МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка додатку з реалізацією методів стиснення графічних форматів даних

Виконав студент групи **К-21і**
спеціальності 122 Комп'ютерні науки
Калєєв Костянтин Олександрович

Керівник **ст. викладач**
Вохменцева Т.Б.

Консультант **д.ф., доцент**
Бучинська І.В.

Рецензент **заступник директора**
- начальник відділу Департаменту
інформації та цифрових рішень
Одеської міської ради
Корчемний П.А.

Одеса 2023

ЗМІСТ

ТЕРМІНИ, СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Зображення – це фізичне відображення або запис візуального сприйняття.

Ймовірність – це абстрактне математичне поняття, пов’язане з нескінченними вибірками експериментальних даних.

Стиснення даних – це процедура перекодування даних для зменшення їх обсягу.

Стиснення зображення – це зменшення розміру (у байтах) графічного файлу, зберігаючи той самий рівень якості зображення.

Частота появи – це величина, яку можна обчислити для кінцевих наборів даних.

ANN – Artificial Neural Network

BPP (bpp) – Bits Per Pixel dB Decibel

CMOS – Complementary Metal-Oxide-Semiconductor

CNN – Convolutional Neural Network

CODEC – Coder-Decoder

CRC – Cyclic Redundancy Check

DCT – Discrete Cosine Transform

DoF – Degrees of Freedom

ET Eye Tracking

FOI – Features of Interest

FPGA – Field-Programmable Gate Array

FPU Floating Point Unit

GOF – Group of Frames

HCI – Human-Computer Interaction

HMI – Human-Machine Interaction

HT – Head Tracking

HVS – Human Visual System

IC – Integrated Circuit

JCU – JPEG Codec Unit

JPEG – Joint Photographic Experts Group

JPEG 2000 – Joint Photographic Experts Group 2000

LED – Light-Emitting Diode

Mbps – Megabit per second 1 Megabit = 1e6 bits

MBps – Megabyte per second 1 Megabyte = 1e6 bytes

MCU – Microcontroller Unit

MIPS – Million Instructions per Second

ML – Machine Learning

MLP – Multilayer Perceptron

MPEG – Moving Picture Experts Group

PCCR – Pupil Center Corneal Reflection

PSNR – Peak Signal-to-Noise Ratio

RAM – Random-Access Memory

RDO – Rate-Distortion Optimization

ROI – Region of Interest

SLE – System of Nonlinear Equations

ВСТУП

Стиснення зображення значно відрізняється від стиснення необроблених двійкових даних. Для стиснення зображень можна використовувати програми стиснення загального призначення, але результат не оптимальний. Це пояснюється тим, що зображення мають певні статистичні властивості, які можуть бути використані спеціально розробленими кодерами. Крім того, можна пожертвувати деякими дрібнішими деталями зображення заради збереження трохи більшої пропускної здатності або місця для зберігання. Це також означає, що в цій області можна використовувати методи стиснення з втратами. Дискретний вейвлет, по суті, є системою підсмугового кодування, і підсмугові кодери були досить успішними у стисненні мови та зображення.

Мета даної кваліфікаційної роботи бакалавра полягає у реалізації вейвлет-перетворення HAAR та перетворення DCT, освоєнні навичок у сфері стиснення зображень та розробці прототипу додатку для стиснення зображень.

Метод стиснення зображення усуває непотрібну та/або нерелевантну інформацію і шифрує те, що залишається. Часто необхідно відкинути непов’язані та ненадлишкові дані, щоб досягти необхідного скорочення. Існує два типи алгоритмів стиснення зображень, без втрат і з втратами, в першу чергу представлені для цієї мети: без втрат і з втратами. В даний час додаткові стратегії включені в фундаментальний метод. В конкретних програмах для стиснення зображень використовуються еволюційні алгоритми нейронної мережі. У цій роботі розглядаються різні доступні методи стиснення зображення. Стиснення зображень суттєво відрізняється від стиснення простих двійкових даних. Щоб вирішити ці проблеми, слід використовувати різні методи стиснення зображення. [1]

Стиснення даних – це процедура перекодування даних для зменшення їх обсягу. Компресія заснована на усуненні надмірності інформації, що міститься у вхідних даних. Дані, які не мають властивостей надмірності (наприклад, випадковий сигнал або шум), не можуть бути стиснуті. Найпростіші алгоритми

стиснення, так звані оптимальні алгоритми кодування, є статистичними і засновані на врахуванні розподілу ймовірностей елементів вхідного повідомлення (текст, зображення, файл). На практиці для наближеної оцінки ймовірностей використовуються частоти появи елементів у вхідному повідомленні. Ймовірність – це абстрактне математичне поняття, пов’язане з нескінченними вибірками експериментальних даних, а частота появи – це величина, яку можна обчислити для кінцевих наборів даних. При досить великій кількості елементів у наборі експериментальних даних, частота елемента близька (з деякою точністю) до його ймовірності [2]. Якщо згадані ймовірності різні, то найбільш імовірні елементи (загальні) можуть порівнювати коротші кодові слова і, навпаки, для малоймовірних елементів порівнювати довші кодові слова. Таким чином можна зменшити середню довжину кодового слова. Оптимальний алгоритм кодування робить це так, що середня довжина кодового слова є мінімальною; як і з меншою довжиною кодування, він стає незворотнім. До таких алгоритмів можна віднести алгоритм Шеннон Фано і метод Хаффман. Недоліком обох методів є те, що вони не можуть кодувати повідомлення більш економно, ніж один біт на елемент повідомлення.

Дана кваліфікаційна робота бакалавра, складається з 61 сторінки, 20 рисунків та 16 джерел посилання.

1 **ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ**

Комп’ютерні та відео-медійні застосунки швидко розвинулися у сфері мультимедіа, яка вимагає високопродуктивних, швидких можливостей цифрового відео та аудіо. Цифрова обробка сигналів широко використовується в багатьох галузях електроніки, комунікаційної та інформаційної техніки. У стисненні сигналів, фільтрації, ідентифікації систем зазвичай використовувані перетворення базуються на базових синусоїдальних функціях, таких як: дискретне перетворення Фур’є, синусне або косинусне перетворення або прямокутних базових функціях: дискретне перетворення Уолша та вейвлетне перетворення (Хаар, Добеші тощо). Усі ці функції є ортогональними, і їх перетворення потребують лише додавання та віднімання. Завдяки цьому їх легко реалізувати на комп’ютері. Це не тільки спрощує обчислення, але також дозволяє використовувати різні (лінійні та нелінійні) фільтри для отримання спектру. Методи Фур’є не завжди є хорошими інструментами для повторного захоплення негладкого сигналу; занадто багато інформації потрібно для локальної реконструкції сигналу. У цих випадках вейвлет-аналіз часто є дуже ефективним, оскільки він забезпечує простий підхід до роботи з локальними аспектами сигналу, отже, особливі властивості Хаара або вейвлет-перетворення дозволяють ефективно аналізувати вихідне зображення в спектральній області [3,4].

Зображення – це фізичне відображення або запис візуального сприйняття. Зображення є важливими документами сучасності; щоб маніпулювати ними в різних програмах, необхідно мати стиснений файл. Стиснення зображення – це зменшення розміру (у байтах) графічного файлу, зберігаючи той самий рівень якості зображення. Це досягається без будь-якої шкоди для зображення. Завдяки зменшенню розміру файлу можна зберігати більше зображень на тій самій кількості дискет або в тому самому обсязі пам’яті. Крім того, це зменшує час, необхідний для завантаження фотографій в Інтернет або завантаження зі сторінок локальної мережі.

Більше чи менше стиснення залежить від мети програми. Стиснення зображення є дуже важливим. Вирішальна частина передачі та зберігання даних зображень через обмеження зберігання. Основна мета стискання зображення полягає в тому, щоб відобразити його з якомога меншою кількістю пікселів. бітів без шкоди для вихідного зображення ключової інформації вмісту. Методи стиснення швидко створюються для стиснення величезних файлів даних, включаючи фотографії з прискореним розширенням технологій в яких величезна кількість зображень. За допомогою правильних процедур дані повинні зберігатися належним чином. Як правило, зображення можна стиснути за допомогою ефективних методів. Існує ряд алгоритмів, які виконують це стиснення багатьма методами, деякі без втрат, а інші ні. Без втрат зберігається ідентична інформація, що й вихідне зображення, і стиснення з втратами даних під час стиснення зображення втрачається. Деякі з цих алгоритмів стиснення були розроблені для окремих типів даних. Тому для інших типів фотографій вони не підійдуть. У деяких алгоритмах можна змінити кілька параметрів, які використовують щоб краще адаптувати стиснення до зображення.

1.1 Аналіз представлення даних

Інформація може бути представлена різними способами. Дискретну інформацію, таку як текст і числа, легко представити в цифрових даних. Цей тип представлення даних відомий як символьні цифрові дані. Таким чином, символічні цифрові дані є абсолютним представленням даних, таких як літера, цифра, символ, знак, машинний код або малюнок. Безперервна інформація, така як мова, музика, аудіо, зображення та відео, часто існує в природному світі як аналогова інформація. Як добре відомо фахівцям у даній галузі техніки, нещодавні досягнення в цифровій комп’ютерній технології дуже великої інтеграції (VLSI) дозволили як дискретну, так і аналогову інформацію представляти цифровими даними. Безперервну інформацію, представлену як цифрові дані, ча-

сто називають дифузними даними. Таким чином, дифузні цифрові дані є представленням даних, які мають низьку щільність інформації і зазвичай нелегко розпізнати людину у своїй рідній формі. Цифрове представлення даних має багато переваг. Наприклад, цифрові дані легше обробляти, зберігати та передавати завдяки своїй властивій високій завадостійкості. Крім того, включення надмірності в представлення цифрових даних дозволяє виявити та/або виправити помилки. Можливості виявлення та/або виправлення помилок залежать від обсягу та типу надлишкових даних, доступної обробки виявлення та виправлення помилок і ступеня пошкодження даних.

Одним із результатів представлення цифрових даних є постійна потреба у збільшенні можливостей обробки, зберігання та передачі даних. Це особливо вірно для дифузних даних, де збільшення точності та роздільної здатності створює експоненціально більшу кількість даних. Стиснення даних широко використовується для зменшення обсягу даних, необхідних для обробки, передачі або зберігання заданої кількості інформації. Загалом існує два типи методів стиснення даних, які можна використовувати окремо або спільно для кодування/декодування даних: стиснення даних без втрат і стиснення даних із втратами. Методи стиснення даних із втратою даних забезпечують неточне представлення вихідних нестиснутих даних таким чином, що декодовані (або реконструйовані) дані відрізняються від вихідних некодованих/нестиснутих даних. Стиснення даних із втратою даних також відоме як необоротне або шумове стиснення. Ентропія визначається як кількість інформації в даному наборі даних. Таким чином, одна очевидна перевага стиснення даних із втратами полягає в тому, що коефіцієнти стиснення можуть перевищувати межу ентропії, і все це за рахунок інформаційного вмісту. Багато методів стиснення даних із втратою даних спрямовані на використання різноманітних властивостей людських органів чуття для видалення даних, які інакше не сприймаються. Наприклад, стиснення візуальних зображень із втратою даних може мати на меті видалення інформаційного вмісту, який перевищує роздільну здатність дисплея або коефіцієнт контрастності.

З іншого боку, методи стиснення даних без втрат забезпечують точне представлення оригінальних нестиснутих даних. Простіше кажучи, декодовані (або реконструйовані) дані ідентичні вихідним незакодованим/нестисненим даним. Таким чином, стиснення даних без втрат має як поточне обмеження мінімальне представлення, яке визначається ентропією даного набору даних.

