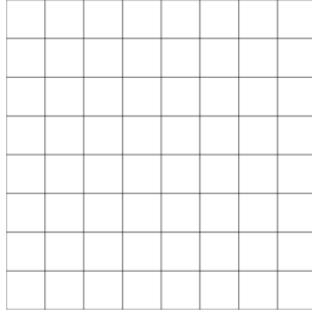


zer jest 0.95, b wynosi 4, indeks sigmy 1023

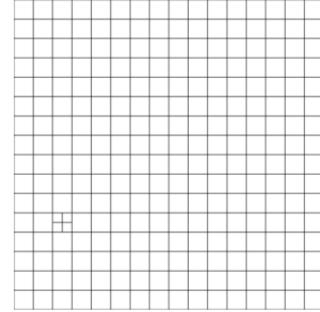


Rysunek 6: Test 6

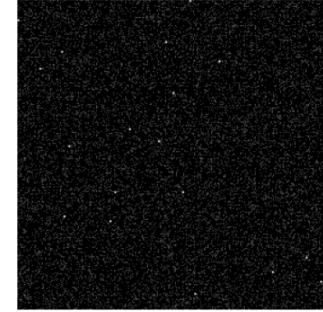
zer jest 0.9, b wynosi 1, indeks sigmy 1



zer jest 0.9, b wynosi 1, indeks sigmy 511

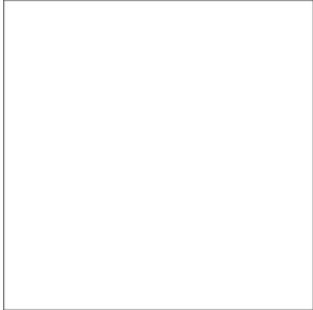


zer jest 0.9, b wynosi 1, indeks sigmy 1023

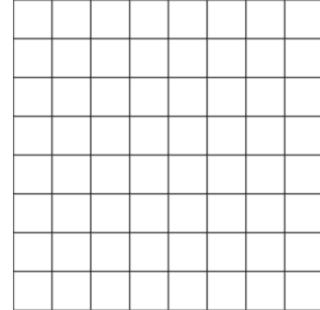


Rysunek 7: Test 7

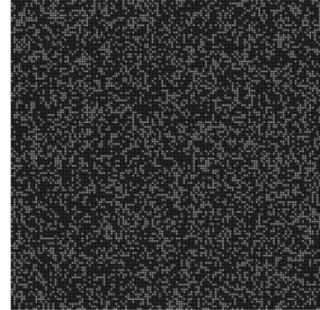
zer jest 0.9, b wynosi 4, indeks sigmy 1



zer jest 0.9, b wynosi 4, indeks sigmy 511

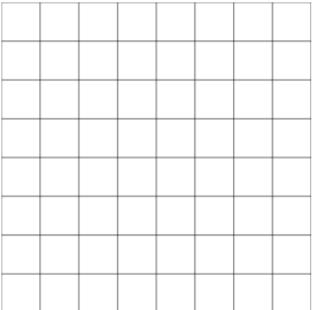


zer jest 0.9, b wynosi 4, indeks sigmy 1023

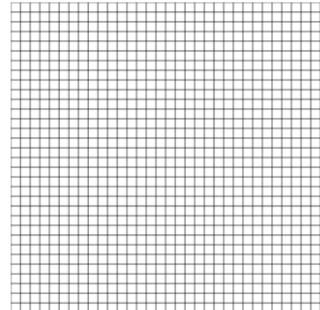


Rysunek 8: Test 8

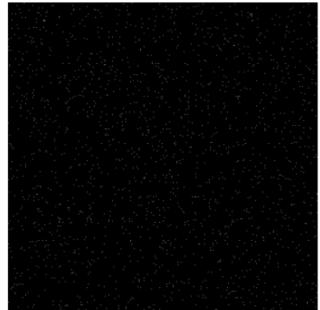
zer jest 0.8, b wynosi 1, indeks sigmy 1



