Penerapan Transformasi Kosinus Diskrit Dalam Kompresi Gambar JPEG

Muhammad Dicky Isra and 13523075[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523075@std.stei.itb.ac.id, [2]mdikiisra98@gmail.com

***Abstrak*—JPEG (Joint Photographic Experts Group) atau JPG merupakan salah satu format gambar yang umum ditemukan sehari-hari. Makalah ini bertujuan untuk mempelajari dan memahami bagaimana cara kompresi JPEG dengan transformasi kosinus diskrit(DCT). Makalah ini akan menunjukkan bagaimana cara atau proses atau algoritma dalam melakukan DCT pada gambar JPEG.**

***Kata Kunci*— DCT, JPEG, JPG.**

I. PENDAHULUAN

Dalam kehidupan sehari hari. Kita banyak berinteraksi dengan gambar-gambar dengan berbagai macam format. Salah satu format gambar yang sering kita jumpai adalah format gambar JPEG atau JPG.

JPEG dapat dipandang sebagai format gambar ataupun jenis kompresi. Format gambar ini adalah formatyang *lossy* yang artinya gambar yang disimpan bukanlah gambar yang sesungguhnya yang, terdapat pengurangan kualitas gambar yang disebabkan oleh kompresinya.

Penggunaan DCT bersama dengan kuantisasi bermanfaat untuk menghilangkan detail yang tidak terlalu kita perhatikan. Kita dapat menentukan seberapa banyak kita ingin mempertahankan kualitas gambar asli atau seberapa banyak kita ingin mengurangi kualitas gambar melalui matriks kuantisasi. Kita dapat melakukan kompresi dengan usaha untuk mengurangi ukuran gambar sebesar mungkin dengan berusaha untuk mempertahankan kualitas gambar sebanyak/sebagus mungkin.

II. LANDASAN TEORI

A. Matriks

Matriks adalah struktur data yang dibentuk atas baris dan kolom yang biasa dinotasikan dengan n x m. Dalam hal ini, n berarti jumlah baris dan m merupakan jumlah kolom.

Contoh matriks A berukuran 3 x 4:

$$A = \begin{bmatrix} 3 & 2 & 4 & 6 \\ 7 & 0 & 8 & -12 \\ 13 & 11 & -1 & 0 \end{bmatrix}$$

Gambar 1. Contoh matriks 3x4

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf

B. Transformasi Kosinus Diskrit

Transformasi Kosinus Diskrit atau yang lebih dikenal sebagai *Discrite Cosine Transform* (DCT) merupakan metode pemrosesan sinyal yang berfokus pada nilai riil, tidak seperti metode transformasi fourier yang juga menghitung bagian kompleksnya.

Terdapat beberapa Formula pada DCT.

1. 1D DCT

Seperti namanya, DCT ini berfokus hanya pada 1 dimensi. Akan diformulakan menjadi

$$G(m) = \sqrt{\tfrac{2}{M}} \cdot \sum_{u=0}^{M-1} g(u) \cdot c_m \cdot \cos\left(\pi \tfrac{m(2u+1)}{2M}\right), \qquad (20.1)$$

for $0 \le m < M$, and the *inverse* transform is

$$g(u) = \sqrt{\tfrac{2}{M}} \cdot \sum_{m=0}^{M-1} G(m) \cdot c_m \cdot \cos\left(\pi \tfrac{m(2u+1)}{2M}\right), \qquad (20.2)$$

for $0 \le u < M$, with

$$c_m = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } m = 0, \\ 1 & \text{otherwise.} \end{cases} \qquad (20.3)$$

C dan D merupakan fungsi basis yang formulasikan sebagai berikut

$$\text{DFT: } C_m^M(u) = \cos\left(2\pi \tfrac{mu}{M}\right), \qquad (20.4)$$

$$\text{DCT: } D_m^M(u) = \cos\left(\pi \tfrac{m(2u+1)}{2M}\right) = \cos\left(2\pi \tfrac{m(u+0.5)}{2M}\right), \qquad (20.5)$$

2. 2D DCT

Selanjutnya, 2D DCT yang memproses data spasial 2 dimensi. Yang diformulasikan sebagai berikut.