Існують різні проблеми, пов'язані з використанням методів стиснення без втрат. Однією з основних проблем, з якими стикаються більшість методів стиснення даних без втрат, є їх чутлива до вмісту поведінка. Це часто називають залежністю від даних. Залежність від даних передбачає, що досягнутий коефіцієнт стиснення значною мірою залежить від вмісту даних, які стискаються. Наприклад, файли бази даних часто мають великі невикористані поля та високу надлишковість даних, пропонуючи можливість стискати дані без втрат у співвідношенні 5 до 1 або більше. На відміну від цього, стислі програмні програми майже не мають надмірності даних і, як правило, не стискаються без втрат краще, ніж 2 до 1. Інша проблема зі стисненням без втрат полягає в тому, що існують значні варіації у ступені стиснення, отриманому при використанні одного методу стиснення даних без втрат для потоків даних, що мають різний вміст і розмір даних. Цей процес відомий як природна варіація. Ще одна проблема полягає в тому, що негативне стиснення може виникнути, коли певні методи стиснення даних впливають на багато типів сильно стиснених даних. Дані з високим ступенем стиснення виглядають випадковими, тому багато методів стиснення даних суттєво розширюють, а не стискають цей тип даних. Для певної програми існує багато факторів, які визначають застосовність різних методів стиснення даних. Ці фактори включають ступінь стиснення, вимоги до обробки кодування та декодування, часові затримки кодування та декодування, сумісність з існуючими стандартами, а також складність і вартість впровадження, а також адаптивність і стійкість до варіацій вхідних даних. У сучасній техніці існує прямий зв'язок між ступенем стиснення та обсягом і складністю необхідної обробки. Одним із обмежуючих факторів у бі-

льшості існуючих методів стиснення даних без втрат рівня техніки є швидкість, з якою виконуються процеси кодування та декодування. Компроміси реалізації апаратного та програмного забезпечення часто визначаються складністю кодера та декодера разом із вартістю. Іншою проблемою, пов'язаною з методами стиснення без втрат, є визначення оптимальної техніки стиснення для даного набору вхідних даних і передбачуваного застосування. Для боротьби з цією проблемою існує багато звичайних методів, що залежать від вмісту, які можна використовувати. Наприклад, дескриптори типів файлів зазвичай додаються до імен файлів, щоб описати прикладні програми, які зазвичай діють на дані, що містяться у файлі. Таким чином можна визначити типи даних, структури даних і формати в даному файлі. Фундаментальні обмеження цієї техніки, що залежить від вмісту, включають:

- надзвичайно велику кількість прикладних програм, деякі з яких не мають опублікованих або документованих форматів файлів, структур даних або дескрипторів типів даних;
- можливість для будь-якого постачальника чи консорціуму стиснення даних отримувати, зберігати та отримувати доступ до величезних обсягів даних, необхідних для ідентифікації відомих дескрипторів файлів і пов’язаних типів даних, структур даних і форматів;
- швидкість, з якою розробляються нові прикладні програми, і необхідність відповідно оновлювати описи даних формату файлу.

1.2 Огляд типів стиснення

Методи, які використовуються для стиснення файлів зображень, зазвичай належать до однієї з двох категорій: із втратами та без втрат. Стиснення із втратами зменшує розмір файлу зображення шляхом остаточного видалення менш важливої інформації, зокрема надлишкових даних. Стиснення з втратами може знизити якість зображення до рівня спотворення, особливо якщо

зображення надто стиснене. Однак якість може бути збережена, якщо компресію застосовувати ретельно. Однією з проблем стиснення з втратами є його незворотність. Якщо стиснення з втратами багаторазово застосовується до того самого зображення, воно дедалі більше спотворюється. Стиснення з втратами даних виявилося цінною стратегією для Інтернету, де часто допускається помірне погіршення якості зображення [5].

Найпоширенішим прикладом стиснення з втратами є JPEG, формат стиснення зображень, який широко використовується в Інтернеті та цифровій фотографії. Цей загальновизнаний формат підтримується численними інструментами та програмами. Крім того, стиснення можна застосовувати в градусах, що дає змогу використовувати стиснення JPEG, яке найкраще забезпечує баланс між розміром файлу та якістю (рис. 1).

Рисунок 1 – Різниця в якості При використанні стиснення в форматі JPEG

Інший підхід до стиснення зображення називається без втрат. Цей метод застосовує стиснення без видалення критичних даних або зниження якості зображення, що призводить до стисненого зображення, яке можна відновити до початкового стану без погіршення чи спотворення. Однак стиснення без втрат не зменшує розмір файлу майже так само, як стиснення з втратами, пропонуючи невелику перевагу з точки зору місця для зберігання, пропускної здатності мережі або швидкості завантаження. Стиснення без втрат зазвичай використовується в ситуаціях, коли якість зображення важливіша за простір на диску чи продуктивність мережі, наприклад для зображень продуктів або для демонстрації творів мистецтва. Одним із найпоширеніших форматів без втрат є PNG,

широко використовуваний формат, який зменшує розмір файлу шляхом визначення шаблонів і стиснення цих шаблонів разом. Хоча файли PNG, як правило, більші за файли JPEG, веб-сайти широко використовують їх, коли потрібно більше деталей зображення, наприклад для логотипів, значків, скріншотів або зображень із текстом. Іншим знайомим форматом без втрат є BMP, власний підхід до стиснення зображень, запроваджений Microsoft і використовується в основному для продуктів Microsoft, зокрема для комп'ютерів Windows [6]. GIF – це формат стиснення, який відноситься до категорії без втрат, хоча існує деяка плутанина щодо того, чи є він із втратами чи без втрат. Зображення GIF обмежені 256 кольорами, тому перетворення зображення з більшою кількістю кольорів у GIF призводить до втрати якості, що іноді пояснюється стисненням із втратами. Але алгоритми стиснення, які використовує GIF, є без втрат. Якщо якість втрачається, це пов'язано з проблемами, пов'язаними з перетворенням файлу. В даний час формат GIF використовується переважно для простих відео та анімацій (рис. 2).

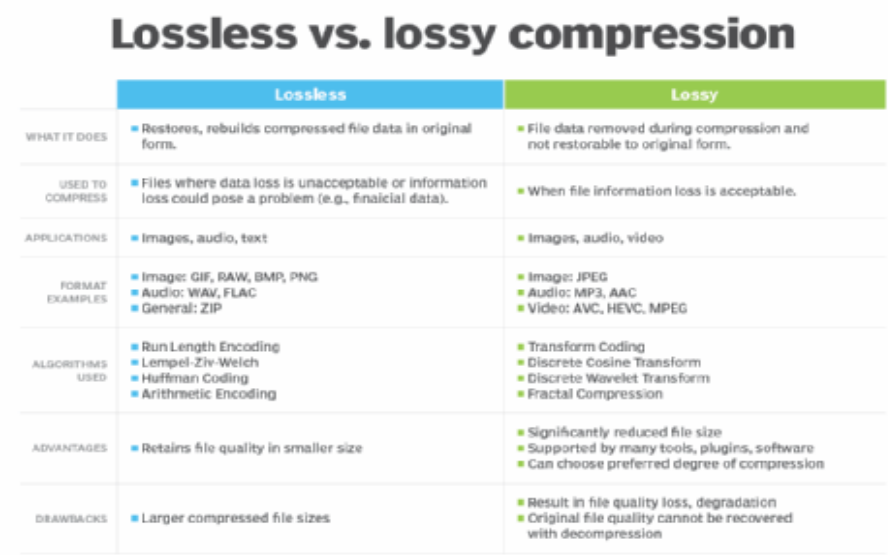

Рисунок 2 – Порівняння відмінностей між стисненням без втрат і стисненням із втратами

Формат стиснення, який досягає успіху, – це WebP від Google, формат зображень, розроблений виключно для Інтернету. На відміну від більшості методів стиснення, WebP підтримує як стиснення без втрат, так і з втратами, що робить його надзвичайно універсальним. Зображення WebP зазвичай займають менше місця на диску, ніж інші формати, але мають порівнянну якість. Більшість основних браузерів підтримують зображення WebP.

Стиснення також можна використовувати для типів файлів, не пов'язаних із зображеннями, таких як текстові або програмні файли, але його використання зазвичай обмежується стисненням без втрат. У текстових і програмних файлах надзвичайно важливо, щоб стиснення було без втрат, оскільки одна помилка може пошкодити значення текстового файлу або спричинити невиконання програми. Формат файлу zip є прикладом стиснення без втрат, який зазвичай використовується для текстових файлів або навіть цілих каталогів файлів [7]. Під час стиснення зображення невелика втрата якості зазвичай непомітна. У ситуаціях, коли є певний допуск до втрати, до файлів можна застосувати більший рівень стиснення, ніж коли допуск до втрати відсутній. З цієї причини графічні зображення зазвичай можна стиснути набагато більше, ніж текстові або програмні файли.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

Вибір програмних засобів для реалізації даної роботи залежить від конкретних вимог та потреб проєкту. Проте, можуть бути розглянуті такі програмні засоби:

- мова програмування;
- бібліотеки та фреймворки;
- інструменти для стиснення зображень;
- середовище розробки.

Вибір мови програмування залежить від особистих навичок та вподобань, а також від підтримки необхідних бібліотек і фреймворків. Наприклад, для розробки застосунку для стиснення зображень можуть бути використані мови програмування, такі як Python, Java, C++ або MATLAB. Вибір бібліотек та фреймворків залежить від необхідних функцій та можливостей. Наприклад, для реалізації вейвлет-перетворення та перетворення DCT можна використати бібліотеки, такі як NumPy, SciPy або OpenCV. Для розробки застосунку можуть бути використані фреймворки, такі як Flask або Django для веб-розробки, або Qt для розробки десктопних застосунків. Для реалізації стиснення зображень можна використати різноманітні алгоритми та методи, такі як JPEG, PNG або дискретне косинусне перетворення (DCT). Вибір конкретного алгоритму залежить від вимог до якості стиснення та швидкодії. Для роботи з програмними засобами можна використовувати різноманітні середовища розробки, такі як PyCharm, Eclipse, Visual Studio або Jupyter Notebook.

Більшість трафіку завантажень складається із зображень. Як наслідок, чим меншим можна зробити зображення для завантаження, тим кращий досвід роботи в мережі застосунок може надати користувачам. Нижче приведені рекомендації, спрямовані на поліпшення ефективності зображень у мережі та забезпечення їх зручного використання шляхом оптимізації їх розміру.

2.1 Огляд форматів зображень

Програми Android зазвичай використовують зображення в одному чи кількох форматів: AVIF, PNG, JPG, JPEG і WebP. Кожен формат має свої особливості, наприклад, рівень стиснення, підтримку прозорості, кольоровий простір тощо. Опис форматів зображень дозволяє ознайомитися з різними варіантами форматів та їхніми перевагами, що сприяє вибору оптимального формату для конкретного використання у програмі Android, а також допомагає розуміти їхні властивості та вибрати найбільш підходящий формат для конкретних потреб.

2.1.1 Формат AVIF: опис та особливості

Android 12 (рівень API 31) і новіші підтримують формат файлу зображень AVIF, який є контейнером для зображень і послідовностей зображень, закодованих з використанням AV1. AVIF використовує внутрішньокадрове кодування, що сприяє стисненню відео, і значно покращує якість зображення при тому самому розмірі файлу порівняно зі старішими форматами, такими як JPEG. Підтримка формату AVIF дозволяє програмам Android використовувати цей формат для зображень, що відкриває нові можливості в забезпеченні високої якості зображень з ефективним використанням простору зберігання.

Основні особливості формату AVIF включають:

- висока стисненість;
- підтримка високої якості;
- підтримка альфа-каналу;
- розширена функціональність.

Формат AVIF використовує потужний кодек AV1, який забезпечує ефективне стиснення зображень без втрати якості. Він може забезпечити значно кращу стисненість порівняно з іншими форматами, такими як JPEG чи PNG. AVIF підтримує широкий спектр кольорових просторів та глибини кольору,

що дозволяє зберігати зображення з високою якістю та точністю кольору. AVIF підтримує альфа-канал, що дозволяє зберігати прозорість у зображеннях. Це особливо корисно для використання зображень з прозорим фоном на веб-сторінках чи в графічних редакторах. AVIF підтримує додаткові функції, такі як масштабування зображень, анімація, HDR (High Dynamic Range) та багато іншого. Це робить його універсальним форматом для різних застосувань.

2.1.2 Формат PNG: опис та особливості

PNG (Portable Network Graphics) є форматом зображень, який використовує безвтратне стиснення для збереження зображень з високою якістю та підтримкою прозорості. Основні особливості формату PNG включають:

- безвтратне стиснення;
- підтримка прозорості;
- висока якість зображення;
- підтримка індексованого кольору.

PNG використовує алгоритми безвтратного стиснення, що дозволяє зберігати зображення без втрати якості. Воно підходить для зображень з простими формами та текстовою інформацією. PNG підтримує альфа-канал, що дозволяє зберігати прозорість у зображеннях. Це корисно для створення зображень з прозорим фоном, які можна використовувати на веб-сторінках чи в графічних редакторах. PNG дозволяє зберігати зображення з високою якістю, зокрема зберігати деталі, градієнти та точність кольору. PNG підтримує індексований кольоровий режим, який дозволяє зменшити розмір файлу, зокрема для зображень з обмеженою палітрою кольорів.

Ключ до зменшення файлів у форматі PNG полягає в зменшенні кількості унікальних кольорів, які використовуються в кожному рядку пікселів, які складають зображення. Використовуючи менше кольорів, покращується потенціал стиснення на всіх інших етапах конвеєра. Зменшення кількості унікаль-

них кольорів має суттєве значення, оскільки ефективність стиснення PNG частково залежить від ступеня зміни кольорів горизонтально суміжних пікселів. Таким чином, зменшення кількості унікальних кольорів у кожному рядку зображень у форматі PNG може допомогти зменшити їхні розміри файлів. Вирішуючи, чи застосовувати цю стратегію, слід пам’ятати, що зменшення кількості унікальних кольорів фактично означає застосування до зображення етапу кодування з втратами. Однак інструмент кодування може не вміти добре оцінювати, наскільки погано виглядає незначна на перший погляд помилка для людського ока. Тому слід виконати цю роботу вручну, щоб забезпечити правильний баланс між ефективним стисненням і прийнятною якістю зображення. Є два особливо корисні підходи, які можна застосувати: прагнення до індексованих форматів та застосування векторного квантування.

