НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

Виконав студент IV курсу, групи ІП-02
(шифр групи)

Гущін Дмитро Олексійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафедри ІПІ, к.т.н., Олійник Ю. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ ,к.ф.-м.н., доцент Рибачук Л.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Гущіну Дмитру Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

керівник проєкту Олійник Юрій Олександрович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту «17» червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розробка функціональних вимог, розробка нефункціональних вимог.
3) Конструювання програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використання

2) Схема структурна класів програмного забезпечення

3) Схема структурна послідовності

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	25.03.2024	
4	Розробка інформаційного забезпечення	01.04.2024	
5	Алгоритмізація задачі	08.04.2024	
6	Обґрунтування вибору використаних технічних засобів	15.04.2024	
7	Розробка програмного забезпечення	29.04.2024	
8	Налагодження програми	06.05.2024	
9	Виконання графічних документів	13.05.2024	
10	Оформлення пояснювальної записки	27.05.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Дмитро ГУЩІН
(підпис) (ініціали, прізвище)

Керівник _____ Юрій ОЛІЙНИК
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 18 таблиць, 14 рисунків та 6 джерел – загалом 58 сторінок.

Дипломний проєкт присвячений розробці та впровадженню методу кодування растрових зображень на основі подібності фрагментів з оператором зсуву.

Мета дослідження полягає в модифікації методу кодування растрових зображень на основі подібності фрагментів шляхом додавання оператора зсуву.

Об'єкт дослідження: закодоване представлення растрового зображення.

Предмет дослідження: методи та алгоритми кодування растрових зображень на основі подібності фрагментів з оператором зсуву.

У розділі "Передпроєктне обстеження предметної області" розглянуто основні поняття та технології, пов'язані з кодуванням та декодуванням зображень, а також проаналізовано існуючі методи та алгоритми.

Розділ "Розроблення вимог до програмного забезпечення" присвячений визначенню функціональних та нефункціональних вимог до програмного забезпечення.

Розділ "Конструювання та розроблення програмного забезпечення" описує процес розробки програмного забезпечення, включаючи вибір технологій, архітектури системи, та реалізацію основних функцій.

У розділі "Аналіз якості та тестування програмного забезпечення" описано процеси тестування для забезпечення надійності та стабільності роботи програми.

Розділ "Розгортання та супровід програмного забезпечення" включає повний опис процесу розгортання програмного забезпечення.

КЛЮЧОВІ СЛОВА: КОДУВАННЯ ЗОБРАЖЕНЬ, ДЕКОДУВАННЯ ЗОБРАЖЕНЬ, WEB-ІНТЕРФЕЙС, PYTHON, OPENCV, TENSORFLOW, MONGODB, КОМПРЕСІЯ ДАНИХ, SSIM.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 18 tables, 14 figures and 6 sources – in total 58 pages.

The diploma project is dedicated to the development and implementation of a raster image encoding method based on fragment similarity with a shift operator.

The aim of the research is to modify the raster image encoding method based on fragment similarity by adding a shift operator.

Object of study: the encoded representation of a raster image.

Subject of study: methods and algorithms for encoding raster images based on fragment similarity with a shift operator.

The section "Pre-project examination of the subject area" examines the basic concepts and technologies related to image encoding and decoding, as well as analyzes existing methods and algorithms.

The section "Development of software requirements" is dedicated to defining the functional and non-functional requirements for the software.

The section "Design and development of software" describes the process of software development, including the choice of technologies, system architecture, and implementation of key functions.

The section "Quality analysis and software testing" describes the testing processes to ensure the reliability and stability of the software.

The section "Deployment and maintenance of software" includes a complete description of the software deployment process.

KEYWORDS: IMAGE ENCODING, IMAGE DECODING, WEB INTERFACE, PYTHON, OPENCV, TENSORFLOW, MONGODB, DATA COMPRESSION, SSIM.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0215.045480.01.91	Технічне завдання	11	
3	А4	КПІ.ІП-0215.045480.02.81	Пояснювальна записка	58	
4	А4	КПІ.ІП-0215.045480.03.12	Текст програми	13	
5	А4	КПІ.ІП-0215.045480.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0215.045480.05.34	Керівництво користувача	7	
7	А3	КПІ.ІП-0215.045480.06.99	Графічний матеріал	4	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-0215.045480.00.90		
Розроб.		Гущін Д.О.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Олійник Ю.Л.					
Н.контр.		Шулькевич Т.В.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-02	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Програмне забезпечення кодування растрових зображень
на основі подібності фрагментів з оператором зсуву
Технічне завдання

КПІ.ІП-0215.045480.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Юрій ОЛІЙНИК

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Дмитро ГУЩІН

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву.