$$G(m,n) = \frac{2}{\sqrt{MN}} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \left[g(u,v) \cdot c_m \cos\left(\frac{\pi(2u+1)m}{2M}\right) \cdot c_n \cos\left(\frac{\pi(2v+1)n}{2N}\right) \right] \quad (20.6)$$

$$= \frac{2 \cdot c_m \cdot c_n}{\sqrt{MN}} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \cdot \left[g(u,v) \cdot D_m^M(u) \cdot D_n^N(v) \right], \quad (20.7)$$

for $0 \le m < M$, $0 \le n < N$, and the inverse transform

$$g(u,v) = \frac{2}{\sqrt{MN}} \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left[G(m,n) \cdot c_m \cos\left(\frac{\pi(2u+1)m}{2M}\right) \cdot c_n \cos\left(\frac{\pi(2v+1)n}{2N}\right) \right] \quad (20.8)$$

$$= \frac{2}{\sqrt{MN}} \cdot \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \left[G(m,n) \cdot c_m \cdot D_m^M(u) \cdot c_n \cdot D_n^N(v) \right], \quad (20.9)$$

for $0 \le u < M$, $0 \le v < N$. The coefficients c_m and c_n in Eqns. (20.7) and (20.9) are the same as in the 1D case (Eqn. (20.3)). Notice

Kita akan menggunakan DCT 2D pada makalah ini sebab gambar adalah data spasial 2 dimensi.

III. PEMBAHASAN

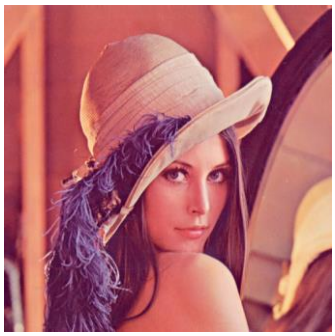
A. Batasan Masalah

Format atau kompresi JPEG merupakan serangkaian proses yang memerlukan beberapa tahapan proses. Menurut [2] Proses tersebut dilakukan dengan tahapan-tahapan sebagai berikut.

1. Konversi warna menjadi bentuk YCbCr
2. Pengambilan sampel ke bawah warna (*down sampling)*
3. Transformasi kosinus diskrit(DCT)
4. Kuantisasi
5. Kompresi tanpa kehilangan (*lossless compression*)

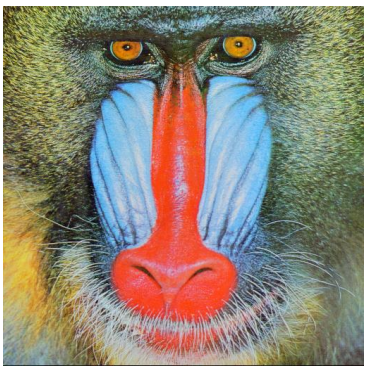
Untuk mempersempit cakupan dari pembahasan makalah ini. Makalah ini akan memfokuskan pembahasan pada proses tahap 3 dan tahap 4, yaitu transformasi kosinus diskrit dan Kuantisasi.

Dalam pembahasan ini, akan digunakan beberapa gambar sebagai bahan percobaan. Gambar tersebut adalah sebagai berikut.



Gambar 2. Lena (512 x 512)

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Koleksi/Citra%20Uji/Lena512warna.bmp



Gambar 3. Baboon (512 x 512)

Sumber:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Koleksi/Citra%20Uji/baboon24.bmp

Gambar 4. Peppers (512 x 512)

Sumber:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Koleksi/Citra%20Uji/peppers512warna.bmp

B. Persiapan Program

Pertama, kita akan mempersiapkan program dengan memasukkan *library* yang diperlukan dan memasukkan gambar yang dibutuhkan dan yang akan digunakan nantinya.