2.1.2.1 Опис підходу індексованих форматів

Будь-яка спроба зменшити колір має починатися зі спроби оптимізувати кольори, щоб була можливість використовувати формат INDEXED під час експорту зображення як PNG. Режим кольорів INDEXED працює шляхом вибору найкращих 256 кольорів для використання та заміни всіх значень пікселів на індекси в цій палітрі кольорів. Результатом є скорочення з 16 мільйонів (потенційних) кольорів до лише 256 кольорів: з 3 (без прозорості) або 4 (з прозорістю) байтів на піксель до 1 байта на піксель. Ця зміна є значним першим кроком зменшення розміру файлу. На рис. 3 показано зображення та його індексований варіант.

Рисунок 3 – Зображення до і після перетворення у формат INDEXED

На рис. 4 показана палітра кольорів для зображення на рис. 3:

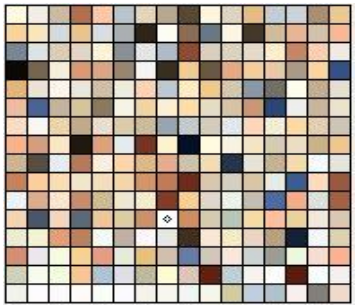

Рисунок 4 – Палітра кольорів для зображення на рис. 3.

Представлення зображення як зображення палітри значно покращує розмір файлу, тому варто перевірити, чи можна конвертувати більшість зображень. Звичайно, не кожне зображення можна точно представити лише за допомогою 256 кольорів. Для правильного вигляду деяких зображень, наприклад, може знадобитися 257, 310, 512 або 912 кольорів. У таких випадках також може бути корисним векторне квантування.

2.1.2.2 Опис підходу векторного квантування

Процес створення індексованого зображення краще описати як векторне квантування (VQ). VQ служить процесом округлення для багатовимірних чисел. У цьому процесі всі кольори на зображенні групуються на основі їх схожості. Для даної групи всі кольори в цій групі замінюються одним значенням центральної точки, що мінімізує помилку для кольорів у цій комірці. На рис. 5 зелені крапки представляють вхідні кольори, а червоні крапки – центральні точки, які замінюють вхідні кольори. Кожна клітинка обмежена синіми лініями [8].

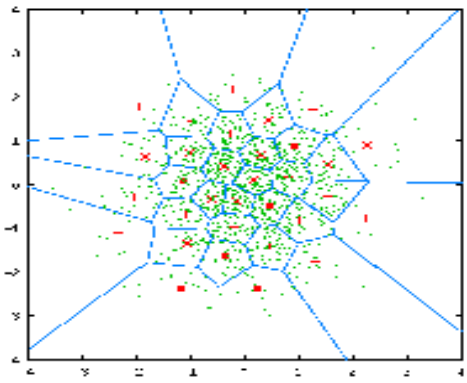

Рисунок 5 – Застосування векторного квантування
до кольорів зображення

Результат застосування VQ до зображення зменшує кількість унікальних кольорів, замінюючи кожну групу кольорів одним кольором, який є «досить близьким» за візуальною якістю.

Ця техніка також дозволяє визначити максимальну кількість унікальних кольорів у зображенні. Наприклад, на рис. 6 показано голову папуги в 16,7 мільйонах кольорів (24 біти на піксель, або bpp) разом із версією, яка дозволяє використовувати лише 16 (3 bpp) унікальних кольорів.

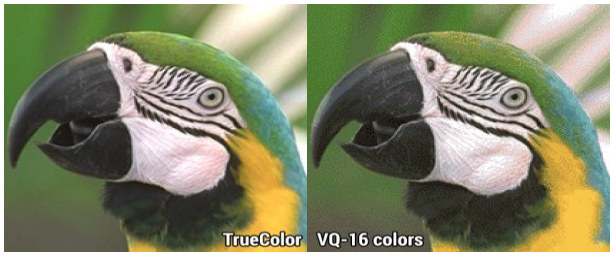

Рисунок 6 – Зображення до та після застосування
векторної кількісної оцінки.

Відразу можна побачити, що є втрата якості; більшість кольорів градієнта було замінено, що додало ефекту смуг зображенню. Для цього зображення потрібно більше 16 унікальних кольорів. Налаштування кроку VQ у конвеєрі може допомогти краще зрозуміти справжню кількість унікальних кольорів, які

використовує зображення, і може допомогти значно зменшити їх кількість. Існує ряд легкодоступних інструментів, які можна використовувати, щоб допомогти реалізувати цю техніку.

2.1.3 Формат JPG: опис та особливості

Якщо використовується зображення JPG, можна внести кілька невеликих змін, які потенційно забезпечать значну економію розміру файлу. До них належать:

- створення файлу меншого розміру за допомогою різних методів кодування (без впливу на якість);
- трохи регулюємо якість, щоб отримати краще стиснення;
- дотримуючись цих стратегій, можна зменшити розмір файлу до 25%.

Вибираючи інструменти, необхідно пам’ятати, що інструменти для експорту фотографій можуть вставляти у зображення непотрібні метадані, наприклад дані GPS. Як мінімум необхідно спробувати використати наявні інструменти, щоб видалити цю інформацію з ваших файлів [5,9].

2.1.4 Формат WebP: опис та особливості

WebP – це новіший формат зображень, який підтримується з Android 4.2.1 (рівень API 17). Цей формат забезпечує чудове стиснення без втрат і втрат для зображень в Інтернеті. Використовуючи WebP, розробники можуть створювати менші та багатші зображення. Файли зображень WebP без втрат у середньому на 26% менші за PNG. Ці файли зображень також підтримують прозорість (також відомий як альфа-канал) за ціною лише на 22% більше байтів. Зображення WebP із втратами на 25-34% менше, ніж порівняльні зображення JPG за еквівалентних показників якості SSIM. У випадках, коли допустиме стиснення RGB із втратами, WebP із втратами також підтримує прозо-

рість, зазвичай створюючи файли розміром у 3 рази меншим, ніж PNG. Є можливість конвертувати існуючі зображення BMP, JPG, PNG або статичні GIF у формат WebP за допомогою Android Studio.

2.2 Вибір оптимального формату

Для різних типів зображень підходять різні формати зображень. JPG і PNG мають дуже різні процеси стиснення, і вони дають зовсім різні результати. Вибір між PNG і JPG часто зводиться до складності самого зображення. На рис. 7 показано два зображення, які виходять досить різними залежно від того, яку схему стиснення застосовує розробник. Зображення ліворуч має багато дрібних деталей, тому стискається ефективніше за допомогою JPG. Зображення праворуч із смугами одного кольору стискається ефективніше за допомогою PNG.

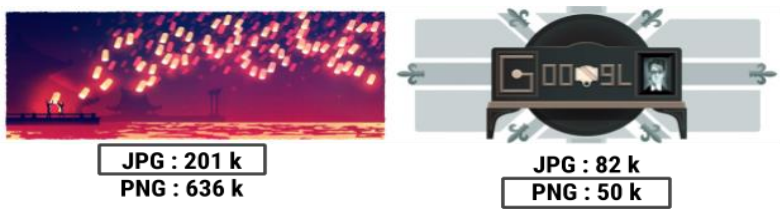

Рисунок 7 – Відповідні випадки для JPG проти PNG

WebP як формат може підтримувати як режими з втратами, так і без втрат, що робить його ідеальною заміною для PNG і JPG. Єдине, про що слід пам’ятати, це те, що він має вбудовану підтримку лише на пристроях з Android 4.2.1 (рівень API 17) і вище. На щастя, більшість пристроїв задовольняють цю вимогу.

На рис. 8 представлена проста візуалізація, яка допоможе вирішити, яку схему стиснення використовувати.

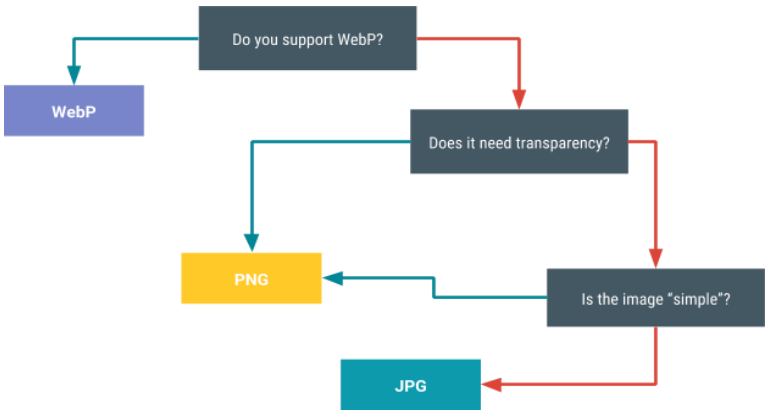

Рисунок 8 – Вибір схеми стиснення

2.3 Оптимізація якості зображень

Для досягнення оптимального балансу між стисненням та якістю зображень існує кілька методів. Один з таких методів використовує скалярні значення і може бути застосований лише до форматів JPG і WebP. Інша техніка використовує переваги бібліотеки Butteraugli і може бути використана для всіх форматів зображень. Ці методи допомагають визначити оптимальне значення якості, щоб забезпечити найкращу комбінацію стиснення і збереження якості зображення.

2.3.1 Аналіз скалярних значення (JPG і WebP)

У форматах JPG і WebP можна використовувати скалярне значення для досягнення балансу між якістю зображення і розміром файлу. Однак, визначення оптимального значення якості для зображення є важливим викликом. Встановлення занадто низького рівня якості призводить до створення компактного файлу, але може позначитися на якості зображення. Зворотньо, встановлення занадто високого рівня якості призводить до збільшення розміру файлу без значної покращення в сприйнятті користувачем. Оптимальне значення якості повинно забезпечувати належний баланс між розміром файлу та якістю зображення для досягнення найкращих результатів.

Найбільш простим рішенням є вибрати якесь немаксимальне значення та використовувати це значення. Однак значення якості впливає на кожне зображення по-різному. Хоча якість 75%, наприклад, може виглядати добре на більшості зображень, у деяких випадках це може бути не так добре. Для великих медіа-застосунків, які завантажують і повторно надсилають мільйони файлів JPG на день, ручне налаштування для кожного ресурсу є недоцільним. Вирішити цю проблему можна, вказавши кілька різних рівнів якості відповідно до категорії зображення. Наприклад, можна встановити 35% як параметр якості для мініатюр, оскільки менше зображення приховує більше артефактів стиснення[6,9].

2.3.2 Огляд технології Butteraugli

Технологія Butteraugli представляє собою бібліотеку, яка використовується для оцінки психовізуальної помилки зображення, тобто моменту, коли спостерігач починає помічати погіршення якості зображення. Основна мета цього проєкту полягає в кількісному визначенні ступеня спотворення стиснутого зображення. Шляхом аналізу різних параметрів та оцінки їх впливу на сприйняття зображення, Butteraugli допомагає визначити, наскільки помітне є зниження якості та спотворення при стисненні зображень.

Butteraugli дозволяє визначити ціль щодо візуальної якості, а потім запустити стиснення PNG, JPG, WebP із втратами та WebP без втрат. Потім можна вибрати зображення, яке є найкращим балансом розміру файлу та рівня Butteraugli. На рис. 9 показано, як Butteraugli використовувався для визначення мінімального рівня якості JPG до того, як візуальне спотворення стало достатньо високим, щоб користувач міг помітити проблему; результатом є зменшення розміру файлу приблизно на 65%.

Рисунок 9 – Зображення до і після застосування технології Butteraugli

Butteraugli дозволяє продовжувати на основі вихідних або вхідних даних. Тобто є можливість шукати найнижчий параметр якості до того, як користувач помітить помітне спотворення в отриманому зображенні, або можна повторно встановлювати рівні спотворення зображення, щоб дізнатися відповідні рівні якості.

2.4 Розгляд розмірів зображень

Зберігати лише одну роздільну здатність зображення на сервері спокусливо. Коли пристрій отримує доступ до зображення, сервер обслуговує його в цій одній роздільній здатності та залишає пристрій зменшувати розмір.