Галузь застосування: обробка зображень.

Наведене технічне завдання поширюється на розробку програмного забезпечення "Кодування та декодування растрових зображень на основі подібності фрагментів з оператором зсуву", котре використовується для кодування растрових зображень шляхом поділу їх на фрагменти, аналізу подібності цих фрагментів та подальшого їхнього кодування з використанням оператора зсуву, що дозволяє підвищити ефективність стиснення зображень та забезпечити високу якість декодування. Програмне забезпечення призначене для використання в галузях, пов'язаних з обробкою та зберіганням зображень, таких як цифрова фотографія, медична візуалізація та інші сфери, де необхідно зберігати великі обсяги зображень. Можливими користувачами програмного забезпечення є фахівці з обробки зображень, розробники програмного забезпечення, дослідники в галузі комп'ютерного зору та інші професіонали, що працюють з великими обсягами растрових зображень.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного забезпечення "Кодування та декодування растрових зображень на основі подібності фрагментів з оператором зсуву" є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для зменшення пам'яті, необхідної для зберігання зображень, а також покращення ефективності їх передачі через мережі зв'язку.

Метою розробки є модифікація методу кодування растрових зображень на основі подібності фрагментів шляхом додавання оператора зсуву.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– завантаження зображення;
– кодування зображення;
– декодування зображення;
– перегляд статистики кодування зображення.

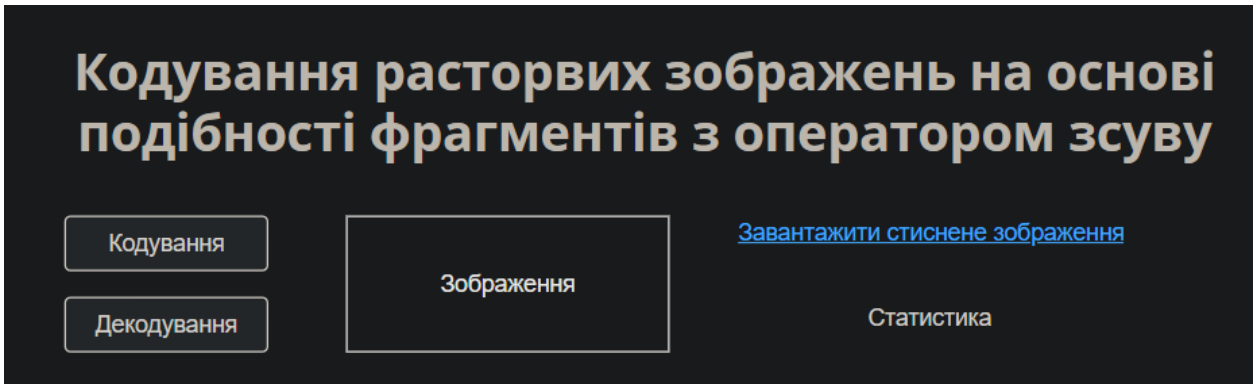


Рисунок 4.1 - Макет користувацького інтерфейсу

4.1.2 Для користувача:

– кодування зображення;
– перегляд ступеня стиснення;
– перегляд подібності оригінального і відновленого зображення;
– можливість завантажити стиснене зображення;
– відновлення зображення.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows, macOS та Linux.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: растрове зображення формату JPG, JPEG, PNG.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: набір байт.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою редактора Visual Studio Code.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді посилання на репозиторій на платформі GitHub, а також у додатках до пояснювальної записки.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема структурна класів програмного забезпечення;
– схема структурна діаграми послідовності.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву**

КПІ.ІП-0215.045480.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE – Integrated Development Environment – інтегроване середовище розробки.

LZMA – Lempel-Ziv-Markov chain-Algorithm – алгоритм стиснення даних.