zer jest 0.8, b wynosi 1, indeks sigmy 511

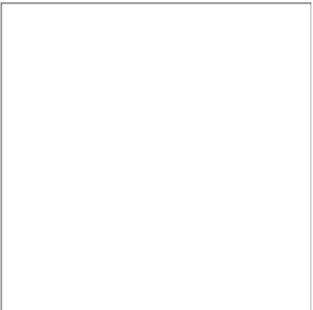


zer jest 0.8, b wynosi 1, indeks sigmy 1023

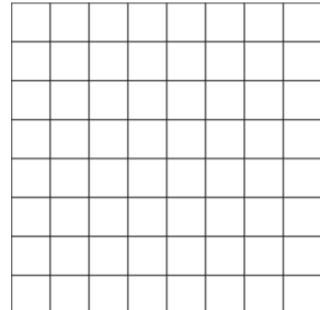


Rysunek 9: Test 9

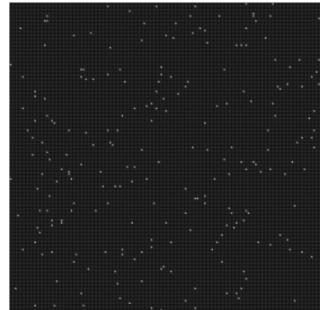
zer jest 0.8, b wynosi 4, indeks sigmy 1