Це рішення зручне для розробника, але потенційно болюче для користувача, оскільки рішення змушує користувача завантажувати набагато більше даних, ніж йому потрібно. Натомість слід зберігати кілька розмірів зображень і обслуговувати розмір, який найбільше підходить для конкретного випадку використання. Наприклад, для мініатюри показ фактичного мініатюрного зображення замість обслуговування та зменшення повнорозмірної версії споживає набагато меншу пропускну здатність мережі

Такий підхід покращує швидкість завантаження та є менш витратним для користувачів, які можуть використовувати тарифні плани з обмеженим або

обмеженим обсягом даних. Подібні дії також призводять до того, що зображення займає менше місця на пристрої та в основній пам’яті. У випадку великих зображень, таких як 4K, цей підхід також позбавляє пристрій від необхідності змінювати розміри зображень перед їх завантаженням.

Реалізація цього підходу вимагає, щоб була серверна служба зображень, щоб надавати зображення з різною роздільною здатністю з належним кешуванням. Існують служби, які можуть допомогти з цим завданням.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

З тих пір, як люди навчилися спілкуватися та обмінюватися інформацією, вони зробили все можливе, щоб зменшити довжину або розмір інформації. До епохи цифрових технологій використовувалися такі методи, як азбука Морзе. Пізніше з'явилися телефони, а передача голосу зазнала нововведень, таких як відсікання високих частот. Швидка перемотка вперед до сучасної ери – зараз маємо справу з інформацією в цифровій формі, швидкість, правдивість і обсяг якої зростають експоненціально. У результаті стиснення даних стало необхідним для ефективного зберігання та передачі.

Зберігання, керування та передача даних стає важливим для передачі даних та інших керованих даними рішень. Це пояснюється тим, що незалежно від ступеня розвитку комп'ютерного обладнання (RAM, ROM, GPU) і форм зв'язку (Інтернет), цих ресурсів мало. Щоб ефективно використовувати ці ресурси, дані часто потрібно стискати, тобто зменшувати до меншого розміру без втрати будь-якої або мінімальної втрати інформації. Можна стискати різні типи даних. Це включає числа, текст, відео, зображення, аудіо або навіть програми та програмне забезпечення. Ці типи даних можна зменшувати в різних співвідношеннях, наприклад 2:1, що означає, що файл даних розміром 100 МБ може займати лише 50 МБ дискового простору після стиснення. Це стиснення, також відоме як ущільнення, виконується за допомогою різних методів стиснення. Методи стиснення даних у цифровому зв'язку стосуються використання спеціальних формул і ретельно розроблених алгоритмів, які використовуються програмним забезпеченням або програмою стиснення для зменшення розміру різних типів даних. Існують певні типи таких технік.[10].

Стиснення даних можна виконати, використовуючи менші рядки бітів (0 і 1) замість вихідного рядка та використовуючи «словник» для розпакування даних, якщо потрібно. Інші методи включають введення покажчиків (посилань) на рядок бітів, з якими знайома програма стиснення, або видалення зайвих символів. Для відео стиснення можна досягти, пропускаючи кожен 3-й

кадр, оскільки це призведе (як можна собі уявити) до зменшення розміру файлу на 1/3. Усе таке стиснення може значно зменшити розмір даних (у випадках до 70% або більше без втрати будь-яких значних даних). Формати стиснення, такі як ZIP, GZIP тощо, використовуються під час передачі даних через Інтернет. Використання методів стиснення даних у цифровому зв'язку значно допомагає зменшити час передачі файлу, вартість зберігання та трафік у мережі.

Проєктування застосунку для реалізації методів стиснення графічних форматів даних включає такі етапи, як: функціональність, архітектура, вибір графічних форматів, реалізація методів стиснення, інтерфейс користувача, тестування та оптимізація та підтримка.

На рис. 10 наведено графічну схему, що представляє етапи проєктування застосунку для реалізації методів стиснення графічних форматів даних.

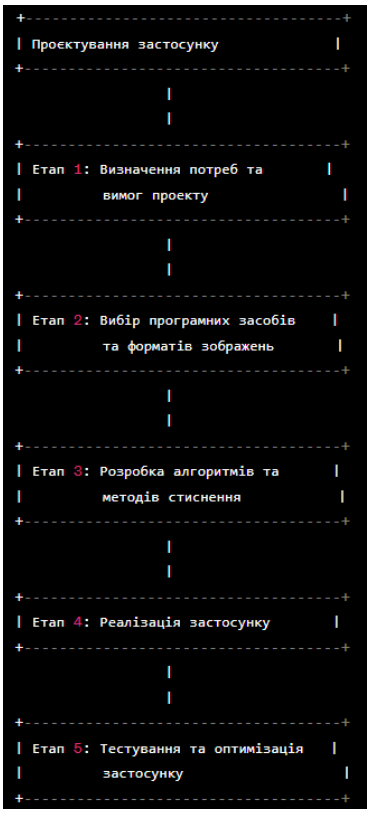

Рисунок 10 – Схема етапів проєктування застосунку для реалізації методів стиснення графічних форматів даних

На першому етапі відбувається, визначення основних функцій, які застосунок повинен виконувати, таких як завантаження зображень, вибір методу стиснення, стиснення зображень, збереження стиснутих зображень тощо. На другому етапі відбувається, розробка архітектури застосунку, включаючи компоненти, модулі та їх взаємозв'язки. Наприклад, можна розділити застосунок на модуль завантаження зображень, модуль стиснення та модуль збереження. Вивчення різних графічних форматів даних, таких як JPEG, PNG, WebP, AVIF тощо. Оцінка їх особливостей, переваг та обмежень для вибору найбільш підходящих форматів для реалізації стиснення. На третьому етапі відбувається, розробка алгоритмів та логіки для реалізації методів стиснення графічних форматів. Це може включати використання різних алгоритмів стиснення, таких як Huffman-кодування, DCT (Discrete Cosine Transform), вейвлет-перетворення та інших. Розробка зручного інтерфейсу користувача, який дозволяє вибирати зображення, виконувати стиснення та переглядати результати, відбувається на четвертому етапі. Інтерфейс повинен бути інтуїтивно зрозумілим та забезпечувати зручність використання для користувача. На п’ятому етапі відбувається, проведення тестування застосунку для перевірки правильності та ефективності реалізованих методів стиснення. Тестування може включати випробування на різних зображеннях з різними форматами та перевірку отриманих результатів. Перевірка можливості оптимізації роботи застосунку для забезпечення швидкості та ефективності стиснення. Крім того, важливо забезпечити підтримку та оновлення застосунку, включаючи виправлення помилок та вдосконалення функціональності.

3.1 Огляд методів стиснення даних

Існують два поширені типи, які завжди виділяються:

- з втратами;
- без втрат.

Щоб зрозуміти техніку стиснення з втратами, необхідно спочатку зрозуміти різницю між даними та інформацією. Дані – це необроблена, часто неорганізована колекція фактів або значень і може означати числа, текст, символи тощо. З іншого боку, інформація вносить контекст, ретельно організовуючи факти.

Щоб пояснити це в контексті, чорно-біле зображення розміром 4×6 дюймів із роздільною здатністю 100 dpi (точок на дюйм) матиме 2 40 000 пікселів. Кожен із цих пікселів містить дані у вигляді числа від 0 до 255, що відображає щільність пікселів (0 означає чорний колір, а 255 – білий).

Це зображення в цілому може містити певну інформацію. Якщо показати зображення в 50 dpi, тобто в 60 000 пікселів, дані, необхідні для збереження зображення, зменшяться, і, можливо, якість також, але інформація залишиться незмінною. Лише після значної втрати даних є можливість втратити інформацію. Нижче наведено пояснення того, як це працює (рис. 11).

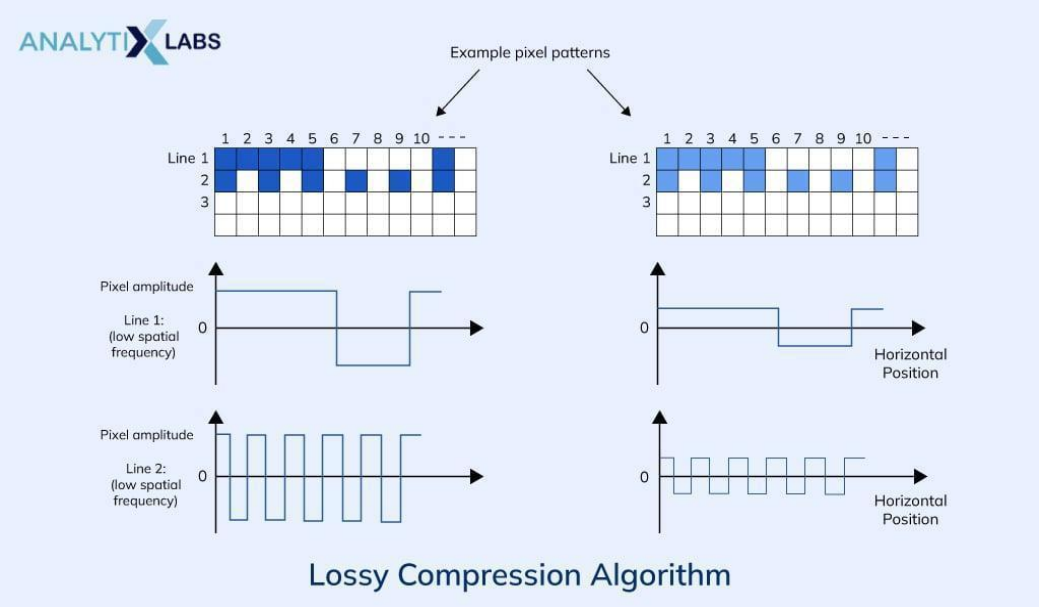

Рисунок 11 – Приклад роботи стиснення

Маючи наведене вище розуміння різниці між даними та інформацією, тепер можна зрозуміти стиснення з втратами. Як випливає з назви, стиснення

з втратами втрачає дані, тобто позбавляється від них, щоб зменшити розмір даних [11].

Перевага стиснення з втратами полягає в тому, що воно відносно швидке, може значно зменшити розмір файлу, і користувач може вибрати рівень стиснення. Це корисно для стиснення таких даних, як зображення, відео та навіть аудіо, використовуючи обмеження людського чуття. Це через обмеження очей і вух, оскільки вони не можуть відчути різницю в якості зображення та звуку до певного моменту.

Недоліком із втратами даних є те, що розпакування даних, стиснутих із втратами, не повертає ті самі дані (щодо якості, розміру тощо). Тим не менш, він буде містити подібну інформацію (це, насправді, корисно в деяких випадках, таких як потокове передавання або завантаження вмісту в Інтернеті). Однак, з іншого боку, постійне завантаження та завантаження файлу може стиснути та, як наслідок, спотворити його до межі розпізнавання, що спричинить постійну втрату інформації. Подібним чином, якщо користувач використовує суворий рівень стиснення, вихідний файл може бути не близьким до оригінального вхідного файлу.

Стиснення без втрат, на відміну від стиснення з втратами, не видаляє жодних даних; замість цього він перетворює його, щоб зменшити його розмір. Щоб зрозуміти концепцію, далі приведено простий приклад. Є уривок тексту, де досить часто повторюється слово «тому що». Термін складається із семи літер, і, використовуючи його скорочену версію, як-от «bcz», можна трансформувати текст. Цю інформацію про заміну «тому що» на «bcz» можна зберегти в словнику для подальшого використання (під час декомпресії).

У той час як стиснення з втратами видаляє зайві або непомітні фрагменти даних, щоб зменшити розмір, стиснення без втрат перетворює їх шляхом кодування за допомогою певної формули чи логіки [12]. Ось як працює стиснення без втрат (рис. 12).

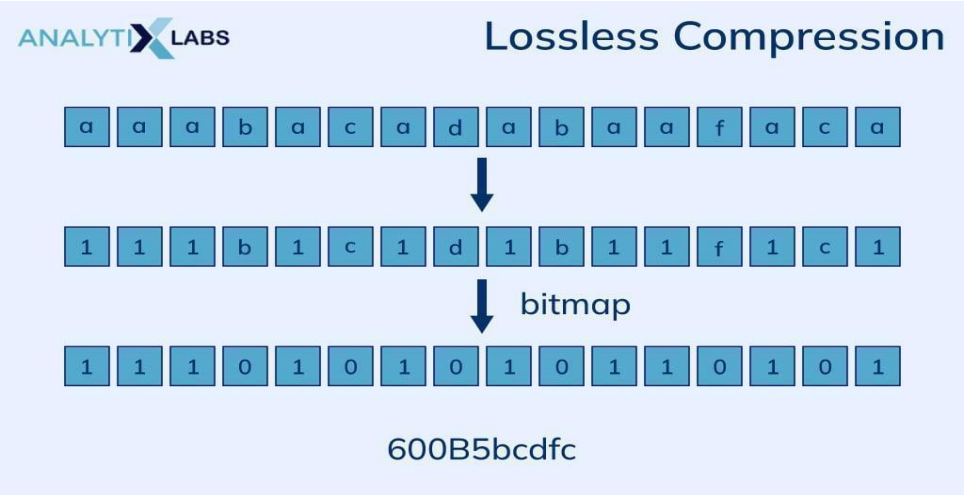

Рисунок 12 – Приклад роботи стиснення без страт

До переваг можна віднести те, що існують типи даних, де стиснення з втратами неможливе. Наприклад, у електронній таблиці, програмному забезпеченні, програмі чи будь-яких даних, що складаються з фактичного тексту чи чисел, втрата не може працювати, оскільки кожне число може бути важливим і не може вважатися зайвим, оскільки будь-яке зменшення негайно призведе до втрати інформації. Тут стиснення без втрат стає вирішальним, оскільки після декомпресії файл можна відновити до початкового стану без втрати даних.