SSIM – Structure Similarity – індекс структурної подібності.

JPEG – Joint Photographic Experts Group – стандарт стиснення зображень.

PNG – Portable Network Graphics – формат зображення.

RLE – Run-Length Encoding – метод стиснення даних.

DCT – Discrete Cosine Transform – алгоритм перетворення сигналу.

LZW – Lempel-Ziv-Welch – алгоритм стиснення даних.

БД – База даних.

ВСТУП

Сьогоднішній темп технологічного розвитку вимагає пошуку ефективних методів оптимізації та покращення функціонування різних систем і процесів. Одним із ключових завдань є стиснення та передача даних, що стає все більш важливим у зв'язку зі зростанням обсягів інформації. Високі вимоги до ресурсів пам'яті та пропускної здатності при зберіганні та передачі зображень створюють потребу у розробці ефективних методів стиснення та кодування. Ці методи мають оптимізувати використання обчислювальних ресурсів та забезпечити більш ефективне управління обсягами даних.

Підставою для виконання даної роботи є необхідність розробки ефективних методів стиснення зображень з використанням алгоритмів, які забезпечують не лише високу ступінь компресії, але й збереження важливих деталей та якості візуального сприйняття. Це дозволить зменшити обсяги пам'яті, необхідні для зберігання зображень, а також покращити ефективність їх передачі через мережі зв'язку.

На сьогоднішній день існує кілька широко використовуваних методів для стиснення зображень. Один з найбільш поширених методів – це стандартний алгоритм стиснення з втратами, такий як JPEG. Цей метод використовує дискретне косинусне перетворення (DCT) для розкладання зображення на компоненти частоти, які потім квантуються та кодуються. Хоча JPEG добре працює для багатьох типів зображень, він не завжди ефективний для стиснення зображень з великою кількістю деталей або текстур. Інші популярні методи – це алгоритми стиснення без втрат. Ці методи використовують різні підходи, такі як кодування RLE (Run-Length Encoding) або LZW (Lempel-Ziv-Welch), для зменшення розміру файлу без втрати якості. Недоліком цих методів є те, що вони не враховують взаємозв'язок між схожими зображеннями, що може призводити до менш ефективного стиснення в порівнянні з методами, які використовують фрагменти подібних зображень.

Стиснення зображень має широкі можливості застосування в різних галузях, включаючи мультимедіа, відеонагляд, а також веб-розробку та мобільні додатки. В мультимедійних застосуваннях воно дозволяє ефективно зберігати та передавати великі обсяги відео та зображень через Інтернет. В галузі відеонагляду воно дозволяє зберігати та аналізувати великі обсяги відеоданих, що підвищує ефективність систем відеонагляду. У веб-розробці та мобільних додатках стиснення зображень зменшує час завантаження сторінок та додатків, що підвищує зручність користування та ефективність роботи.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Зображення – це візуальне відображення об'єкта, яке може бути збережене та відтворене за допомогою різних технологій. Основні типи зображень – це векторні і растрові.

Растрові зображення, також відомі як бітові маски, представляють собою масиви пікселів. Піксель – це найменша одиниця растрового зображення, яка представляє собою квадратну область на екрані та визначається з кольором та яскравістю. Розмір растрового зображення вимірюється у пікселях по ширині та висоті. Растрові зображення ідеально підходять для фотографій та складних зображень, але вони можуть страждати від втрати якості при масштабуванні. Якість зображення визначається його роздільною здатністю. Роздільна здатність – це кількість пікселів на зображенні. Чим вища роздільна здатність, тим більше деталей буде видно на зображенні.

Векторні зображення складаються з геометричних об'єктів, таких як лінії, криві, полігони, які описують форму та кольори об'єктів. Вони зберігаються у вигляді математичних формул, що дозволяє масштабувати їх без втрати якості. Векторні зображення ідеально підходять для створення логотипів, іконок, малюнків та інших графічних об'єктів, які не потребують фото реалістичності. У цьому дослідженні використовуються саме растрові зображення, оскільки вони є найбільш поширеними. Окрім цього, робота з векторною графікою є складнішою і вимагає спеціальних інструментів.

Кодування або стиснення – це процес перетворення цифрового зображення у формат, який потребує менше місця для зберігання або передачі. Це робиться шляхом видалення надмірної інформації з зображення без значного впливу на його візуальну якість.

