Звіт

До лабораторної роботи №3

З предмету «Розробка мультимедійних систем»

Виконала студентка групи ІПС-41

Оніщенко Дар'я

Наша програма у загальному складається з 3 основних блоків:

- Компресор на основі DCT (Discrete Cosine Transform)
- Компресор на основі DWT (Discrete Wavelet Transform)
- Модуль порівняння PSNR (peak signal-to-noise ratio) для вимірювання рівня спотворень вираховуючи через MSE (mean square error)

Та допоміжних файлів, таких як:

- Utils – для загальних математичних перетворень (квантування / порогова обробка, реконструкція (декодування))
- Plotting – для візуалізації роботи алгоритмів стиснення та фінального результату

У якості вхідного зображення для демонстрації роботи скрипту візьмемо обкладинку вінілової платівки каверу Боббі Вомака на «Fly me to the moon» (Френк Сінатра). Завантажимо дві версії обкладинки з різними розмірами під назвами sample.png(578×578) та sample_512.png (512×512). На основі цієї різниці в розмірах, перевіримо чи коректно працює перевірка вхідних даних у функції зчитування та переведення зображення у grayscale – load_image_grayscale(path: str). Вона має перевірити вхідне зображення відповідати умові розмірності рівній степеню двійки для зручніших подальших перетворень та розбиття на під матрицю розміру 8×8. У разі невідповідності умові, виводиться відповідне повідомлення у консоль та переходимо до наступного зображення з папки вхідних зображень.

Косинусне перетворення (DCT). Основна ідея – перетворити зображення в частотну область, обрізати високочастотні компоненти та відновити зображення з меншою кількістю інформації. Перш за все потрібно розбити вхідну матрицю, що ми отримали після переведення зображення у ч/б:

Зображення ділиться на невеликі квадратичні блоки розміру 8, що відповідає процесу block-based processing. Цей процес допомагає застосувати DCT локально без артефактів. У даному коді за це відповідні функції

blockify/unblockify (перетворенням 2D масиву пікселів у 4D масив блоків і навпаки; transpose забезпечує правильне розташування блоків, що ми розбили, використовуючи індекси):

```
def blockify(img, block_size=8):  2 usages  OnishchenkoDaria
    h, w = img.shape
    blocks = img.reshape(h // block_size, block_size,
                         w // block_size, block_size)
    return blocks.transpose(0, 2, 1, 3)

def unblockify(blocks, block_size=8):  2 usages  OnishchenkoDaria
    h_blocks, w_blocks, _, _ = blocks.shape
    return blocks.transpose(0, 2, 1, 3).reshape(h_blocks * block_size, w_blocks * block_size)
```

Далі до кожного блоку застосовуємо двовимірне 2D DCT (працюємо з матрицями).

$$C_{k,l}(m,n) = \cos\left(\frac{k\pi}{2M}(2m+1)\right)\cos\left(\frac{l\pi}{2N}(2n+1)\right).$$

Результатом є матриця коефіцієнтів DCT, де:

- верхній лівий кут → низькі частоти (загальні форми, плавні градієнти)
- нижній правий кут → високі частоти (дрібні деталі, шум).

Далі ключовий етап для зниження кількісної інформації зображення, себто стиснення – квантізація (чим сильніша частота, тим більше її округляємо).

$$\hat{X}_{ij}(k,l) = \text{round}\left[\frac{X_{ij}(k,l)}{Q(k,l)}\right]$$

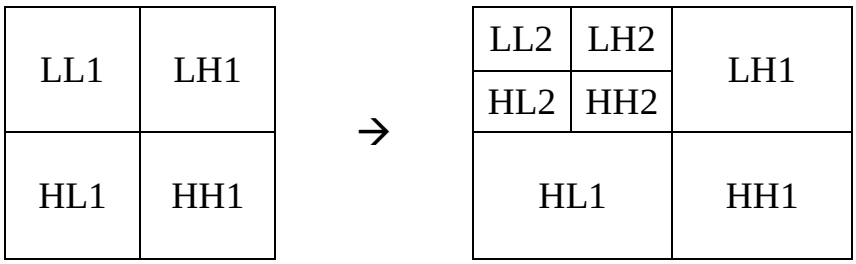
Далі здійснюємо відсікання високих частот. Для додаткового стиснення залишаються лише певні низькочастотні коефіцієнти (верхній лівий квадрат матриці; keep_ratio=0.5 означає, що залишаються лише 50% блоків від початку, а решта обнуляється).

Далі для відновлення зображення займемося декомпресією, де по суті в зворотному порядку застосуємо формули квантізації та DCT, що ми використовували раніше, та через unblockify складемо всі блоки у правильному порядку.