Pembuat kode membuat fungsi fungsi untuk membantu dan membuat program lebih modular.

```python
def zigzag(matrix: np.ndarray) -> np.ndarray:
    """
    computes the zigzag of a quantized block
    :param numpy.ndarray matrix: quantized matrix
    :returns: zigzag vectors in an array
    """
    # initializing the variables
    h = 0
    v = 0
    v_min = 0
    h_min = 0
    v_max = matrix.shape[0]
    h_max = matrix.shape[1]
    i = 0
    output = np.zeros((v_max * h_max))

    while (v < v_max) and (h < h_max):
        if ((h + v) % 2) == 0:  # going up
            if v == v_min:
                output[i] = matrix[v, h]  # first line
                if h == h_max:
                    v = v + 1
                else:
                    h = h + 1
                i = i + 1
            elif (h == h_max - 1) and (v < v_max):  # last column
                output[i] = matrix[v, h]
                v = v + 1
                i = i + 1
            elif (v > v_min) and (h < h_max - 1):  # all other cases
                output[i] = matrix[v, h]
                v = v - 1
                h = h + 1
                i = i + 1
        else:  # going down
            if (v == v_max - 1) and (h <= h_max - 1):  # last line
                output[i] = matrix[v, h]
                h = h + 1
                i = i + 1
            elif h == h_min:  # first column
                output[i] = matrix[v, h]
                if v == v_max - 1:
                    h = h + 1
                else:
                    v = v + 1
                i = i + 1
            elif (v < v_max - 1) and (h > h_min):  # all other cases
                output[i] = matrix[v, h]
                v = v + 1
                h = h - 1
                i = i + 1
        if (v == v_max - 1) and (h == h_max - 1):  # bottom right element
            output[i] = matrix[v, h]
            break
    return output
```

Fungsi zigzag yang digunakan untuk menulis kembali matriks hasil matriks dct dan kuantisasi dalam bentuk linear yan disusun dalam pola zigzag.

```python
def trim(array: np.ndarray) -> np.ndarray:
    """
    in case the trim_zeros function returns an empty array, add a zero to the array to use as the DC component
    :param numpy.ndarray array: array to be trimmed
    :return numpy.ndarray:
    """
    trimmed = np.trim_zeros(array, 'b')
    if len(trimmed) == 0:
        trimmed = np.zeros(1)
    return trimmed
```

Fungsi trim untuk membantu komponen DC

```python
def run_length_encoding(array: np.ndarray) -> list:
    """
    finds the intermediary stream representing the zigzags
    format for DC components is <size><amplitude>
    format for AC components is <run_length, size> <Amplitude of non-zero>
    :param numpy.ndarray array: zigzag vectors in array
    :returns: run length encoded values as an array of tuples
    """
    encoded = list()
    run_length = 0
    eob = ("EOB",)

    for i in range(len(array)):
        for j in range(len(array[i])):
            trimmed = trim(array[i])
            if j == len(trimmed):
                encoded.append(eob)  # EOB
                break
            if i == 0 and j == 0:  # for the first DC component
                encoded.append((int(trimmed[j]).bit_length(), trimmed[j]))
            elif j == 0:  # to compute the difference between DC components
                diff = int(array[i][j] - array[i - 1][j])
                if diff != 0:
                    encoded.append((diff.bit_length(), diff))
                else:
                    encoded.append((1, diff))
                run_length = 0
            elif trimmed[j] == 0:  # increment run_length by one in case of a zero
                run_length += 1
            else:  # intermediary steam representation of the AC components
                encoded.append((run_length, int(trimmed[j]).bit_length(), trimmed[j]))
                run_length = 0
            # send EOB
        if not (encoded[len(encoded) - 1] == eob):
            encoded.append(eob)
    return encoded
```

Fungsi run_length_encoding yang digunakan untuk proses *run length encoding*.

```python
def get_freq_dict(array: list) -> dict:
    """
    returns a dict where the keys are the values of the array, and the values are their frequencies
    :param numpy.ndarray array: intermediary stream as array
    :return: frequency table
    """
    #
    data = Counter(array)
    result = {k: d / len(array) for k, d in data.items()}
    return result
```

Fungsi untuk mengubah array menjadi kamus frekuensi

```python
def find_huffman(p: dict) -> dict:
    """
    returns a Huffman code for an ensemble with distribution p
    :param dict p: frequency table
    :returns: huffman code for each symbol
    """
    # Base case of only two symbols, assign 0 or 1 arbitrarily; frequency does not matter
    if len(p) == 2:
        return dict(zip(p.keys(), ['0', '1']))

    # Create a new distribution by merging lowest probable pair
    p_prime = p.copy()
    a1, a2 = lowest_prob_pair(p)
    p1, p2 = p_prime.pop(a1), p_prime.pop(a2)
    p_prime[a1 + a2] = p1 + p2

    # Recurse and construct code on new distribution
    c = find_huffman(p_prime)
    ca1a2 = c.pop(a1 + a2)
    c[a1], c[a2] = ca1a2 + '0', ca1a2 + '1'

    return c
```