Стиснення зображень може бути з втратами і без втрат. Під час стиснення з втратами деяка інформація втрачається, що може призвести до

зниження якості зображення. Цей тип стиснення дозволяє значно зменшити розмір файлу, але може призвести до втрати різкості та появи артефактів. Стиснення без втрат зберігає всю інформацію, що дозволяє точно відтворити оригінал. Це важливо для зображень, де кожен піксель має значення, наприклад, медичні зображення. Однак це сильно обмежує ступінь стиснення і вимагає більше обчислювальних ресурсів.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації кодування зображень. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

Метод Хаффмана

Метод Хаффмана – це алгоритм кодування префіксів, який використовується для стиснення даних без втрат. Він був розроблений Девідом Хаффманом у 1952 році і є одним з найпоширеніших алгоритмів стиснення даних [1].

Алгоритм аналізує відносну частоту кожного символу (в нашому випадку – інтенсивності пікселів) в зображенні. Ця інформація використовується для побудови дерева Хаффмана. На основі частотної таблиці створюється бінарне дерево, в якому символи з найменшою частотою розташовані ближче до кореня, а символи з найбільшою частотою – далі від нього. Кожен символ (інтенсивність пікселя) замінюється бітовою послідовністю відповідно до його положення в дереві Хаффмана. Код для символу буде двійковим представленням шляху від кореневого вузла до листа, який відповідає цьому символу. Чим більша його частота, тим менше бітів використовується для його кодування. Замінені бітові послідовності збираються в одну компактну послідовність, що дозволяє стиснути дані. Для відновлення оригінального зображення використовується те ж саме дерево Хаффмана, щоб розкодувати бітові послідовності назад у символи.

Метод Хаффмана добре підходить для стиснення зображень без втрат, оскільки він забезпечує хорошу стискаючу ефективність при мінімальній втраті інформації. Однак він може бути менш ефективним для зображень з великим динамічним діапазоном або з великою кількістю унікальних значень пікселів.

RLE

Кодування довжин серій (RLE) – це простий алгоритм стиснення даних без втрат, який використовується для стиснення даних, що містять довгі послідовності однакових символів [2].

Алгоритм замінює послідовність однакових символів одним символом та підрахунком кількості повторень цього символу. Наприклад, рядок "AAAABBBCCDAA" можна стиснути до "4A3B2C1D2A", де "4A" означає 4 символи "A", "3B" - 3 символи "B" і т.д.

Переваги методу стиснення RLE полягають у його простоті та швидкості. Цей метод легко і швидко застосовується до даних з великою кількістю послідовних повторень, що дозволяє досягти значного ступеня стиснення без значних витрат на обчислення.

Недоліки RLE включають обмежену ефективність стиснення для даних з низькою кількістю повторень або з великою кількістю унікальних символів. Крім того, для збереження інформації про кількість повторень потрібно додаткове місце, що може призводити до збільшення розміру стисненого файлу у випадку, коли повторень немає багато.

Алгоритм часто використовується для стиснення растрових зображень, таких як логотипи та іконки. RLE може використовуватися для стиснення тексту, але зазвичай він не такий ефективний, як інші алгоритми стиснення тексту, такі як метод Хаффмана.

LZW

Метод стиснення LZW (Lempel-Ziv-Welch) використовується для зменшення розміру даних шляхом заміни повторюваних фрагментів на короткі коди [3].

Алгоритм LZW використовує словник, який містить усі можливі символи, що зустрічаються в даних. Спочатку словник містить лише символи, які є окремими символами вхідних даних. Алгоритм сканує вхідні дані посимвольно, шукаючи послідовності символів, які вже існують у словнику. Якщо знайдено відповідність, алгоритм зберігає код цієї відповідності та продовжує сканування з наступного символу після відповідності. Якщо відповідність не знайдено, алгоритм створює новий елемент у словнику, що складається з уже знайденої найдовшої відповідності та наступного символу. Також зберігається код цієї відповідності. Вихідними даними є послідовність кодів, які відповідають знайденим відповідностям або доданим елементам словника.