Недоліком, можна назвати те що існує обмеження на стиснення даних. Якщо дані вже стиснуті, їхнє повторне стиснення призведе до незначного зменшення їх розміру або зовсім не зменшиться. Крім того, він менш ефективний проти файлів більшого розміру.

3.2 Переваги та недоліки методів стиснення даних

Є кілька переваг використання різних методів стиснення даних, розглянутих вище, таких як:

– зменшує дисковий простір, який займає файл;
– читання та запис файлів можна зробити швидко;
– збільшує швидкість передачі файлів через Інтернет та інші мережі.

Навіть маючи низку переваг методів стиснення даних, існує компроміс, оскільки вартість завжди пов’язана зі стисненням файлу. Ця вартість призводить до певних недоліків, таких як:

- час обробки складних алгоритмів стиснення даних може бути дуже високим, особливо якщо дані великі;
- деякі алгоритми стиснення потребують ресурсів і можуть спричинити брак пам’яті машини;
- існує залежність від програмного забезпечення, яке розпаковує стислі файли;
- пов’язана вартість стиснення також може бути грошовою, оскільки певне програмне забезпечення вимагає сплати ліцензійної плати;
- під час процесів декомпресії можуть виникнути проблеми з несумісністю;
- будь-яка помилка під час передачі стиснених даних може призвести до значної втрати інформації.

3.3 Опис моделі стиснення даних

У разі, якщо посилатись на дослідницьку статтю або техніку стиснення технічних даних у форматі pdf можна знайти численні типи моделей стиснення даних, які використовують різні алгоритми стиснення, що стосуються двох описаних вище методів стиснення (рис. 13).

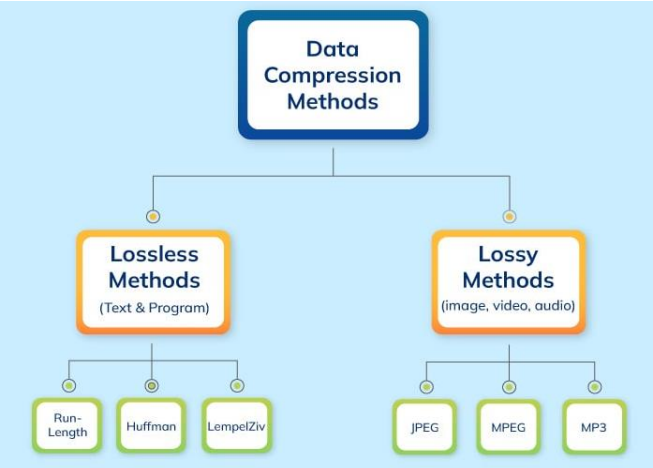

Рисунок 13 – Типи моделей стиснення даних, які використовують різні алгоритми стиснення

3.4 Опис моделі безвтратного стиснення

Модель безвтратного стиснення є одним з підходів до ефективного зменшення розміру даних без втрати інформації. Вона базується на алгоритмах, які забезпечують точне відновлення початкових даних без будь-яких змін чи втрат. Опис моделі безвтратного стиснення включає докладне вивчення та огляд різних методів та алгоритмів, що застосовуються для досягнення максимальної стисненості без втрати інформації. Деякі з популярних алгоритмів безвтратного стиснення включають:

- алгоритми Хаффмана – ці алгоритми використовуються для створення оптимальних префіксних кодів, де кожному символу присвоюється унікальний бітовий код на основі його частоти входження;
- алгоритми Лемпеля-Зіва (LZ) – ці алгоритми використовуються для створення словникових кодів, де повторювані фрагменти даних замінюються на вказівники на попередні входження цих фрагментів;
- алгоритми арифметичного кодування – ці алгоритми використовують математичні моделі для представлення даних з більшою точністю, використовуючи раціональні числа;

– алгоритми проголошення без втрат – ці алгоритми використовуються для стиснення даних шляхом виявлення та заміни повторюваних шаблонів чи послідовностей даних.

Опис моделі безвтратного стиснення також може включати порівняльний аналіз різних алгоритмів, їхню продуктивність та рівень стиснення, а також розгляд практичних аспектів їх використання в реальних застосунках.

Модель безвтратного стиснення є важливою складовою багатьох систем обробки та передачі даних, включаючи мережі зв'язку, зберігання даних та обробку мультимедійних файлів. Розуміння та ефективне використання моделей безвтратного стиснення може допомогти забезпечити оптимальне використання ресурсів та поліпшити продуктивність системи.

Найпоширенішими моделями, заснованими на техніці без втрат, є:

– RLE (кодування довжини серії);
– кодувальник словника (LZ77, LZ78, LZR, LZW, LZSS, LZMA, LZMA2);
– прогноз за частковою відповідністю (PPM);
– здування;
– змішування вмісту;
– кодування Хаффмана;
– адаптивне кодування Хаффмана;
– кодування Шеннона Фано;
– арифметичне кодування;
– кодування Lempel Ziv Welch;
– ZСтандарт;
– Bzip2 (Burrows and Wheeler).

Найпоширеніші моделі, засновані на техніці з втратами: перетворення кодування, дискретне косинусне перетворення, дискретне вейвлет-перетворення та фрактальне стиснення.

3.5 Використання моделей на основі нейронних мереж

Деякі моделі на основі нейронної мережі також використовуються для стиснення, наприклад:

- стиснення на основі багатошарового перцептрона (MLP) (використовується для стиснення зображення);
- стиснення на основі згорткової нейронної мережі (CNN), наприклад Deep Coder (використовується для стиснення відео);
- стиснення на основі Generative AdversualNetwork (GAN) (використовується для стиснення в реальному часі).

3.6 Аналіз методів стиснення даних

Стиснення даних використовується, коли є потреба зменшити розмір даних. Стиснення зображення – деякі цифрові камери стискають зображення для ефективного зберігання. Крім того, щоб усунути зниження швидкості камери через великі необроблені зображення, виконується стиснення. Ось чому зображення часто зберігаються у форматі JPEG, який використовує техніку стиснення даних із втратою даних, порівняно з PNG, який використовує стиснення без втрат. Стиснення диска – більшість операційних систем включають певну форму стиснення для зберігання даних на диску. Хоча стиснені файли вимагають більше часу для доступу ОС, плюсом є висока швидкість обробки, досягнута цим стисненням. Архівні файли – кілька файлів можна помістити в один файл, який потім можна стиснути. Тут стають у нагоді такі формати файлів, як ZIP і RAR, і це полегшує передачу файлів і створення їх резервних копій. Стиснення аудіо – кілька років тому команда, яка працювала над стандартами аудіо- та відеосистем, визначила переваги представлення аудіоданих у цифровому вигляді. Ця група, відома як MPEG (Motion Pictures Experts Group), розробила механізм кодування аудіо-відео, відомий як MPEG-1. Найпоширені-

шою формою стиснення аудіо є mp3, тобто MPEG-1 Layer 3, де кожен наступний рівень стає складнішим і видаляє зайві дані (рис. 14). Mp3 – це стиснення з втратами, що може знизити якість; однак його аналоги без втрат, такі як аудіо у форматі WAV і FLAC, також можна використовувати. Усі ці формати використовуються для зберігання аудіо на CD/DVD, трансляції музики на веб-сайтах тощо [8].

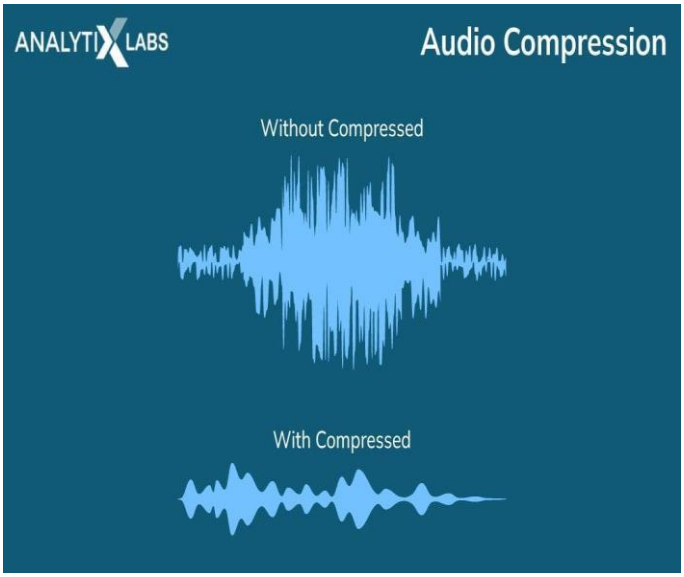

Рисунок 14 – Приклад стиснення аудіофайлу

Традиційно такі пристрої, як «Ключ Морзе» та «Приймач додаткового коду», використовувалися для передачі та отримання інформації азбукою Морзе, що є типом стиснення даних. Сьогодні різне програмне забезпечення реалізує різні типи моделей, побудованих на алгоритмах стиснення даних. Наприклад, у Linux і Windows використовуються такі програми, як gzip і ZIP відповідно, тоді як у macOS StuffIt є стандартним інструментом для стиснення даних. Для досягнення стиснення у відео використовується формат GIF (Graphics Interchange Format), а для зображень – JPEG (Joint Photographic Experts Group) [12].

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Аналіз дискретного вейвлет-перетворення Хаара Вейвлет Хаара (FHT)

Дискретне вейвлет-перетворення Хаара Вейвлет Хаара (FHT) є одним з перших відомих ортогональних вейвлетів. Деталізуюча функція у нього має вигляд прямокутних імпульсів меандру (значення 1 в інтервалі [0,0.5] і -1 в інтервалі [0.5,1]). Апроксимуюча функція (рис. 15) має значення 1 в інтервалі [0,1] і 0 за межами цього інтервалу. Вейвлети Хаара добре локалізовані в просторі, але не дуже добре локалізовані в частотної області, оскільки меандр має широкий спектр частот (теоретично нескінченний) [4].

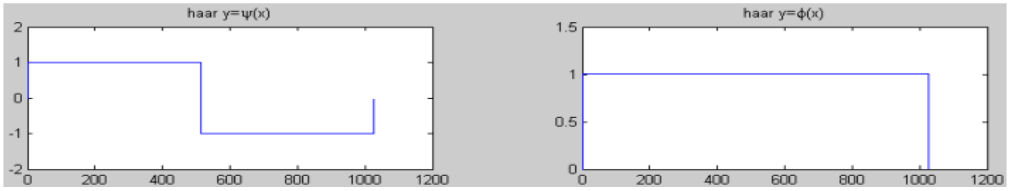

Рисунок 15 – Деталізуюча і апроксимуюча функції Вейвлета Хаара

Перетворення Хаара в загальному вигляді для одновимірного сигналу (відліків) виглядає наступним чином. Нехай є одновимірний дискретний сигнал S. Кожній парі елементів з індексами 2 j і 2 j +1, j – ∅, поставимо у відповідність два значення (4.1, 4.2):

$$a_j = \frac{s_{2j} + s_{2j+1}}{2} a_j = \frac{s_{2j} + s_{2j+1}}{2} \quad (4.1)$$

$$d_j = \frac{s_{2j} - s_{2j+1}}{2} d_j = \frac{s_{2j} - s_{2j+1}}{2} \quad (4.2)$$

де j а є апроксимирована версія сигналу s , а d_j містить деталізуючу інформацію, необхідну для відновлення сигналу s.

Нижче описано двовимірний сигнал як s-матрицю кінцевого розміру. Слід застосувати до кожного рядка матриці один крок одновимірного вейвлет перетворення. В результаті отримано дві матриці, рядки яких містять апроксимуючу і деталізуючу складові рядків вихідної матриці. До кожного стовпчика обох матриць також можна застосувати крок одновимірного перетворення. В результаті виходить чотири матриці (рис. 16). Перша є апроксимуючою складовою вихідного сигналу (огрубілим варіантом), інші три містять деталізуючу інформацію – вертикальну, горизонтальну і діагональну. Таким чином, двовимірне перетворення зводиться до композиції одновимірних перетворень.

