**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет**

**імені Петра Могили**

**Факультет комп’ютерних наук**

**Кафедра комп'ютерної інженерії**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,

д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

# Метод стиснення проміжних кадрів відео для вбудованих систем

Спеціальність 123 Комп’ютерна інженерія

123 – КБР.01 – 405.22010508

***Студент:*** _____ Д. В. Доценко
*підпис*
«__» _____ 202__ р.

***Керівник: канд. техн. наук, доцент*** _____ Я. М. Крайник
*підпис*
«__» _____ 202__ р.

**Миколаїв – 2024**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Чорноморський національний університет імені Петра Могили**
**Факультет комп'ютерних наук**
**Кафедра комп'ютерної інженерії**

ЗАТВЕРДЖУЮ

Зав. кафедри _____ І. М. Журавська

«_____» _____ 2024 р.

# ЗАВДАННЯ
## на виконання кваліфікаційної бакалаврської роботи

Видано студенту групи 405 факультету комп'ютерних наук

Доценко Дмитро Володимирович
*(прізвище, ім'я, по батькові студента)*

1. Тема кваліфікаційної роботи

Метод стиснення проміжних кадрів відео для вбудованих систем

Затверджена наказом по ЧНУ ім. Петра Могили від 30.01.2024 № 17.

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Розроблений метод стиснення проміжних кадрів відео для вбудованих систем. Для реалізації цієї задачі було досліджено існуючі методи стиснення зображень та відео, було виявлено їх переваги та недоліки. Розробити новий комбінований метод стиснення, який поєднуватиме різні підходи для досягнення оптимального балансу між ефективністю, якістю та обчислювальною складністю. Провести оцінку розробленого методу на різних наборах тестових даних, щоб підтвердити його ефективність та порівняти його з існуючими методами.

4. Перелік питань, що підлягають розробці
Аналіз предметної області та постановка задачі. Огляд існуючих методів стиснення зображень та відео. Розробка нового комбінованого методу стиснення проміжних кадрів відео. Оцінка розробленого методу та порівняння з існуючими методами стиснення.

5. Перелік графічних матеріалів
Графіки та діаграми, що ілюструють ефективність та якість різних методів стиснення. Схеми та блок-схеми, що пояснюють роботу запропонованого методу стиснення. Таблиці з результатами експериментів._____

6. Завдання до спеціальної частини
Проведення аналізу виробничого приміщення та робочих місць. Проведення оцінки природнього освітлення в приміщенні. Оцінка наявних факторів виробничого приміщення. Оцінка ефективності заходів щодо покращення умов праці працівників._____

7. Консультанти:

| Консультант | Кафедра (організація) | Частина роботи |
|---|---|---|
| Алексєєва Анна Олександрівна | Кафедра екології Медичного інституту | Спеціальна частина з охорони праці |
| | | |
| | | |

Керівник роботи
канд. техн. наук, доцент Крайник Ярослав Михайлович_____
*(посада, прізвище, ім'я, по батькові)*

_____
*(підпис)*

Завдання прийнято до виконання

Доценко Дмитро Володимирович_____
*(прізвище, ім'я, по батькові студента)*

_____
*(підпис)*

Дата видачі завдання «____» _____ 20_____ р.

**КАЛЕНДАРНИЙ ПЛАН**

**виконання кваліфікаційної роботи**

Тема: Метод стиснення проміжних кадрів відео для вбудованих систем

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1 | Розробка та затвердження завдання на виконання КР | 30.01.24 | 30.01.24 | Виконав |
| 2 | Огляд літератури за темою роботи | 15.01.24 | 18.02.24 | Виконав |
| 3 | Складання календарного плану БКР | 19.02.24 | 06.03.24 | Виконав |
| 4 | Аналіз предметної області | 19.02.24 | 04.03.24 | Виконав |
| 5 | Розробка проєктних рішень | 23.02.24 | 09.03.24 | Виконав |
| 6 | Моделювання та конструювання АПЗ | 20.03.24 | 27.03.24 | Виконав |
| 7 | Перевірка працездатності, тестування та апробація розробленого АПЗ, аналіз результатів тестування | 28.03.24 | 21.04.24 | Виконав |
| 8 | Відгук керівника КР | 28.05.24 | 07.06.24 | Виконав |
| 9 | Оформлення БКР та презентації | 26.05.24 | 04.06.24 | Виконав |
| 10 | Попередній захист | 28.05.24 | 28.05.24 | Виконав |
| 11 | Рецензування | 12.05.24 | 20.05.24 | Виконав |
| 12 | Завершення оформлення КР та презентації | 28.05.24 | 10.06.24 | Виконав |
| 13 | Захист бакалаврської кваліфікаційної роботи | 24.06.24 | 27.06.24 | Виконав |

Розробив здобувач ВО Доценко Дмитро Володимирович _____

*(прізвище, ім'я, по батькові)* *(підпис)*

«____» _____ 20__ р.

Керівник роботи канд. техн. наук, доцент Крайник Ярослав Михайлович _____

*(посада, прізвище, ім'я, по батькові)* *(підпис)*

«____» _____ 20__ р.

# АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Метод стиснення проміжних кадрів відео для вбудованих систем»
Студент 405 гр.: Доценко Дмитро Володимирович
Керівник: канд. техн. наук, доцент Крайник Ярослав Михайлович

Кваліфікаційна бакалаврська робота присвячена розвитку методів стиснення зображень з використанням алгоритмів QOI та Хаффмана, які включають попередню обробку зображень, перетворення у простір кольорів YCrCb, стисненні вузлових точок та стисненні основної області. Актуальність дослідження полягає в необхідності розвитку методів, які б дозволили ефективно зменшувати обсяги даних зображень та відео, забезпечуючи при цьому високу якість та адаптивність до різних типів контенту.

Об'єктом дослідження є методи стиснення зображень та їхній вплив на обсяг пам'яті та якість зображення. Предметом дослідження є комбінований метод стиснення проміжних кадрів, що поєднує попередню обробку та стиснення вузлових точок без втрат за допомогою алгоритму Хаффмана.

Робота пройшла апробацію на трьох конференціях, а саме: XXVI Всеукраїнській науково-практичної конференції «Могилянські читання – 2023» (Миколаїв, 2023 р.), XXI Міжнародній науковій конференції «Ольвійський форум – 2023» (Миколаїв, 2023 р.), XXIII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій» (Одеса, 2023 р.).Також, по матеріалам кваліфакаційної бакалаврської роботи була написана та опублікована стаття «Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite Ok Image і Хаффмана» в науково фаховому виданні «Електронне моделювання», випуск №2 (2024 р.).

Пояснювальна записка бакалаврської роботи складається зі вступу, трьох розділів, висновків, переліку джерел посилання та чотирьох додатків. У вступі визначається актуальність теми, сформульовані мета, об’єкт, предмет роботи та завдання для досягнення поставленої мети. У першому розділі проводиться аналіз існуючих технологій потрібних для подальшої роботи. У другому розділі розроблюється новий метод стиснення зображень. В третьому розділі проводяться експериментаді дослідження з розробленням програм для стиснення як окремих зображень, так і відео.

В цілому, кваліфікаційна бакалаврська робота містить 71 сторінку (без додатків), 37 рисунків, 2 таблиці, 24 джерел посилання.

***Ключові слова:*** *зображення, стиснення, алгоритм Хаффмана, вузлові точки, ефективність, якість, порівняння, алгоритм QOI*

**ABSTRACT**

of the Bachelor’s Thesis

"Method of compressing intermediate video frames for embedded systems"

Student: Dotsenko D. V.

Supervisor: Krainyk Y. M.

The bachelor's thesis is devoted to the development of image compression methods using QOI and Huffman algorithms, which include image pre-processing, conversion to YCrCb color space, compression of nodal points and compression of the main region. The relevance of the research lies in the need to develop methods that would allow to effectively reduce the volume of image and video data, while ensuring high quality and adaptability to different types of content.

The object of research is image compression methods and their impact on memory capacity and image quality. The subject of the research is a combined method of compression of intermediate frames, which combines pre-processing and compression of nodal points without loss using the Huffman algorithm.

The work was approved at three conferences, namely: XXVI All-Ukrainian Scientific and Practical Conference "Mohyla Readings - 2023" (Mykolaiv, 2023), XXI International Scientific Conference "Olvian Forum - 2023" (Mykolaiv, 2023), XXIII All-Ukrainian scientific and technical conference of young scientists, graduate students and students "State, achievements and prospects of information systems and technologies" (Odesa, 2023). Also, based on the materials of the qualifying bachelor thesis, the article "Image compression method using pre-processing and Quite Ok Image and Huffman algorithms" in the scientific publication "Electronic Modeling", issue No. 2 (2024).

An explanatory note of a bachelor's thesis consists of an introduction, three chapters, conclusions, a list of reference sources and four appendices. The introduction determines the relevance of the topic, formulates the goal, object, subject of work and tasks to achieve the set goal. The first section analyzes the existing technologies needed for further work. In the second section, a new image compression method is developed. In the third section, experimental studies are conducted with the development of programs for compressing both individual images and videos.

In general, the bachelor's thesis contains 71 pages (without appendices), 37 figures, 2 tables, 24 reference sources.

***Keywords:*** *image, compression, Huffman algorithm, node values, efficiency, quality, comparison, QOI algorithm.*

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

CABAC – Context-Adaptive Binary Arithmetic Coding
DCT – Discrete Cosine Transform
FLIF – Free Lossless Image Format
GIF – Graphics Interchange Format
JPEG – Joint Photographic Experts Group
LZW – Lempel-Ziv-Welch
MANIAC – Meta-Adaptive Near-zero Integer Arithmetic Coding
MOS – Mean Opinion Score
PNG – Portable Network Graphics
PSNR – Peak Signal-to-Noise Ratio
QOI – Quite OK Image format
RGB – Red Green Blue
SSI – Structural Similarity Index
WebP – Web Picture format
YCrCb – Luminance (Y) and Chrominance (Cr and Cb)

# ВСТУП

Сучасні цифрові технології перетворили мультимедійне представлення на невід'ємну частину нашого життя, використовуючи їх широко в різних пристроях з високопродуктивними можливостями відображення контенту. Робота з зображеннями стала важливою не лише в повсякденному житті, але й у професійній сфері, забезпечуючи високу точність та наглядність у медицині, відеоіграх, рекламі, медіа і багатьох інших галузях. Проте, з використанням зображень часто виникають проблеми, пов'язані з великим обсягом пам'яті, який вони займають. Це породжує проблеми з їх зберіганням та обробкою, особливо гостро це питання стає у контексті відеофайлів та зображень, що займають значний обсяг пам'яті. Таким чином, розробка ефективних методів стиснення зображень без втрат стає актуальною проблемою.

Актуальність дослідження полягає в необхідності розвитку методів, які б дозволили ефективно зменшувати обсяги даних зображень та відео, забезпечуючи при цьому високу якість та адаптивність до різних типів контенту. Це сприятиме зниженню витрат на зберігання та передачу даних, що є надзвичайно важливим у сучасному інформаційному суспільстві.

***Мета роботи*** полягає у розробці комбінованого методу стиснення проміжних кадрів, що використовує попередню обробку та стиснення вузлових точок без втрат за допомогою алгоритму Хаффмана. Демонстрація, що поєднання таких методів може призвести до кращих результатів у порівнянні з їх використанням окремо. Поставлена мета вимагає вирішення таких ***завдань***:

1) провести аналіз існуючих методів стиснення зображень;
2) розробити комбінований метод стиснення;
3) порівняти ефективність та якість розробленого методу з існуючими аналогами;
4) провести інтегральну оцінку виробничого приміщення.

***Об'єктом дослідження*** є методи стиснення зображень та їхній вплив на обсяг пам'яті та якість зображення.

***Предметом дослідження*** є комбінований метод стиснення проміжних кадрів, що поєднує попередню обробку та стиснення вузлових точок без втрат за допомогою алгоритму Хаффмана.

Результати даного дослідження можуть бути корисними для розробки ефективних методів стиснення зображень для вбудованих систем, що мають обмежені ресурси. Подальша робота над цими методами може призвести до зменшення обсягу пам'яті, який займають проміжні кадри, а в подальшому і відео, зберігаючи при цьому високу якість та деталізацію.

*Апробація результатів* кваліфікаційної роботи відбулася під час XXVI Всеукраїнської науково-практичної конференції «Могилянські читання – 2023» (Миколаїв, 2023 р.), XXI Міжнародній науковій конференції «Ольвійський форум – 2023» (Миколаїв, 2023 р.), XXIII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій» (Одеса, 2023 р.).Також, по матеріалам кваліфакаційної бакалаврської роботи була написана та опублікована стаття «Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite Ok Image і Хаффмана» в науково фаховому виданні «Електронне моделювання», випуск №2 (2024 р.).

*Публікації.* Основні положення роботи опубліковані у збірниках матеріалів XXVI Всеукраїнської науково-практичної конференції «Могилянські читання – 2023», XXI Міжнародній науковій конференції «Ольвійський форум – 2023», XXIII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій», та опубліковано статтю в журналі «Електронне моделювання», випуск №2.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Постановка задачі

У представленій роботі ставиться завдання розробити ефективний метод стиснення проміжних кадрів відео для вбудованих систем. Цей метод повинен відповідати наступним вимогам:

1) висока ефективність стиснення: метод повинен забезпечувати значне зменшення обсягу даних, що передаються та зберігаються, для оптимізації використання ресурсів вбудованої системи;

2) збереження якості зображення та відео: незважаючи на стиснення, метод повинен забезпечувати прийнятну якість відео для кінцевого користувача. Це особливо важливо для застосунків, де візуальна інформація є критично важливою;

3) низька обчислювальна складність: алгоритми стиснення повинні бути обчислювально ефективними, щоб забезпечити роботу в реальному часі на вбудованих системах з обмеженими ресурсами;

4) адаптивність: метод повинен бути адаптивним до різних типів відеоконтенту та умов передачі даних. Це дозволить досягти оптимальної продуктивності в різних сценаріях використання;

5) сумісність: розроблений метод повинен бути сумісним з існуючими стандартами та форматами відео, щоб забезпечити його інтеграцію з іншими компонентами системи.

Для досягнення поставленої мети планується дослідити та проаналізувати існуючі методи стиснення зображень та відео, виявити їх переваги та недоліки. На основі цього аналізу буде розроблено новий комбінований метод стиснення, який поєднуватиме різні підходи для досягнення оптимального балансу між ефективністю, якістю та обчислювальною складністю.

Також буде проведено експериментальну оцінку розробленого методу на різних наборах тестових даних, щоб підтвердити його ефективність та

порівняти його з існуючими методами. Результати цієї оцінки дозволять зробити висновки про придатність методу для використання у вбудованих системах та визначити напрямки подальших досліджень.

## 1.2 Огляд форматів стиснення зображення

Напрямок стиснення зображень, який став каталізатором розвитку відповідних алгоритмів, виявив своє використання в різних галузях, що призвело до поступового ускладнення методів стиснення. Сучасний стан досліджень у цій області свідчить про збільшення обчислювальної потужності та підвищення якості отриманих результатів. Важливим аспектом є поділ стиснення зображень на дві категорії: стиснення без втрат, де інформація не втрачається, та стиснення з втратами, де відкидається частина інформації для отримання кращого результату.

Методи стиснення з втратами ґрунтуються на меншій чутливості людського ока до високочастотних компонент та більшій чутливості до низькочастотних змін. Також використовується особливість, що яскравість краще сприймається порівняно з кольоровою інформацією. Це дає можливість відкидати частину інформації, яка не суттєво впливає на кінцеве представлення зображення, щоб досягти кращого коефіцієнта стиснення.

Важливим аспектом у розумінні процесу стиснення зображень є усвідомлення того, що цей процес складається з ряду етапів, кожен із яких може відноситися до одного з двох основних видів стиснення: без втрат або з втратами. Це визначається особливостями та метою кожного конкретного етапу. Значущою особливістю є те, що результати попереднього етапу можуть служити основою для подальшого вдосконалення наступного.

### 1.2.1 Опис формату стиснення JPEG

Один із найкласичніших прикладів цього підходу виявляється у форматі JPEG (Joint Photographic Experts Group), який використовує каскад етапів для досягнення оптимального стиснення зображень.

Перший етап цього процесу полягає в перетворенні зображення у представлення YCrCb, де компоненти Y відповідають за яскравість, а Cr і Cb за колір. Це перетворення дозволяє використовувати особливості сприйняття людським оком різних компонентів зображення.

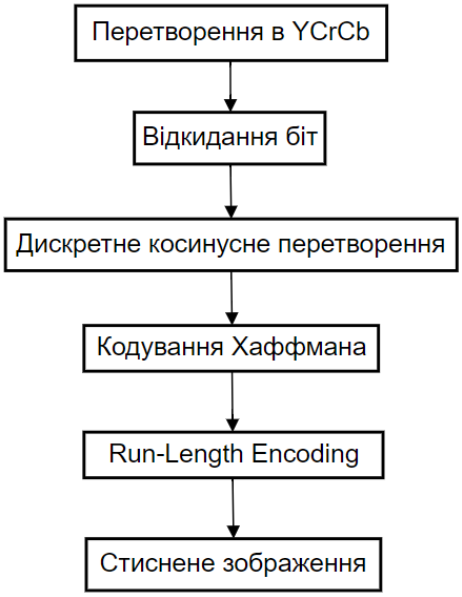



Рисунок 1.1 – Етапи стиснення зображення за допомогою JPEG

Другий етап – відкидання найменших значущих біт. Це призводить до втрати деякої деталізації, але дозволяє ефективно зменшити об'єм інформації, що зберігається. Далі застосовується дискретне косинусне перетворення, яке перетворює просторовий сигнал в частотний домен. Це дозволяє концентрувати енергію сигналу в областях з найбільшою важливістю.

Після дискретного косинусного перетворення використовується кодування Хаффмана, що дозволяє ефективно представляти частотні компоненти зображення з використанням меншої кількості бітів для більш часто зустрічаючих значень. Останнім етапом є кодування Run–Length Encoding, яке використовується для подальшого зменшення довжини послідовності бітів та зменшення об'єму збереженого зображення. Такий каскад етапів дозволяє досягти високої ефективності стиснення та зменшення об'єму даних при збереженні задовільної якості зображення. Розуміння цього

процесу стиснення є ключовим для розробки нових та вдосконалення існуючих методів стиснення зображень без втрат в якості.

### 1.2.2 Опис формату стиснення PNG

Іншим широко розповсюдженим форматом стиснення зображень є Portable Network Graphics (PNG). PNG – це широко використовуваний формат стиснення зображень без втрат, розроблений як покращена та вільна альтернатива формату GIF. PNG підтримує палітри індексованих кольорів (8 біт на піксель), зображення у відтінках сірого (16 біт на піксель) та повнокольорові зображення без палітри (48 біт на піксель), а також зображення з 8-бітовим альфа-каналом прозорості.

PNG використовує два етапи стиснення: попередню фільтрацію та стиснення даних.

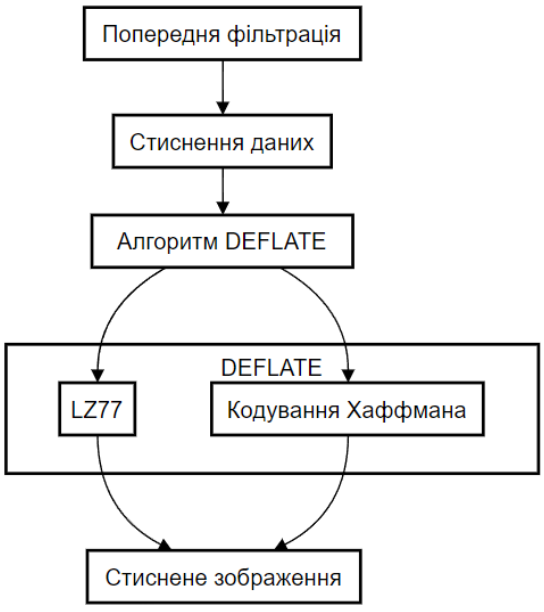



Рисунок 1.2 – Етапи стиснення зображення за допомогою PNG

Попередня фільтрація застосовується для зменшення надмірності даних у зображенні, що робить його більш піддатливим до стиснення. Після фільтрації дані стискаються за допомогою алгоритму DEFLATE, який є комбінацією алгоритму LZ77 та кодування Хаффмана.