Fungsi untuk melakukan encoding huffman

```python
def lowest_prob_pair(p):
    # Return pair of symbols from distribution p with lowest probabilities
    sorted_p = sorted(p.items(), key=lambda x: x[1])
    return sorted_p[0][0], sorted_p[1][0]
```

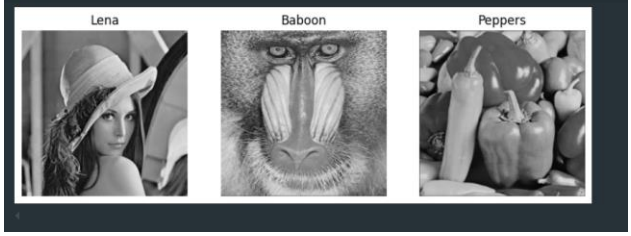
Fungsi untuk mencari simbol dengan probabilitas terkecik

Semua fungsi-fungsi tersebut kemudian akan digunakan kemudian di program utama. Selanjutnya pada program utama, akan dilakukan *import library* yang diperlukan dalam program ini.

```python
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

Tahap selanjutnya, kita perlu memasukkan gambar ke dalam program. Gambar yang akan kita masukkan akan kita konversi menjadi gambar abu atau *gambar grayscale* kemudian kita juga akan mengkonversinya menjadi matriks numpy agar dapat mempermudah kalkulasi saat dibutuhkan.

```
lena = np.array(Image.open('./assets/Lena512warna.bmp').convert('L'))
baboon = np.array(Image.open('./assets/baboon24.bmp').convert('L'))
peppers = np.array(Image.open('./assets/peppers512warna.bmp').convert('L'))

fig, axs = plt.subplots(1, 3, figsize=[10,5])

axs[0].imshow(lena, cmap=plt.get_cmap('gray'))
axs[0].set_title("Lena")

axs[1].imshow(baboon, cmap=plt.get_cmap('gray'))
axs[1].set_title("Baboon")


axs[2].imshow(peppers, cmap=plt.get_cmap('gray'))
axs[2].set_title("Peppers")

for ax in axs:
    ax.axis('off')
✓ 0.2s
```



C. Analisis Kompresi JPEG

Dalam percobaan ini, kita akan menggunakan program yang telah dibuat oleh orang lain [3]. Kita akan melakukan analis serta mencoba menggunakan gambar yang kita miliki untuk dianalisis dan mencari tau bagaimana proses dan hasil kompresi.

```
# read image
imgOriginal = cv2.imread('lena.bmp', cv2.IMREAD_COLOR)
# convert BGR to YCrCb
img = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2YCR_CB)
```

Pertama kita akan memasukkan gambar yang kita perlukan dan kemudian mengkonversikannya dalam YCrCb

```
width = len(img[0])
height = len(img)
y = np.zeros((height, width), np.float32) + img[:, :, 0]
cr = np.zeros((height, width), np.float32) + img[:, :, 1]
cb = np.zeros((height, width), np.float32) + img[:, :, 2]
# size of the image in bits before compression
totalNumberOfBitsWithoutCompression = len(y) * len(y[0]) * 8 + len(cb) * len(cb[0]) * 8
+ len(cr) * len(cr[0]) * 8
```

Kita juga akan menghitung jumlah bits sebelum kompresi agar kemudian diakhir kita dapat membandingkan hasil dan mencari rasio kompresi

```
# channel values should be normalized, hence subtract 128
y = y - 128
cr = cr - 128
cb = cb - 128
```

Kemudian, kita akan melakukan penggeseran nilai Y, Cr, Cb. Pada awalnya ketiga nilai ini memiliki *range* pada 0 sampai 255, tetapi sekarang kita akan menggeser ke kiri sebanyak 128. Sehingga *range* saat ini adalah -128 sampai 127.