zer jest 0.8, b wynosi 4, indeks sigmy 511

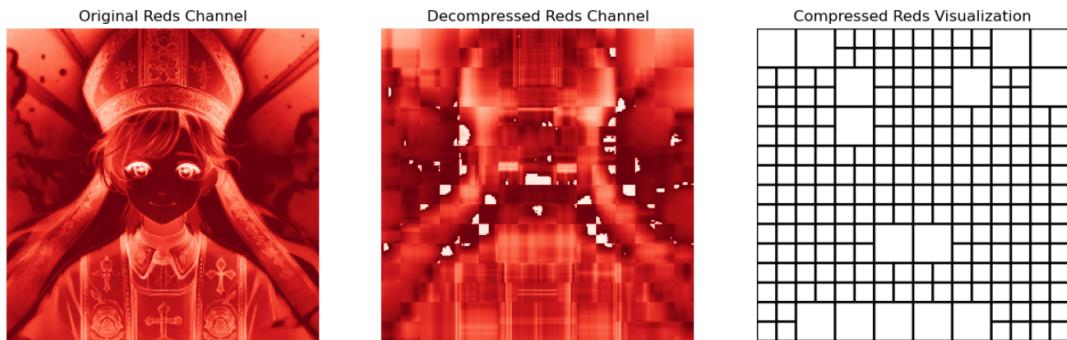


zer jest 0.8, b wynosi 4, indeks sigmy 1023

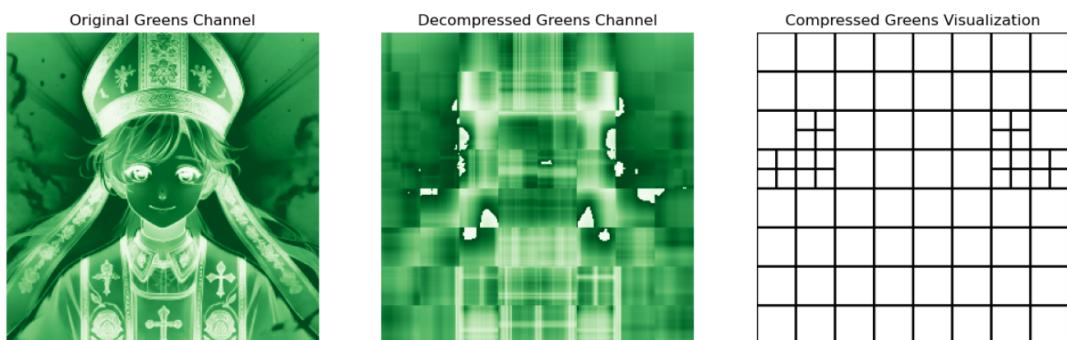


Rysunek 10: Test 10

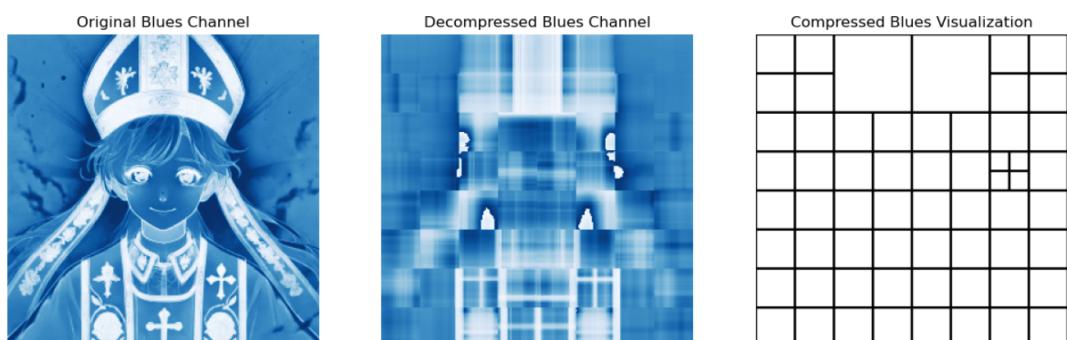
6 Kompresja obrazu



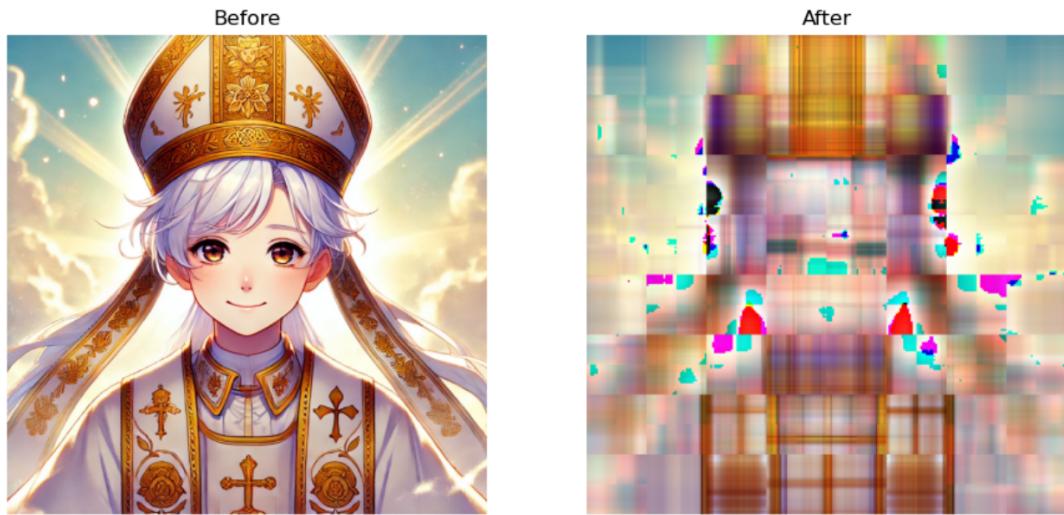
Rysunek 11: Test dla $r = 1, \epsilon = 1$



Rysunek 12: Test dla $r = 1, \epsilon = 1$



Rysunek 13: Test dla $r = 1, \epsilon = 1$



Rysunek 14: Wynik dla $r = 1, \epsilon = 1$



Rysunek 15: Test dla $r = 1, \epsilon = \sigma_{2^k}$



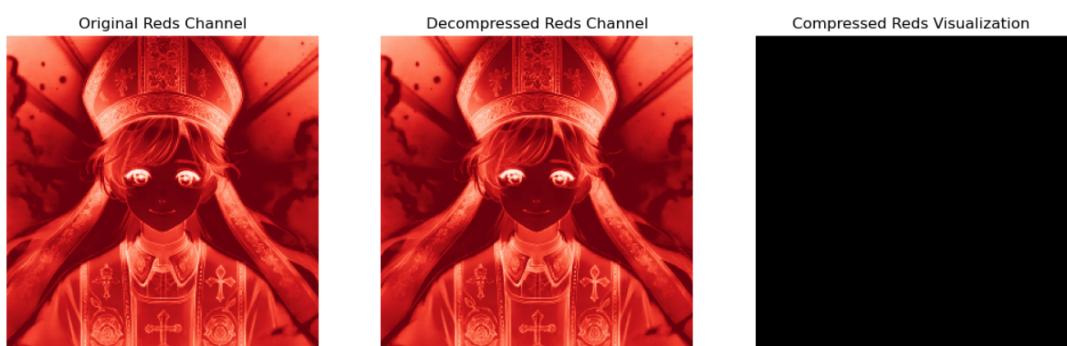
Rysunek 16: Test dla $r = 1, \epsilon = \sigma_{2^k}$