Дискретне вейвлет-перетворення (DWT) з використанням вейвлета Хаара. Основна ідея — розкласти зображення на аппроксимаційні та деталізовані коефіцієнти, обрізати дрібні деталі та відновити зображення з меншою кількістю даних. У результаті втілення кількох рівнів розкладу отримуємо список коефіцієнтів:

- cA (LL) — аппроксимаційні коефіцієнти на найнижчій частоті (загальні форми зображення)
- (cH, cV, cD) — детальні коефіцієнти: горизонтальні, вертикальні та діагональні деталі на кожному рівні (LH, HL, HH)

У форматі матриці вона має наступний вигляд, де з кожним наступним рівнем, ми розкладаємо далі рекурсивно матриці LLn , n – кількість рівнів:



Після цього проводимо операцію Thresholding встановлює в нуль усі коефіцієнти, які менші за поріг, що зменшує кількість збереженої інформації та дозволяє стискати зображення, при цьому зберігаючи помітні деталі.

Далі аналогічно DTC проводимо декомпресію: обернені вейвлет перетворення (об’єднуючи аппроксимаційні та детальні коефіцієнти в 2D зображення) та складання блоків докупи через unblockify.

Peak Signal-to-Noise Ratio (PSNR) порівняння. PSNR вимірює відношення максимальної можливої потужності сигналу до потужності шуму, де шум – це різниця між оригіналом і відновленим зображенням. Чим більше PSNR, тим краща якість відновленого зображення, типові значення:

30 дБ	Хороша якість
40 дБ	Дуже висока якість
До 20 дБ	Погана якість

У обчисленнях використовує формулу:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{\text{MSE}} \right)$$

- MAX_I — максимальне значення пікселя (для 8-бітного зображення $MAX_I = 255$)

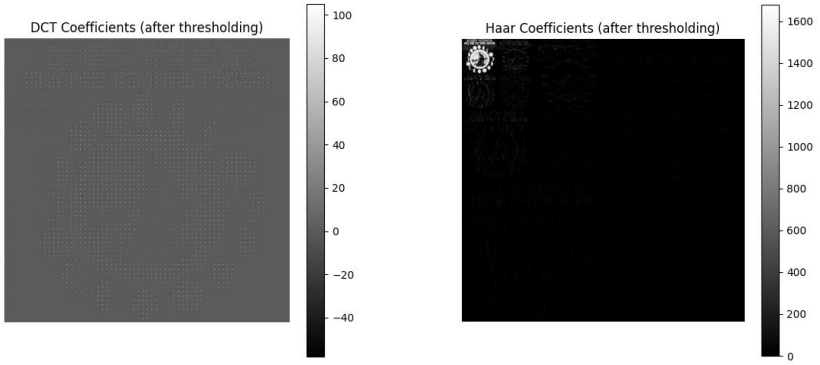
У свою чергу MSE вимірює середню квадратуру різниці між оригіналом і відновленим зображенням і чим менше значення MSE, тим ближче відновлене зображення до оригіналу. Формула:

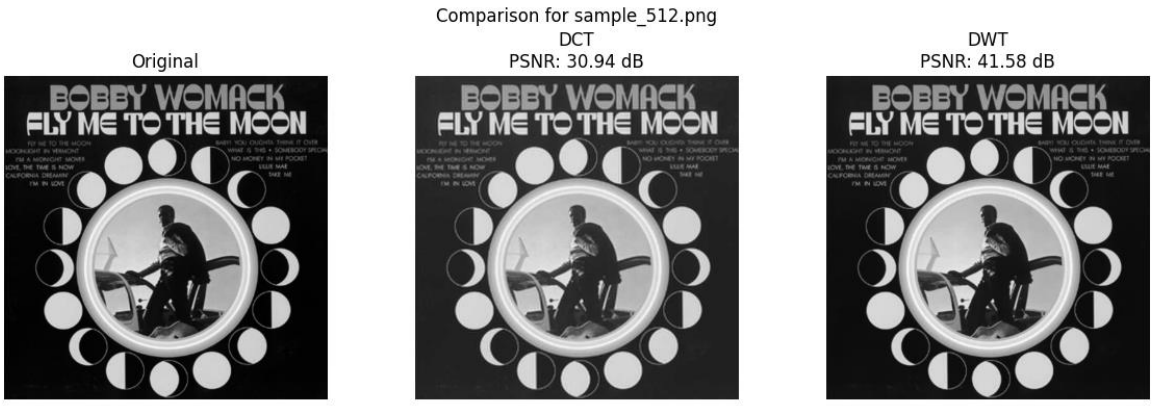
$$\mathrm{MSE} = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - K(i,j)]^2$$

- $I(i,j)$ — значення пікселя оригінального зображення.
- $K(i,j)$ — значення пікселя відновленого зображення.
- M, N — висота і ширина зображення.

З недоліків середньоквадратична помилка не враховує візуальну важливість (кольори і загальне сприйняття).

Демонстрація результатів і висновки.





Як можемо спостерігати, алгоритми працюють як належить і видають прекрасний результат. У висновку, якщо ми порівняємо з оригіналом, можна припустити, що метод Хаара впорався трохи краще з передачею кольору ніж DCT, але обидва результати входять в межу норми похибки MSE.