```
# 4: 2: 2 subsampling is used # another subsampling scheme can be used
# thus chrominance channels should be sub-sampled
# define subsampling factors in both horizontal and vertical directions
SSH, SSV = 2, 2
# filter the chrominance channels using a 2x2 averaging filter # another type of filter
can be used
crf = cv2.boxFilter(cr, ddepth=-1, ksize=(2, 2))
cbf = cv2.boxFilter(cb, ddepth=-1, ksize=(2, 2))
crSub = crf[::SSV, ::SSH]
cbSub = cbf[::SSV, ::SSH]
```

Selanjutnya, kita akan melakukan *down sampling*. Pada program ini, kita akan menggunakan rasio 4 : 2 : 2

```
# check if padding is needed,
# if yes define empty arrays to pad each channel DCT with zeros if necessary
yWidth, yLength = ceil(len(y[0]) / windowSize) * windowSize, ceil(len(y) / windowSize) * windowSize
if (len(y[0]) % windowSize == 0) and (len(y) % windowSize == 0):
    yPadded = y.copy()
else:
    yPadded = np.zeros((yLength, yWidth))
    for i in range(len(y)):
        for j in range(len(y[0])):
            yPadded[i, j] += y[i, j]

# chrominance channels have the same dimensions, meaning both can be padded in one loop
cWidth, cLength = ceil(len(cbSub[0]) / windowSize) * windowSize, ceil(len(cbSub) / windowSize) *
windowSize
if (len(cbSub[0]) % windowSize == 0) and (len(cbSub) % windowSize == 0):
    crPadded = crSub.copy()
    cbPadded = cbSub.copy()
# since chrominance channels have the same dimensions, one loop is enough
else:
    crPadded = np.zeros((cLength, cWidth))
    cbPadded = np.zeros((cLength, cWidth))
    for i in range(len(crSub)):
        for j in range(len(crSub[0])):
            crPadded[i, j] += crSub[i, j]
            cbPadded[i, j] += cbSub[i, j]
```

Karena kita akan melakukan perhitungan dengan pembagian blok 8x8. Hal itu menyebabkan adanya kemungkinan bahwa terdapat gambar yang tidak dapat sepenuhnya dibagi menjadi 8x8. Sehingga kita perlu menambahkan *padding* atau data tambahan yang sedemikian sehingga membuat gambar kita dapat dibagi-bagi menjadi blok 8x8.

```
# get DCT of each channel
# define three empty matrices
yDct, crDct, cbDct = np.zeros((yLength, yWidth)), np.zeros((cLength, cWidth)), np.zeros((cLength,
cWidth))

# number of iteration on x axis and y axis to calculate the luminance cosine transform values
hBlocksForY = int(len(yDct[0]) / windowSize)  # number of blocks in the horizontal direction for
luminance
vBlocksForY = int(len(yDct) / windowSize)  # number of blocks in the vertical direction for luminance
# number of iteration on x axis and y axis to calculate the chrominance channels cosine transforms
values
hBlocksForC = int(len(crDct[0]) / windowSize)  # number of blocks in the horizontal direction for
chrominance
vBlocksForC = int(len(crDct) / windowSize)  # number of blocks in the vertical direction for
chrominance
```

Selanjutnya kita akan mempersiapkan 3 variabel yang akan digunakan dalam kalkulasi DCT. Kita akan melakukan operasi DCT kepada setiap channel, Y, Cr, dan Cb.

```
# define 3 empty matrices to store the quantized values
yq, crq, cbq = np.zeros((yLength, yWidth)), np.zeros((cLength, cWidth)), np.zeros((cLength, cWidth))
# and another 3 for the zigzags
yZigzag = np.zeros(((vBlocksForY * hBlocksForY), windowSize * windowSize))
crZigzag = np.zeros(((vBlocksForC * hBlocksForC), windowSize * windowSize))
cbZigzag = np.zeros(((vBlocksForC * hBlocksForC), windowSize * windowSize))
```

Kita juga akan mempersiapkan 3 variabel yang akan digunakan untuk menyimpan hasil kuantisasi dari matriks yang setelah dilakukan DCT. Lalu juga kita juga akan membuat 3 variabel untuk menyimpan vektor atau array nilai matriks kuantisasi yang bentuknya diubah menjadi linear dengan pola zigzag.