Rysunek 17: Test dla $r = 1, \epsilon = \sigma_{2^k}$



Rysunek 18: Wynik dla $r = 1, \epsilon = \sigma_{2^k}$



Rysunek 19: Test dla $r = 1, \epsilon = \sigma_{2^{k-1}}$



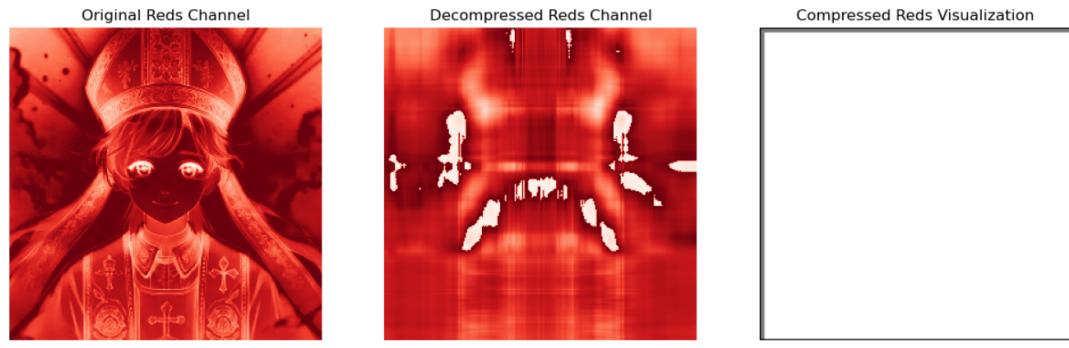
Rysunek 20: Test dla $r = 1, \epsilon = \sigma_{2^{k-1}}$



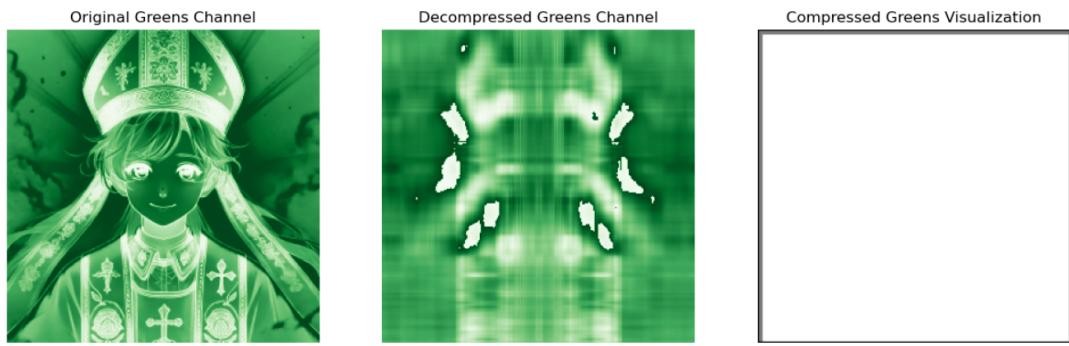
Rysunek 21: Test dla $r = 1, \epsilon = \sigma_{2^{k-1}}$



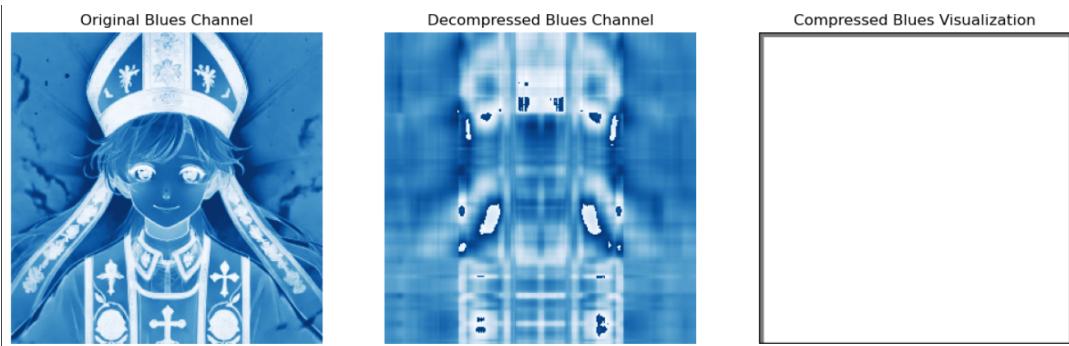
Rysunek 22: Wynik dla $r = 1, \epsilon = \sigma_{2^{k-1}}$