Переваги LZW включають високу ступінь стиснення для різних типів даних, зокрема текстових, зображень і звукових файлів. Цей метод добре працює з даними, які містять багато повторюваних фрагментів або шаблонів. Також LZW є відносно простим у реалізації і швидким у роботі, особливо для комп'ютерних систем з достатньою кількістю пам'яті.

Недоліки LZW включають високі вимоги до обсягу пам'яті для зберігання словника кодів, що може призвести до збільшення розміру стисненого файлу. Крім того, алгоритм може бути менш ефективним для даних без повторюваних фрагментів або для коротких файлів.

Щодо застосування, LZW широко використовується у комп'ютерних системах для стиснення текстових файлів, зображень (наприклад, у форматах GIF і TIFF) і аудіо-даних. Його також можна знайти у різних мережевих протоколах, таких як компресія даних у форматі gzip та у багатьох інших програмах і форматах файлів.

DCT

Дискретне косинусне перетворення (DCT) – це математичний алгоритм, який використовується для аналізу та стиснення сигналів, зокрема зображень і аудіо. Алгоритм DCT перетворює вхідний сигнал з часової і просторової області у частотну область [4].

DCT перетворює сигнал на суму косинусних функцій з різними частотами. У контексті зображень, DCT розбиває зображення на блоки пікселів та перетворює кожен блок у набір коефіцієнтів, які представляють частоти, що присутні у цьому блоку. Нижчі коефіцієнти DCT представляють низькочастотні компоненти сигналу (загальні риси зображення), а вищі коефіцієнти представляють високочастотні компоненти (деталі зображення). Зазвичай, DCT використовується в стандартах стиснення зображень, таких як JPEG, для зменшення обсягу даних. Після DCT велика частина високочастотної інформації, яка не важлива для людського сприйняття, може бути відкинута або стиснута, що дозволяє досягти значних ступенів стиснення з мінімальним втрачанням якості.

JPEG

JPEG (Joint Photographic Experts Group) – це стандарт стиснення зображень, який використовується для зменшення обсягу файлів з мінімальним впливом на якість. У стандарті JPEG використовується комбінація різних методів стиснення, включаючи DCT, квантування, та кодування диференціалів.

Спочатку JPEG перетворює зображення з RGB (червоний, зелений, синій) в модель YCbCr (Y – яскравість, Cb – різниця між синім та Y, Cr – різниця між червоним та Y). Це робиться тому, що людське око більш чутливе до яскравості, ніж до кольору. Кожен компонент YCbCr розбивається на блоки 8x8 пікселів. Для кожного блоку виконується DCT, яке перетворює просторові дані блоку в частотні дані. Нижчі частоти містять основну візуальну інформацію, а вищі – деталі. Коефіцієнти DCT квантуються за допомогою спеціальної матриці квантування. Ця матриця призначає більші значення ваги низькочастотним коефіцієнтам та менші значення ваги високочастотним коефіцієнтам. Таким чином, відкидаються менш важливі деталі зображення. За допомогою методу кодування, такого як Хаффмана, стиснуті коефіцієнти DCT упаковуються для ефективного зберігання.

Переваги стандарту JPEG включають ефективність стиснення, підтримку кольорів та параметризацію якості, що дозволяє користувачам налаштовувати рівень стиснення зображень. JPEG є одним з найпоширеніших форматів зображень і сумісний з більшістю програм для роботи з зображеннями та операційними системами.

Однак у JPEG є певні недоліки. Перш за все, під час стиснення деталі можуть бути втрачені, особливо при високому ступені стиснення. Це може призводити до втрати різких країв та текстур у зображенні. Крім того, JPEG відомий своєю схильністю до артефактів стиснення, таких як блочна артефактизація та втрата деталізації, особливо помітна на зображеннях з високим контрастом або різкими переходами кольорів.

Вейвлет-перетворення

Вейвлет-перетворення – це метод стиснення зображень, який базується на використанні вейвлет-аналізу для розкладання зображення на різні компоненти частот та просторових деталей. Цей метод дозволяє ефективно виділити і зберегти важливу інформацію, використовуючи менше даних, ніж інші методи стиснення [5].