```
# define quantization tables
QTY = np.array([[16, 11, 10, 16, 24, 40, 51, 61],  # luminance quantization table
                [12, 12, 14, 19, 26, 48, 60, 55],
                [14, 13, 16, 24, 40, 57, 69, 56],
                [14, 17, 22, 29, 51, 87, 80, 62],
                [18, 22, 37, 56, 68, 109, 103, 77],
                [24, 35, 55, 64, 81, 104, 113, 92],
                [49, 64, 78, 87, 103, 121, 120, 101],
                [72, 92, 95, 98, 112, 100, 103, 99]])

QTC = np.array([[17, 18, 24, 47, 99, 99, 99, 99],  # chrominance quantization table
                [18, 21, 26, 66, 99, 99, 99, 99],
                [24, 26, 56, 99, 99, 99, 99, 99],
                [47, 66, 99, 99, 99, 99, 99, 99],
                [99, 99, 99, 99, 99, 99, 99, 99],
                [99, 99, 99, 99, 99, 99, 99, 99],
                [99, 99, 99, 99, 99, 99, 99, 99],
                [99, 99, 99, 99, 99, 99, 99, 99]])
# define window size
windowSize = len(QTY)
```

Proses kuantisasi menggunakan tabel kuantisasi seperti

di atas.

```
yCounter = 0
for i in range(vBlocksForY):
    for j in range(hBlocksForY):
        yDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize]
        = cv2.dct(
            yPadded[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize +
            windowSize])
        yq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] =
        np.ceil(
            yDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize +
            windowSize] / QTY)
        yZigzag[yCounter] += zigzag(
            yq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize +
            windowSize])
        yCounter += 1
yZigzag = yZigzag.astype(np.int16)
```

Pada proses ini, kita melakukan perhitungan untuk kanal Y dalam yang disimpan dalam bentuk array linear.

```
# either crq or cbq can be used to compute the number of blocks
cCounter = 0
for i in range(vBlocksForC):
    for j in range(hBlocksForC):
        crDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] = cv2.dct(
            crPadded[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize])
        crq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] = np.ceil(
            crDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] / QTC)
        crZigzag[cCounter] += zigzag(
            crq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize])
        cbDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] = cv2.dct(
            cbPadded[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize])
        cbq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] = np.ceil(
            cbDct[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize] / QTC)
        cbZigzag[cCounter] += zigzag(
            cbq[i * windowSize: i * windowSize + windowSize, j * windowSize: j * windowSize + windowSize])
        cCounter += 1
crZigzag = crZigzag.astype(np.int16)
cbZigzag = cbZigzag.astype(np.int16)
```

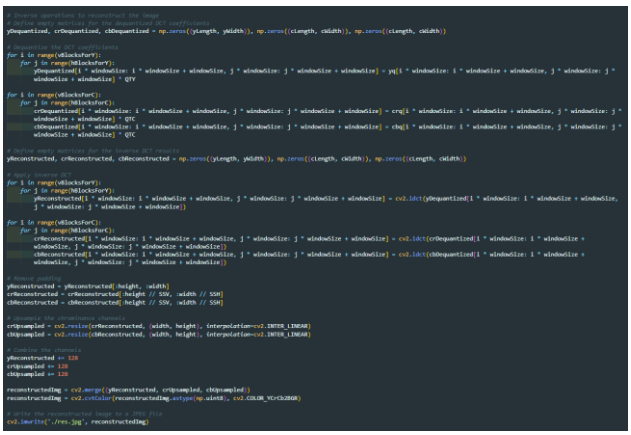
Kita juga akan melakukan proses yang sama untuk mendapatkan nilai pada kanal Cb dan Cr.