До переваг PNG можна віднести:

1) стиснення без втрат: PNG зберігає всі деталі вихідного зображення, що робить його ідеальним для зберігання графіки, логотипів, іконок та інших зображень, де важлива точність відтворення кольорів та деталей;

2) підтримку прозорості: PNG підтримує альфа-канал, що дозволяє зберігати зображення з прозорими ділянками. Це особливо корисно для вебграфіки та елементів дизайну;

3) широка підтримка: PNG є відкритим стандартом і підтримується більшістю сучасних веббраузерів, графічних редакторів та інших програм.

До недоліків можна віднести PNG:

1) у порівнянні з форматами стиснення з втратами, такими як JPEG, PNG зазвичай забезпечує менший ступінь стиснення. Це означає, що файли PNG можуть бути більшими за розміром, ніж файли JPEG з тією ж якістю зображення;

2) PNG не є найкращим вибором для зберігання фотографій, оскільки його алгоритм стиснення не оптимізований для фотографічного контенту. Для фотографій зазвичай рекомендується використовувати формати з втратами, такі як JPEG.

Переваги формату PNG включають збереження всіх деталей без втрат, підтримку прозорості для зображень та широку підтримку стандарту серед сучасних програм. Однак, до недоліків можна віднести менший ступінь стиснення порівняно з JPEG, що призводить до більших розмірів файлів, а також неоптимальність для зберігання фотографій у порівнянні з іншими форматами, такими як JPEG.

### 1.2.3 Опис формату стиснення QOI

Завдяки прогресу в області інформаційних технологій та математичного моделювання, ми можемо використовувати як складні, так і досить прості алгоритми для стиснення зображень без втрат. Одним з таких алгоритмів є The Quite OK Image Format (QOI), розроблений у 2021 році.

QOI – це відносно новий формат стиснення зображень без втрат, розроблений у 2021 році. Він був створений з метою забезпечення швидкого та простого стиснення, при цьому зберігаючи високу якість зображення. QOI особливо ефективний для стиснення зображень з невеликою кількістю кольорів або зображень, де сусідні пікселі мають схожі значення кольору.

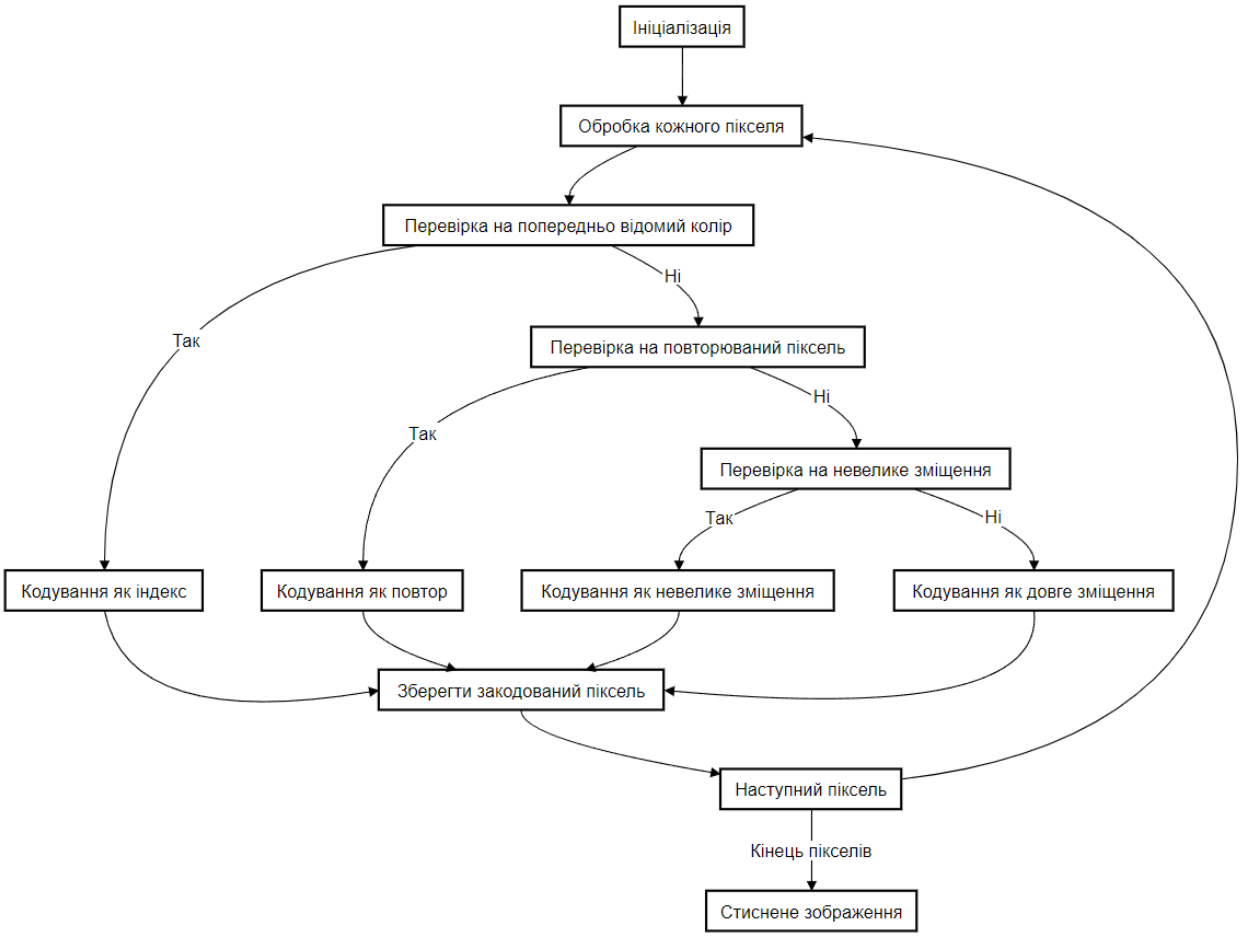



Рисунок 1.3 – Етапи стиснення зображення за допомогою QOI

Основні принципи роботи QOI включають кодування кожного пікселя зображення окремо, використовуючи різні режими кодування в залежності від значення пікселя та контексту (попередніх пікселів). QOI має кілька режимів кодування, кожен з яких оптимізований для певних ситуацій:

1) QOI_OP_RGB: пряме кодування кольору пікселя (4 байти;

2) QOI_OP_RGBA: пряме кодування кольору пікселя з альфа-каналом (4 байти);

3) QOI_OP_INDEX: кодування пікселя за допомогою індексу в таблиці попередніх кольорів (1 байт);

4) QOI_OP_DIFF: кодування різниці між поточним пікселем та попереднім пікселем (1 або 2 байти);

5) QOI_OP_LUMA: кодування різниці яскравості та кольоровості між поточним пікселем та попереднім пікселем (1 або 2 байти);

6) QOI_OP_RUN: кодування довжини серії однакових пікселів (1 байт).

QOI підтримує таблицю з 64 попередніх кольорів, яка використовується для кодування пікселів у режимі QOI_OP_INDEX. Це дозволяє ефективно стискати зображення з обмеженою палітрою кольорів.

До переваги QOI можна віднести:

1) швидкість: QOI є одним з найшвидших алгоритмів стиснення зображень без втрат, як при стисненні, так і при розпакуванні;

2) простота: алгоритм QOI є відносно простим для розуміння та реалізації;

3) ефективність: QOI забезпечує хороший ступінь стиснення для багатьох типів зображень, особливо для зображень з невеликою кількістю кольорів або з плавними градієнтами;

З недоліків можна відзначити QOI:

1) обмежену ефективність для деяких типів зображень: QOI може бути менш ефективним для стиснення зображень з високою деталізацією або з великою кількістю різких переходів кольору;

2) його відносну новизну: QOI є відносно новим форматом, тому його підтримка в програмному забезпеченні та бібліотеках може бути обмеженою.

Загалом, QOI є перспективним форматом стиснення зображень без втрат, який пропонує гарний компроміс між швидкістю, простотою та ефективністю. Він може бути особливо корисним для застосунків, де

швидкість стиснення та розпакування є критично важливою, наприклад, у вебграфіці або мобільних іграх.

### 1.2.4 Опис алгоритму Хаффмана

Далі, розглянемо алгоритм Хаффмана, який використовується для стиснення вузлових точок зображення, які є ключовими для відтворення зображення. Алгоритм Хаффмана – це метод стиснення даних без втрат, який працює шляхом присвоєння коротших кодів символам, що зустрічаються частіше, і довших кодів символам, що зустрічаються рідше. Це дозволяє зменшити загальний розмір даних, зберігаючи при цьому всю інформацію.

Алгоритм починається з підрахунку частоти появи кожного символу у вхідних даних. Далі будується бінарне дерево, де кожен лист представляє символ, а його вага дорівнює частоті появи цього символу. Побудова починається з двох вузлів з найменшими вагами, які об'єднуються в один вузол з вагою, що дорівнює сумі їх ваг.

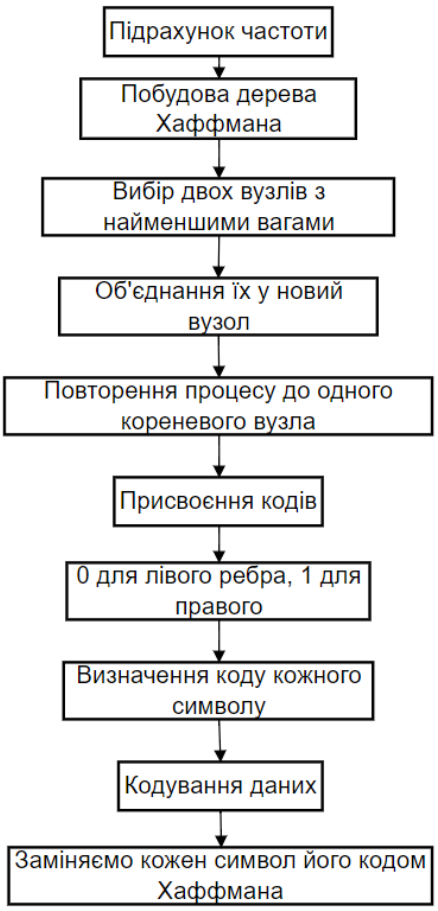



Рисунок 1.4 – Процес кодування Хаффмана

Цей процес повторюється, поки не залишиться один кореневий вузол. Кожному ребру дерева присвоюється значення 0 або 1 (наприклад, 0 для лівого ребра та 1 для правого). Код кожного символу визначається шляхом обходу дерева від кореня до відповідного листа, записуючи значення ребер по дорозі. Вхідні дані кодуються шляхом заміни кожного символу його кодом Хаффмана.

Для декодування закодованих даних використовується те саме дерево Хаффмана, яке було побудовано під час кодування. Процес декодування полягає в обході дерева від кореня, доки не буде знайдено лист, що відповідає поточному бінарному коду. Після знаходження листа, його символ додається до вихідних даних, і процес обходу дерева починається знову з кореня.

Але незважаючи на доступність алгоритму Хаффмана, досі існують проблеми, пов'язані зі стисненням зображень без втрат. Однією з основних

проблем є пошук найкращого підходу до стиснення вузлових точок. Від цього залежать важливі аспекти в багатьох сферах, де обробка і передача зображень є критично важливими, таких як комп'ютерний зір та обробка зображень.

Оптимізація процесу стиснення вузлових точок може значно покращити ефективність алгоритмів розпізнавання образів та аналізу зображень, зменшуючи при цьому обчислювальні витрати. Таким чином, актуальність цієї теми полягає в необхідності розвитку ефективних методів стиснення вузлових точок зображень, зокрема, за допомогою таких алгоритмів, як Хаффмана та QOI.

## 1.3 Огляд методів стиснення зображень

Стиснення зображень є важливою галуззю досліджень та розробок, оскільки зображення займають значний обсяг пам'яті. Для ефективного зберігання та передачі зображень було розроблено безліч методів стиснення, кожен з яких має свої особливості та переваги.

Методи стиснення зображень можна класифікувати за різними критеріями, наприклад за типом стиснення та за принципом роботи. Спочатку розглянемо типи стиснення, вони поділяються на стиснення без втрат, та відповідно з втратами.

### 1.3.1 Стиснення без втрат та стиснення з втратами

При стисненні без втрат усі дані зображення зберігаються, і відновлене зображення є точною копією оригіналу. Ці методи використовуються в тих випадках, коли важливо зберегти всі деталі зображення, наприклад, у медичних зображеннях або технічних кресленнях. До стиснення без втрат входять алгоритми LZ77 та LZ78, ці алгоритми використовують словник для зберігання повторюваних послідовностей даних у зображенні. LZ77 використовує ковзне вікно для пошуку повторень, а LZ78 будує словник динамічно під час стиснення. Також алгоритм LZW (Lempel-Ziv-Welch) – це більш удосконалена версія алгоритму LZ78, який використовує більш

ефективний спосіб побудови словника. LZW використовується у форматі PNG.

При стисненні з втратами деякі дані зображення відкидаються, що призводить до деякої втрати якості. Однак ці методи дозволяють досягти значно більшого ступеня стиснення, ніж методи без втрат. Стиснення з втратами широко використовується у фотографіях та відео, де деяка втрата якості може бути прийнятною. Прикладами алгоритмів стиснення з втратами є JPEG (Joint Photographic Experts Group), який є найпоширенішим форматом стиснення з втратами, який використовує дискретне косинусне перетворення (DCT) та квантування для зменшення обсягу даних. JPEG забезпечує гарний компроміс між ступенем стиснення та якістю зображення. JPEG 2000 – це більш сучасний формат стиснення з втратами, який використовує вейвлет-перетворення замість DCT. JPEG 2000 забезпечує кращу якість зображення при високих ступенях стиснення, але вимагає більше обчислювальних ресурсів. Та алгоритм WebP – це формат стиснення зображень, розроблений Google, який підтримує як стиснення з втратами, так і без втрат. WebP використовує різні методи стиснення, включаючи прогнозування та кодування ентропії. Далі, розглянемо стиснення за принципом роботи.

### 1.3.2 Опис методів стиснення

Методи стиснення на основі словника використовують техніку побудови словника послідовностей даних, які часто зустрічаються у зображенні. Під час стиснення ці повторювані послідовності замінюються посиланнями на відповідні записи у словнику, що дозволяє значно зменшити обсяг даних, що зберігаються.

Одним з найбільш відомих прикладів такого методу є алгоритм LZW (Lempel-Ziv-Welch), який використовується у форматі PNG. LZW створює динамічний словник під час обробки даних, додаючи нові послідовності в міру їх виявлення у вхідному потоці. При стисненні зображення за допомогою LZW кожна послідовність символів, що вже міститься у словнику, замінюється

коротшим кодом, що вказує на її позицію у словнику. Це дозволяє досягти значного зменшення розміру файлу без втрати інформації.

Методи стиснення на основі словника є ефективними для зображень з великою кількістю повторюваних елементів або патернів, оскільки вони дозволяють ефективно зменшити надмірність даних. У результаті, такі методи широко використовуються в різних форматах графічних файлів та в багатьох інших застосуваннях, де потрібне без втрат стиснення даних.

Стиснення на основі перетворення перетворює зображення у інший простір, де дані можуть бути представлені більш компактно. Наприклад, JPEG використовує дискретне косинусне перетворення для перетворення зображення у частотну область, де високочастотні компоненти можуть бути відкинуті з меншим впливом на візуальну якість. Інші приклади перетворень, що використовуються у стисненні зображень, включають вейвлет-перетворення (JPEG 2000) та дискретне перетворення Фур'є .

Стиснення на основі прогнозування передбачає значення пікселів на основі сусідніх пікселів та кодують лише різницю між передбаченим та фактичним значенням. Це дозволяє зменшити кількість бітів, необхідних для представлення кожного пікселя. Різні методи прогнозування можуть використовувати різні моделі для передбачення значень пікселів, наприклад, лінійну інтерполяцію або більш складні моделі на основі машинного навчання.

Метод стиснення на основі фракталів використовує фрактальну геометрію для представлення зображення у вигляді набору фрактальних кодів. Фрактали – це самоподібні структури, які можна описати математичними рівняннями. Стиснення на основі фракталів може забезпечити дуже високий ступінь стиснення, але воно вимагає значних обчислювальних ресурсів і не завжди підходить для всіх типів зображень.

Вибір методу стиснення зображень залежить від багатьох факторів, включаючи:

1) типу зображення: різні типи зображень (фотографії, графіка, медичні зображення тощо) можуть вимагати різних методів стиснення;

2) вимоги до якості: якщо важливо зберегти всі деталі зображення, то слід використовувати стиснення без втрат. Якщо деяка втрата якості допустима, то можна використовувати стиснення з втратами для досягнення більшого ступеня стиснення;

3) обчислювальних ресурсів: деякі методи стиснення вимагають більше обчислювальних ресурсів, ніж інші. При виборі методу слід враховувати доступні обчислювальні потужності;

4) вимог до швидкості: якщо швидкість стиснення та розпакування є критично важливою, то слід вибирати методи, які забезпечують швидку обробку зображень.

Розуміння різних методів стиснення зображень та їх класифікації є важливим для вибору оптимального методу для конкретного завдання.

## 1.4 Сучасні тенденції в стисненні зображень без втрат

Сучасні тенденції в стисненні зображень без втрат постійно еволюціонують, відображаючи нові відкриття в галузі обробки зображень, швидкий технологічний прогрес та зростаючі вимоги до якості та ефективності.

Деякі з ключових сучасних тенденцій включають використання штучного інтелекту в стисненні зображень. Штучний інтелект, зокрема методи глибокого навчання, стають все більш популярними у вдосконаленні методів стиснення зображень без втрат. Нейронні мережі можуть навчитися ефективно визначати та вилучати непотрібну інформацію зображень, що дозволяє зберігати якість зображення при зменшенні його розміру.

Високопродуктивні математичні методи, такі як оптимізація та алгоритми машинного навчання, застосовуються для розробки нових підходів до стиснення зображень. Ці методи дозволяють ефективніше використовувати інформацію в зображеннях та зменшити розмір файлів без значного втручання в якість зображення.

З розвитком високопродуктивних пристроїв і додатків зростають вимоги до швидкості та якості стиснення зображень. Сучасні методи стиснення повинні бути достатньо ефективними для швидкої передачі та обробки великих обсягів даних, а також забезпечувати високу якість зображення для задоволення потреб користувачів.

Розробка спеціалізованих методів для конкретних застосувань таких як медицина, аерокосмічна промисловість та анімація, існують унікальні вимоги до стиснення зображень. Включають розробку спеціалізованих методів стиснення, які враховують ці специфічні вимоги та забезпечують оптимальний баланс між ефективністю та якістю.

З підвищенням потужності обчислювальних систем і розвитком технологій облікових записів у хмарі зростає інтерес до можливостей стиснення великих обсягів зображень без втрат. Включають розробку методів та інфраструктури для ефективного зберігання, передачі та обробки великих обсягів даних у реальному часі.

Всі ці тенденції спрямовані на покращення ефективності та якості стиснення зображень без втрат у відповідь на зростаючі вимоги та потреби різних галузей та користувачів.

## 1.5 Виклики у стисненні зображень без втрат

Стиснення зображень без втрат стикається з рядом викликів, які потрібно вирішувати для досягнення оптимального балансу між розміром файлу та якістю зображення. Деякі з основних викликів у цій області включають:

1) збереження високої якості зображення: одним з основних викликів є збереження високої якості зображення після стиснення. При стисненні без втрат необхідно враховувати те, щоб деталі та текстури зображення залишалися чіткими та відтворювалися точно так, як у вихідному зображенні;

2) збереження структурної та семантичної інформації: іншим важливим викликом є збереження структурної та семантичної інформації зображення. При стисненні необхідно зберегти ключові аспекти, які визначають сутність та розуміння зображення, щоб воно залишалося зрозумілим та корисним для кінцевого користувача;

3) ефективність стиснення: забезпечення ефективності стиснення є ще одним важливим викликом. Стиснення зображень повинно зменшувати розмір файлу таким чином, щоб він був прийнятним для передачі та зберігання, але при цьому не знижувати якість зображення великою мірою;

4) врахування різноманітності контенту: зображення можуть містити різноманітність контенту, від простих текстур до складних сцен з багатьма об'єктами. Врахування цієї різноманітності є важливим викликом у стисненні зображень, оскільки різні типи контенту можуть вимагати різних методів стиснення для досягнення оптимальних результатів;

5) відповідність стандартам та форматам: ще одним викликом є відповідність стандартам та форматам зображень. Забезпечення сумісності стиснених зображень з різними пристроями та програмним забезпеченням може бути складною задачею, особливо в умовах швидкого технологічного розвитку.

Розв'язання цих викликів вимагає поєднання різних підходів, включаючи розробку нових алгоритмів стиснення, використання штучного інтелекту, оптимізацію параметрів стиснення та вивчення властивостей різних типів зображень.