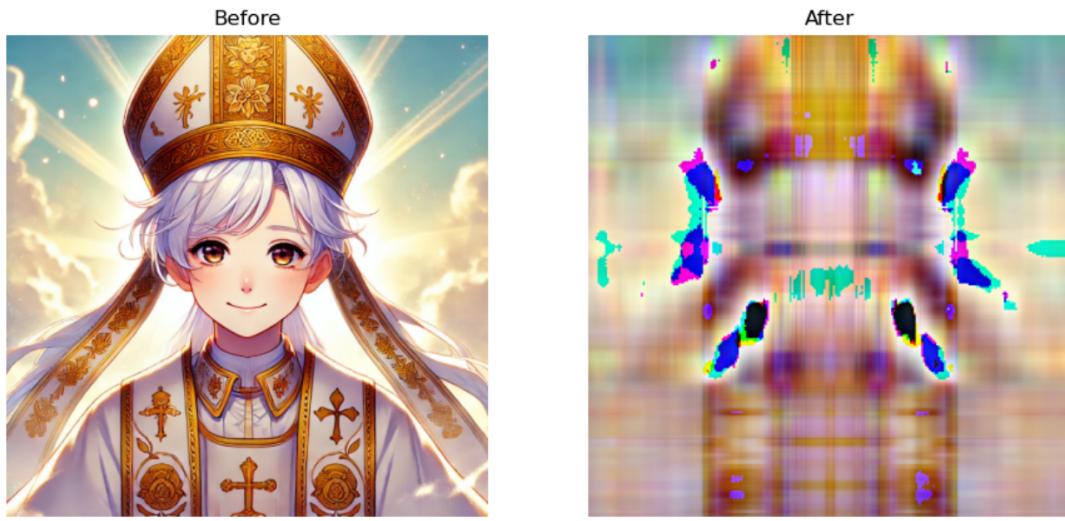
Rysunek 23: Test dla $r = 2, \epsilon = 1$



Rysunek 24: Test dla $r = 2, \epsilon = 1$



Rysunek 25: Test dla $r = 2, \epsilon = 1$



Rysunek 26: Wynik dla $r = 2, \epsilon = 1$



Rysunek 27: Test dla $r = 2, \epsilon = \sigma_{2^k}$



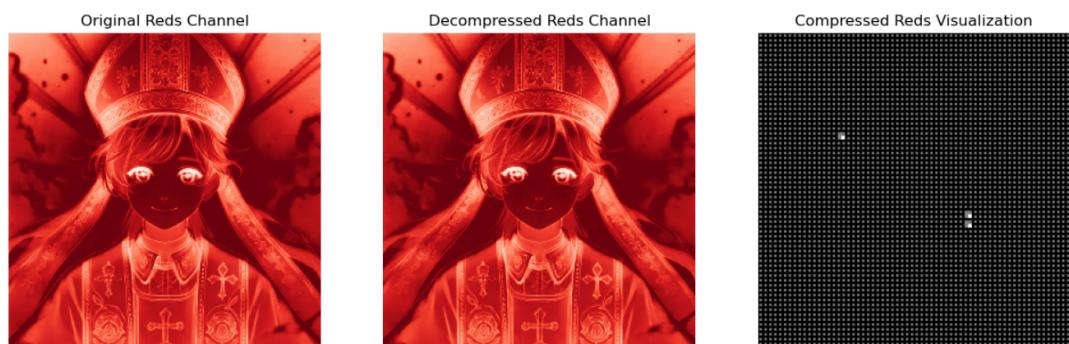
Rysunek 28: Test dla $r = 2, \epsilon = \sigma_{2^k}$



Rysunek 29: Test dla $r = 2, \epsilon = \sigma_{2^k}$



Rysunek 30: Wynik dla $r = 2, \epsilon = \sigma_{2^k}$



Rysunek 31: Test dla $r = 2, \epsilon = \sigma_{2^{k-1}}$



Rysunek 32: Test dla $r = 2$, $\epsilon = \sigma_{2^{k-1}}$



Rysunek 33: Test dla $r = 2$, $\epsilon = \sigma_{2^{k-1}}$



Rysunek 34: Wynik dla $r = 2$, $\epsilon = \sigma_{2^{k-1}}$

7 Podsumowanie

Z praktycznej części zadania wynika, że najmniej stratna kompresja uzyskiwana jest dla niższych maksymalnych rzędów macierzy oraz dla wyższych progów ϵ . Natomiast jest to kosztem większej ilości obliczeń i niewielkiem obniżeniu wielkości obrazu.