```
# find the run length encoding for each channel
# then get the frequency of each component in order to form a Huffman dictionary
yEncoded = run_length_encoding(yZigzag)
yFrequencyTable = get_freq_dict(yEncoded)
yHuffman = find_huffman(yFrequencyTable)

crEncoded = run_length_encoding(crZigzag)
crFrequencyTable = get_freq_dict(crEncoded)
crHuffman = find_huffman(crFrequencyTable)

cbEncoded = run_length_encoding(cbZigzag)
cbFrequencyTable = get_freq_dict(cbEncoded)
cbHuffman = find_huffman(cbFrequencyTable)
```

Memasuki tahap 5, yaitu *lossless compression*. Kita akan melakukan *encoding* yang bernama *run length*. Kita juga akan melakukan Huffman encoding agar dapat menghemat ruang untuk data yang sering muncul.

```
# calculate the number of bits to transmit for each channel
# and write them to an output file
file = open("Compressed1mage.asfh", "w")
yBitsToTransmit = str()
for value in yEncoded:
    yBitsToTransmit += yHuffman[value]

crBitsToTransmit = str()
for value in crEncoded:
    crBitsToTransmit += crHuffman[value]

cbBitsToTransmit = str()
for value in cbEncoded:
    cbBitsToTransmit += cbHuffman[value]

if file.writable():
    file.write(yBitsToTransmit + "\n" + crBitsToTransmit + "\n" + cbBitsToTransmit)
file.close()

totalNumberOfBitsAfterCompression = len(yBitsToTransmit) + len(crBitsToTransmit) + len
(cbBitsToTransmit)
print(
    "Compression Ratio is " + str(
        np.round(totalNumberOfBitsWithoutCompression / totalNumberOfBitsAfterCompression, 1)))
```

Selanjutnya kita akan menghitung bits pada setiap kanal dan kemudian menuliskannya ke dalam file. Kita juga dapat mengetahui berapa total dari bits yang dihasilkan dari setiap kanal.

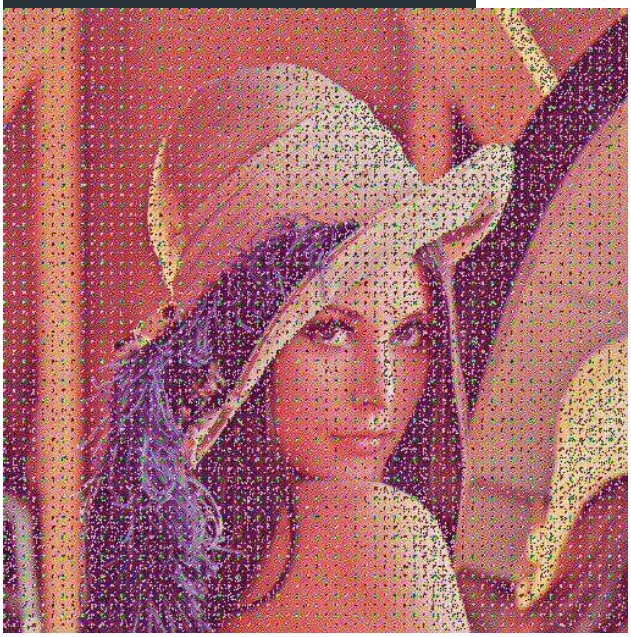


Penulis juga membuat kode tambahan untuk membalikkan semua proses tadi untuk membangun kembali gambar dengan data baru yang telah dikompresi.

D. Hasil

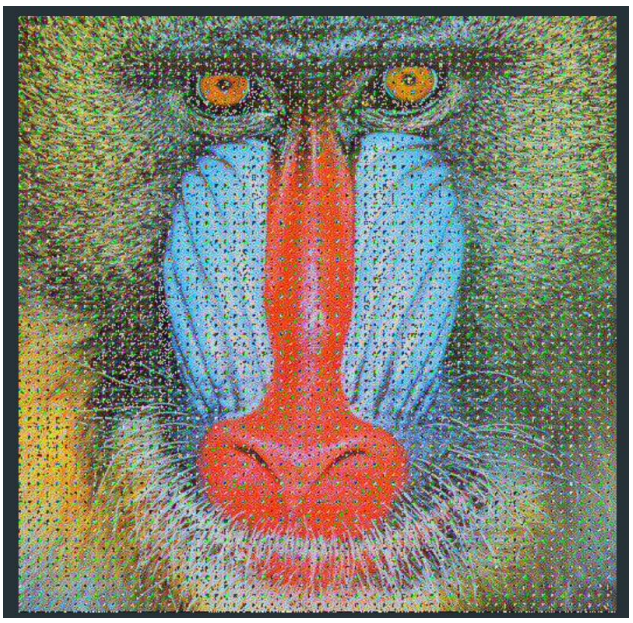
Setelah melakukan proses sebelumnya, kita sekarang bisa menghitung rasio dari kompresi. Dengan semua hal ini untuk melakukan kalkulasi terhadap 3 gambar kita. Kita akan mendapat hasil sebagai berikut

