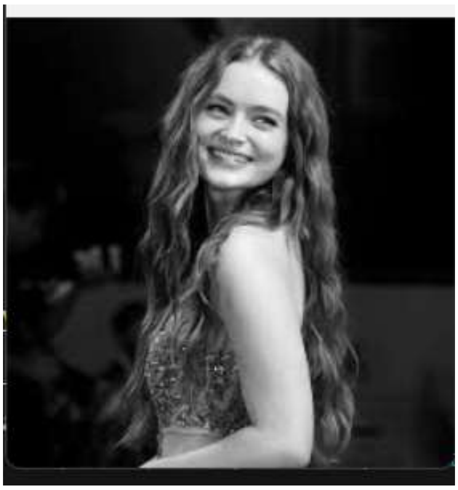
```
# JPEG Compression using DCT

import cv2
import numpy as np
from scipy.fftpack import dct, idct

# Read Image
I = cv2.imread(r"D:\DIP_Images\lab2_part1.jpg")

# Check if image loaded
if I is None:
    print("Error: Image not found. Check the path.")
    exit()

# Convert to Grayscale if RGB
if len(I.shape) == 3:
    I = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)

# Convert to double (float)
I = np.float64(I)

# JPEG Quantization Matrix
Q = np.array([
    [16,11,10,16,24,40,51,61],
    [12,12,14,19,26,58,60,55],
    [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]
])

# Compression Factor
factor = 10
Q = Q * factor

blockSize = 8

m, n = I.shape

reconstructed = np.zeros((m, n))

# 2D DCT Function
def dct2(block):
    return dct(dct(block.T, norm='ortho').T, norm='ortho')

# 2D IDCT Function
def idct2(block):
    return idct(idct(block.T, norm='ortho').T, norm='ortho')

# Block Processing
for i in range(0, m - blockSize + 1, blockSize):
    for j in range(0, n - blockSize + 1, blockSize):

        block = I[i:i+8, j:j+8]
        block = block - 128

        dctBlock = dct2(block)

        quantBlock = np.round(dctBlock / Q)

        dequantBlock = quantBlock * Q

        idctBlock = idct2(dequantBlock)

        idctBlock = idctBlock + 128

        reconstructed[i:i+8, j:j+8] = idctBlock

# Convert to uint8
reconstructed = np.uint8(np.clip(reconstructed, 0, 255))

I_uint8 = np.uint8(I)

# Display Images
cv2.imshow("Original Image", I_uint8)
cv2.imshow("Reconstructed Image After High JPEG Compression", reconstructed)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original Image



Recounstructed