## Висновки до розділу 1

Аналіз предметної області стиснення зображень показав, що ця галузь є критично важливою для багатьох секторів, включаючи графічні застосунки, вебтехнології та комп'ютерний зір. Розвиток методів стиснення спрямований на покращення якості результатів та зменшення обсягу даних.

Стиснення зображень поділяється на дві основні категорії: стиснення без втрат та стиснення з втратами. Кожна категорія має свої переваги та недоліки, і вибір методу залежить від конкретних вимог до якості зображення та ступеня стиснення. Стиснення без втрат гарантує повне відновлення оригіналу зображення, що є важливим для деяких застосувань. Однак, цей тип стиснення зазвичай забезпечує менший ступінь стиснення порівняно зі стисненням з втратами. Стиснення з втратами дозволяє досягти більшого ступеня стиснення за рахунок деякої втрати якості зображення. Цей тип стиснення широко використовується для фотографій та відео, де деяка втрата якості може бути прийнятною.

Існуючі методи стиснення зображень, такі як JPEG, PNG, JPEG 2000 та WebP, широко використовуються в різних застосунках. Проте, вони мають свої обмеження, особливо в контексті вбудованих систем з обмеженими ресурсами. Тому розробка нових, більш ефективних методів стиснення залишається актуальною задачею.

Одним з перспективних напрямків є комбінований підхід до стиснення зображень, який поєднує різні методи для досягнення оптимального балансу між ефективністю, якістю та обчислювальною складністю. У цій роботі ми зосередимося на розробці такого комбінованого методу, який використовуватиме попередню обробку зображень, формування блоку даних вузлових точок, стиснення за алгоритмом Хаффамана та QOI..

Очікується, що запропонований метод дозволить досягти кращих результатів стиснення порівняно з існуючими методами. Це відкриває нові можливості для оптимізації зберігання та передачі зображень у різних застосунках, таких як відеоспостереження, мобільні пристрої та інші.

# 2 МЕТОДИ ТА АЛГОРИТМИ СТИСНЕННЯ ЗОБРАЖЕНЬ

## 2.1 Метод стиснення проміжних кадрів відео

У сучасному цифровому світі відеоконтент стає все більш популярним засобом передачі інформації. Від стрімінгових платформ до соціальних мереж відеофайли стали невід'ємною частиною нашого повсякденного життя. Ключовим аспектом передачі цих відео є технології та формати стиснення, які дозволяють зменшити розмір файлу без значної втрати якості зображення.

Одним з найпоширеніших стандартів стиснення відео є MPEG. Незважаючи на ефективність MPEG для багатьох застосувань, його складність може бути надмірною для деяких ситуацій або пристроїв з обмеженими ресурсами, таких як мобільні телефони або відеокамери з низькою обчислювальною потужністю.

Тому існує нагальна потреба у розробці нового, спрощеного методу стиснення, який би забезпечував високу якість зображення при меншому обсязі обчислень, ніж MPEG.

### 2.1.1 Роль опорних та проміжних кадрів у стисненні відео

Опорні кадри – це ключовий елемент у системах стиснення відео, зокрема у таких форматах, як MPEG. Вони служать «орієнтиром» для інших кадрів, які йдуть за ними, і, таким чином, відіграють важливу роль у процесі стиснення. Опорні кадри, часто називають ключовими кадрами, кодуються незалежно, тобто їх стиснення не залежить від інформації з попередніх або наступних кадрів. Вони подібні до звичайних стиснутих зображень і є основою для стиснення наступних «проміжних» кадрів. Роль опорних кадрів у відео можна порівняти з функцією книжкового закладника під час читання. Вони допомагають декодеру «орієнтуватися» і знати, де почати відтворення або як правильно відтворити наступні кадри. Завдяки їх незалежності від інших кадрів, опорні кадри часто мають більший розмір порівняно з проміжними кадрами. Однак їх наявність дозволяє здійснити більш ефективне стиснення

для кадрів, що слідують, використовуючи різницю між кадрами замість повного кодування кожного окремого кадру.

Проміжні кадри, відомі також як «різницеві кадри», є основою більшості сучасних методів стиснення відео. На відміну від опорних кадрів, які кодуються повністю, проміжні кадри базуються на різницях між сусідніми кадрами. Вони представляють зміни, що відбулися з моменту останнього опорного кадру. Замість того щоб кодувати весь кадр, система стиснення зосереджується лише на тих пікселях, які змінилися порівняно з опорним кадром.

Для визначення змін у проміжних кадрах система порівнює кожен піксель поточного кадру з відповідним пікселем у попередньому кадрі. Різниця між значеннями пікселів визначає, наскільки цей конкретний піксель змінився. Якщо піксель не змінився, він може бути пропущений або закодований як «без змін».

Одним із способів оптимізації зберігання різницевої інформації є використання алгоритму Хаффмана. Цей метод безвтратного стиснення даних базується на частоті появи певних значень різниці у відеокадрі. Значення, які з'являються частіше, отримують коротші коди, в той час як рідше зустрічаються значення отримують довші коди. В результаті, загальний обсяг даних може бути значно зменшений без втрати якості.

### 2.1.2 Перевірка різниці опорного та проміжного кадру

Для цього дослідження була розроблена програма, яка порівнює два зображення (які можуть розглядатися як опорний та проміжний кадри) і створює теплову карту різниць між ними. Ця карта наочно демонструє, де і наскільки пікселі двох зображень відрізняються один від одного. При запуску програми необхідно вказати чотири аргументи: ширину зображення, висоту зображення, файл першого зображення (опорний кадр) та файл другого зображення (проміжний кадр) (рис. 2.1).

а) б)

Рисунок 2.1 – Порівняння кадрів: а) опорний кадр, б) проміжний кадр

Програма спочатку читає обидва зображення піксель за пікселем. Для кожного пікселя визначається різниця в значеннях між опорним і проміжним кадрами. Якщо різниця дорівнює нулю (тобто пікселі однакові), відповідна область на тепловій карті зображується білим кольором, а напис «0» вказує на відсутність відмінностей. Якщо існує різниця, область пікселя на тепловій карті зображується жовтим кольором, а величина різниці відображається чорним кольором на цій області (рис. 2.2).

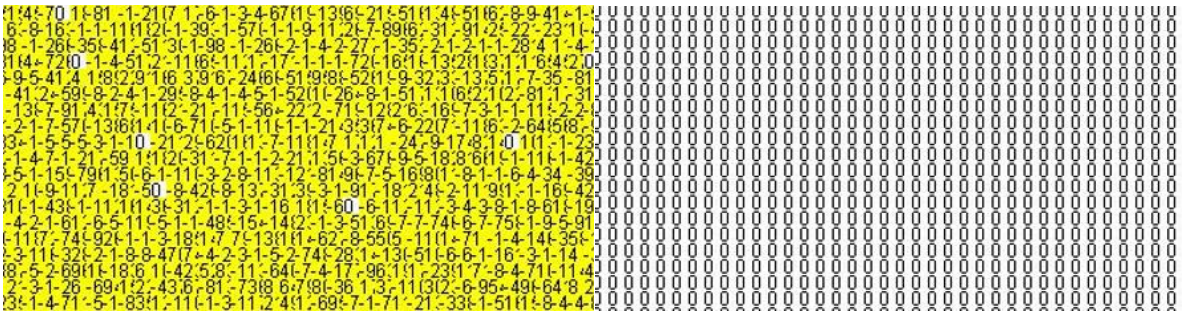


Рисунок 2.2 – Теплова карта різниць між опорним та проміжним кадром

Далі, результати розрахунків різниці зберігаються в список списків, де кожен внутрішній список відповідає одному з трьох теплових карт. Після обробки всіх пікселів, різниці з усіх трьох карт записуються у бінарний файл. Цей файл потім можна стиснути за допомогою алгоритму Хаффмана, щоб отримати кінцевий стиснений результат.

### 2.1.3 Результати теплової карти різниць

Після реалізації цього методу було проведено тестування, і результати виявилися обнадійливими. Наприклад, для бінарного файлу з перевірки зображень (рис. 2.1), первісний розмір файлу становив 6220800 байтів. Після застосування алгоритму Хаффмана розмір файлу скоротився до 959778 байтів, що відповідає стисненню у 6,4815 разів.

Основна причина такого ефективного стиснення полягає в тому, що велика частина послідовних кадрів відео має мінімальні зміни, і, як результат, більшість значень різниці складаються з однакових чисел (найчастіше нулів). Тому алгоритм Хаффмана, який є особливо ефективним для даних з великою кількістю повторюваних значень, виявився ідеальним для цієї задачі.

## 2.2 Попередня обробка зображень

Алгоритм попередньої обробки зображень, який застосовується до алгоритму QOI, є комплексним процесом, спрямованим на оптимізацію та покращення подальшого стиснення. Основний принцип роботи цього алгоритму включає в себе перетворення кольорового простору зображення з RGB (червоний, зелений, синій) в YCrCb (яскравість та відхилення кольору). Далі, розберемо перетворення кольорового простору зображення з RGB в YCrCb.

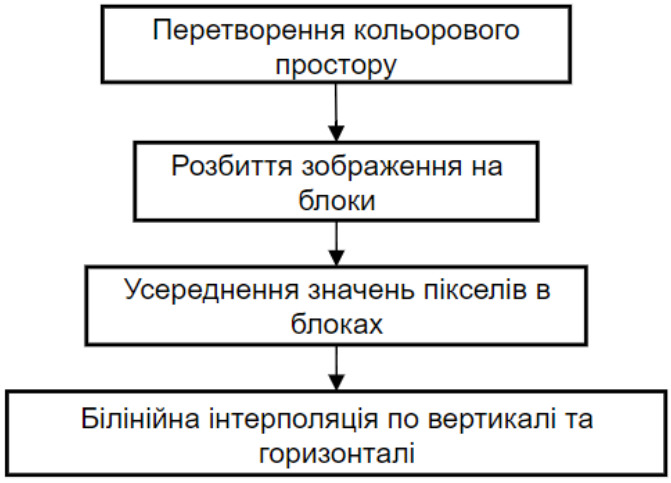



Рисунок 2.3 – Етапи попередньої обробки зображення

RGB, що є стандартом для цифрових зображень, ґрунтується на принципі адитивного синтезу кольору, де основні кольори об'єднуються в різних пропорціях для формування інших кольорів. YCrCb – це кольоровий простір, який відокремлює компонент яскравості (Y) від компонентів кольору (Cr та Cb). Це перетворення важливе, оскільки дозволяє обробляти яскравість та кольорові компоненти окремо, підвищуючи ефективність подальшої обробки зображення.

Після перетворення зображення з RGB до YCrCb, наступним кроком є розбиття зображення на блоки однакового розміру, наприклад, 4x4 або 8x8 пікселів. Кожен блок обробляється окремо. Усереднення значень пікселів в кожному блоці зменшує ентропію зображення, роблячи значення в блоках менш різноманітними, що сприяє кращому стисненню. Цей крок використовує властивості локальної однорідності в зображеннях, де сусідні пікселі часто мають схожі кольори.

Додатково, обробка зображення по блоках включає використання білінійної інтерполяції по вертикалі та горизонталі. Цей процес заміщає значення пікселів у блоках середнім значенням, обчисленим шляхом лінійної інтерполяції відносно значень пікселів по вертикалі та горизонталі.

## 2.3 Вплив параметрів алгоритму попередньої обробки на ефективність стиснення

Алгоритм попередньої обробки зображень відіграє важливу роль у забезпеченні ефективності стиснення без втрат. Він включає кілька параметрів, які можуть суттєво впливати на кінцевий результат стиснення. Розглянемо детальніше вплив цих параметрів на коефіцієнт стиснення та якість зображення.

### 2.3.1 Вплив розміру блоку вузлових точок та кількості відкинутих найменш значущих біт

Розмір блоку, на якій розбивається зображення перед обробкою, є одним з ключових параметрів. Збільшення розміру блоку призводить до зменшення кількості вузлових точок, що зберігаються, і, відповідно, до більшого ступеня стиснення. Однак, це також може призвести до втрати деталей та погіршення якості зображення, особливо якщо зображення містить багато дрібних деталей або різких переходів кольору. З іншого боку, зменшення розміру блоку дозволяє зберегти більше деталей, але призводить до меншого ступеня стиснення.

Кількість відкинутих найменш значущих бітів (англ. Least Significant Bit, LSB) при усередненні значень пікселів у блоці також впливає на ефективність стиснення. Відкидання більшої кількості LSB призводить до більшого ступеня стиснення, але також до більшої втрати інформації та погіршення якості зображення. Навпаки, збереження більшої кількості LSB дозволяє зберегти більше деталей, але призводить до меншого ступеня стиснення.

Тип інтерполяції, який використовується для відновлення значень пікселів між вузловими точками, також може впливати на якість зображення. Білінійна інтерполяція є простим та швидким методом, але вона може призводити до деякого згладжування та розмиття деталей. Більш складні методи інтерполяції, такі як кубічна або сплайн-інтерполяція, можуть забезпечити кращу якість зображення, але вони вимагають більше обчислювальних ресурсів.

Крім перерахованих вище параметрів, алгоритм попередньої обробки може включати й інші параметри, такі як метод вибору вузлових точок, тип фільтрації зображення тощо. Вплив цих параметрів на ефективність стиснення та якість зображення також потребує дослідження.

Для дослідження впливу параметрів алгоритму попередньої обробки на ефективність стиснення необхідно провести серію експериментів з різними

наборами зображень та різними значеннями параметрів. У кожному експерименті слід оцінювати коефіцієнт стиснення та якість зображення за допомогою об'єктивних та суб'єктивних метрик.

Об'єктивні метрики, такі як пікове відношення сигнал/шум (англ. Peak Signal-to-Noise Ratio) та структурна подібність (англ. Structural Similarity Index), дозволяють кількісно оцінити ступінь спотворення зображення після стиснення. Суб'єктивні метрики, такі як візуальна оцінка якості (англ. Mean Opinion Score), враховують сприйняття якості зображення людиною. Результати експериментів дозволять визначити оптимальні значення параметрів алгоритму попередньої обробки для різних типів зображень та різних вимог до якості та ступеня стиснення.

## 2.4 Загальна схема стиснення

Загальна схема стиснення складається з двох етапів: попередньої обробки зображення та стиснення основної області зображення. Вона може бути представлена такою як продемонстровано на рис. 2.3.

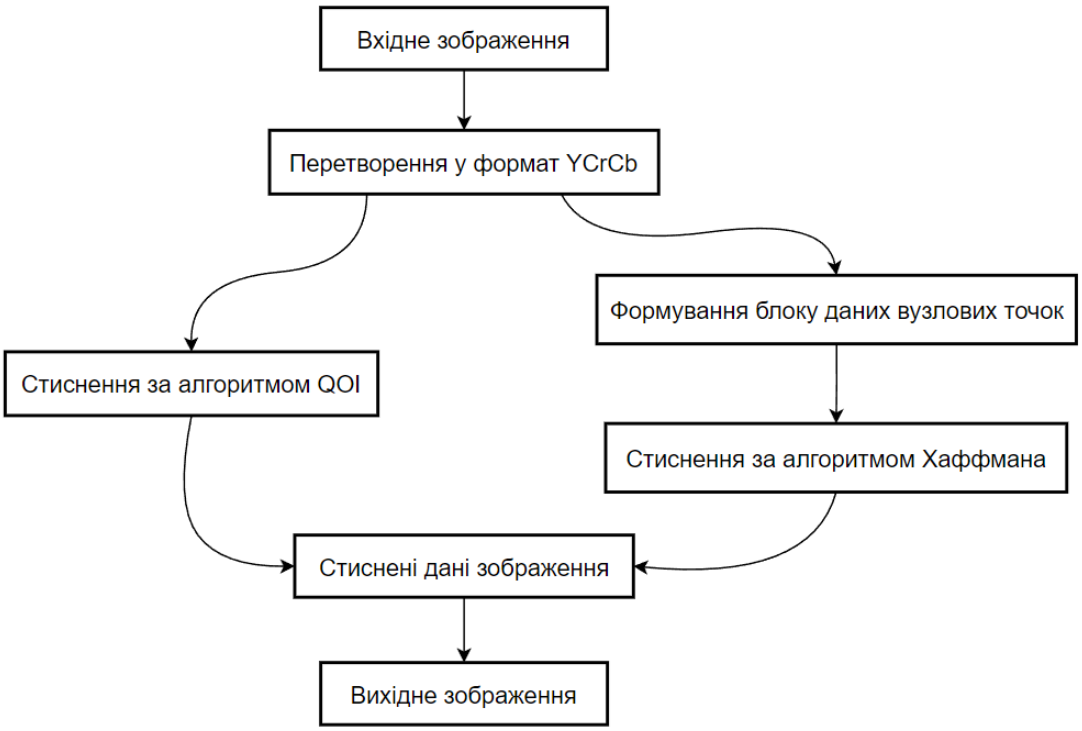



Рисунок 2.4 – Загальна схема стиснення зображення

Такий двоетапний процес дозволяє згладити деталі та знизити відмінності між пікселями, що допомагає знизити ентропію та покращити ефективність стиснення зображення. Використання згладжування зображення перед його стисненням відіграє ключову роль у підготовці для оптимального використання алгоритму QOI.

## 2.5 Стиснення основної області

Після алгоритму попередньої обробки зображення, основна область зображення готова до стиснення. Далі результат попередньої стадії обробки передається на вхід QOI – алгоритму стиснення без втрат, який був розроблений з метою простого і ефективного стиснення 24-бітних та 32-бітних кольорових зображень. QOI використовує прості з точки зору обчислень процедури.

Процес стиснення основної області зображення в QOI включає кроки сканування зображення піксель за пікселем та стиснення кожної пікселя за допомогою відповідного алгоритму. Алгоритм продовжує сканування та кодування, поки всі пікселі зображення не будуть оброблені і закодовані.

В результаті отримується стиснуте зображення, яке займає значно менше місця в порівнянні з оригінальним зображенням. При цьому якість стиснення можна контролювати за допомогою параметру кількості відкинутих останніх значущих біт у основній області.

## 2.6 Виділення вузлових точок зображення

Далі розглянемо стиснення вузлових точок за допомогою алгоритму Хаффмана. При стисненні зображення важливо зберегти ключові аспекти, які визначають якість та впізнаваність оригінального зображення. Одним з підходів, які дозволяють досягти цього є виділення вузлових точок зображення окремі пікселі, що мають важливе значення для загальної структури та семантики зображення. Зазвичай, вузлові точки вибираються на основі різноманітних критеріїв, в залежності від конкретних потреб та цілей

процесу стиснення. Зображення розглядається як набір вузлових точок, між якими можуть проводитися інтерполяції для відновлення інтервенуючих пікселів. Тобто, ми виділяємо кожні N пікселів зображення, які будуть слугувати базою для подальшого відновлення зображення.

Розглядаються два основних підходи до стиснення вузлових точок: пряме стиснення та стиснення різниць між вузловими точками.

У першому випадку вузлові точки стискаються безпосередньо за допомогою алгоритму Хаффмана. Алгоритм Хаффмана – це метод безвтратного стиснення, що використовує принципи статистичної ентропії для створення оптимізованого дерева Хаффмана, яке використовується для кодування та декодування даних. Цей підхід дозволяє зменшити загальну кількість бітів, потрібних для представлення кожної кольорової компоненти вузлової точки.

У другому випадку розглядаємо стиснення вузлових точок у формі різниць. Тобто, замість стиснення окремих вузлових точок, ми стискаємо різниці між послідовними вузловими точками, а далі виконується стиснення різниць використовуючи алгоритм Хаффмана. Цей підхід особливо ефективний, коли вузлові точки мають подібні значення або коли значення пікселів незначно змінюються з кадру на кадр.

Замість того, щоб кодувати велику кількість даних, які можуть бути подібними, ми кодуємо лише невеликі різниці між ними. Таким чином, ми можемо значно зменшити обсяг даних, необхідних для представлення зображення, при цьому підтримуючи високу якість зображення навіть після процесу стиснення та відтворення.

### 2.6.1 Стиснення вузлових точок зображення за допомогою алгоритма Хаффмана

При стисненні вузлових точок алгоритмом Хаффмана, кожна вузлова точка розглядається як символ, що потребує кодування. Алгоритм Хаффмана використовує статистику вузлових точок для побудови оптимального дерева

Хаффмана, що далі використовується для генерації бінарних кодів для кожної вузлової точки.

Алгоритм Хаффмана є жадібним алгоритмом, що означає, що він намагається зробити найкращий можливий вибір на кожному етапі побудови дерева Хаффмана, не зважаючи на оптимізацію кінцевого результату. Це робить алгоритм відносно простим для розуміння та реалізації, але водночас дивовижно ефективним для багатьох задач стиснення. Для кожної вузлової точки або різниці між вузловими точками генерується унікальний бінарний код. Цей код потім використовується для представлення вузлових точок під час зберігання або передачі стиснутого зображення.

