

Algorytmy Macierzowe laboratorium 3.

Hierarchiczna kompresja macierzy

Antoni Kucharski, Joachim Grys

3.12.2024

1 Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z algorytmem kompresji macierzy, który działa na zasadzie budowania drzewa podmacierzy — każdy węzeł reprezentuje pewną podmacierz (lub całą macierz) a dzieci powstają w wyniku jej podziału na daną liczbę kawałków (najczęściej na 4 podmacierze).

Następnie należało wykorzystać algorytm do kompresji wybranego obrazu kolorowego kompresując osobno kanał czerwieni, zieleni i niebieskiego wybranej bitmapy.

2 CompressTree

2.1 Reprezentacja drzewa

1. Korzeń drzewa to cała macierz
2. Korzeń posiada 4 podmacierze — 4 dzieci
3. Każdy kolejny węzeł posiada 4 dzieci
4. Liście są skompresowane za pomocą SVD
5. Decyzja dotycząca podziału na podmacierze czy kompresji (liść) zależy od wartości *warunku dopuszczalności*

2.2 Warunek dopuszczalności

Warunku dopuszczalności nie trzeba definiować jednoznacznie. Może to być:

- maksymalny dopuszczalny rozmiar podmacierzy (jeśli macierz jest większa niż maksymalny dopuszczalny rozmiar, to dzielimy ją dalej na 4 podmacierze),
- maksymalna głębokość drzewa (jeśli aktualna głębokość drzewa jest mniejsza niż dopuszczalny rozmiar, to dzielimy dalej)
- jeśli liczba wartości osobliwych macierzy jest większa niż zadane r to dzielimy dalej
- jeśli największa wartość osobliwa macierzy jest większa niż zadane ϵ to dzielimy dalej

2.3 Kod

```
from sklearn.utils.extmath import randomized_svd
import numpy as np
from numpy.linalg import svd

class CompressTree:
    def __init__(self, matrix, row_min, row_max, col_min, col_max):
        self.matrix = matrix
        self.row_min = row_min
        self.row_max = row_max
        self.col_min = col_min
        self.col_max = col_max

        self.leaf = False
        ...
        UL | UR
        -----
        DL | DR
        ...
    self.childs = [[None, None], [None, None]]

    def make_leaf(self, U, Sigma, V):
        self.leaf = True
        self.u = U
        self.s = Sigma
        self.v = V
```

```

def create_tree(self, r, epsylon):

    U, Sigma, V = randomized_svd(self.matrix[self.row_min:self.row_max, self.col_min: self.col_max], n_components=r)
    if self.row_max == self.row_min + r:
        self.make_leaf(U, Sigma, V)
    elif Sigma[r-1] <= epsylon:
        self.make_leaf(U, Sigma, V)
    else:
        rows = [self.row_min, (self.row_min + self.row_max)//2, self.row_max]
        cols = [self.col_min, (self.col_min + self.col_max)//2, self.col_max]
        for i in range(2):
            for j in range(2):
                self.childs[i][j] = CompressTree(self.matrix, rows[i], rows[i+1], cols[j], cols[j+1])
                self.childs[i][j].create_tree(r, epsylon)

def decompress(self, dest_matrix):
    if self.leaf:
        r = len(self.s)
        sigma = np.zeros((r,r))
        np.fill_diagonal(sigma, self.s)
        dest_matrix[self.row_min:self.row_max, self.col_min: self.col_max] =
            self.u @ sigma @ self.v

    else:
        for i in range(2):
            for j in range(2):
                self.childs[i][j].decompress(dest_matrix)

def compare(self, new_matrix):
    return np.sum(np.square(self.matrix - new_matrix))

```

3 Rysowanie Drzewa

```
def draw_tree(root, title=''):
    image = np.ones(root.matrix.shape)*255

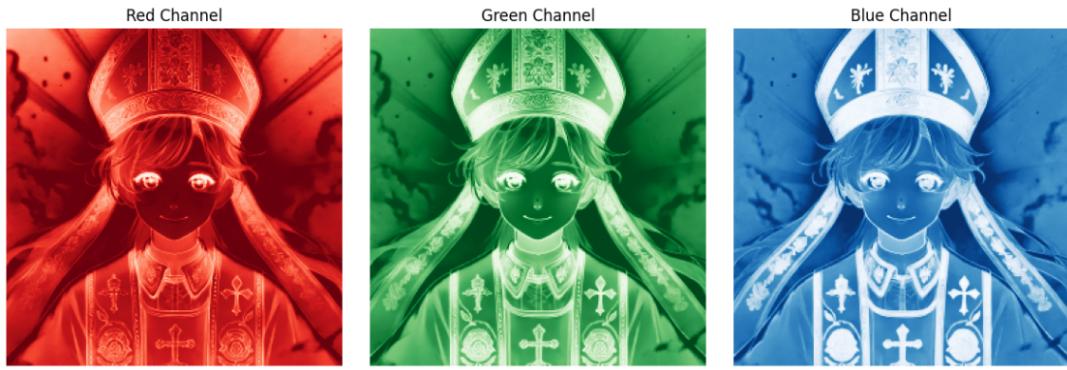
    Q = deque()
    Q.append(root)
    while Q:
        v = Q.pop()
        if v.leaf:
            r = len(v.s)
            gray = 125
            image[v.row_min:v.row_max, v.col_min:v.col_min+r] = gray*np.ones((v.
                row_max - v.row_min, r))#np.zeros((v.row_max - v.row_min, min(r,v
                .col_max - v.col_min)))
            image[v.row_min:v.row_min + r, v.col_min:v.col_max] =gray*np.ones((r
                , v.col_max - v.col_min)) #np.zeros((min(r,v.row_max - v.row_min
                ) , v.col_max - v.col_min))
            image[v.row_min, v.col_min:v.col_max] = np.zeros((1,v.col_max - v.
                col_min))
            image[v.row_max-1, v.col_min:v.col_max] = np.zeros((1,v.col_max - v.
                col_min))
            image[v.row_min:v.row_max,v.col_min] = np.zeros(v.row_max-v.row_min)
            image[v.row_min:v.row_max,v.col_max-1] = np.zeros(v.row_max-v.
                row_min)
        else:
            for i in range(2):
                for j in range(2):
                    Q.append(v.childs[i][j])

    plt.imshow(image,cmap = "gray", vmin=0, vmax=255)
    plt.title(title)
    plt.show()
```