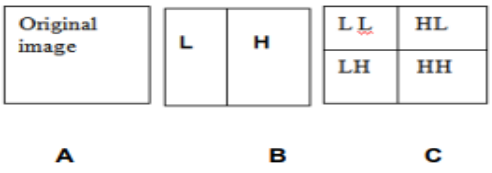

Рисунок 16 – Схема двовимірного вейвлет-перетворення

A – оригінальне зображення;

B – перший біг уздовж ряду;

C – перший пробіг уздовж колони.

Масив зображення розбивається на дві половини, що містять перетворені дані та коефіцієнти деталізації. коефіцієнти перетворених даних є результатами фільтра низьких частот, тоді як коефіцієнти деталізації є результатами фільтра високих частот. Після перетворення зображення в рядку зображення потім трансформується вздовж колонка. Цей процес повторюється до трьох ітерацій [13].

Наприклад, для s-матриці:

$$s = \begin{bmatrix} 13 & 12 & 16 & 15 \\ 15 & 14 & 11 & 19 \\ 12 & 13 & 18 & 15 \\ 11 & 14 & 17 & 16 \end{bmatrix}$$

На першому етапі при застосуванні вейвлет-перетворення до кожного рядка вихідної матриці отримуємо 2 матриці:

$$W_L = \begin{bmatrix} (13+12)/2 & (16+15)/2 \\ (15+14)/2 & (11+19)/2 \\ (12+13)/2 & (18+15)/2 \\ (11+14)/2 & (17+16)/2 \end{bmatrix} = \begin{bmatrix} 12{,}5 & 15{,}5 \\ 14{,}5 & 15 \\ 12{,}5 & 16{,}5 \\ 12{,}5 & 16{,}5 \end{bmatrix}$$

$$W_H = \begin{bmatrix} (13-12)/2 & (16-15)/2 \\ (15-14)/2 & (11-19)/2 \\ (12-13)/2 & (18-15)/2 \\ (11-14)/2 & (17-16)/2 \end{bmatrix} = \begin{bmatrix} 0{,}5 & 0{,}5 \\ 0{,}5 & -4 \\ -0{,}5 & 1{,}5 \\ -1{,}5 & 0{,}5 \end{bmatrix}$$

Далі слід застосувати перетворення Хаара до кожного стовпця матриць WL WH, та отримуємо матриці WLL, WLH, WHL, WHH.

WLL характеризує низькочастотну складову:

$$W_{LL} = \begin{bmatrix} (12{,}5+14{,}5)/2 & (15{,}5+15)/2 \\ (12{,}5+12{,}5)/2 & (16{,}5+16{,}5)/2 \end{bmatrix} = \begin{bmatrix} 13{,}5 & 15{,}25 \\ 12{,}5 & 16{,}5 \end{bmatrix}$$

$$W_{LL} = \begin{bmatrix} (12{,}5+14{,}5)/2 & (15{,}5+15)/2 \\ (12{,}5+12{,}5)/2 & (16{,}5+16{,}5)/2 \end{bmatrix} = \begin{bmatrix} 13{,}5 & 15{,}25 \\ 12{,}5 & 16{,}5 \end{bmatrix}$$

WLH – вертикальне відхилення:

$$W_{LH} = \begin{bmatrix} (12{,}5-14{,}5)/2 & (15{,}5-15)/2 \\ (12{,}5-12{,}5)/2 & (16{,}5-16{,}5)/2 \end{bmatrix} = \begin{bmatrix} -1 & 0{,}25 \\ 0 & 0 \end{bmatrix}$$

$$W_{LH} = \begin{bmatrix} (12{,}5-14{,}5)/2 & (15{,}5-15)/2 \\ (12{,}5-12{,}5)/2 & (16{,}5-16{,}5)/2 \end{bmatrix} = \begin{bmatrix} -1 & 0{,}25 \\ 0 & 0 \end{bmatrix}$$

WHL – горизонтальне відхилення:

$$W_{HL} = \begin{bmatrix} (0{,}5+0{,}5)/2 & (0{,}5-4)/2 \\ (-0{,}5-1{,}5)/2 & (1{,}5+0{,}5)/2 \end{bmatrix} = \begin{bmatrix} 0{,}5 & -1{,}75 \\ -1 & 1 \end{bmatrix}$$

$$W_{HL} = \begin{bmatrix} (0{,}5+0{,}5)/2 & (0{,}5-4)/2 \\ (-0{,}5-1{,}5)/2 & (1{,}5+0{,}5)/2 \end{bmatrix} = \begin{bmatrix} 0{,}5 & -1{,}75 \\ -1 & 1 \end{bmatrix}$$

WHH – діагональне відхилення:

$$W_{HL} = \begin{bmatrix} (0{,}5-0{,}5)/2 & (0{,}5+4)/2 \\ (-0{,}5+1{,}5)/2 & (1{,}5-0{,}5)/2 \end{bmatrix} = \begin{bmatrix} 0 & 2{,}25 \\ 0.5 & 0.5 \end{bmatrix}$$

$$W_{HL} = \begin{bmatrix} (0{,}5-0{,}5)/2 & (0{,}5+4)/2 \\ (-0{,}5+1{,}5)/2 & (1{,}5-0{,}5)/2 \end{bmatrix} = \begin{bmatrix} 0 & 2{,}25 \\ 0.5 & 0.5 \end{bmatrix}$$

Реконструкція вихідної матриці відбувається в зворотному порядку за формулами:

$$s'_{2j} = a_j + d_j s'_{2j} = a_j + d_j \tag{4.3}$$

$$s'_{2j+1} = a_j - d_j s'_{2j+1} = a_j - d_j \tag{4.4}$$

Перший етап реконструкції початкового сигналу:

$$W_L = \begin{bmatrix} (13{,}5-1) & (15{,}25+0{,}25) \\ (12{,}5+0) & (16{,}5+0) \\ (13{,}5+1) & (15{,}25-0{,}25) \\ (12{,}5-0) & (16{,}5-0) \end{bmatrix} = \begin{bmatrix} 12{,}5 & 15{,}5 \\ 12{,}5 & 16{,}5 \\ 14{,}5 & 15 \\ 12{,}5 & 16{,}5 \end{bmatrix}$$

Другий етап реконструкції початкового сигналу:

$$W_H = \begin{bmatrix} (0{,}5+0) & (-1{,}75+2{,}25) \\ (-1+0{,}5) & (1+0{,}5) \\ (0{,}5-0) & (-1{,}75-2{,}25) \\ (-1-0{,}5) & (1-0{,}5) \end{bmatrix} = \begin{bmatrix} 0{,}5 & 0{,}5 \\ -0{,}5 & 1{,}5 \\ 0{,}5 & -4 \\ -1{,}5 & 0{,}5 \end{bmatrix}$$

Таким чином, перетворення Хаара – це пара фільтрів, які поділяють сигнал на низькочастотну і високочастотну складові. Щоб отримати вихідний сигнал, потрібно просто знову об'єднати ці складові. Для стиснення отримані вейвлет-коефіцієнти кодуються тим чи іншим способом.

4.2 Аналіз дискретного косинусне перетворення Фур'є (DCT)

Аналіз спектрів – одна з основних задач цифрової обробки сигналів. Основою цифрового спектрального аналізу є дискретне перетворення Фур'є (DCT), яке переводить послідовність, яка задана в часовій області, в послідовність, відповідну компонентам спектру [14].

Дискретне перетворення Фур'є має вигляд:

$$F_n = \sum_{k=0}^{N-1} x_k e^{-j\frac{2\pi}{N}kn}, \; n = 0,\dots,N-1. \tag{4.5}$$

Зворотне перетворення:

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{j\frac{2\pi}{N}kn}, \; k = 0,\dots,N-1. \tag{4.6}$$

В матричній формі DCT виглядає:

$$F = \frac{1}{\sqrt{N}} E_N X \quad . \tag{4.7}$$

Сама матриця ядра DCT носить назву матриці дискретних експоненціальних функцій (ДЕФ). При цьому рядки матриці визначають набір ортогональних функцій або базис розкладання [15].

$$E_N = \begin{bmatrix} w^0 & w^0 & w^0 & \dots & w^0 \\ w^0 & w^1 & w^2 & \dots & w^{N-1} \\ w^0 & w^2 & w^4 & \dots & w^{2(N-1)} \\ w^0 & w^3 & w^6 & \dots & w^{3(N-1)} \\ \dots & \dots & \dots & \dots & \dots \\ w^0 & w^{N-1} & w^{2(N-1)} & \dots & w^{(N-1)(N-1)} \end{bmatrix}$$

$$E_N = \begin{bmatrix} w^0 & w^0 & w^0 & \dots & w^0 \\ w^0 & w^1 & w^2 & \dots & w^{N-1} \\ w^0 & w^2 & w^4 & \dots & w^{2(N-1)} \\ w^0 & w^3 & w^6 & \dots & w^{3(N-1)} \\ \dots & \dots & \dots & \dots & \dots \\ w^0 & w^{N-1} & w^{2(N-1)} & \dots & w^{(N-1)(N-1)} \end{bmatrix}$$

де

$$W^k = e^{-j2\pi k/N} \tag{4.8}$$

При виконанні перетворення Фур'є рядки матриці ядра задають набір ортогональних функцій, за якими виконується розкладання вихідного сигналу. Кожен елемент вектору результату визначає внесок відповідної ортогональної функції в формування вихідного сигналу.

Для перетворення Фур'є, як і для будь-якого ортогонального перетворення, матриця ядра перетворення E_N зворотна, що дозволяє виконати як пряме, так і зворотне перетворення [4]:

$$\begin{cases} F = \frac{1}{\sqrt{N}} E_N X. \\ X = \frac{1}{\sqrt{N}} E_N^{-1} F. \end{cases} \quad (4.9)$$

При цьому матриця ядра зворотного перетворення має властивість $E_N^{-1} \equiv E_N^*$, де E_N^* – ермітово-спряжена матриця. Поняття ермітово-спряженої матриця передбачає, що матриця зворотного перетворення є транспонованою по відношенню до E_N і елементи її є комплексно спряжені до W_{ij}.

Нижче представлені основні властивості матриці ядра перетворення E_N. Коефіцієнти такої матриці мають наступні властивості:

– симетричність $W_{ij} \equiv W_{ji}$;
– мультиплікативність $W^{k+m} \equiv W^k * W^m$;
– циклічність: $W^{k+N} \equiv W^k$ або $W^{-k} \equiv W^{N-k}$.

4.3 Вибір мови програмування та середовища розробки

В якості мови програмування обрано мову C#. На даний момент мова програмування C# є однією з найпотужніших, швидко зростаючих і популярних мов в IT-індустрії. В даний час на ній написані різні застосунки: від невеликих настільних програм до великих веб-порталів і веб-сервісів, що обслуговують мільйони користувачів кожен день.

У порівнянні з іншими мовами C# досить молода, але в той же час вона вже пройшла довгий шлях. Перша версія мови прийшла з випуском Microsoft Visual Studio .Net в лютому 2002. C# – мова з C-подібним синтаксисом і схожа на C++ і Java. Тому, якщо розробник знайомий з однією з цих мов, то буде простіше освоїти C#. C# є об'єктно-орієнтованою і в цьому відношенні багато

чого було перейнято у Java і C++. Наприклад, C# підтримує поліморфізм, успадкування, перевантаження оператора та статичний ввід. Об'єктно-орієнтований підхід дозволяє вирішувати завдання побудови великих, але в той же час гнучких, масштабованих і розширюваних застосунків. І C# продовжує активно розвиватися, і з кожною новою версією є все більше і більше цікавих функцій, таких як лямбда, динамічні посилання, асинхронні методи.

Мова C# є найбільш відомою новинкою в області створення мов програмування. З'явившись на світло в надрах Microsoft, вона з перших своїх кроків отримала потужну підтримку. Мова визнана міжнародним співтовариством. Компілятори Microsoft будуються відповідно до міжнародних стандартів мови. Керівником групи, що створює мову C#, є співробітник Microsoft Андреас Хейлсберг. Як зазначав сам Андреас Хейлсберг, C# створювалась як мова компонентного програмування, і в цьому одне з головних переваг мови, спрямоване на можливість повторного використання створених компонентів. Створювані компілятором компоненти є саме документовані, крім коду містять метаінформацію, що описує компоненти, і тому можуть виконуватися на різних платформах. В якості мови програмування для виконання кваліфікаційної роботи бакалавра було обрано мову C#. Слід відзначити наступні важливі фактори, які вплинули на вибір даної мови програмування:

- C# створювалась розвивається паралельно з каркасом Framework.Net і в повній мірі враховує всі його можливості;
- C# є повністю об'єктно-орієнтованою мовою;
- C# є потужною об'єктною мовою з можливостями успадкування та універсалізації;
- C# є спадкоємцем мови C++ (загальний синтаксис, загальні оператори мови полегшують перехід від мови C++ до C #. Зберігши основні риси свого батька, мова стала простіше і надійніше).