Коли зображення потрібно відновити, бінарні коди декодуються за допомогою того ж самого дерева Хаффмана, що дозволяє відновити початкові вузлові точки та, відповідно, відтворити зображення. Такий підхід гарантує високу якість відтворення зображень незалежно від ступеня стиснення.

Ефективність алгоритму Хаффмана особливо помітна в ситуаціях, коли дані мають велику кількість повторюваних значень. У таких випадках він дозволяє значно зменшити обсяг даних без втрати якості. Це особливо важливо для систем з обмеженими ресурсами, таких як вбудовані системи або мобільні пристрої, де обмеженість пам'яті та обчислювальної потужності є критичними факторами.

Крім того, використання алгоритму Хаффмана може спростити процес зберігання та передачі стиснутих даних, що робить його привабливим вибором для різноманітних застосувань у відео та зображеннях. Вибір методу стиснення вузлових точок залежить від конкретних вимог до якості зображення та обмежень системи, в якій він буде застосовуватися.

## 2.7 Математична модель алгоритму стиснення

Запропонований метод стиснення зображень базується на комбінованому підході, який включає кілька етапів: попередню обробку зображень, перетворення кольорового простору, розбиття на блоки,

застосування алгоритмів QOI та Хаффмана. Розглянемо математичну модель кожного з цих етапів.

На етапі попередньої обробки відбувається фільтрація зображення для зменшення шумів та артефактів. Нехай $I$ – це початкове зображення, представлене у вигляді матриці пікселів розміру $M \times N$. Відфільтроване зображення $I_f$ отримується за допомогою згортки з ядром фільтру $K$:

$$I_f(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I(i-u, j-v) \times K(u,v), \quad (2.1)$$

де $(i,j) -$ координати пікселя, а $K -$ ядро фільтра розміром $(2k+1) \times (2k+1)$.

Для зменшення кореляції між кольоровими компонентами зображення використовується перетворення кольорового простору з RGB до YCbCr. Перетворення кожного пікселя (R,G,B) в нові координати (Y,Cb,Cr) здійснюється за формулами:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B \quad (2.2)$$

Формула для обчислення Y використовує вагові коефіцієнти для кожного кольору RGB.

$$C_r = (R - Y) \times 0{,}713 + 128 \quad (2.3)$$

Червона колірність визначається різницею між червоним кольором і значенням Y, з урахуванням вагового коефіцієнту.

$$C_r = (B - Y) \times 0{,}564 + 128 \quad (2.4)$$

Синя колірність обчислюється аналогічно червоній колірності. Формула перетворення зображення з формату RGB в формат YCrCb базується на математичних виразах, які конвертують значення кольорів RGB у компоненти яскравості (Y) та дві колірні складові (Cr та Cb). Усі ці формули дозволяють ефективно розділити яскравісні та колірні компоненти зображення.

Після перетворення кольорового простору зображення розбивається на невеликі блоки розміром $n \times n$. Нехай $B_{i,j}$ позначає блок, розташований у $i$-му рядку та $j$-му стовпці:

$$B_{i,j} = \{I_f(m,n) \mid i \times n \leq m < (i+1) \times n, j \times n \leq n < (j+1) \cdot n\}. \quad (2.5)$$

Алгоритм QOI здійснює швидке стискання зображення за рахунок використання таблиць квантування та коефіцієнтів перетворення Фур'є. Кожен блок піддається дискретному косинусному перетворенню:

$$F(u,v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} B_{i,j}(x,y) \times \cos\left(\frac{2x+1)u\pi}{2n}\right) \times \cos\left(\frac{2y+1)v\pi}{2n}\right), \quad (2.6)$$

де $F(u,v)$ −коефіцієнти дискретного косинусного перетворення, а $u,v$ −індекси в частотному домені.

Квантування здійснюється за допомогою квантувальної матриці $Q(u,v)$:

$$F(u,v) = \left[\frac{F(u,v)}{Q(u,v)}\right]. \quad (2.7)$$

Для подальшого стиснення отриманих коефіцієнтів використовується алгоритм Хаффмана. Нехай $S -$ множина коефіцієнтів після квантування. Кожен коефіцієнт кодується за допомогою відповідного коду Хаффмана:

$$C(s) = Huffman(s),\ s \in S, \quad (2.8)$$

де $C(s) -$ код Хаффмана для коефіцієнта $s$.

Таким чином, математична модель алгоритму стиснення включає обробку початкового зображення, перетворення його у новий кольоровий простір, розбиття на блоки, застосування дискретного косинусного перетворення, квантування та стиснення за допомогою кодування Хаффмана. Кожен з цих етапів спрямований на зменшення обсягу даних при збереженні якості зображення.

## 2.8 Порівняння з іншими методами стиснення без втрат

Для оцінки ефективності запропонованого методу стиснення проміжних кадрів відео, проведемо його порівняння з іншими відомими методами стиснення зображень без втрат, такими як PNG, WebP без втрат та FLIF.

PNG (англ. Portable Network Graphics) – це широко використовуваний формат стиснення зображень без втрат, який підтримує прозорість. Він використовує алгоритм стиснення LZW, який ефективно працює з зображеннями, що містять великі області однакового кольору. Однак, PNG може бути менш ефективним для стиснення фотографій або зображень з високою деталізацією.

WebP (англ. Web Picture Format) – це формат стиснення зображень, розроблений Google, який підтримує як стиснення з втратами, так і без втрат. WebP без втрат використовує різні методи стиснення, включаючи прогнозування та кодування ентропії. Він зазвичай забезпечує кращий ступінь стиснення, ніж PNG, при збереженні високої якості зображення.

FLIF (англ. Free Lossless Image Format) – це відносно новий формат стиснення зображень без втрат, який претендує на найкращий ступінь стиснення серед усіх існуючих форматів. Він використовує комбінацію різних методів стиснення, включаючи MANIAC (англ. Meta-Adaptive Near-zero Integer Arithmetic Coding) та CABAC (англ. Context-Adaptive Binary Arithmetic Coding).

Очікується, що запропонований метод забезпечить кращий ступінь стиснення, ніж PNG, завдяки використанню більш ефективних алгоритмів стиснення та попередньої обробки зображень. Порівняння з WebP та FLIF буде більш складним, оскільки ці формати також використовують сучасні методи стиснення. Проте, запропонований метод може мати перевагу завдяки своїй спеціалізації на стисненні відеоданих та можливості адаптації до різних типів відеоконтенту.

## 2.9 Можливості та обмеження запропонованого методу

Запропонований метод стиснення зображень має численні переваги, які роблять його ефективним у різних контекстах. Основною перевагою є високий ступінь стиснення, що досягається завдяки комбінованому підходу, який включає попередню обробку зображень, перетворення кольорового простору, розбиття на блоки та алгоритми QOI та Хаффмана. Такий підхід дозволяє досягти значного зменшення обсягу даних без втрати якості зображення, що є критично важливим для багатьох застосувань, де збереження чіткості та деталізації є пріоритетом.

Метод демонструє високу ефективність для різних типів зображень, включаючи фотографії та зображення з високою деталізацією. Це дозволяє використовувати його в різних галузях, таких як системи відеоспостереження та архівування графічних даних. Важливою особливістю є здатність методу адаптуватися до різних умов використання, що дозволяє оптимізувати його для конкретних завдань і покращити результати стиснення. Крім того, завдяки модульній структурі, метод може бути інтегрований з іншими сучасними алгоритмами стиснення, що дозволяє підвищити його ефективність та швидкодію.

Однак, незважаючи на численні переваги, запропонований метод має й певні обмеження. Однією з головних проблем є складність реалізації. Комбінований підхід вимагає використання кількох етапів обробки зображень, що ускладнює його реалізацію та вимагає додаткових ресурсів для обробки.

Порівняння з іншими сучасними методами стиснення, такими як WebP та FLIF, може бути більш складним, оскільки ці формати також використовують сучасні методи стиснення, що може знижувати відносну перевагу запропонованого методу. Крім того, відкидання найменш значущих бітів для покращення коефіцієнта стиснення може призвести до втрати якості зображення, що є критичним для деяких застосувань.

Для досягнення максимальної ефективності запропонований метод вимагає додаткової оптимізації для конкретних задач та умов використання, що може потребувати значних витрат часу та ресурсів.

Загалом, запропонований метод стиснення зображень демонструє значний потенціал для застосування в різних галузях. Однак, для досягнення максимальної ефективності, необхідно подальше вдосконалення та адаптація методу до специфічних умов використання, що потребує значних зусиль та ресурсів.

## 2.10 Вплив типу зображення на ефективність стиснення

Ефективність стиснення зображень значною мірою залежить від типу зображення, яке піддається стисненню. Важливо враховувати різноманітні характеристики зображень, такі як кількість деталей, рівень шуму, колірна гамма та частотний вміст, щоб обрати оптимальні методи стиснення. У цьому розділі ми розглянемо, як різні типи зображень впливають на ефективність запропонованого методу стиснення.

Фотографії високої роздільної здатності зазвичай містять велику кількість деталей і мають широку кольорову гаму. Для таких зображень метод перетворення кольорового простору (наприклад, з RGB до YCbCr) може бути особливо ефективним, оскільки він дозволяє розділити яскравість та колірну інформацію, що полегшує подальше стиснення. Крім того, використання дискретного косинусного перетворення у поєднанні з алгоритмами квантування та ентропійного кодування, такими як Хаффман, дозволяє зменшити розмір файлу без значних втрат якості.

Медичні зображення, такі як рентгенівські знімки, МРТ та КТ, мають високі вимоги до якості та деталізації, оскільки навіть незначні втрати можуть призвести до неправильних діагнозів. У таких випадках важливо застосовувати методи стиснення без втрат або з мінімальними втратами якості. Алгоритми, що базуються на кодуванні без втрат, такі як QOI, можуть бути ефективними для збереження всіх важливих деталей зображення.

Технічні креслення, графіки та діаграми зазвичай містять багато різких ліній та однорідних областей. Для таких зображень методи квантування та кодування без втрат є особливо ефективними. Алгоритмм Хаффмана може бути використаний для кодування повторюваних елементів, таких як лінії та текст, що дозволяє значно зменшити обсяг даних без втрат важливої інформації.

Зображення з низьким рівнем деталізації, такі як іконки або прості графічні елементи, можуть бути ефективно стиснуті за допомогою простих алгоритмів, які використовують усереднення та білінійну інтерполяцію. У таких випадках методи стиснення з втратами, які відкидають найменш значущі біти, можуть забезпечити високу ступінь стиснення без помітного погіршення якості зображення.

Результати дослідження показують, що тип зображення значно впливає на ефективність стиснення. Для фотографій високої роздільної здатності, медичних зображень, технічних креслень та зображень з низьким рівнем деталізації необхідно застосовувати різні методи стиснення, що враховують їх специфічні характеристики. Використання комбінованого підходу дозволяє досягти високої ефективності стиснення для різних типів зображень. Подальші дослідження можуть бути спрямовані на вдосконалення методів стиснення з урахуванням специфічних властивостей різних типів зображень, що забезпечить ще більшу ефективність та збереження якості.

## 2.11 Перспективи розвитку методу

Запропонований метод стиснення зображень має значний потенціал для подальшого розвитку та вдосконалення. Існує кілька напрямків, які можуть суттєво покращити ефективність та застосовність методу у різних галузях.

Одним з найбільш перспективних напрямків є інтеграція методів машинного навчання, зокрема глибоких нейронних мереж, у процес стиснення зображень. Використання моделей глибокого навчання може допомогти в автоматичному визначенні оптимальних параметрів для квантування та

кодування, а також у відновленні зображень після стиснення з мінімальною втратою якості. Наприклад, автоенкодери можуть бути використані для навчання ефективних представлень зображень, які забезпечують високе стиснення при збереженні основних деталей.

Подальший розвиток методу може включати адаптивне квантування, яке динамічно налаштовує рівні квантування залежно від характеристик конкретного зображення. Це дозволить зменшити втрати якості для важливих деталей зображення та підвищити загальний ступінь стиснення. Використання адаптивного підходу також може включати аналіз локальної складності зображення та застосування різних стратегій квантування до різних частин зображення.

Оскільки складність реалізації є одним з обмежень запропонованого методу, перспективним напрямком є оптимізація обчислювальних витрат. Це може включати як оптимізацію алгоритмів, так і використання спеціалізованих апаратних засобів, таких як графічні процесори та програмовані логічні інтегральні схеми. Такі оптимізації дозволять прискорити процес стиснення та зменшити енергоспоживання, що є важливим для застосувань у мобільних пристроях та вбудованих системах.

Також, подальші дослідження можуть бути спрямовані на вдосконалення методів кодування, таких як покращення алгоритму Хаффмана або інтеграція більш ефективних алгоритмів ентропійного кодування, таких як арифметичне кодування або новіші методи, розроблені для специфічних типів зображень. Це може додатково зменшити обсяг стиснених даних та покращити ефективність методу.

Для забезпечення широкого застосування методу необхідно провести більш широке тестування в реальних умовах. Це включає застосування методу до різних типів зображень, таких як медичні зображення, фотографії високої роздільної здатності, супутникові знімки та інші. Таке тестування допоможе виявити слабкі місця методу та визначити області для подальшого вдосконалення.

Перспективи розвитку методу стиснення зображень є дуже широкими та багатообіцяючими. Інтеграція з машинним навчанням, адаптивне квантування, оптимізація обчислювальних витрат, вдосконалення методів кодування, тестування в реальних умовах та розробка стандартів – всі ці напрямки можуть суттєво покращити ефективність та застосовність методу. Подальші дослідження та інновації у цих напрямках дозволять зробити метод ще більш потужним інструментом для стиснення зображень у різних галузях.

## Висновки до розділу 2

У цьому розділі було розглянуто різні методи та алгоритми стиснення зображень, включаючи перетворення кольорового простору, розбиття на блоки, усереднення, білінійну інтерполяцію, стиснення без втрат QOI, та стиснення вузлових точок за допомогою алгоритму Хаффмана. Було проведено аналіз ефективності цих методів, а також їх комбінацій, що показало переваги комбінованого підходу для досягнення кращого стиснення зображень без втрат.

Введення математичної моделі алгоритму стиснення дозволило формалізувати процеси попередньої обробки зображень, перетворення кольорового простору, розбиття на блоки, застосування алгоритмів QOI та Хаффмана. Це забезпечило чітке розуміння кожного етапу та його впливу на загальну ефективність стиснення.

Результати дослідження показали, що попередня обробка зображень з використанням перетворення кольорового простору, розбиття на блоки та усереднення дозволяє знизити ентропію зображення та покращити ефективність стиснення алгоритмом QOI. Крім того, стиснення вузлових точок зображення за допомогою алгоритму Хаффмана у представленні як різниці виявилося більш ефективним, ніж пряме стиснення вузлових точок.

Було також досліджено можливості та обмеження запропонованого методу. Основними можливостями є висока ефективність стиснення, збереження якості зображення та здатність адаптуватися до різних типів

зображень. Водночас основними обмеженнями є складність реалізації та вимоги до обчислювальних ресурсів.

Досліджено вплив відкидання найменш значущих бітів на ефективність стиснення алгоритмом QOI. Результати показали, що відкидання певної кількості бітів може призвести до значного покращення коефіцієнта стиснення без суттєвої втрати якості зображення.

Перспективи розвитку методу включають інтеграцію з методами машинного навчання, адаптивне квантування, оптимізацію обчислювальних витрат, поліпшення методів кодування, реальні додатки та тестування, а також розробку стандартів. Всі ці напрямки можуть суттєво покращити ефективність та застосовність методу у різних галузях.

У цілому, результати дослідження підтверджують, що комбінований підхід до стиснення зображень є перспективним напрямком для розробки ефективних методів стиснення зображень без втрат. Подальші дослідження та інновації у цьому напрямку дозволять зробити метод ще більш потужним інструментом для стиснення зображень у різних галузях.

# 3 ПРАКТИЧНЕ ЗАСТОСУВАННЯ МЕТОДУ СТИСНЕННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

У цьому розділі буде описано практичне застосування розробленого методу стиснення. Розглянемо розробку програм для стиснення зображень та відео, включаючи опис їх функціональності, реалізації та результати експериментів.

## 3.1 Розробка програми для стиснення окремих зображень

### 3.1.1 Опис програми

Програма для стиснення зображень реалізує комбінований метод, що складається з попередньої обробки зображення, стиснення основної області та стиснення вузлових точок.

Попередня обробка включає зчитування вхідного зображення; перетворення кольорового простору з RGB у YCrCb, розбиття зображення на блоки заданого розміру (наприклад, 4x4 пікселі), усереднення значень пікселів у кожному блоці з відкиданням заданої кількості найменш значущих бітів та застосування білінійної інтерполяції для відновлення значень пікселів між вузловими точками (алгоритм Хаффмана). Вузлові точки зберігаються в окремий масив.

Стиснення основної області здійснюється за допомогою алгоритму QOI, який стискає пікселі, що не є вузловими точками. Стиснені дані записуються у вихідний файл.

Стиснення вузлових точок включає підрахунок частоти появи кожної вузлової точки або різниці між вузловими точками, побудову дерева Хаффмана на основі підрахованих частот та кодування вузлових точок або різниць за допомогою алгоритму Хаффмана. Стиснені дані записуються у вихідний файл.

### 3.1.2 Попередня обробка зображень

Перед початком процесу стиснення зображень необхідно виконати їх попередню обробку. Цей етап включає перетворення зображень у відповідний кольоровий формат та виділення вузлових точок (рис. 3.1).

```
def convert_to_ycrcb(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith(('.png', '.jpg', '.jpeg'))])

    # Перетворення кожного зображення у формат YCrCb і збереження у
вихідну папку
    for frame_file in frame_files:
        frame_path = os.path.join(input_folder, frame_file)
        frame = cv2.imread(frame_path)
        ycrcb_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2YCrCb)  #
        output_path = os.path.join(output_folder, frame_file)
        cv2.imwrite(output_path, ycrcb_frame)
        print(f"Зображення {frame_file} перетворено та збережено у
{output_path}")
```

Рисунок 3.1 – Функція для перетворення у формат YCrCb

Функція приймає два параметри: шлях до вхідної папки з зображеннями (*input_folder*) та шлях до вихідної папки для збереження перетворених зображень (*output_folder*). Далі, створюється вихідна папка, зчитуються всі файли з розширенням *.png*, *.jpg*, або *.jpeg* з вхідної папки і сортуються за іменами.

Для кожного зображення виконується:

– зчитування шляху до файлу;

– завантаження зображення за допомогою *cv2.imread*;

– перетворення зображення з формату RGB у YCrCb за допомогою *cv2.cvtColor* (opencv використовує формат RGB за замовчуванням);

– збереження перетвореного зображення у вихідну папку за допомогою *cv2.imwrite*.

На рис. 3.2 показано результат перетворення зображення з простору кольорів RGB у YCrCb.

– вхідне зображення у просторі кольорів RGB, на якому видно яскраві насичені кольори автомобіля та його оточення;

– зображення після перетворення у простір кольорів YCrCb. На цьому зображенні можна помітити, що кольори стали більш приглушеними, і виділяються інші компоненти, такі як яскравість (Y) та колірні складові (Cr та Cb).

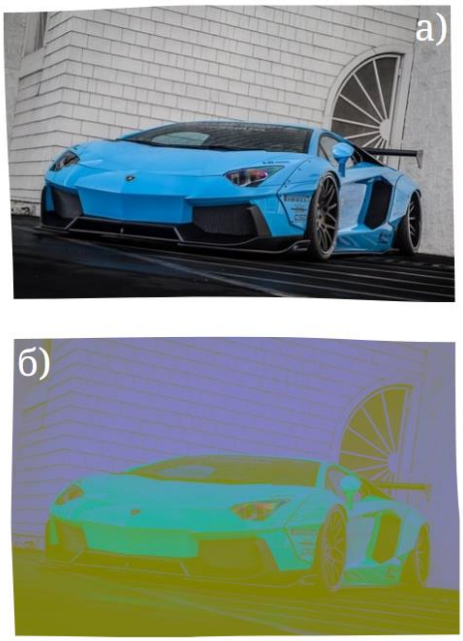



Рисунок 3.2 – Результат перетворення в YCrCb, а) вхідне зображення, б) вихідне зображення

Перетворення зображення у простір кольорів YCrCb дозволяє розділити інформацію про яскравість і кольорові компоненти, що може бути корисним для подальшої обробки зображення, такої як виділення ключових точок або стиснення. Простір YCrCb використовується в багатьох відеостандартах і системах стиснення зображень завдяки його здатності краще моделювати сприйняття кольорів людиною.