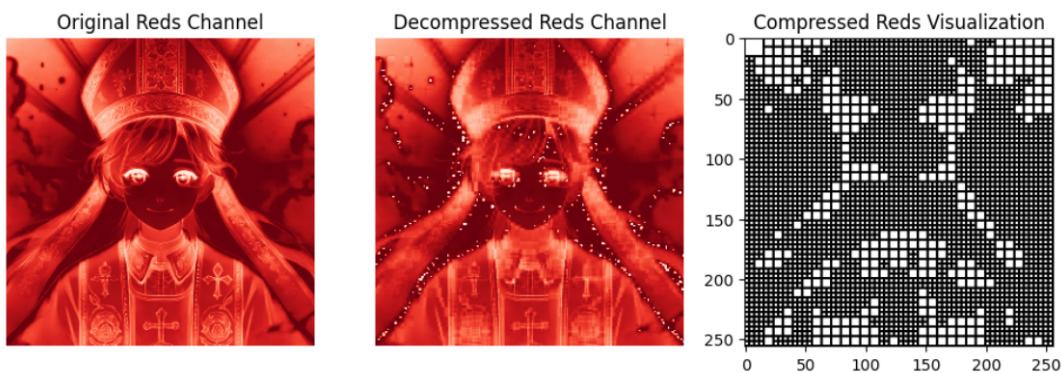
4 Wyniki



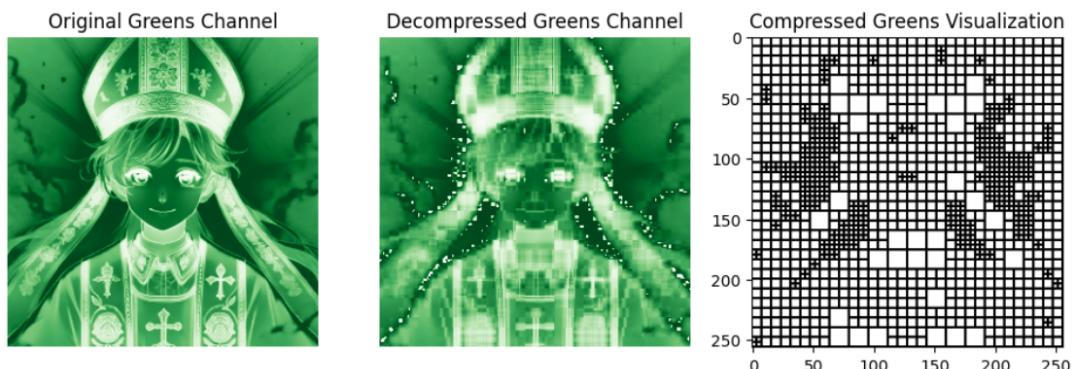
Rysunek 1: Obraz przed jakąkolwiek operacją



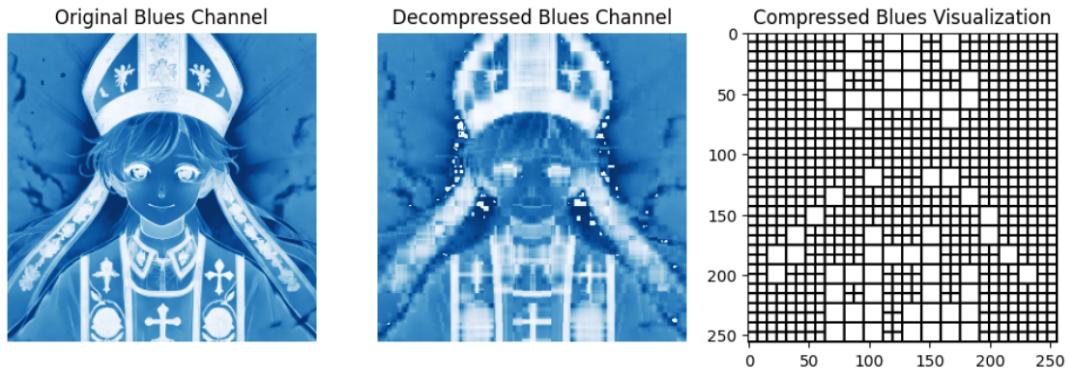
Rysunek 2: Rozkład obrazu na składowe



Rysunek 3: Kompresja kanału czerwieni



Rysunek 4: Kompresja kanału zieleni



Rysunek 5: Kompresja kanału niebieskiego



Rysunek 6: Rysunek przed i po działaniu kompresji

```
Compression results (number of compressed bytes):

red
Before: 0, After: 175

green
Before: 7, After: 159

blue
Before: 9, After: 126
```

Rysunek 7: Wyniki kompresji dla poszczególnych kanałów