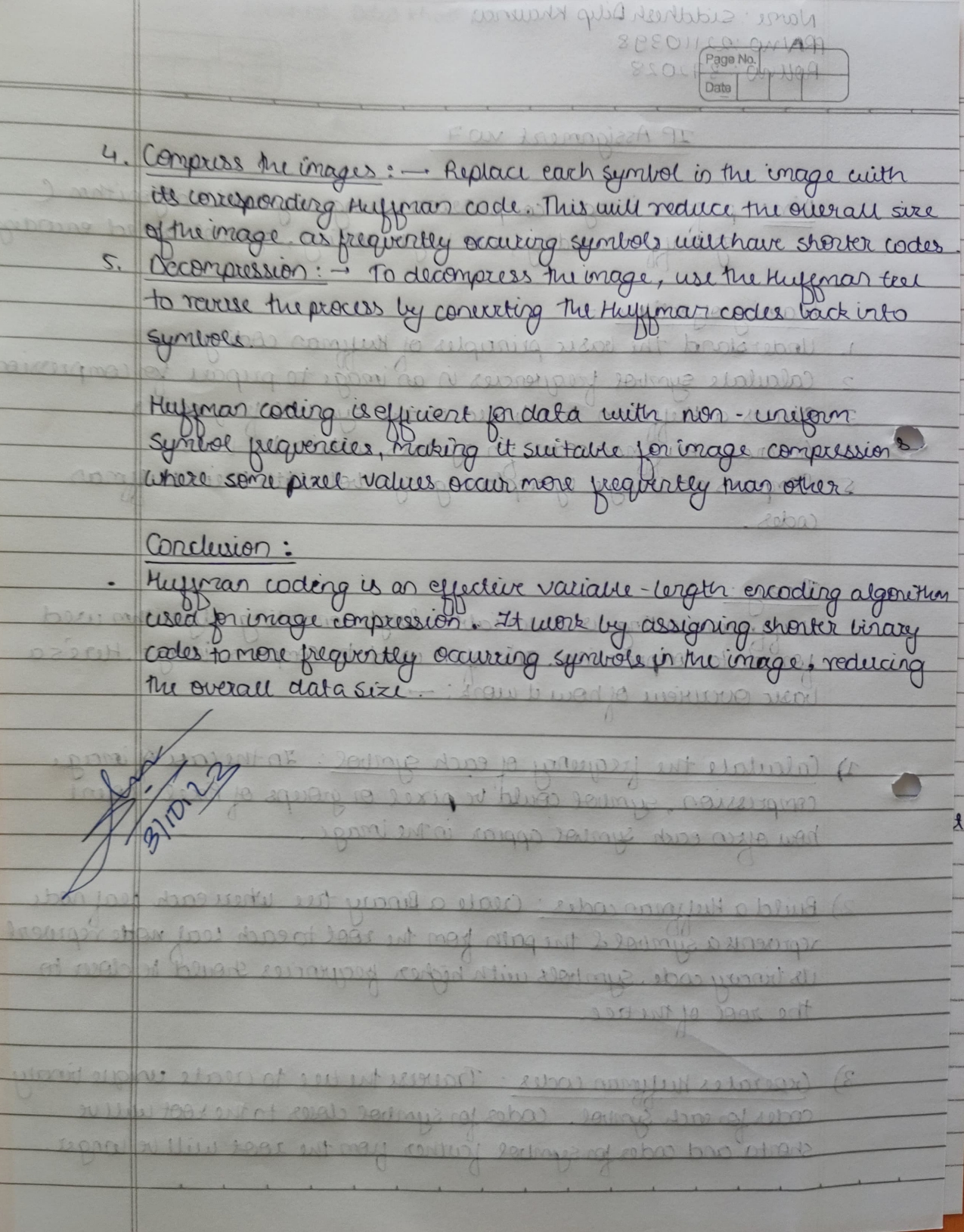
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Bansilal Ramnath Agarwal Charitable Trust's  Vishwakarma Institute of Information  Technology    **Department of**  **Artificial Intelligence and Data Science** | | | |
| Name: Siddhesh Dilip Khairnar |  | | | |
| Class: TY | Division: B | | | Roll No: 372028 |
| Semester: V | | Academic Year: 2023-2024 | | |
| Subject Name & Code: Image Processing: ADUA31205(B) | | | | |
| Title of Assignment: Perform image compression using any basic algorithm (e.g., Huffman coding, run length coding, symbol-based encoding). | | | | |
| Date of Performance: 04-10-2023 | | | Date of Submission: 01-11-2023 | |

**ASSIGNMENT NO. 7**

A piece of paper with writing on it

Description automatically generated



Program Code:

import cv2

import numpy as np

from collections import Counter

from heapq import heappush, heappop, heapify

class Node:

    def \_\_init\_\_(self, char, freq):

        self.char = char

        self.freq = freq

        self.left = None

        self.right = None

    def \_\_lt\_\_(self, other):

        return self.freq < other.freq

def calc\_freq(image):

    # Calculate frequency of each pixel value

    freq\_dict = Counter(image.flatten())

    return freq\_dict

def huffman\_tree(freq\_dict):

    # Create priority queue from frequency dict

    heap = [[weight, Node(char, weight)] for char, weight in freq\_dict.items()]

    heapify(heap)

    # Build Huffman Tree

    while len(heap) > 1:

        lo = heappop(heap)

        hi = heappop(heap)

        node = Node(None, lo[0] + hi[0])

        node.left = lo[1]

        node.right = hi[1]

        heappush(heap, [node.freq, node])

    return heap[0][1]

def huffman\_encoding(node, binary\_string='', code={}):

    # Generate Huffman encoding for each pixel value

    if node is not None:

        if node.char is not None:

            code[node.char] = binary\_string

        huffman\_encoding(node.left, binary\_string + '0', code)

        huffman\_encoding(node.right, binary\_string + '1', code)

    return code

def compress\_image(image, code):

    # Compress image using Huffman encoding

    rows, cols = image.shape

    compressed\_image = ''

    for i in range(rows):

        for j in range(cols):

            compressed\_image += code[image[i, j]]

    return compressed\_image

# Load grayscale image

image = cv2.imread(

    "C:/Users/asus/Downloads/EvXjoAkUYAE5K7O.jpg", cv2.IMREAD\_GRAYSCALE)

# Calculate frequency of each pixel value

freq\_dict = calc\_freq(image)

# Build Huffman Tree

tree = huffman\_tree(freq\_dict)

# Generate Huffman encoding for each pixel value

code = huffman\_encoding(tree)

# Compress image using Huffman encoding

compressed\_image = compress\_image(image, code)

print('Huffman Codes:')

for pixel\_value, huffman\_code in code.items():

    print(f'{pixel\_value} -> {huffman\_code}')

# Calculate average number of bits

average\_bits = sum(len(code[pixel\_value]) \* freq for pixel\_value,

                   freq in freq\_dict.items()) / sum(freq\_dict.values())

print('Average number of bits:', average\_bits)

Output:

A screenshot of a computer

Description automatically generated