Після перетворення зображень у простір кольорів YCrCb, наступним кроком є виділення вузлових точок та їх стиснення. Це допомагає зменшити кількість даних, які необхідно обробляти, і підвищити ефективність стиснення. Для цього ми використовуємо розбиття зображення на блоки, обчислення середнього значення кожного блоку і інтерполяцію для відновлення зображення.

Для виділення вузлових точок зображення розбивається на блоки фіксованого розміру. (рис. 3.3).

```
def split_into_blocks(image, block_size):
    width, height = image.size
    pixels = np.array(image)
    blocks = []
    for i in range(0, height, block_size):
        for j in range(0, width, block_size):
            blocks.append(pixels[i:i+block_size,
j:j+block_size])
    return blocks
```

Рисунок 3.3 – Розбиття зображення на блоки

Це дозволяє обробляти зображення частинами, зменшуючи складність обчислень. Наступним кроком є обчислення середнього кольору для кожного блоку (рис. 3.4).

```
def average_blocks(blocks, bits_to_discard):
    new_blocks = []
    for block in blocks:
        avg_color = np.mean(block, axis=(0, 1))
        avg_color = (avg_color // (2 ** bits_to_discard))
* (2 ** bits_to_discard)
        new_blocks.append(avg_color.astype(np.uint8))
    return new_blocks
```

Рисунок 3.4 – Обчислення середнього значення блоку

Середній колір використовується як представник кольору для всього блоку, що дозволяє значно зменшити кількість даних.

```
def bilinear_interpolation(blocks, block_size, width, height):
    result = np.zeros((height, width, 3), dtype=np.uint8)
    num_blocks_width = width // block_size
    num_blocks_height = height // block_size

    for i in range(num_blocks_height):
        for j in range(num_blocks_width):
            block_color = blocks[i * num_blocks_width + j]
            result[i * block_size:(i + 1) * block_size, j *
block_size:(j + 1) * block_size] = block_color

    return Image.fromarray(result, 'YCbCr')
```

Рисунок 3.5 – Інтерполяція для відновлення зображення

Для відновлення зображення з блоків використовується білінійна інтерполяція. Вона дозволяє заповнити зображення кольорами, які були обчислені для кожного блоку (рис. 3.5).

На рис. 3.6 зображено результат виділення вузлових точок та подальшої обробки зображення.

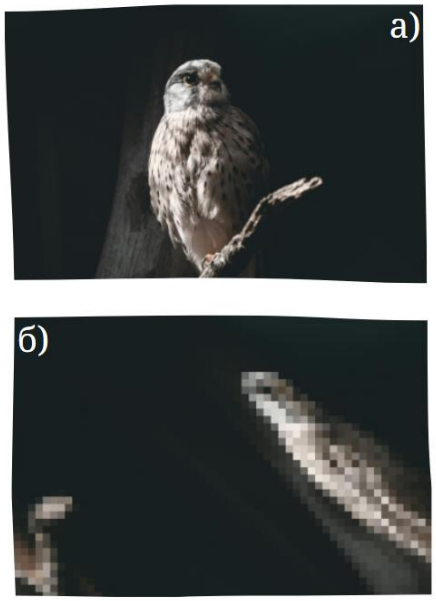



Рисунок 3.6 – Результат виділення вузлових точок: а) – вхідне зображення, б) – виділення блоку вузлових точок

Для демонстрації роботи зображення було розбите на блоки розміром 32x32 пікселі, після чого для кожного блоку було визначено середній колір і створено нове зображення, де кожен блок має однорідний колір. Це призводить до суттєвого зниження деталізації, але зберігає загальну структуру та колірні характеристики оригінального зображення. Тому в таких випадках краще використовувати блоки розміром 2x2 або 4x4 пікселі.

Для кодування вузлових точок застосуємо алгоритм Хаффмана. Цей метод ефективний для зменшення розміру даних за рахунок використання частотності символів.

Спочатку будуємо дерево Хаффмана на основі частотності символів у даних. Це дозволяє визначити оптимальні коди для кожного символу (рис. 3.7).

```
def build_huffman_tree(frequency):
    heap = [Node(freq, sym) for sym, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        new_node = Node(left.freq + right.freq, left.symbol +
right.symbol, left, right)
        heapq.heappush(heap, new_node)

    return heap[0]
```

Рисунок 3.7 – Побудова дерева Хаффмана

Функція *build_huffman_tree* приймає на вхід словник частотності символів і повертає корінь побудованого дерева Хаффмана.

Після побудови дерева Хаффмана, генеруємо коди для кожного символу, проходячи дерево від кореня до листків (рис. 3.8).

```
    def huffman_codes(node, val=''):
        codes = {}
        newVal = val + str(node.huff)
        if node.left:
            node.left.huff = '0'
            codes.update(huffman_codes(node.left, newVal))
        if node.right:
            node.right.huff = '1'
            codes.update(huffman_codes(node.right, newVal))

        if not node.left and not node.right:
            codes[node.symbol] = newVal

        return codes
```

Рисунок 3.8 – Генерація кодів Хаффмана

Функція *huffman_codes* приймає корінь дерева Хаффмана і повертає словник, де кожному символу відповідає його код Хаффмана.

Далі кодуємо вихідні дані, використовуючи згенеровані коди Хаффмана, і доповнюємо їх до байтового вирівнювання (рис. 3.9).

```
def huffman_encoding(data, codes):
    encoded_data = ''.join(codes[byte] for byte in data)
    return encoded_data

def pad_encoded_data(encoded_data):
    extra_padding = 8 - len(encoded_data) % 8
    for i in range(extra_padding):
        encoded_data += "0"
    padded_info = "{0:08b}".format(extra_padding)
    encoded_data = padded_info + encoded_data
    return encoded_data

def get_byte_array(padded_encoded_data):
    b = bytearray()
    for i in range(0, len(padded_encoded_data), 8):
        byte = padded_encoded_data[i:i+8]
```

Рисунок 3.9 – Стиснення даних за допомогою кодів Хаффмана

Функції *huffman_encoding*, *pad_encoded_data* та *get_byte_array* забезпечують процес кодування і формування байтового масиву.

Останнім кроком є зчитування даних з файлу, виконання всіх попередніх етапів і збереження стиснених даних у новий файл (рис. 3.10).

```
def compress(input_data):
    frequency = calculate_frequency(input_data)
    huffman_tree = build_huffman_tree(frequency)
    codes = huffman_codes(huffman_tree)

    encoded_data = huffman_encoding(input_data, codes)
    padded_encoded_data = pad_encoded_data(encoded_data)
    compressed_data = get_byte_array(padded_encoded_data)

    return compressed_data
```

Рисунок 3.10 – Повний процес стиснення

Функція *compress* зчитує дані з вхідного файлу, виконує всі етапи кодування і зберігає стиснені дані у вихідний файл.

Таким чином, алгоритм Хаффмана дозволяє ефективно стискати дані шляхом побудови оптимальних кодів для частих символів і використання цих кодів для кодування вхідних даних.

Проведемо аналіз ефективності стиснення двох підходів на різних зображеннях, а саме алгоритму Хаффмана напряму та у представлені як різниці:


Рисунок 3.11 – Використання алгоритму Хаффмана напряму

На рис. 3.11 з гістограми видно, що розподіл значень кольорових компонент для обраного зображення залежить від самого зображення і може значно варіюватись при його зміні.

Розмір файлу після стиснення складає 273,737 байт, що становить приблизно 70% від початкового розміру (388,848 байт). Коефіцієнт стиснення для цього підходу складає 1,42052. Це означає, що стиснення вузлових точок з використанням алгоритму Хаффмана напряму може бути ефективним у випадках, коли розподіл значень кольорових компонент відповідає наведеному вище прикладу, коли певна частина значень має значно більшу частоту у порівнянні з іншими.


Рисунок 3.12 – Використання алгоритму Хаффмана у представленні як різниці

На гістограмі на рис. 3.12 продемонстровано використання алгоритму Хаффмана у представленні як різниці видно, що кількість кольорових компонент має значно більший розмах значень, що характеризується тим, що більшість значень знаходяться у межах, які вказують на незначну різницю між значеннями.

Розмір файлу після стиснення становить 219,344 байт, що складає близько 60% від початкового розміру (388,848 байт). Коефіцієнт стиснення для цього підходу складає 1,77278. Це вказує на те, що підхід здатний забезпечити кращу якість стиснення та може уніфікувати обробку зображень, оскільки одне і те саме дерево Хаффмана може бути використано для кодування різних зображень за рахунок однакового розподілу значень кольорових компонент.

### 3.1.3 Аналіз ефективності стиснення

За результатами дослідження порівняння двох підходів стиснення вузлових точок за допомогою алгоритму Хаффмана, можна зробити наступні висновки:

– у випадку використання алгоритму Хаффмана напряму для стиснення вузлових точок, спостерігається середня ефективність стиснення. Коефіцієнт стиснення коливається в межах приблизно 1.24–1.42. Загалом, цей підхід зменшує кількість бітів, не втрачаючи точність вузлових точок;

– у випадку використання алгоритму Хаффмана у представленні як різниці, спостерігається вища ефективність стиснення. Коефіцієнт стиснення коливається в межах приблизно 1.60-2.63. Цей підхід особливо ефективний у випадках, коли вузлові точки мають незначні зміни.

Важливо відзначити, що попередня обробка зображень алгоритмом QOI також впливає на ефективність стиснення. Результати в табл. 3.1 вказують на покращення ефективності QOI після використання алгоритму попередньої обробки. Це підтверджує, що комбінація різних методів стиснення може призводити до суттєвого покращення результатів.

Табл. 3.1 та 3.2 конкретизують результати експериментів, представляючи розмір файлу до та після стиснення, коефіцієнти стиснення та використання кодування різниці для алгоритму Хаффмана. Ці дані визначаються різноманітністю зображень та підкреслюють важливість вибору оптимального методу стиснення в залежності від конкретних властивостей вхідних даних.

У таблиці нижче показані результати експериментів щодо ефективності обробки зображень алгоритмом попередньої обробки, та порівняння показників прямого застосування QOI до зображення яке було перетворено у формат YCrCb, але без ігнорування найменших значущих біт, та показників після використання алгоритму попередньої обробки, де передбачалося відкидання 3 таких бітів у кольоровій компоненті.

Таблиця 3.1 – Результати досліджень методу попередньої обробки зображень

| Image | Size | BMP | QOI no processing | Коефіцієнт стиснення | QOI proc 3 | Коефіцієнт стиснення |
|---|---|---|---|---|---|---|
| 1 | 1920×1280 | 7201 | 1152 | 6,250868056 | 468 | 15,38675214 |
| 2 | 1920×1440 | 8101 | 6035 | 1,342336371 | 1806 | 4,485603544 |
| 3 | 1920×1432 | 8056 | 8279 | 0,97306438 | 2254 | 3,574090506 |
| 4 | 1920×2828 | 15908 | 14467 | 1,099606 | 4082 | 3,89710926 |
| 5 | 1920×2888 | 16246 | 10603 | 1,532207866 | 2303 | 7,05427703 |
| 6 | 1920×1080 | 6076 | 2177 | 2,790996785 | 33 | 184,1212121 |
| 7 | 1920×1252 | 7043 | 6678 | 1,054657083 | 1368 | 5,148391813 |
| 8 | 1920×1440 | 8101 | 2764 | 2,93089725 | 390 | 20,77179487 |
| 9 | 1920×1280 | 7201 | 3131 | 2,299904184 | 163 | 44,17791411 |
| 10 | 1920×1280 | 7201 | 3265 | 2,205513017 | 377 | 19,10079576 |

Далі розглянемо результати досліджень щодо стиснення вузлових точок за допомогою алгоритму Хаффмана.

Таблиця 3.2 – Результати стиснення вузлових точок

| Image | Розмір файлу до, байт | Розмір після | Коеф. стиснення | Кодування різниці |
|---|---|---|---|---|
| 1 | 388,848 байт | 273,737 байт | 1.42052 | - |
| 2 | 388,848 байт | 311,869 байт | 1.24683 | - |
| 3 | 388,848 байт | 228,558 байт | 1.70131 | - |
| 4 | 388,848 байт | 219,344 байт | 1.77278 | + |
| 5 | 388,848 байт | 243,102 байт | 1.59953 | + |
| 6 | 388,848 байт | 147,708 байт | 2.63255 | + |

Дані таблиць показують, що алгоритм Хаффмана у представленні як різниці є більш ефективним для стиснення вузлових точок, ніж пряме стиснення. Це пояснюється тим, що алгоритм Хаффмана у представленні як

різниці використовує статистику сусідніх вузлових точок для кодування різниці між ними. Коли вузлові точки мають незначні зміни, різниця між ними також буде невеликою. Це дозволяє алгоритму Хаффмана кодувати різницю між вузловими точками більш ефективно, ніж самі вузлові точки.

## 3.2 Розробка програми для стиснення проміжних кадрів відео

Наступним етапом є розробка програми для стиснення проміжних кадрів відео. Стиснення відео є важливою частиною обробки даних, оскільки воно дозволяє зменшити об'єм даних, зберігаючи при цьому необхідну якість зображення.

На даному етапі ми зосередимося на кількох ключових процесах:

– розбиття відео на кадри: початкове відео буде розбито на окремі кадри, що дозволить обробляти їх індивідуально;

– вибірка опорних кадрів: з розбитих кадрів будуть відібрані опорні кадри, які слугуватимуть базовими точками для стиснення;

– перетворення в колірний простір YCrCb: опорні кадри будуть перетворені в колірний простір YCrCb для більш ефективного стиснення;

– стиснення проміжних кадрів алгоритмом Хаффмана: проміжні кадри будуть стиснуті як різниці між поточним і попереднім кадром за допомогою алгоритму Хаффмана;

– Стиснення основної області за допомогою алгоритму QOI;

– після стиснення всі кадри будуть зібрані назад у відео та збережені.

Цей процес дозволяє значно зменшити розмір відеофайлу, зберігаючи його якість на високому рівні. Далі будуть наведені деталі реалізації кожного з цих кроків.

На рис. 3.13 зображено процес розділення кадрів відео на опорні та проміжні. Це перший крок у стисненні відео, який включає зчитування кадрів

з вхідної папки, вибірку опорних кадрів та розподіл решти кадрів як проміжних.

```
def split_frames(input_folder, key_frame_interval):
    key_frames_folder = os.path.join(input_folder, "key_frames")
    intermediate_frames_folder = os.path.join(input_folder,
"intermediate_frames")

    os.makedirs(key_frames_folder, exist_ok=True)
    os.makedirs(intermediate_frames_folder, exist_ok=True)

    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith('.png')])

    for i, frame_file in enumerate(frame_files):
        frame_path = os.path.join(input_folder, frame_file)
        if i % key_frame_interval == 0:
            os.rename(frame_path, os.path.join(key_frames_folder,
frame_file))
        else:
            os.rename(frame_path,
os.path.join(intermediate_frames_folder, frame_file))
```

Рисунок 3.13 – Процес розділення кадрів відео на опорні та проміжні

Функція *split_frames* приймає на вхід папку з кадрами та інтервал опорних кадрів, після чого розподіляє кадри по відповідним папкам.

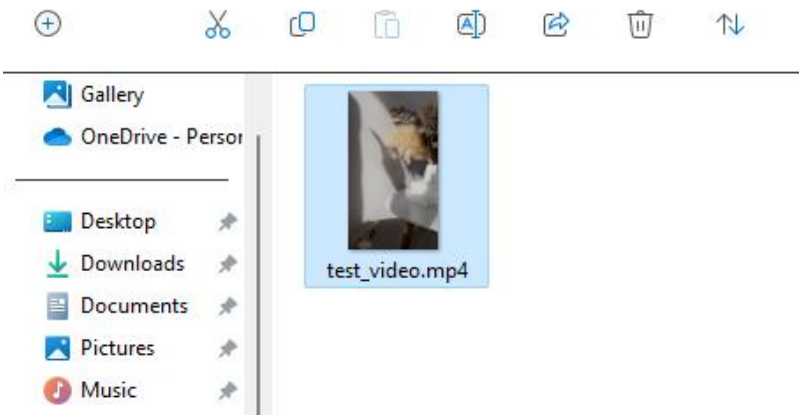



Рисунок 3.14 – Вхідне відео для стиснення

На рис. 3.14 зображено початкове відео, яке буде використовуватися для стиснення. Відео завантажується та підготовлюється для подальшої обробки.


Рисунок 3.15 – Розбиття відео на кадри

На рис. 3.15 показано процес розбиття відео на окремі кадри. Це необхідний етап для подальшої обробки та стиснення кадрів індивідуально.

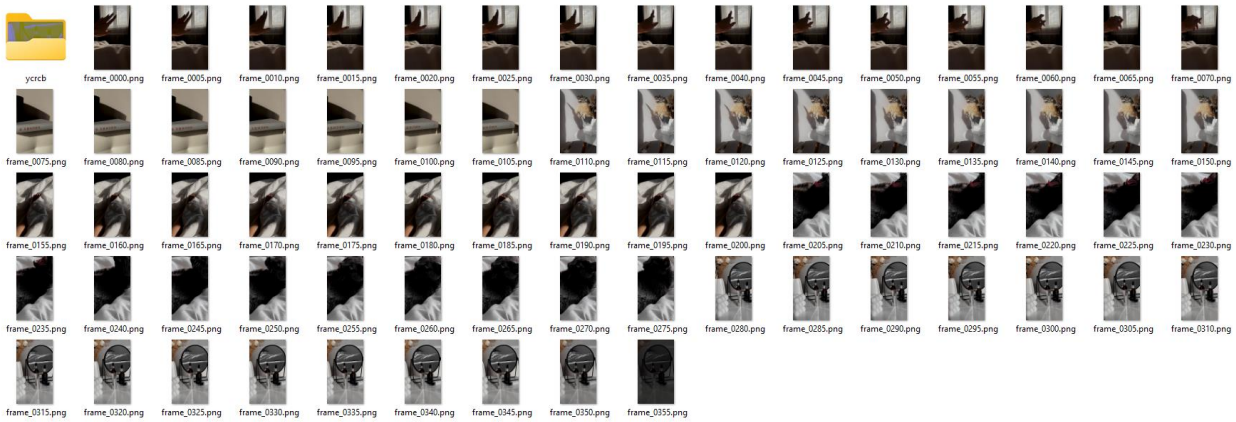



Рисунок 3.16 – Вибірка опорних кадрів

Після вибірки опорних кадрів, їх потрібно перетворити в простір кольорів YCrCb. Це перетворення є важливим кроком у підготовці кадрів для подальшого стиснення. Рисунок 3.17 та 3.18 демонструє процес перетворення.

```
def convert_to_ycrcb(input_folder):
    output_folder = os.path.join(input_folder, "ycrcb")
    os.makedirs(output_folder, exist_ok=True)

    frame_files = sorted([f for f in os.listdir(input_folder)
if f.endswith('.png')])

    for frame_file in frame_files:
        frame_path = os.path.join(input_folder, frame_file)
        frame = cv2.imread(frame_path)
        ycrcb_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2YCrCb)
        output_path = os.path.join(output_folder, frame_file)
        cv2.imwrite(output_path, ycrcb_frame)
```

Рисунок 3.17 – Перетворення кадрів в простір кольорів YCrCb

Функція *convert_to_ycrcb* призначена для перетворення всіх кадрів у вхідній папці в простір кольорів YCrCb та збереження перетворених кадрів у нову папку. Перетворення кадрів в YCrCb:

– за допомогою циклу for здійснюється ітерація по всіх файлах кадрів;

– кожний файл кадру зчитується за допомогою *cv2.imread*;

– кадр перетворюється з простору кольорів RGB в YCrCb за допомогою *cv2.cvtColor*;

– перетворений кадр зберігається у нову папку *output_folder* за допомогою cv2.imwrite.

Функція також виводить повідомлення про успішне перетворення та збереження кадрів.

На рис. 3.18 можна побачити повне перетворення опорних кадрів відео у формат YCrCb.