Завдяки каркасу Framework.Net, який став надбудовою над операційною системою, програмісти C# отримують переваги роботи з віртуальною машиною. Framework.Net представляє потужну платформу для створення застосунків. Можна виділити такі основні можливості:

- багатомовна підтримка;
- кросплатформеність;
- потужна бібліотека класів;
- різноманітні технології.

Платформа заснована на загальній мові виконання (CLR), яка дозволяє .NET підтримувати декілька мов: разом з C#, це також VB.NET, C++, F #, а також різні діалекти інших мов, пов'язаних з .NET, такі як Delphi.NET. Під час компіляції код на будь-якій з цих мов складається в збірку на загальну проміжну мову (CIL), свого роду .NET асемблера. Таким чином, можна зробити окремі модулі одного і того ж застосунка в окремих мовах.

.NET – це портативна платформа (з деякими обмеженнями). Наприклад, остання версія .NET Framework наразі підтримується на більшості сучасних операційних систем Windows (Windows 10/8.1/8/7/Vista). І з моно проєкту, можна створювати програми, які будуть працювати на інших операційних системах Linux, у тому числі Android і IOS мобільних платформах [9]. .NET – це бібліотека класів, яка є спільною для всіх підтримуваних мов. І будь-яка програма, написана в C# – текстовий редактор, чат, або складний веб-сайт в деякому роді, буде працювати з бібліотеками .NET. Загальні мовні виконавчі файли та бібліотека базових класів є основою для цілого стеку технологій, розробники яких можуть використовувати їх для створення застосунків. Наприклад, технологія ADO.NET призначена для роботи з базами даних у цьому стеку технологій, для створення графічних програм можна з – багатою інтерфейсною технологією WPF. Для створення Websites – ASP.NET тощо.

Також примітна особливість C# та .NET Framework, автоматична колекція сміття. Це означає, що в більшості випадків не потрібно дбати про звільнення пам'яті, на відміну від C++. Вищезгадане мовне середовище буде викликати колектор сміття і очищати пам'ять.

Середовищем розробки обрано MS Visual Studio Community Edition 2019 року, яка є безкоштовною середовищем розробки. Нижче перераховані її основні переваги:

- інтуїтивний стиль кодування;
- більш висока швидкість розробки;
- можливості налагодження (вбудований відлагоджувач може працювати як відлагоджувач рівня вихідного коду, так і відлагоджувач машинного рівня);
- можливість управління проєктом;
- вбудована функція управління вихідним кодом з підтримкою технології;
- IntelliSence – технологія авто доповнення Microsoft, дописує назву функції при введенні початкових літер.

4.4 Опис реалізації програмного коду

Після запуску програми на екрані з’являється головне вікно (рис. 17).

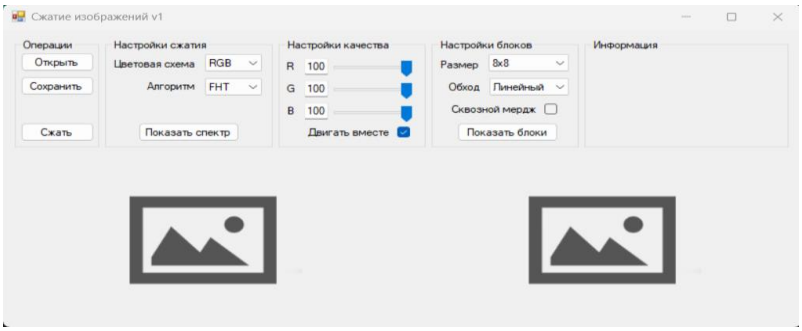

Рисунок 17 – Головне вікно програми

У верхній частині вікна знаходиться головне меню, яке складається з наступних розділів: «Операції», «Налаштування стиснення», «Налаштування якості», «Налаштування блоків» та «Інформація».

Розділ «Операції» складається з таких елементів, як:

– «Відкрити» – відчиняє вікно для вибору зображення;
– «Зберегти» – відчиняє вікно для збереження зображення;
– «Стиснути» – починає операцію стиснення.

Розділ «Налаштування стиснення» складається з таких елементів, як:

– «Кольорова схема» – дозволяє обрати одну з кольорових схем: RGB, YCbCr, HSV;
– «Алгоритм» – дозволяє обрати однин з алгоритмів стиснення: FHT, DCT;
– «Показати спектр» – показує спектр зображення розбитого на блоки у яких можна зробити стиснення (рис. 18).

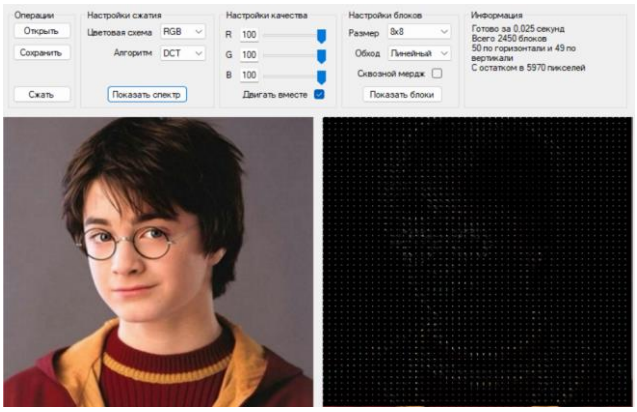

Рисунок 18 – Спектр зображення

«Налаштування якості» – дозволяє встановити якість зображення після стиснення шляхом зміни кольорової насиченості для обраної схеми.

Розділ «Налаштування блоків» складається з таких елементів, як:

– «Розмір» – дозволяє встановити розмір блоків при стисненні від 2x2 до 512x512;

– «Обхід» – дозволяє обрати тип обходу блоків: «Лінійний» або «Зиг-заг»;
– «Наскрізний мердж» – дозволяє встановити режим злиття блоків;
– «Показати блоки» – показує блоки як буде розділене зображення (рис. 19) та виводить інформацію про блоки.

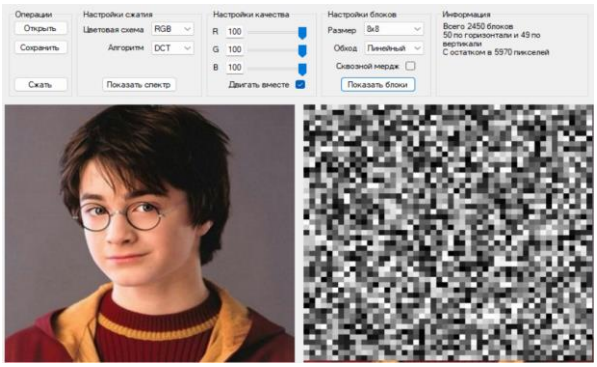

Рисунок 19 – Блоки зображення при стисненні

«Інформація» – блок у якому відображається інформація про спектр, блоки або стиснення (час виконання операція, ступінь стиснення, кількість блоків та інші). На рис. 20 представлено результат стиснення.

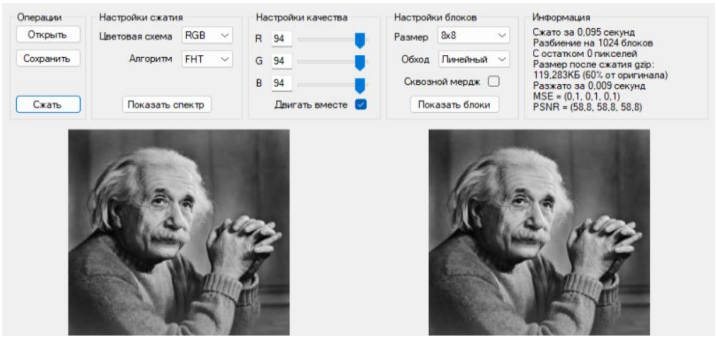

Рисунок 20 – Результат стиснення

Уривок коду для функції переводу одного піксела в іншу колірну схему представлено в додатку А.

ВИСНОВКИ

У роботі проаналізовані кілька різних методів стиснення зображень. Більшість методів стиснення зображень можна розділити на одну з двох категорій. Важко порівнювати продуктивність різних методів стиснення, якщо набори даних і показники продуктивності, які використовуються, не є еквівалентними. Певні програми, зокрема ті, що мають справу з технологіями безпеки, можуть ефективно використовувати деякі з цих стратегій. Після аналізу кожного методу було виявлено, що стратегії стиснення зображень без втрат є кращими за методи стиснення зображень із втратами з точки зору ефективності. Стиснення з втратами призводить до більшого співвідношення, ніж стиснення без втрат. Для досягнення оптимального значення якості зображення використовуються різні методи. Один з них використовує скалярні значення і підходить для форматів зображень, таких як JPG і WebP. Цей метод дозволяє збалансувати якість і розмір файлу, але потребує правильного вибору значення якості для досягнення оптимальних результатів. Також, технологія Butteraugli, який є бібліотекою для перевірки психовізуальної помилки зображень, дозволяє кількісно оцінити ступінь спотворення стисненого зображення. Це допомагає визначити момент, коли глядач починає помічати погіршення якості зображення, що є важливим фактором при виборі оптимальних параметрів стиснення. Дана робота спрямована на забезпечення швидкого та ефективного стиснення даних з використанням комбінації незалежного від вмісту стиснення даних. В одному аспекті роботи спосіб стиснення даних містить етапи:

- аналізують даних вхідного потоку даних для ідентифікації типу даних блоку даних, причому потік вхідних даних містить безліч різнорідних типів даних;
- виконання залежного від вмісту стиснення даних для блоку даних, якщо тип даних блоку даних ідентифіковано;
- виконання незалежного від вмісту стиснення даних для блоку даних, якщо тип даних блоку даних не визначено.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Noura Q. E., Mohammed S. A. Features of Application of Data Compression Methods. QALAAI ZANIST JOURNAL. 2021. №6(3), P. 969–983. URL: https://doi.org/10.25212/lfu.qzj.6.3.34 (дата звернення 21.04.2023)
2. Резуненко А.А. Способы кодирования изображений независимыми блоками. П.: ПВІЗ, 2004.
3. Bracewell R.N. The Fourier Transform and its Applications. McGRAW-HILL, International Editions. 2000.
4. Grochenig K., Madych W.R. Multiresolution Analysis, Haar Bases, and Self-Similar Tilings of Rn . IEEE Trans. Inform. Theory. 1992. №38. P. 556–568.
5. ITU-T Rec T.800. URL: https://www.itu.int/rec/T-REC-T.800/en (дата звернення 15.03.2023)
6. ITU-T Rec T.808. URL: https://www.itu.int/rec/T-REC-T.808/en (дата звернення 06.04.2023)
7. Johnston, N., Vincent, D., Minnen, D., Covell, M., Singh, S., Chinen, T., Toderici, G. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. P. 4385-4393.
8. Working with GeoMedia Professional, Appendix E «Raster Information». Compression Techniques. DJA080791, SJ**690. V. 6.0.
9. Starosolski R. Application of reversible denoising and lifting steps to DWT in lossless JPEG 2000 for improved bitrates. Signal Process Image Commun. 2015, 39(A): 249±63.
10. Королев А.В. Теоретические основы компактного представления изображений на основе устранения версификационной избыточ ности. Х.: ХВУ, 2003. 329 с.
11. Starosolski R. Reversible Denoising and Lifting Based Color Component Transformation for Lossless Image Compression. 2016.

12. Королёва Н.А., Стрюк А. Ю. Способ сжатия видеоданных, основанный на дискретном волновом преобразовании. Інформаційно-керуючі системи на залізничному транспорті, 2000. № 6. С. 67–70.
13. Schiopu, I., Munteanu, A. Residual-error prediction based on deep learning for lossless image compression. Electronics Letters. 2018. 54(17), P.1032–1034.
14. Різуненко А.О. Методи та інформаційна технологія стиску зображень в автоматизованих системах на основі вейвлет-перетворень. Х.: НАКУ ХАІ, 2005. 190 с.
15. DCT. URL: https://www.researchgate.net/profile/Ivana-Jovanovic-10/publication/253906898/figure/fig1/AS:298198519304194@1448107475630/Example-of-an-overcomplete-set-of-81-2D-DCT-basis-functions-for-patches-of-size-8-8.png (дата звернення 10.02.2023)
16. Wickerhauser M. Acoustic signal compression with wavelet packets. In Chui C. ed. Wavelets :A Tutorial in Theory and Applications, Academic Press, Boston. 1992. P.679–700.