Вейвлети – це короткі хвилеподібні функції, які локалізовані як в часі, так і в частоті. Це дозволяє їм ефективно виявляти як високочастотні, так і низькочастотні компоненти сигналу на різних часових інтервалах. Під час вейвлет-перетворення зображення розбивається на рівні деталізації, представлені як набори коефіцієнтів вейвлет-перетворення. Ці коефіцієнти зберігають інформацію про частотні складові та просторові деталі зображення. Потім коефіцієнти можна стиснути, видаляючи незначні деталі та використовуючи різні методи кодування для зменшення розміру даних.

Однією з переваг вейвлет-перетворення є його здатність забезпечувати якісний стиснення зображення при збереженні важливих деталей із високою точністю. Крім того, вейвлет-перетворення дозволяє ефективно видаляти шум та інші небажані артефакти зображення.

Проте, існують деякі недоліки. Деякі алгоритми вейвлет-перетворення можуть бути складними у реалізації та вимагати значних обчислювальних ресурсів. Крім того, підбір оптимальних параметрів для кожного зображення може бути складним завданням.

Незважаючи на деякі недоліки, вейвлети широко застосовуються в різних областях завдяки своїм можливостям аналізувати нестаціонарні сигнали та виявляти локальні особливості.

Кодування на основі фрагментів

Розроблений метод кодування растрових зображень використовує підхід, що полягає в розбитті вхідного зображення на невеликі фрагменти. Потім для кожного фрагменту виконується пошук подібних фрагментів у базі даних. За допомогою цього пошуку кожному вхідному фрагменту призначається ідентифікатор, який відповідає подібному фрагменту з бази даних. Після цього набір ідентифікаторів стискається, щоб зменшити обсяг даних.

Метод містить два основні етапи роботи. Перший етап – це наповнення бази даних фрагментами, коли для кожного фрагменту зображення зберігається відповідний ідентифікатор. Другий етап – це безпосередньо процес кодування зображення, коли для кожного фрагменту вхідного зображення встановлюється відповідний ідентифікатор, і згруповані ідентифікатори стискаються для подальшої передачі або зберігання.

Отже, метод розділяє зображення на менші частини, знаходить подібні фрагменти у базі даних, і використовує цю інформацію для стиснення даних, що дозволяє зберегти значну кількість інформації при зменшенні обсягу даних.

Кодування на основі фрагментів з оператором зсуву

Дипломний проєкт – це модифікація методу кодування на основі фрагментів [6], яка базується на виявленні та використанні подібності між фрагментами зображення за допомогою оператора зсуву.

Головною відмінністю цього методу є можливість врахування малих зсувів об'єктів на зображенні, що дозволяє знаходити більше подібних фрагментів і отримувати кращі результати стиснення. Крім того, завдяки використанню оператора зсуву, можна досягти більшої гнучкості при виборі фрагментів для порівняння.

Проте, цей метод може бути більш вимогливим до обчислювальних ресурсів через необхідність враховувати додаткові варіанти зсуву фрагментів. Крім того, збільшення кількості операцій порівняння може призвести до збільшення часу обробки.

В таблиці 1.2 наведено порівняння методу кодування зображень на основі фрагментів з аналогами.

Таблиця 1.2 – Порівняння методів кодування

Метод	**Переваги**	**Недоліки**	**Застосування**
Метод Хаффмана	Простота реалізації	Не завжди забезпечує максимальне стиснення	Текстові дані, стиснення без втрат
RLE	Ефективний для даних з довгими серіями однакових символів	Низька ступінь стиснення для зображень зі складними структурами	Зображення з простими структурами, текстові дані
LZW	Ефективний для даних з повторюваними послідовностями	Може бути обчислювально складним	Текст, зображення
DCT	Висока якість стиснення зображення	Втрата інформації	Текст, зображення
JPEG	Високий ступінь стиснення, широко використовується	Втрата інформації, артефакти стиснення	Фотографії, зображення з високою деталізацією
Вейвлет-перетворення	Здатний забезпечувати високу якість стиснення і збереження деталей зображення	Може бути обчислювально складним	Фотографії, зображення з високою деталізацією

Продовження таблиці 1.2 – Порівняння методів кодування

Метод	**Переваги**	**Недоліки**	**Застосування**
Кодування на основі фрагментів	Ефективний для неоднорідних даних	Потребує зберігання бази даних з фрагментами зображення, може бути обчислювально складним	Фотографії, зображення з високою деталізацією
Кодування на основі фрагментів з оператором зсуву	Вища ступінь деталізації в порівнянні з кодуванням на основі фрагментів	Потребує зберігання бази даних з фрагментами зображення, може бути обчислювально складним	Фотографії, зображення з високою деталізацією

1.3 Опис бізнес-процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель (рисунок 1.3).