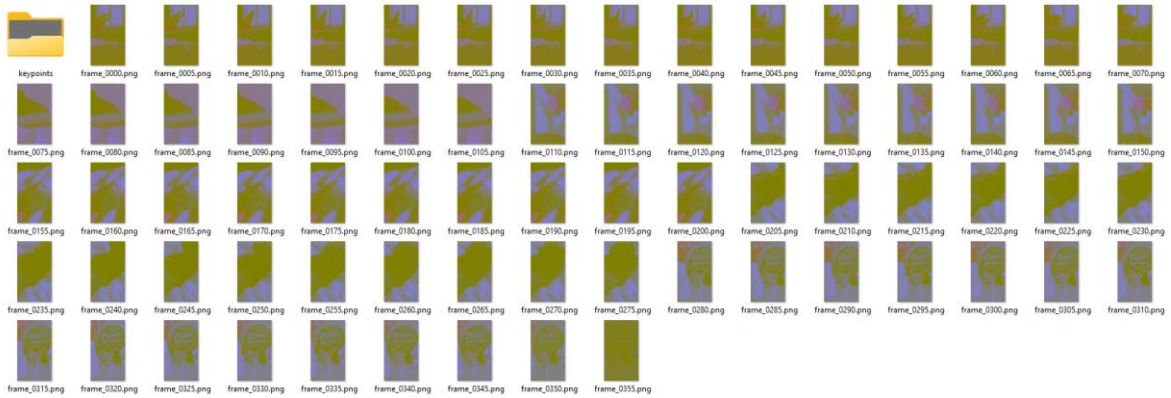



Рисунок 3.18 – Перетворення опорних кадрів в YCrCb

Наступним етапом є виділення проміжних кадрів відносно опорних, та їх наступне стиснення за допомогою алгоритму Хаффмана у представленні як різниці (рис. 3.19).

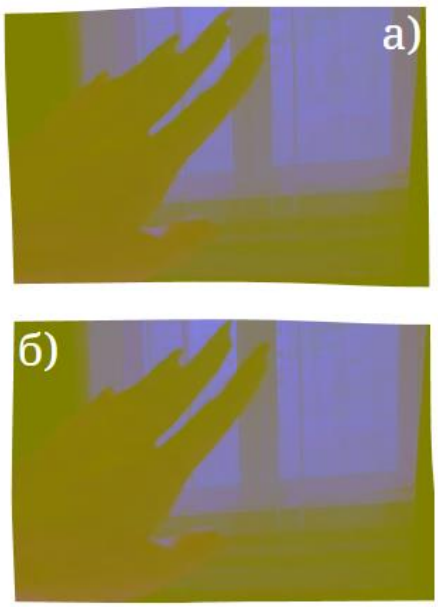



Рисунок 3.19 – Виділення кадрів, а) опорний кадр, б) проміжний кадр

Перед початком стиснення, варто згенерувати теплову карту різниць між двома кадрами. Рисунок 3.20 демонструє блок коду для генерації теплової карти різниць між проміжним та опорним кадром. Цей процес є важливим для візуалізації різниць між кадрами, що дозволяє більш ефективно стискати відео шляхом обробки тільки змінних частин кадрів.

Клас *ImageHeatMapGenerator* призначений для створення теплових карт, які відображають різницю між опорним та проміжним кадром відео. Програма приймає кілька аргументів командного рядка та генерує теплові карти та бінарний файл з різницями.

```
for (int i = 0; i < 3; i++) {
            BufferedImage heatMapImage = new BufferedImage(width * 10, height *
10, BufferedImage.TYPE_INT_RGB);
            Graphics graphics = heatMapImage.getGraphics();
            graphics.setColor(Color.white);
            graphics.fillRect(0, 0, heatMapImage.getWidth(),
heatMapImage.getHeight());
            graphics.setColor(Color.black);
            for (int j = 0; j < height; j++) {
                for (int k = 0; k < width; k++) {
                    int startVal = startFile.read();
                    int compareVal = compareFile.read();
                    int diff = startVal - compareVal;
                    binData.get(i).add(diff);
                    if (diff == 0) {
                        graphics.setColor(Color.white);
                        graphics.fillRect(k*10, j * 10, 10, 10);
                        graphics.setColor(Color.black);
                        graphics.drawString("0", k * 10, (j+1) * 10);
                    } else {
                        graphics.setColor(Color.yellow);
                        graphics.fillRect(k * 10, j * 10, 10, 10);
                        graphics.setColor(Color.BLACK);
                        graphics.drawString(Integer.toString(diff), k * 10,
(j+1) * 10);
            }
            String filename = String.format("heatMap%d.jpg", i);
            ImageIO.write(heatMapImage, "jpg", new File(filename));
        }
        for (int i = 0; i < binData.get(0).size(); i++) {
            for (int j = 0; j < 3; j++) {
                outputDiffFile.write(binData.get(j).get(i));
```

Рисунок 3.20 – Блок коду для генерації теплових карт

Розглянемо генерація теплових карт більш детальніше:

– цикл виконується тричі, один раз для кожного кольорового каналу;

– створюється нове зображення *heatMapImage* з розміром, що в 10 разів перевищує оригінальний розмір (для кращої візуалізації);

– встановлюється білий фон зображення та чорний колір для тексту.

Розглянемо обчислення різниць між пікселями:

– вкладені цикли проходять через кожен піксель зображення;

– для кожного пікселя зчитується значення з обох файлів (*startFile* та *compareFile*);

– обчислюється різниця між відповідними значеннями пікселів (*diff*);

– різниця додається до списку *binData* для відповідного кольорового каналу.

Відображення різниць на тепловій карті:

– якщо різниця дорівнює нулю, піксель на тепловій карті зберігається білим з текстом «0»;

– якщо різниця не дорівнює нулю, піксель заповнюється жовтим кольором з текстом, що показує значення різниці;

– збереження теплових карт та запис різниць у бінарний файл відбувається наступним чином:

– зображення теплової карти зберігається у файл з ім'ям *heatMapX.jpg*, де *X* - індекс кольорового каналу;

– різниці для кожного кольорового каналу записуються у *outputDiffFile*;

– після завершення запису, файл закривається.

Завдяки цьому підходу можна ефективно ілюструвати та обробляти зміни між кадрами відео. Результати обробки двох кадрів можна побачити на рис. 3.21 та 3.22.

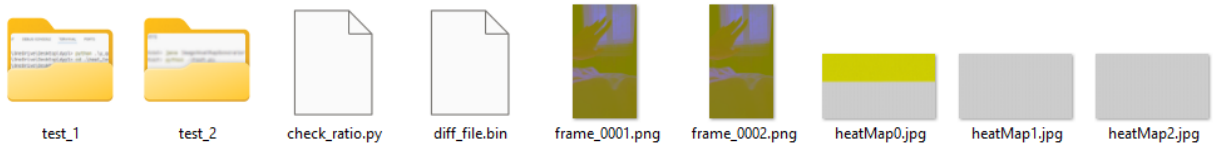
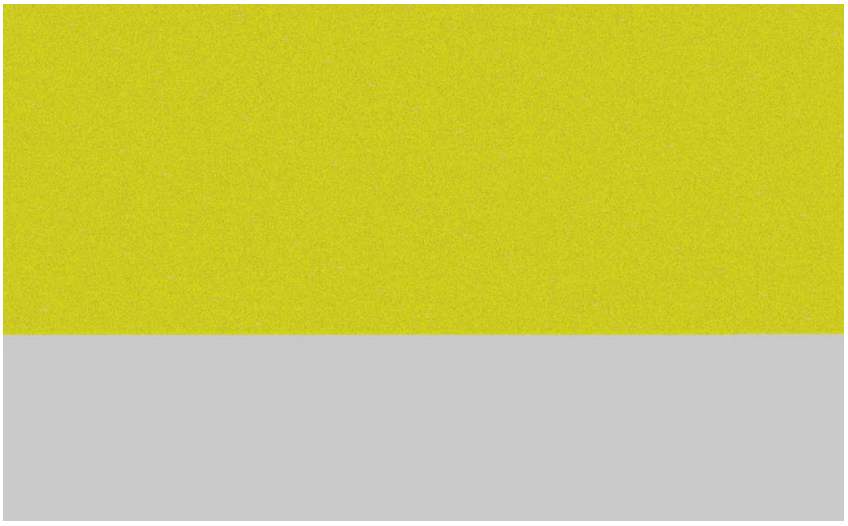



Рисунок 3.21 – Результат використання програми для генерації теплових карт

Рисунок 3.22 ілюструє теплову карту, створену на основі різниць між пікселями опорного та проміжного кадрів.



а)



б)

Рисунок 3.22 – Генерація теплової карти різниць, а) файл *heatMap0.jpg*, б) файл *heatMap1.jpg*

В області зображення, де відмінностей немає, використовується білий колір, а для відображення відмінностей використовується жовтий колір. Кожен жовтий квадрат представляє різницю в значенні пікселя, яка також показана числовим значенням всередині квадрата. Такий підхід допомагає візуально оцінити відмінності між кадрами та їх розподіл по всьому зображенню.

Наступним етапом є стиснення бінарного файлу, для отримання результатів. Ключовими параметрами є розмір оригінального кадру, розмір стиснуного кадру, та ступінь стиснення (рис. 3.23). Демонстрація коду програми продемонстрована на рис. 3.24–3.28.

Функція *calculate_frequency* обчислює частоту кожного символу в переданих даних, повертаючи об'єкт *Counter* з підрахунком символів.

```
def calculate_frequency(data):
    return Counter(data)

def build_huffman_tree(frequency):
    heap = [Node(freq, sym) for sym, freq in frequency.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        new_node = Node(left.freq + right.freq, left.symbol +
right.symbol, left, right)
        heapq.heappush(heap, new_node)

    return heap[0]
```

Рисунок 3.24 – Використання функцій *calculate_frequency* та *build_huffman_tree*

Функція *build_huffman_tree* будує дерево Хаффмана на основі частот символів. Спочатку створюється мін-купка (*heap*) з вузлів, після чого виконується побудова дерева шляхом злиття двох вузлів з найменшими частотами доти, доки не залишиться один вузол.

Функція *def huffman_codes* генерує коди Хаффмана для кожного символу, обходячи дерево Хаффмана. Лівим гілкам відповідає '0', правим - '1'. Листові вузли містять кінцеві коди (рис. 3.25).

```
def huffman_codes(node, val=''):
    codes = {}
    newVal = val + str(node.huff)

    if node.left:
        node.left.huff = '0'
        codes.update(huffman_codes(node.left, newVal))
    if node.right:
        node.right.huff = '1'
        codes.update(huffman_codes(node.right, newVal))

    if not node.left and not node.right:
        codes[node.symbol] = newVal

    return codes
```

Рисунок 3.25 – Функція *def huffman_codes*

Функція *def huffman_encoding* виконує кодування даних за допомогою згенерованих кодів Хаффмана, повертаючи рядок закодованих даних.

```
def huffman_encoding(data, codes):
    encoded_data = ''.join(codes[byte] for byte in data)
    return encoded_data

def pad_encoded_data(encoded_data):
    extra_padding = 8 - len(encoded_data) % 8
    for i in range(extra_padding):
        encoded_data += "0"

    padded_info = "{0:08b}".format(extra_padding)
    encoded_data = padded_info + encoded_data
    return encoded_data
```

Рисунок 3.26 – Функції *def huffman_encoding* та *def pad_encoded_data*

А *def pad_encoded_data* додає заповнення (*padding*) до закодованих даних, щоб їх довжина була кратна 8. Інформація про кількість доданих бітів зберігається на початку закодованого рядка.

```
def get_byte_array(padded_encoded_data):
    b = bytearray()
    for i in range(0, len(padded_encoded_data), 8):
        byte = padded_encoded_data[i:i+8]
        b.append(int(byte, 2))
    return b
```

Рисунок 3.27 – Функція *def get_byte_array*

Вона перетворює закодовані дані з заповненням у байтовий масив, де кожен байт представлений 8-бітовим бінарним значенням.

```
def compress(input_file, output_file):
    with open(input_file, 'rb') as f:
        data = f.read()

    frequency = calculate_frequency(data)
    huffman_tree = build_huffman_tree(frequency)
    codes = huffman_codes(huffman_tree)

    encoded_data = huffman_encoding(data, codes)
    padded_encoded_data = pad_encoded_data(encoded_data)
    b = get_byte_array(padded_encoded_data)

    with open(output_file, 'wb') as f:
        f.write(bytes(b))

    return len(data), len(b)
```

Рисунок 3.28 – Функція *def compress*

Остання функція *def compress* виконує всі кроки стиснення даних: читає вхідний файл, обчислює частоти символів, будує дерево Хаффмана, генерує коди, кодує дані, додає заповнення та зберігає результат у вихідний файл. Вона

також повертає розміри оригінального та стисненого файлів. Результати стиснення бінарного файлу продемонстровані на рис. 3.29.

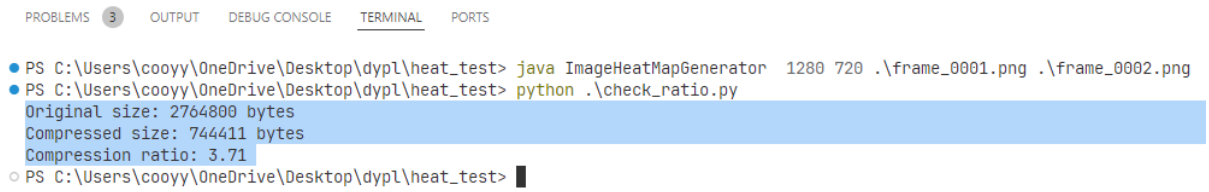



Рисунок 3.29 – Стиснення бінарного файлу, результати стиснення у представленні як різниці

З результатів можна побачити ступінь стиснення дорівнює 3,71, що свідчить про досить непогане стиснення у представленні як різниці проміжного та опорного кадру.

Рисунок 3.30 демонструє результати стиснення за допомогою алгоритму Хаффмана напряму. Також, з результатів можна побачити порівняння стиснення оригінального зображення в RGB форматі, та вже після перетворення в YCrCb формат.

```
Original size: 42467328 bytes
YCrCb compressed size: 21633821 bytes
YCrCb compression ratio: 1.96
RGB compressed size: 26247047 bytes
RGB compression ratio: 1.62
```

Рисунок 3.30 – Результат стиснення алгоритмом Хаффмана напряму, як звичайного зображення RGB, так і перетвореного в YCrCb

Для стиснення Хаффманом напряму можна побачити різницю ступення стиснення в 0,34 рази. Що говорить про досить суттєвий вплив цього перетворення.

На рис. 3.30 можна побачити результати стиснення за допомогою алгоритму QOI вже попередньо обробленого зображення.

```
YCrCb QOI compressed size: 16150441 bytes
YCrCb QOI compression ratio: 2.63
RGB QOI compressed size: 19654732 bytes
RGB QOI compression ratio: 2.16
```

Рисунок 3.31 – Результат стиснення основної області за допомогою QOI, як звичайного зображення RGB, так і перетвореного в YCrCb

З результатів вже бачимо різницю в цілих 0,47 разів, результат є більшим, ніж звичайне стиснення Хаффманом.

**Висновки до розділу 3**

У розділі 3 було розглянуто та експериментально перевірено два підходи до стиснення зображень та проміжних кадрів відео за допомогою алгоритмів Хаффмана та QOI.

В підрозділі 3.1 була створена програма для стиснення окремих зображень, яка базувалася попередній обробці та стисненні основної області, тобто перетворенні RGB в YCrCb, розбитті зображення на блоки, стисненні вузлових точок алгоритмом Хаффмана та стисненні основної області алгоритмом QOI. Також було досліджено ефективність використання алгоритму Хаффмана для стиснення вузлових точок. Результати показали, що алгоритм Хаффмана напряму забезпечує помірне стиснення з коефіцієнтом 1.24–1.42, тоді як використання алгоритму в представленні як різниці підвищує ефективність до 1.60–2.63.

Підрозділ 3.2 фокусувався на стисненні проміжних кадрів відео. Було розроблено програму, яка використовує алгоритм Хаффмана у представленны як різниці для обробки кадрів, що дозволяє покращити ефективність стиснення. Результати показали, що попередня обробка зображень та наступне стиснення вузлових точок за допомогою алгоритму Хаффмана у представленні як різниці забезпечує значне покращення коефіцієнта стиснення без суттєвої втрати якості зображення.

Експериментальні дослідження підтвердили, що комбіноване використання методів перетворення кольорового простору, розбиття на блоки, усереднення та білінійної інтерполяції, а також алгоритмів QOI та Хаффмана, є ефективним підходом до стиснення зображень та відео. Зокрема, стиснення вузлових точок у представленні як різниці за допомогою алгоритму Хаффмана виявилося більш ефективним, ніж пряме стиснення, що підтверджує перспективність цього підходу для подальшого розвитку методів стиснення даних.

# ВИСНОВКИ

У ході виконання роботи були досягнуті всі основні завдання та реалізована мета дослідження, пов'язані з розробкою та впровадженням методу стиснення зображень та відео. Детальніше розглянемо результати по кожному з розділів.

На початковому етапі роботи було проведено детальний огляд існуючих форматів стиснення зображень та методів, які застосовуються в сучасних технологіях. Було проаналізовано такі формати, як JPEG, PNG, WebP та інші. Виявлено їхні переваги та недоліки, а також актуальні тенденції та виклики у галузі стиснення зображень без втрат якості. Це дозволило сформулювати чітку постановку задачі для подальших етапів дослідження, яка полягала у розробці нового методу стиснення зображень з урахуванням сучасних вимог до ефективності та якості.

На основі проведеного аналізу був запропонований новий метод стиснення зображень, який включає попередню обробку зображень та використання алгоритму QOI для стиснення основної області. Запропонована методика полягає у виділенні вузлових точок зображення та їх подальшому стисненні алгоритмом Хаффмана. Було також проведено оцінку впливу параметрів попередньої обробки на загальну ефективність стиснення, що дозволило оптимізувати процес та підвищити якість стиснення без значних втрат інформації.

Для підтвердження ефективності запропонованого методу було розроблено програму для стиснення окремих зображень та проведено тестування на різних наборах даних. Результати показали, що новий метод дозволяє досягти високого ступеня стиснення без втрати якості. Крім того, була створена програма для стиснення проміжних кадрів відео, що дозволило оцінити її результативність у реальних умовах. Експериментальні дослідження продемонстрували переваги запропонованого методу у порівнянні з традиційними підходами, такими як JPEG та PNG.

У процесі виконання роботи були успішно виконані наступні завдання:

– проведено детальний аналіз сучасних методів та алгоритмів стиснення зображень та відео;

– розроблено та реалізовано новий метод стиснення зображень, який включає попередню обробку та використання алгоритмів QOI та Хаффмана;

– створено програмне забезпечення для реалізації запропонованого методу та проведено його тестування на різних наборах даних;

– проведено експериментальну перевірку ефективності методу, яка підтвердила його переваги у порівнянні з існуючими підходами.

Подальші дослідження можуть бути спрямовані на оптимізацію розробленого методу для його застосування в різних галузях, а також на інтеграцію нових алгоритмів стиснення для підвищення ефективності та швидкодії. Зокрема, можна дослідити можливість застосування запропонованого методу для стиснення зображень у медичних системах, системах відеоспостереження та інших областях, де важливо зберегти високу якість зображення при мінімальному обсязі даних.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Krainyk Y. Combined Run-Length and Huffman Encoding for Image Compression. Aug., 2022. 15 p. (Preprint. Petro Mohyla Black Sea National University). DOI: 10.21203/rs.3.rs-1982410/v1.

2. Krainyk Y. M., Dotsenko D. V. Method of Image Compression Using Image Preprocessing, and Huffman and Quite Ok Image Algorithms. *Èlektronnoe modelirovanie*. 2024. Vol. 46, no. 2. P. 75–87. DOI: 10.15407/emodel.46.02.075

3. Yang Y., Peng Y., Liu Zh. A Fast Algorithm for YCbCr to RGB Conversion. *IEEE Transactions on Consumer Electronics*. 2007. Vol. 53, Is. 4. P. 1490–1493. DOI: 10.1109/tce.2007.4429242.

4. Stabno M., Wrembel R. RLH: Bitmap compression technique based on run-length and Huffman encoding. Information Systems. 2009. Vol. 34, Is 4—5. P. 400—414. DOI: 10.1016/j.is.2008.11.002.

5. Miano J. Compressed image file formats: JPEG, PNG, GIF, XBM, BMP. Reading, Ma : Addison Wesley Longman, 1999. 264 c.

6. Крайник Я., Дзяман Є. Застосування алгоритму QOI у стисненні зображень. *Інформаційні технології та інженерія* : Всеукр. науково-практ. конф. молодих вчен., аспірантів і студентів. 2023. С. 70—71.

7. Крайник Я., Доценко Д. Метод стиснення проміжних кадрів відео. *Могилянські читання* : Всеукр. науково-практ. конф. 2023. С. 416—417.

8. Крайник Я., Доценко Д. Стиснення вузлових точок зображення для алгоритму Хаффмана. *Ольвійський форум* : Міжнародній науковій конф. 2023. С. 35—36.