Додаток А –
функції переводу одного піксела в іншу колірну схему

```
            public static void RGB_to_YCbCr(float r, float g, float b, out
float y, out float cb, out float cr) {
                y = (float)(0+(0.229*r)+(0.587*g)+(0.114*b));
                cb = (float)(128-(0.168736*r)-(0.331264*g)+(0.5)*b);
                cr = (float)(128+(0.5*r)-(0.418688*g)-(0.081312*b));
            }

            public static void RGB_to_HSV(float r, float g, float b, out
float h, out float s, out float v)
            {
                r /= 255.0f;
                g /= 255.0f;
                b /= 255.0f;

                // h:0-360.0, s:0.0-1.0, v:0.0-1.0
          float max = Math.Max(r, Math.Max(g, b));
                float min = Math.Min(r, Math.Min(g, b));

                v = max;

                if (max == 0.0f) {
                    s = 0;
                    h = 0;
                }
                else if (max - min == 0.0f) {
                    s = 0;
                    h = 0;
                }
                else {
                    s = (max - min) / max;
                    if (max == r) {
                        h = 60 * ((g - b) / (max - min)) + 0;
                    }
                    else if (max == g) {
                        h = 60 * ((b - r) / (max - min)) + 120;
                    }
                    else {
                        h = 60 * ((r - g) / (max - min)) + 240;
                    }
                }

                if (h < 0) h += 360.0f;

                h /= 2;   // dst_h : 0-180
                s *= 255; // dst_s : 0-255
                v *= 255; // dst_v : 0-255
            }

            public static void YCbCr_to_RGB(float y, float cb, float cr, out
float r, out float g, out float b) {
```

```
                r = (float)(y+1.402*(cr-128));
                g = (float)(y-0.34414*(cb-128)-0.71414*(cr-128));
                b = (float)(y+1.772*(cb-128));
            }

            public static void YCbCr_to_HSV(float y, float cb, float cr, out
float h, out float s, out float v) {
                float r, g, b;
                YCbCr_to_RGB(y, cb, cr, out r, out g, out b);
                RGB_to_HSV(r, g, b, out h, out s, out v);
            }

            public static void HSV_to_YCbCr(float h, float s, float v, out
float y, out float cb, out float cr) {
                float r, g, b;
                HSV_to_RGB(h, s, v, out r, out g, out b);
                RGB_to_YCbCr(r, g, b, out y, out cb, out cr);
            }

            public static void HSV_to_RGB(float h, float s, float v, out
float r, out float g, out float b) {
                h *= 2.0f; // 0-360
                s /= 255.0f; // 0.0-1.0
                v /= 255.0f; // 0.0-1.0
                r = g = b = 0;

                int hi = (int)(h / 60.0f) % 6;
                float f = (h / 60.0f) - hi;
                float p = v * (1.0f - s);
                float q = v * (1.0f - s * f);
                float t = v * (1.0f - s * (1.0f - f));

                switch (hi) {
                    case 0: r = v; g = t; b = p; break;
                    case 1: r = q; g = v; b = p; break;
                    case 2: r = p; g = v; b = t; break;
                    case 3: r = p; g = q; b = v; break;
                    case 4: r = t; g = p; b = v; break;
                    case 5: r = v; g = p; b = q; break;
                }
                r *= 255; // dst_r : 0-255
                g *= 255; // dst_r : 0-255
                b *= 255; // dst_r : 0-255
        }
```

Програмний код для згортки зображення

```
            public static byte[] compressImageValues(
                float[] imageValues,
                int width,
                int height,
                string colorScheme,
                string method,
                Tuple<float,float,float> quality,
                int blockSize,
                string travelMode,
```

```
                bool crossMerge
            )
            {
                if (imageValues == null)
                    return null;

                //узнаем размер остатка
                long   remainderSize   =   getRemainderSize(width,   height,
blockSize);

                //бьем изображение на блоки
                var blocks = splitImageValuesIntoBlocks(imageValues, width,
height, blockSize);
                int pixelsInBlock = blockSize * blockSize;
                var elementsAmountInOneBlock = pixelsInBlock * 3;
                var  elementsAmountInAllBlocks  =  elementsAmountInOneBlock  *
blocks.Length;

                //применяем прямое преобразоваение к каждому блоку
                switch (method) {
                    case "FHT":
                        MathUtils.mapFunctionToEachBlock(MathUtils.fht_2d,
blocks);
                        break;
                    case "DCT":
                        MathUtils.mapFunctionToEachBlock(MathUtils.dct_2d,
blocks);
                        break;
                }

                // Обнуляем малозначимые коэффициенты
                // узнаем, сколько элементов должно быть обнулено
                int[] q = new int[3];
                q[0] = (int)MathUtils.MapInterval(100-quality.Item1, 0, 100,
0, pixelsInBlock);
                q[1] = (int)MathUtils.MapInterval(100-quality.Item2, 0, 100,
0, pixelsInBlock);
                q[2] = (int)MathUtils.MapInterval(100-quality.Item3, 0, 100,
0, pixelsInBlock);

                switch (method) {
                    case "FHT": {
                        Parallel.For(0, blocks.Length, i => {
                       //В преобразовни Хаара малозначащие коэффициенты - это те,
                       //которые близки к нулю.

       //Поэтому для каждого канала создадим список пар вида (индекс, значение)
       //отсортируем по модулю значения и оставим только индексы и обнулим
       // необходимое количество элементов по первым q[j] из этих индексов

                            float[][] channels = new float[3][];
                            for (int j = 0; j < 3; ++j) {
                                channels[j] = new float[pixelsInBlock];
                                for (int k = 0; k < pixelsInBlock; ++k)
                                    channels[j][k] = blocks[i][k*3+j];
                            }
```

```
                            int[][] indexes = new int[3][];
                            for (int j = 0; j < 3; ++j)
                                indexes[j] = channels[j].Select((elem, ind)
=> new { Index = ind, Value = elem })
                                                        .OrderBy(x        =>
Math.Abs(x.Value))
                                                        .Take(q[j])
                                                        .Select(x         =>
x.Index)
                                                        .ToArray();

                            for (int j = 0; j < 3; ++j)
                                for (int k = 0; k < q[j]; ++k)
                                    blocks[i][indexes[j][k]*3 + j] = 0;
                        });
                        break;
                    }
                    case "DCT": {
      //малозначимые коэффициенты скапливаются в правом нижнем углу,
      //так что обнуляем начиная оттуда и пока не обнулим столько, сколько надо
                        int[] indexes = ZigZag.getIndexes(blockSize);
                        Parallel.For(0, blocks.Length, i => {
                            for (int channel = 0; channel < 3; ++channel)
                                for (int j = 0; j < q[channel]; ++j)
                                    blocks[i][indexes[pixelsInBlock - 1 - j]
* 3 + channel] = 0;
                        });
                        break;
                    }
                }


                //если нужно - обходим блоки зиг-загом
                if (travelMode == "Зиг-заг") {
                    ZigZag.precompute(blockSize);
                    Parallel.For(0, blocks.Length, i => {
                        blocks[i] = toZigzag(blocks[i]);
                    });
                }

                float[] values = new float[elementsAmountInAllBlocks];
                //укладываем блоки
                if (crossMerge){ //с кросс-мерджем
                    crossMergeBlocks(blocks, values);
                }
                else { //без кросс-мерджа
                    Parallel.For(0, blocks.Length, block_i => {
                        for (int i = 0; i < elementsAmountInOneBlock; ++i)
                            values[block_i * elementsAmountInOneBlock + i] =
blocks[block_i][i];
                    });
                }

                // если есть остаток
                if (remainderSize != 0) {
```

```
                    //то получаем его
                    float[] remainder = getRemainder(imageValues, width,
height, blockSize);

                    //увеличиваем values, что бы он влез
                    Array.Resize(ref values, values.Length +
remainder.Length);

                    //дописываем его в конец
                    for (int i = 0; i < remainder.Length; ++i)
                        values[elementsAmountInAllBlocks + i] = remainder[i];
                }

                //сжимаем
                byte[] compressedBytes =
compressBytes(floatArrayAsByteArray(values));

                //сюда запишем итоговый .compresed (то есть заголовок и сжатые
байты(уложенные блоки + остаток))
                byte[] resultBytes = new byte[Constants.HeaderSize +
compressedBytes.Length + remainderSize*4];

                //записываем заголовок
                writeHeader_v1(resultBytes, width, height, colorScheme,
method, blockSize, travelMode, crossMerge);

                //дописываем сжатые данные
                for (int i = 0; i < compressedBytes.Length; ++i)
                    resultBytes[Constants.HeaderSize + i] =
compressedBytes[i];

                return resultBytes;
            }
```

Приклад уривку коду для розгортання зображення:

```
            static public float[] decompressImageBytes(
                byte[] compressedBytes,
                out int width,
                out int height,
                out string colorScheme,
                out string method,
                out int blockSize,
                out string travelMode,
                out bool crossMerge
            ) {
                //считваем заголовок, что бы узнать информацию об изображении
                readHeader_v1(
                    compressedBytes,
                    out width,
                    out height,
                    out colorScheme,
                    out method,
                    out blockSize,
                    out travelMode,
                    out crossMerge
```

```
                );

                int blocksAmount = getTotalBlocksAmount(blockSize, width,
height);
                int elementsAmountInOneBlock = blockSize * blockSize * 3;

                //разжимаем основную часть (уложенне блоки и остаток)
                float[]                        values                        =
byteArrayAsFloatArray(decompressBytes(compressedBytes, Constants.HeaderSize));

                //восстанавливаем блоки
                float[][] blocks = new float[blocksAmount][];

                if (crossMerge) //с кросс-мерджем  {
                    restoreCrossMergedBlocks(values,           blocksAmount,
elementsAmountInOneBlock, blocks);
                }
                else //без кросс-мерджа {
                    Parallel.For(0, blocks.Length, i => {
                      float[] temp = new float[elementsAmountInOneBlock];
                      for (int j = 0; j < elementsAmountInOneBlock; ++j)
                         temp[j] = values[i * elementsAmountInOneBlock + j];
                      blocks[i] = temp;
                    });
                }

                //если надо - восстанавливаем линейность из зигзага
                if (travelMode == "Зиг-заг") {
                    ZigZag.precompute(blockSize);
                    Parallel.For(0, blocks.Length, i => {
                        blocks[i] = fromZigzag(blocks[i]);
                    });
                }

                //к каждому блоку применяем обратное преобразование
                switch (method) {
                    case "FHT":

MathUtils.mapFunctionToEachBlock(MathUtils.inv_fht_2d, blocks);
                        break;
                    case "DCT":

MathUtils.mapFunctionToEachBlock(MathUtils.inv_dct_2d, blocks);
                        break;
                }

                float[]   resultValues   =   mergeBlocksToImageValues(blocks,
width, height, blockSize);

                //записываем остаток, если он был
                long   remainderSize   =   getRemainderSize(width,   height,
blockSize);
                if (remainderSize != 0) {
                    float[] remainder = new float[remainderSize];
                    for (int i = 0; i < remainderSize; ++i)
                        remainder[i]      =      values[blocks.Length      *
```

```
elementsAmountInOneBlock + i];

                    writeRemainderIntoImageValues(remainder,   resultValues,
width, height, blockSize);
                }

                return resultValues;
            }

            //записать в начало массива заголовок
            static void writeHeader_v1(
                byte[] bytes,
                int width,
                int height,
                string colorScheme,
                string method,
                int blockSize,
                string travelMode,
                bool crossMerge
            ) {
                bytes[0]  = 1;
                bytes[1]  = (byte)(width);
                bytes[2]  = (byte)(width >> 8);
                bytes[3]  = (byte)(height);
                bytes[4]  = (byte)(height >> 8);
                bytes[5]  = (byte)((colorScheme == "RGB") ? 0 : (colorScheme
== "YCbCr" ? 1 : 2));
                bytes[6]  = (byte)((method == "FHT") ? 0 : 1);
                bytes[7]  = (byte)(blockSize);
                bytes[8]  = (byte)(blockSize >> 8);
                bytes[9]  = (byte)((travelMode == "Линейный") ? 0 : 1);
                bytes[10] = (byte)(crossMerge ? 1 : 0);
            }

            //прочитать заголовок из начала массива
            static void readHeader_v1(
                byte[] bytes,
                out int width,
                out int height,
                out string colorScheme,
                out string method,
                out int blockSize,
                out string travelMode,
                out bool crossMerge
            )
            {
                if (bytes[0] != 1)
                    throw new Exception("Cant read header: wrong verson " +
bytes[0].ToString());

                colorScheme = method = travelMode = "";

                width  = (bytes[2] << 8) + bytes[1];
                height = (bytes[4] << 8) + bytes[3];
                switch (bytes[5]) {
                    case 0: colorScheme = "RGB"; break;
```

```
            case 1: colorScheme = "YCbCr"; break;
            case 2: colorScheme = "HSV"; break;
        }
        switch (bytes[6]) {
            case 0: method = "FHT"; break;
            case 1: method= "DCT"; break;
        }
        blockSize = (bytes[8] << 8) + bytes[7];
        switch (bytes[9]){
            case 0: travelMode = "Линейный"; break;
            case 1: travelMode = "Зиг-заг"; break;
        }
        crossMerge = (bytes[10] == 1);
    }
```