```
Compression Ratio is 10.1
```

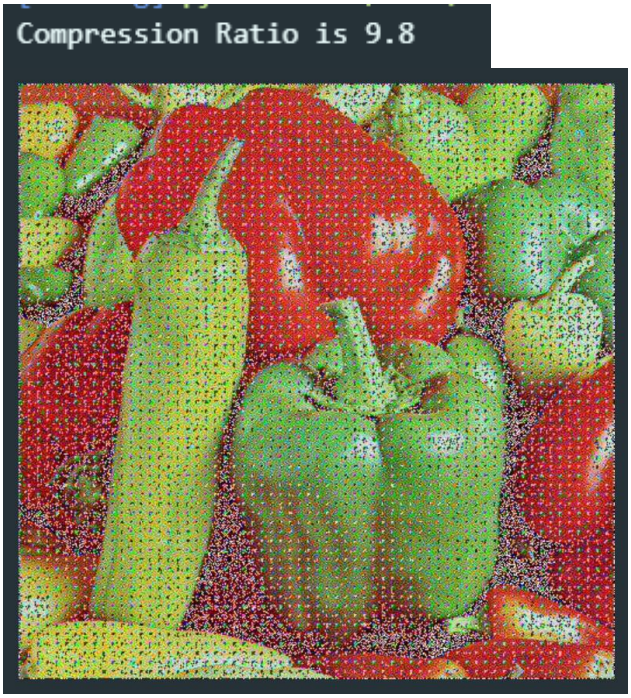


Hasil kompresi pada Lena menghasilkan rasio kompresi sebesar 10.1

```
Compression Ratio is 8.1
```



Hasil kompresi Baboon menghasilkan gambar baru dengan rasio kompresi 8.1



Untuk foto peppers menghasilkan kompresi dengan tingkat rasio kompresi sebesar 9.8.

V. KESIMPULAN

Transformasi kosinus diskrit merupakan transformasi standar yang digunakan dalam kompresi JPEG. Penggunaannya memungkinkan kita mengubah data spasial menjadi data frekuensi yang linear.

Penggunaan DCT dapat membantu untuk menghilangkan bagian yang tidak terlalu diperhatikan.

VII. UCAPAN TERIMA KASIH

Puji dan Syukur atas kehadirat Allah SWT, penulis dapat menyelesaikan makalah berjudul "Penerapan Transformasi Kosinus Diskrit Dalam Kompresi Gambar JPEG". Tak lupa penulis ucapkan terima kasih dan rasa syukur kepada :

1. Orang tua penulis yang senantiasa memberikan dukungan terbaik kepada penulis dalam kondisi apapun
2. Keluarga penulis yang senantiasa memberikan dukungan moral yang sangat baik.
3. Bapak Ir. Rila Mandala, M.Eng., Ph.D. selaku dosen mata kuliah Aljabar Linier dan Geometri yang selalu memberikan ilmu pengetahuan, inspirasi, dan bimbingan sehingga penulis bisa menyelesaikan makalah dan mendapat banyak pengetahuan baru satu semester ke belakang.
4. Teman-teman.
5. Penulis-penulis lain yang telah mempublikasikan hasil karyanya.

REFERENSI

[1] Munir, Rinaldi. 2023." Review Matriks". https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf (diakses pada 1 Desember 2025)

[2] Wilhelm Burger, Mark J. Burge. 2016. Digital Image Processing: An Algorithmic Introduction Using Java 2nd Edition (diakses pada 1 Desember 2025)

[3] 2022. https://github.com/mVirtuoso21/JPEG-Image-Compressor/tree/main (diakses pada 2 Desember 2025)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 2 Januari 2025

Muhammad Dicky Isra dan 13523075