9. Крайник Я., Доценко Д. Алгоритм попередньої обробки зображень для алгоритму QOI. *Стан, досягнення та перспективи інформаційних систем та технологій* : Всеукр. науково-техн. конф. молодих вчен., аспірантів і студентів. 2023. С. 300—301.

10. Dominic S., QOI — The Quite OK Image Format. URL: https://qoiformat.org/qoi-specification.pdf (Last accessed: 10.05.2024).

11. Apostolico A. Fast gapped variants for Lempel-Ziv-Welch compression. *Information and Computation*. 2007. Vol. 205, Is. 7. P. 1012–1026. DOI: 10.1016/j.ic.2007.03.001.

12. Ziv J. The Universal LZ77 Compression Algorithm Is Essentially Optimal for Individual Finite-Length $N$-Blocks. *IEEE Transactions on Information Theory*. 2009. Vol. 55, Is 5. P. 1941—1944. DOI: 10.1109/tit.2009.2016069.

13. Wu Y.. Learned Block-based Hybrid Image Compression. *IEEE Transactions on Circuits and Systems for Video Technology*. 2021. C. 1. DOI: 10.1109/tcsvt.2021.3119660.

14. Jagadish H., Pujar and Lohit M., Kadlaskar A. New Lossless Method Of Image Compression And Decompression Using Huffman Coding Techniques. *Journal of Theoretical and Applied Information Technology*. 2010

15. A Novel Image Compression method using Wavelet Coefficients and Huffman Coding / S. Thomas. *Journal of Engineering Research*. 2023. DOI: 10.1016/j.jer.2023.08.015.

16. Hybrid compression technique for image hiding using Huffman, RLE and DWT / P. V. Materials Today: *Proceedings*. 2022. DOI: 10.1016/j.matpr.2021.12.346.

17. Hu Y.-C., Chang C.-C. A new lossless compression scheme based on Huffman coding scheme for image compression. *Signal Processing: Image Communication*. 2000. Vol. 16, no. 4. P. 367–372. DOI: 10.1016/s0923-5965(99)00064-8.

18. Adjustable compression method for still JPEG images / J. Mora Pascual. *Signal Processing: Image Communication*. 2015. Vol. 32. P. 16–32. DOI: 10.1016/j.image.2015.01.004.

19. Koc B., Arnavut Z., Koçak H. The pseudo-distance technique for parallel lossless compression of color-mapped images. *Computers & Electrical Engineering*. 2015. Vol. 46. P. 456–470. DOI: 10.1016/j.compeleceng.2015.01.003.

20. Wang J., Zeng L. A region-based hierarchical image compression method with simulated visual perception. *Digital Signal Processing*. 2024. Vol. 145. P. 104-339. DOI: 10.1016/j.dsp.2023.104339.

21. Liu R., Li M., Ma L. Efficient in-situ image and video compression through probabilistic image representation. *Signal Processing*. 2023. P. 109-268. DOI: 10.1016/j.sigpro.2023.109268.

22. System and method for a multi-primary wide gamut color system : patent US11069280B2 *United States*. Applied on 28.10.2020 ; published on 20.07.2021. 143 p. URL: https://patents.google.com/patent/US11069280B2/en?q=(RGB)&oq=RGB (дата звернення: 10.05.2024).

23. Image processing method, image forming apparatus, and storage medium : patent US20220053107A1 *United States*. Applied on 17.08.2021 ; published on 17.02.2022. 13 p. URL: https://patents.google.com/patent/US20220053107A1/en?q=(YCrCb)&oq=YCrCb (дата звернення: 10.05.2024).

24. Video compression method, device and computer readable storage medium : patent CN109618173B *China*. Applied on 17.12.2018 ; published on 12.04.2019. 17 p. URL:https://patents.google.com/patent/CN109618173B/en?q=(YCrCb)&oq=YCrCb (дата звернення: 10.05.2024).

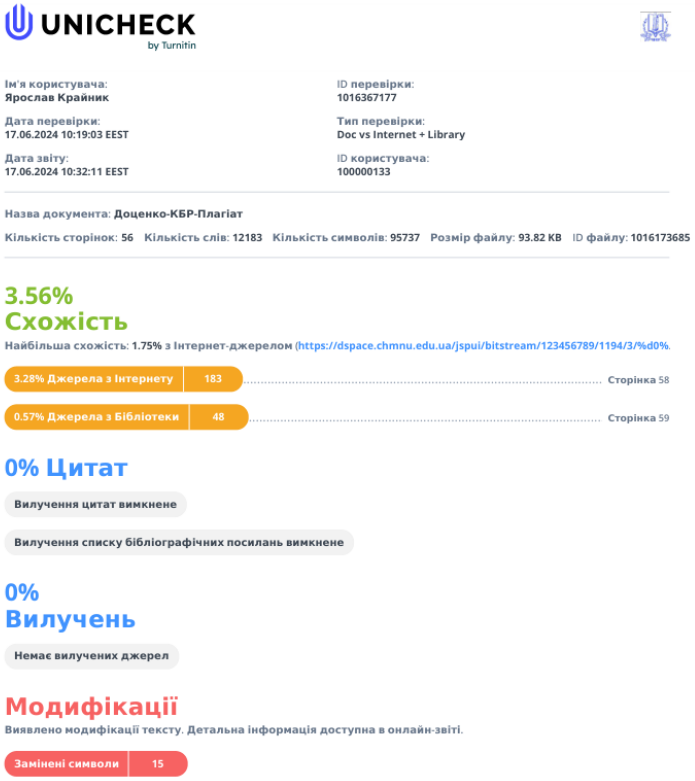
# ДОДАТОК А
## Довідка
## про перевірку на унікальність пояснювальної записки

бакалаврської кваліфікаційної роботи на тему:
«Метод стиснення проміжних кадрів відео для вбудованих систем»

студента спеціальності 123 «Комп’ютерна інженерія», 405 групи
Доценко Дмитро Володимирович
прізвище, ім’я, по-батькові

Перевірку тексту здійснено сервісом: онлайн-сервіс Unicheck

Результат перевірки тексту бакалаврської кваліфікаційної роботи: схожість складає 3,56 %.




Здобувач: Керівник:

канд. техн. наук, доцент

_____Д. В. Доценко _____Я. М. Крайник
підпис ініціали, прізвище підпис ініціали, прізвище

Дата: «___» _____ 2024 р.

# ДОДАТОК Б

## Код для генерації теплової карти різниць

```
package heat_test;
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.ArrayList;

public class ImageHeatMapGenerator {
    public static void main(String[] args) throws IOException {
        if (args.length < 4) {
            System.err.println("Not enough arguments");;
            return;
        }
        int width = Integer.parseInt(args[0]);
        int height = Integer.parseInt(args[1]);
        // the next argument should specify the file to compare with and the
last one is for the file under comparison
        InputStream startFile = new FileInputStream(args[2]);
        InputStream compareFile = new FileInputStream(args[3]);
        OutputStream outputDiffFile = new FileOutputStream("diff_file.bin");
        ArrayList<ArrayList<Integer>> binData = new ArrayList<>();
        for (int i = 0; i < 3; i++) {
            binData.add(new ArrayList<>());
        }
        for (int i = 0; i < 3; i++) {
            BufferedImage heatMapImage = new BufferedImage(width * 10, height
* 10, BufferedImage.TYPE_INT_RGB);
            Graphics graphics = heatMapImage.getGraphics();
            graphics.setColor(Color.white);
            graphics.fillRect(0, 0, heatMapImage.getWidth(),
heatMapImage.getHeight());
            graphics.setColor(Color.black);
            for (int j = 0; j < height; j++) {
                for (int k = 0; k < width; k++) {
                    int startVal = startFile.read();
                    int compareVal = compareFile.read();
                    int diff = startVal - compareVal;
                    binData.get(i).add(diff);
                    if (diff == 0) {
```

```
                    graphics.setColor(Color.white);
                    graphics.fillRect(k*10, j * 10, 10, 10);
                    graphics.setColor(Color.black);
                    graphics.drawString("0", k * 10, (j+1) * 10);
                } else {
                    graphics.setColor(Color.yellow);
                    graphics.fillRect(k * 10, j * 10, 10, 10);
                    graphics.setColor(Color.BLACK);
                    graphics.drawString(Integer.toString(diff), k * 10,
(j+1) * 10);
                }
            }
        }
        String filename = String.format("heatMap%d.jpg", i);
        ImageIO.write(heatMapImage, "jpg", new File(filename));
    }
    for (int i = 0; i < binData.get(0).size(); i++) {
        for (int j = 0; j < 3; j++) {
            outputDiffFile.write(binData.get(j).get(i));
        }
    }
    outputDiffFile.close();
  }
}
```

# ДОДАТОК В

## Код для стиснення бінарного файлу

```
import heapq
import os
from collections import defaultdict, Counter

class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''

    def __lt__(self, nxt):
        return self.freq < nxt.freq

def calculate_frequency(data):
    return Counter(data)

def build_huffman_tree(frequency):
    heap = [Node(freq, sym) for sym, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        new_node = Node(left.freq + right.freq, left.symbol + right.symbol,
left, right)
        heapq.heappush(heap, new_node)

    return heap[0]

def huffman_codes(node, val=''):
    codes = {}
    newVal = val + str(node.huff)

    if node.left:
        node.left.huff = '0'
        codes.update(huffman_codes(node.left, newVal))
    if node.right:
```

```
        node.right.huff = '1'
        codes.update(huffman_codes(node.right, newVal))

    if not node.left and not node.right:
        codes[node.symbol] = newVal

    return codes

def huffman_encoding(data, codes):
    encoded_data = ''.join(codes[byte] for byte in data)
    return encoded_data

def pad_encoded_data(encoded_data):
    extra_padding = 8 - len(encoded_data) % 8
    for i in range(extra_padding):
        encoded_data += "0"

    padded_info = "{0:08b}".format(extra_padding)
    encoded_data = padded_info + encoded_data
    return encoded_data

def get_byte_array(padded_encoded_data):
    b = bytearray()
    for i in range(0, len(padded_encoded_data), 8):
        byte = padded_encoded_data[i:i+8]
        b.append(int(byte, 2))
    return b

def compress(input_file, output_file):
    with open(input_file, 'rb') as f:
        data = f.read()

    frequency = calculate_frequency(data)
    huffman_tree = build_huffman_tree(frequency)
    codes = huffman_codes(huffman_tree)

    encoded_data = huffman_encoding(data, codes)
    padded_encoded_data = pad_encoded_data(encoded_data)
    b = get_byte_array(padded_encoded_data)

    with open(output_file, 'wb') as f:
        f.write(bytes(b))
```

```
    return len(data), len(b)

def main():
    input_file = 'diff_file.bin'
    output_file = 'compressed_diff_file.bin'

    original_size, compressed_size = compress(input_file, output_file)
    compression_ratio = original_size / compressed_size

    print(f'Original size: {original_size} bytes')
    print(f'Compressed size: {compressed_size} bytes')
    print(f'Compression ratio: {compression_ratio:.2f}')

if __name__ == '__main__':
    main()
```

# ДОДАТОК Г

## Код для стиснення відео

```
import cv2
import os
import heapq
from collections import defaultdict

def video_to_frames(video_path, output_folder):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Не вдалося відкрити відео: {video_path}")
        return

    frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_path = os.path.join(output_folder,
f"frame_{frame_count:04d}.png")
        cv2.imwrite(frame_path, frame)
        frame_count += 1

    cap.release()
    print(f"Збережено {frame_count} кадрів у папку: {output_folder}")

def split_frames(input_folder, key_frame_interval):
    key_frames_folder = os.path.join(input_folder, "key_frames")
    intermediate_frames_folder = os.path.join(input_folder,
"intermediate_frames")

    os.makedirs(key_frames_folder, exist_ok=True)
    os.makedirs(intermediate_frames_folder, exist_ok=True)

    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith('.png')])

    for i, frame_file in enumerate(frame_files):
```

```
        frame_path = os.path.join(input_folder, frame_file)
        if i % key_frame_interval == 0:
            os.rename(frame_path, os.path.join(key_frames_folder,
frame_file))
        else:
            os.rename(frame_path, os.path.join(intermediate_frames_folder,
frame_file))

    print(f"Опорні кадри збережено у папку: {key_frames_folder}")
    print(f"Проміжні кадри збережено у папку: {intermediate_frames_folder}")

def convert_to_ycrcb(input_folder):
    output_folder = os.path.join(input_folder, "ycrcb")
    os.makedirs(output_folder, exist_ok=True)

    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith('.png')])

    for frame_file in frame_files:
        frame_path = os.path.join(input_folder, frame_file)
        frame = cv2.imread(frame_path)
        ycrcb_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2YCrCb)
        output_path = os.path.join(output_folder, frame_file)
        cv2.imwrite(output_path, ycrcb_frame)

    print(f"Опорні кадри перетворено у YCrCb та збережено у папку:
{output_folder}")

def extract_keypoints(input_folder):
    output_folder = os.path.join(input_folder, "keypoints")
    os.makedirs(output_folder, exist_ok=True)

    orb = cv2.ORB_create()
    keypoints_dict = {}

    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith('.png')])

    for frame_file in frame_files:
        frame_path = os.path.join(input_folder, frame_file)
        frame = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
        keypoints, descriptors = orb.detectAndCompute(frame, None)
        keypoints_dict[frame_file] = (keypoints, descriptors)
```

```
        keypoints_frame = cv2.drawKeypoints(frame, keypoints, None, color=(0,
255, 0))
        output_path = os.path.join(output_folder, frame_file)
        cv2.imwrite(output_path, keypoints_frame)

    print(f"Вузлові точки виділено та збережено у папку: {output_folder}")
    return keypoints_dict

class Node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left = left
        self.right = right
        self.huff = ''

    def __lt__(self, other):
        return self.freq < other.freq

def huffman_encoding(data):
    frequency = defaultdict(int)
    for item in data:
        frequency[item] += 1

    heap = [Node(freq, symbol) for symbol, freq in frequency.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        left.huff = '0'
        right.huff = '1'
        newNode = Node(left.freq + right.freq, left.symbol + right.symbol,
left, right)
        heapq.heappush(heap, newNode)

    huffman_code = {}
    def generate_code(node, current_code=''):
        if node is None:
            return
        if node.left is None and node.right is None:
            huffman_code[node.symbol] = current_code
```

```
        generate_code(node.left, current_code + '0')
        generate_code(node.right, current_code + '1')

    root = heapq.heappop(heap)
    generate_code(root)
    return huffman_code

def compress_keypoints(keypoints_dict):
    all_descriptors = []
    for key, (keypoints, descriptors) in keypoints_dict.items():
        if descriptors is not None:
            all_descriptors.extend(descriptors.flatten().tolist())

    huffman_code = huffman_encoding(all_descriptors)

    compressed_data = ''.join(huffman_code[item] for item in all_descriptors)
    print("Стиснення завершено за допомогою алгоритму Хаффмана.")
    return compressed_data, huffman_code

def convert_to_rgb(input_folder):
    output_folder = os.path.join(input_folder, "rgb_converted")
    os.makedirs(output_folder, exist_ok=True)

    frame_files = sorted([f for f in os.listdir(input_folder) if
f.endswith('.png')])

    for frame_file in frame_files:
        frame_path = os.path.join(input_folder, frame_file)
        frame = cv2.imread(frame_path)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_YCrCb2BGR)
        output_path = os.path.join(output_folder, frame_file)
        cv2.imwrite(output_path, rgb_frame)

    print(f"Кадри перетворено з YCrCb у RGB та збережено у папку:
{output_folder}")

def compare_file_sizes(original_frame_path, compressed_folder):
    original_size = os.path.getsize(original_frame_path)

    compressed_rgb_folder = os.path.join(compressed_folder, "rgb_converted")
    compressed_frame_path = os.path.join(compressed_rgb_folder,
os.path.basename(original_frame_path))
    compressed_size = os.path.getsize(compressed_frame_path)
```

```
    print(f"Розмір оригінального кадру: {original_size} байт")
    print(f"Розмір стиснутого та перетвореного кадру: {compressed_size}
байт")

# Приклад використання
video_path = "input/test_video.mp4"
output_folder = "output"
key_frame_interval = 5  # Встановіть інтервал опорних кадрів

# Спочатку розбиваємо відео на кадри
video_to_frames(video_path, output_folder)

# Тепер розділяємо кадри на опорні та проміжні
split_frames(output_folder, key_frame_interval)

# Перетворюємо опорні кадри з RGB у YCrCb
key_frames_folder = os.path.join(output_folder, "key_frames")
convert_to_ycrcb(key_frames_folder)

# Виділяємо вузлові точки на опорних кадрах
ycrcb_folder = os.path.join(key_frames_folder, "ycrcb")
keypoints_dict = extract_keypoints(ycrcb_folder)

# Стискаємо вузлові точки за допомогою алгоритму Хаффмана
compressed_data, huffman_code = compress_keypoints(keypoints_dict)

# Порівнюємо розмір оригінального і стиснутого кадру
original_folder = output_folder
random_frame = os.path.join(original_folder, "output/frame_0000.png")
compare_file_sizes(random_frame, key_frames_folder)
```

# ДОДАТОК Д

## Матеріали апробації роботи

### В.1 XXI Міжнародній науковій конференції «Ольвійський форум – 2023»

Міністерство освіти і науки України
Національна академія наук України
Південний науковий центр НАН та МОН України
Чорноморський національний університет імені Петра Могили
Первинна профспілкова організація ЧНУ імені Петра Могили
Інститут української археографії та джерелознавства ім. М.С. Грушевського НАНУ
Державний архів Миколаївської області
ДУ «Національний науковий центр радіаційної медицини НАМН України»
Державний аграрний університет Молдови (Кишинів)
Університет гуманітарних та природничих наук ім. Яна Длугоша (Польща)
Університет імені Адама Міцкевича (Польща)
Leipzig University of Applied Sciences (Німеччина)
Ca` Foscari University, Venice (Італія).

**ОЛЬВІЙСЬКИЙ ФОРУМ – 2023:**
**стратегії країн Причорноморського регіону**
**в геополітичному просторі**

*XVII Міжнародна наукова конференція*

**ТЕЗИ**

**ТЕХНІЧНІ НАУКИ**
**СТАЛИЙ РОЗВИТОК УНІВЕРСИТЕТСЬКОЇ СИСТЕМИ ОСВІТИ**

*15–18 червня 2023 р., м. Миколаїв,Україна*

Миколаїв – 2023

УДК 004.627

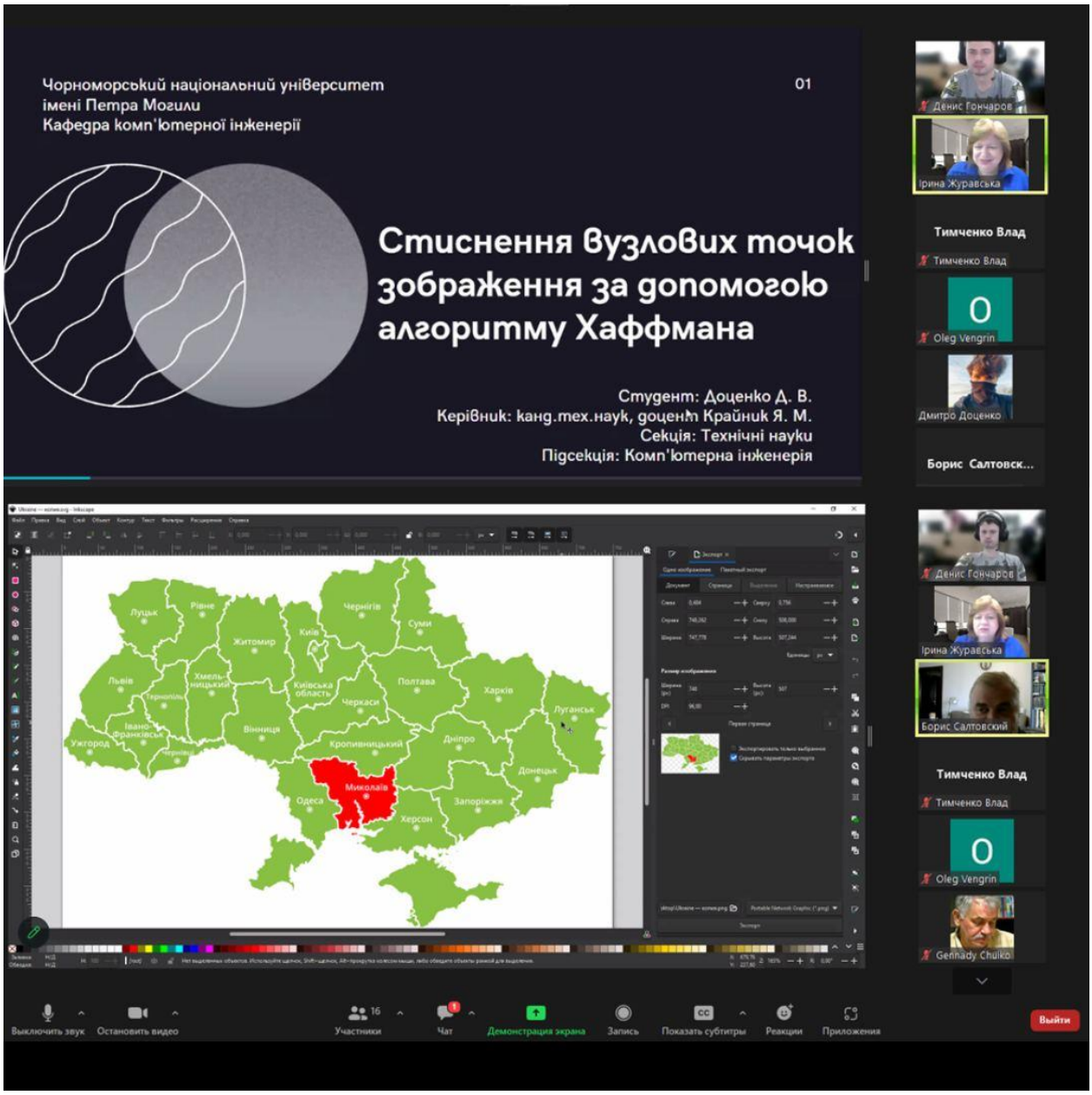
***Крайник Я. М.,***
канд. техн. наук, доцент кафедри комп'ютерної інженерії,
ЧНУ імені Петра Могили, м. Миколаїв, Україна
***Доценко Д. В.,***
бакалаврант,
ЧНУ імені Петра Могили, м. Миколаїв, Україна

## СТИСНЕННЯ ВУЗЛОВИХ ТОЧОК ЗОБРАЖЕННЯ ЗА ДОПОМОГОЮ АЛГОРИТМУ ХАФФМАНА

Алгоритм Хаффмана є одним з найефективніших методів стиснення даних, який використовується для зменшення розміру інформації без втрати даних. Цей алгоритм заснований на використанні частоти символів у наборі даних для кодування за допомогою бітових послідовностей, що мають різну довжину. Актуальність цієї теми полягає в пошуку оптимального підходу до стиснення вузлових точок, що є одним із завдань у багатьох сферах, де обробка і передача зображень відіграють важливу роль. Наприклад, у сфері комп'ютерного зору та обробки зображень, стиснення вузлових точок може покращити ефективність алгоритмів розпізнавання образів та аналізу зображень, зменшуючи обчислювальні витрати.

У цьому дослідженні пропонується виділення вузлових точок як етап стиснення зображення. Він полягає у виділенні кожних N пікселів зображення, які будуть слугувати базою для подальшого відновлення зображення. Порівнюються два підходи до стиснення вузлових точок з використанням алгоритму Хаффмана: напряму та у представленні як різниці. У першому випадку ми стискаємо вузлові точки безпосередньо за допомогою алгоритму Хаффмана [1], що дозволяє зменшити загальну кількість бітів для представлення кольорової компоненти. У другому випадку ми розглядаємо стиснення вузлових точок у формі різниці. Замість стиснення окремих вузлових точок, ми стискаємо різниці між послідовними точками за допомогою алгоритму Хаффмана. Цей підхід особливо корисний у випадках, коли вузлові точки мають подібні значення та незначно змінюються з кадру на кадр. В результаті, ми кодуємо лише різниці, що дозволяє зберегти простір для зберігання даних.

Таким чином, обидва підходи використовують кодування Хаффмана для стиснення, проте в першому випадку ми зменшуємо кількість бітів, не втрачаючи точність вузлових точок, а в другому випадку ми стискаємо різниці між ними, що особливо корисно для подібних і не-


Чорноморський національний університет
імені Петра Могили
Кафедра комп'ютерної інженерії
01
Стиснення вузлових точок зображення за допомогою алгоритму Хаффмана
Студент: Доценко Д. В.
Керівник: канд.мех.наук, доцент Крайник Я. М.
Секція: Технічні науки
Підсекція: Комп'ютерна інженерія
Денис Гончаров
Ірина Журавська
Тимченко Влад
Oleg Vengrin
Дмитро Доценко
Борис Салтовск...
Борис Салтовский
Gennady Chulko
Выключить звук
Остановить видео
Участники
Чат
Демонстрация экрана
Запись
Показать субтитры
Реакции
Приложения
Выйти

## В.2 XXIII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення та перспективи інформаційних систем і технологій»

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Одеський національний технологічний університет**
**Університет Інформатики і прикладних знань, м.Лодзь, Польща**
**Національний технічний університет України «Київський політехнічний інститут»**
**Навчально-науковий інститут комп'ютерних систем і технологій «Індустрія 4.0» ім. П.М. Платонова**

**XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів**

# «СТАН, ДОСЯГНЕННЯ ТА ПЕРСПЕКТИВИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ»

***Матеріали конференції***

Одеса

**20-21 квітня 2023 р.**

*Матеріали конференції «Стан, досягнення та перспективи інформаційних систем і технологій»*

# Розділ 5.

# Комп'ютерні телекомунікаційні мережі та технології

**УДК 004.627**

## АЛГОРИТМ ПОПЕРЕДНЬОЇ ОБРОБКИ ЗОБРАЖЕНЬ ДЛЯ АЛГОРИТМУ QOI

**ДОЦЕНКО Д. В.** (dotsenko.d@chmnu.edu.ua), **КРАЙНИК Я. М.** (yaroslav.krainyk@chmnu.edu.ua)
Чорноморський національний університет імені Петра Могили

*Розроблено алгоритм попередньої обробки зображень для алгоритму стиснення зображень без втрат QOI.*

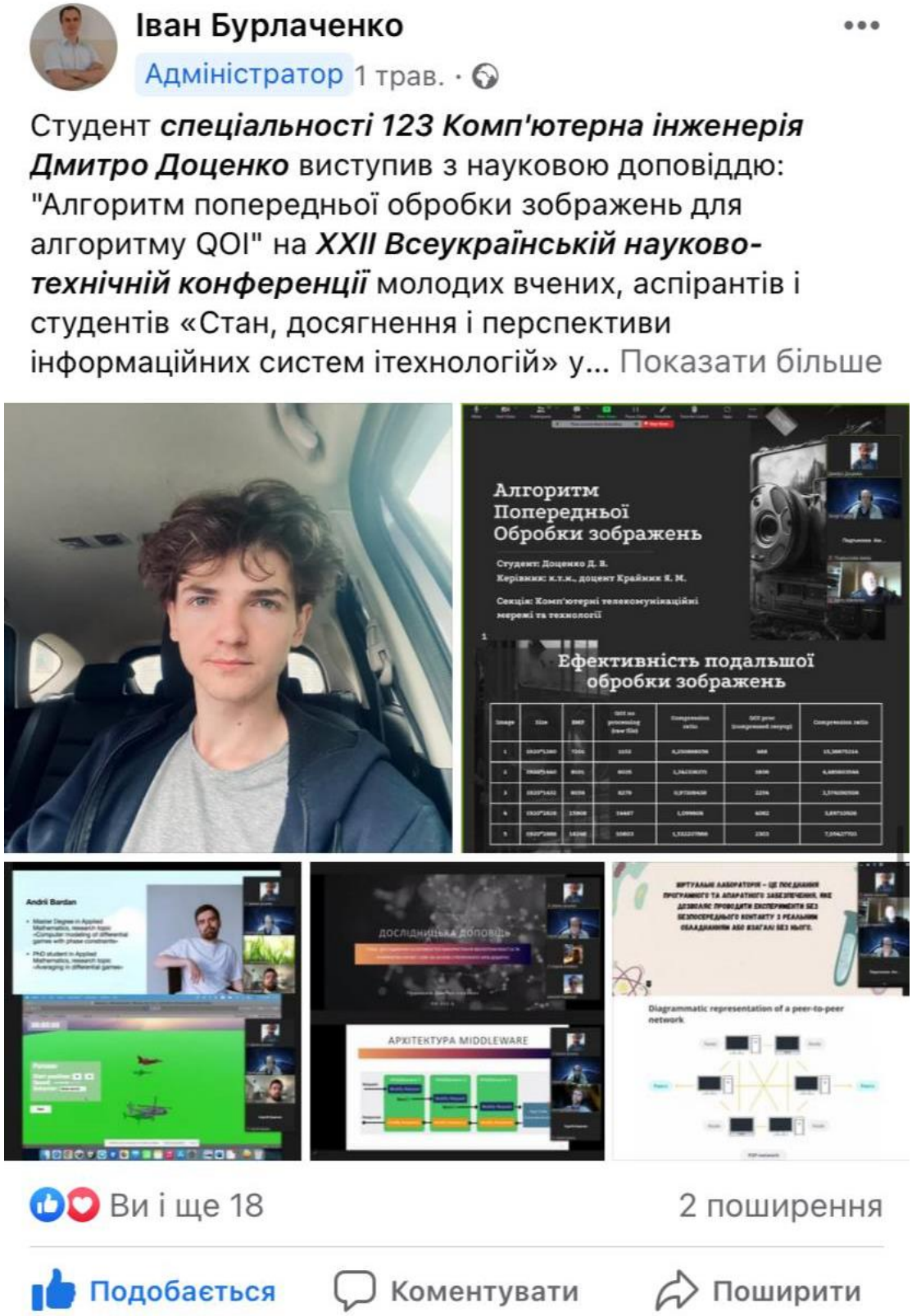
**Постановка проблеми:** У сучасному світі зображення стали невід'ємною частиною нашого життя, вони використовуються в різних галузях від медицини до мультимедіа. Проте, зображення займають багато місця, що робить їх важкими для зберігання та обробки. Тому, виникає проблема стиснення зображень без втрат, яке дозволить зменшити їх розмір, не втрачаючи якість зображення.

**Перелік вирішених задач:** З метою вирішення проблеми стиснення зображень без втрат було розроблено алгоритм попередньої обробки зображень для алгоритму QOI, який також доповнюється методом обробки зображень по блоках різного розміру.

**Виклад суті дослідження:** Спочатку варто розповісти що ж таке “QOI”. The Quite OK Image Format (QOI) - це алгоритм стиснення зображень без втрат, винайдений Домініком Шаблевським в кінці 2021 року. QOI забезпечує стиснення без втрат 24-бітних або 32-бітових кольорових растрових зображень за допомогою комбінації методів прогнозного кодування та ентропійного кодування. Прогнозне кодування використовує той факт, що сусідні пікселі зображення часто мають схожий колір. Алгоритм аналізує значення кольору сусідніх пікселів, щоб передбачити значення поточного пікселя. Потім різниця між прогнозованим значенням і фактичним значенням кодується за допомогою ентропійного кодування, що призводить до більш компактного представлення зображення. Ентропійне кодування використовує переваги статистичних властивостей даних для досягнення стиснення. У QOI ентропійне кодування використовується для стиснення значень різниці. Це робиться за допомогою варіанту алгоритму LZ77, який є популярним алгоритмом для стиснення даних без втрат. Загалом QOI забезпечує гарний баланс між ступенем стиснення та якістю зображення за допомогою комбінації цих методів.

Ми пропонуємо використати алгоритм попередньої обробки зображення, який базується на перетворенні зображення з простору кольорів RGB в YCrCb. Це стандартна практика в обробці зображень, яка дозволяє розділити зображення на яскравість (Y) та кольорові складові (Cr та Cb). Це дає змогу збільшити ефективність подальшої обробки зображення. Після перетворення зображення з простору кольорів RGB в YCrCb, проводиться обробка зображення по блоках різного розміру, тобто 4*4, 8*8 та інші. Кожен блок розглядається окремо, і значення пікселів в ньому усереднюються. Це забезпечує зменшення ентропії зображення, оскільки значення в блоках стають менш різноманітними, що сприяє кращому стисненню зображення.

Обробка зображення по блокам використовує білінійну апроксимацію по вертикалі та горизонталі. Це означає, що значення пікселів у блоках замінюються на середнє значення,


Іван Бурлаченко
Адміністратор 1 трав. ·
Студент спеціальності 123 Комп'ютерна інженерія Дмитро Доценко виступив з науковою доповіддю: "Алгоритм попередньої обробки зображень для алгоритму QOI" на XXII Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Стан, досягнення і перспективи інформаційних систем ітехнологій» у... Показати більше
Алгоритм Попередньої Обробки зображень
Ефективність подальшої обробки зображень
Andrii Bardan
АРХІТЕКТУРА MIDDLEWARE
Diagrammatic representation of a peer-to-peer network
Ви і ще 18
2 поширення
Подобається
Коментувати
Поширити

## В.3 XXVI Всеукраїнській науково-практичної конференції «Могилянські читання – 2023»

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України

ЧНУ

# «МОГИЛЯНСЬКІ ЧИТАННЯ – 2023:

## досвід та тенденції розвитку суспільства в Україні: глобальний, національний та регіональний аспекти»

XXVI Всеукраїнська науково-практична конференція

# ТЕЗИ ДОПОВІДЕЙ

Миколаїв, 6–10 листопада 2023 року

Миколаїв – 2023

**Тези доповідей**



**Підсекція:**
**ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ СИСТЕМИ**

**Тези доповідей**

УДК 004.627

*Доценко* ***Д. В.,***
*студент 4-го курсу,*
***Крайник Я. М.,***
*канд. тех. наук, доцент,доцент кафедри КІ,*
*ЧНУ імені Петра Могили, м. Миколаїв, Україна,*
*Aerotech Ltd, м. Дніпро, Україна, Radar Systems, м. Київ, Україна;*
*t. Aleksandrów Łódzki, Poland*

## МЕТОД СТИСНЕННЯ ПРОМІЖНИХ КАДРІВ ВІДЕО

У сучасному цифровому світі відеоконтент стає все більш поширеним засобом передачі інформації. Від стрімінгових сервісів до соціальних мереж – відеофайли стали невід'ємною частиною нашого щоденного життя. Основою передачі цих відео є специфікації та формати стиснення, які дозволяють зменшувати розмір файлу без високої втрати якості зображення. Один з найбільш поширених стандартів стиснення відео – MPEG. Хоча MPEG виявився ефективним для багатьох застосувань, його складність може виявитися завеликою для певних ситуацій або обмежених ресурсів, таких як мобільні пристрої або відеокамери з обмеженою обчислювальною потужністю. Отже, існує актуальна потреба у розробці нового, спрощеного методу стиснення, який би забезпечив відмінну якість при меншому обсязі обчислень, порівняно з MPEG.

Опорні кадри – це ключова концепція в системах стиснення відео, особливо в таких форматах, як MPEG. Вони служать як «референс» або «відправна точка» для інших кадрів, які слідують за ними, і, таким чином, відіграють важливу роль у процесі стиснення. Опорні кадри – часто називаються ключовими кадрами – це ті кадри, які кодуються незалежно. Це означає, що їх стиснення не базується на інформації з попередніх чи наступних кадрів. Вони подібні до звичайних стиснутих зображень, і саме вони служать основою для стиснення наступних «проміжних» кадрів. Роль опорних кадрів у відео можна порівняти з роллю книжкової закладки під час читання. Вони допомагають декодеру «орієнтуватися», де почати відтворення або як правильно відтворити наступні кадри. Через їх незалежність від інших кадрів, опорні кадри часто мають вищий розмір порівняно з проміжними. Однак їх присутність дозволяє здійснити більш ефективне стиснення для кадрів, які слідують за ними, використовуючи різницю між кадрами замість повного кодування кожного окремого кадру.

Проміжні кадри, відомі також як «різницеві кадри», є серцем більшості сучасних технік стиснення відео. На відміну від опорних кадрів, які кодуються повністю, проміжні кадри базуються на різницях між сусідніми кадрами. Проміжні кадри представляють зміни, які відбулися від часу опорного кадру. Замість того, щоб кодувати весь кадр, система стиснення зосереджується лише на тих пікселях, які змінилися в порівнянні із опорним кадром. Для визначення змін у проміжних кадрах система порівнює кожен піксель поточного кадру з відповідним пікселем у попередньому кадрі. Різниця між значеннями пікселів визначає, наскільки цей конкретний піксель змінився. Якщо піксель не змінився, він може бути пропущений або кодується як «без змін».

Одним зі способів оптимізації зберігання різницевої інформації є використання алгоритму Хаффмана [1]. Цей метод безвтратного стиснення даних базується на частоті появи певних значень різниці у відеокадрі. Значення, які з'являються частіше, отримують коротші коди, в той час як значення, що зустрічаються рідше, отримують довші коди. В результаті, загальний обсяг даних може бути значно зменшений без втрати якості.

Для цього дослідження було розроблено програму для перевірки різниці, вона призначена для порівняння двох зображень (які можуть вважатися опорним і проміжним кадрами) і генерує «теплову карту» різниць між ними. Така карта дає візуальне уявлення про те, де і як пікселі двох зображень відрізняються. При запуску програми очікується чотири аргументи: ширина зображення; висота зображення; перший файл – опорний кадр; другий файл – проміжний кадр (рис. 1).


Рисунок 1 – Опорний кадр (а), проміжний кадр (б)

Далі програма читає обидва зображення піксель за пікселем, після чого для кожного пікселя визначається різниця в значеннях між опорним та проміжним кадром. Якщо різниця дорівнює нулю (тобто пікселі однакові),

**В.4 Публікація статті «Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite Ok Image і Хаффмана» в науково фаховому виданні «Електронне моделювання», випуск №2 (2024 р.).**



DOI: https://doi.org/10.15407/emodel.46.01.075
УДК 004.627

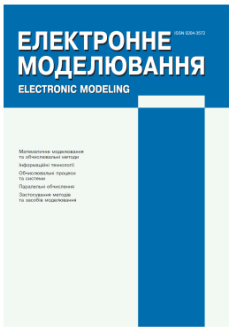
**Я.М. Крайник**, канд. тех. наук, **Д.В. Доценко**
Чорноморський національний університет ім. Петра Могили
Україна, 54003, Миколаїв, вул. 68 Десантників, 10
e-mail: yaroslav.krainyk@chmnu.edu.ua, dotsenko.d@chmnu.edu.ua

## Метод стиснення зображень з використанням попередньої обробки та алгоритмів Quite Ok Image і Хаффмана

Представлено метод стиснення зображень, який базується на комбінованому підході з використанням попередньої обробки зображень та алгоритму Хаффмана. Запропоновано організацію циклу обробки відповідно до цього методу та наведено експериментальні результати роботи методу на тестовому наборі зображень. Комбінований підхід дозволяє досягти кращих результатів у порівнянні з прямим використанням одного з методів. У ході процесу стиснення виконується перетворення у відповідний кольоровий формат. Далі відбувається розділення зображення на 2 блоки даних, основну область та область вузлових точок. До них застосовуються різні методи стиснення з урахуванням структури даних. Проведено порівняння використання алгоритму Хаффмана до області пікселів без додаткових перетворень та з представленням у вигляді різниці, яке показало, що представлення у вигляді різниці дозволяє уніфікувати розподіл значень для стиснення та покращити результат стиснення. Результати обох процесів комбінуються та передаються далі або для передачі на інші пристрої або для збереження інформації у стисненому стані. Запропонований метод стиснення може бути використаний при реалізації нових форматів стиснення у вбудованих системах, які мають обмежені обчислювальні потужності, але потребують роботи з графічними даними.

*Ключові слова: зображення, стиснення, алгоритм Хаффмана, вузлові точки, ефективність, якість, порівняння, алгоритм QOI.*

Сучасні цифрові технології надають великі можливості для мультимедіа представлення і відзначаються широким використанням в розробці різних пристроїв з високопродуктивними можливостями відображення контенту. Одним з ключових аспектів цього є робота з зображеннями, які стали необхідною частиною як повсякденного життя, так і професійної сфери. Зображення допомагають забезпечити високу точність та наочність в різних галузях, починаючи від медицини, де вони використо-




UK EN +38 (044) 424-14-66 em@ipme.kiev.ua

ЕЛЕКТРОННЕ МОДЕЛЮВАННЯ

ПРО ЖУРНАЛ АРХІВ НОМЕРІВ ДЛЯ АВТОРІВ РЕДКОЛЕГІЯ КОНТАКТИ

Головна / АРХІВ НОМЕРІВ / 2024 рік / Том 46, № 2 (2024)

Електронне моделювання

Том 46, №2 (2024)

https://doi.org/10.15407/emodel.46.02

ЗМІСТ