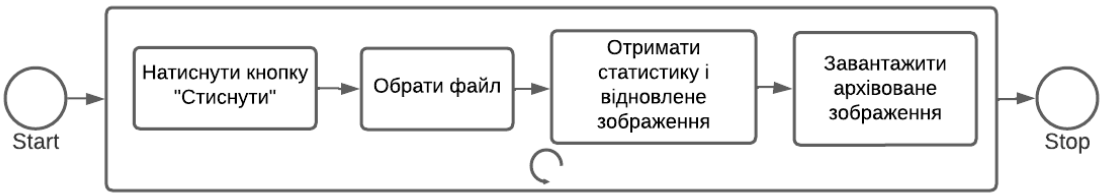


Рисунок 1.3 – Схема бізнес-процесу

Опис послідовності дій користувача:

а) натиснути кнопку «Стиснути»;

б) обрати зображення;

в) отримати декодоване зображення і статистику;

г) завантажити архівоване зображення.

1.4 Постановка задачі

Метою даного дослідження є розробка та вдосконалення методів стиснення та кодування зображень на основі подібності фрагментів з застосуванням оператора зсуву, що дозволить зменшити обсяги пам'яті, необхідні для зберігання зображень, та підвищити ефективність передачі даних, забезпечуючи при цьому збереження якості зображення на задовільному рівні.

На вході кодування отримуємо растрове зображення формату JPG, JPEG або PNG, а на виході отримуємо масив байтів, що зберігає стиснутий масив ідентифікаторів фрагментів в базі. Це дозволить ефективно зберігати та передавати зображення за допомогою мережі зв'язку або зменшити вимоги до обсягу пам'яті для зберігання. Головними задачами є реалізація алгоритмів стиснення та декомпресії, вибір оптимальних методів для зберігання даних та забезпечення високої якості відновлення зображень після стиснення.

Для досягнення мети повинні бути вирішені такі завдання:

а) аналіз існуючих методів стиснення та кодування зображень;
б) збір великого обсягу зображень для формування бази фрагментів;
в) оновлення алгоритму кодування растрових зображень за рахунок додавання оператора зсуву.;
г) програмна реалізація алгоритму кодування растрових зображень на основі подібності фрагментів з оператором зсуву;
д) створення веб інтерфейсу для тестування і подальшого використання розробленого алгоритму;
е) дослідження ефективності роботи програмного забезпечення.

Висновки до розділу

У цьому розділі виконано детальний аналіз предметної області, пов'язаної зі стисненням растрових зображень, що є важливою складовою багатьох сучасних інформаційних систем і мереж. Аналіз предметної області

підкреслив необхідність розробки вдосконалених методів стиснення, які могли б зменшити обсяги даних і покращити їх передачу та збереження.

Аналіз існуючих алгоритмів стиснення виявив, що кожен з них має свої особливі переваги та обмеження залежно від контексту їх застосування. Наприклад, RLE ефективний для зображень з однорідними областями, тоді як JPEG краще справляється з фотографіями завдяки своїй здатності мінімізувати видимість помилок стиснення в деталізованих областях.

Завданням проєкту є розробка оновленого алгоритму стиснення зображень на основі фрагментів за рахунок використання оператора зсуву, який має значно підвищити ефективність стиснення. Основні задачі включають реалізацію вдосконалених алгоритмів стиснення та декомпресії, вибір оптимальних методів зберігання даних та забезпечення високої якості зображень після стиснення, що є критично важливим для використання у реальних системах та застосуваннях.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головною функцією програмного забезпечення є кодування растрового зображення. Для цього необхідно натиснути кнопку «Стиснути» і обрати відповідний файл. Окрім цього, після кодування є можливість перегляду індексу схожості, відсотка стиснення, декодованого зображення і завантаження архівованого зображення.

Детальніше можна побачити на рисунку 2.1.

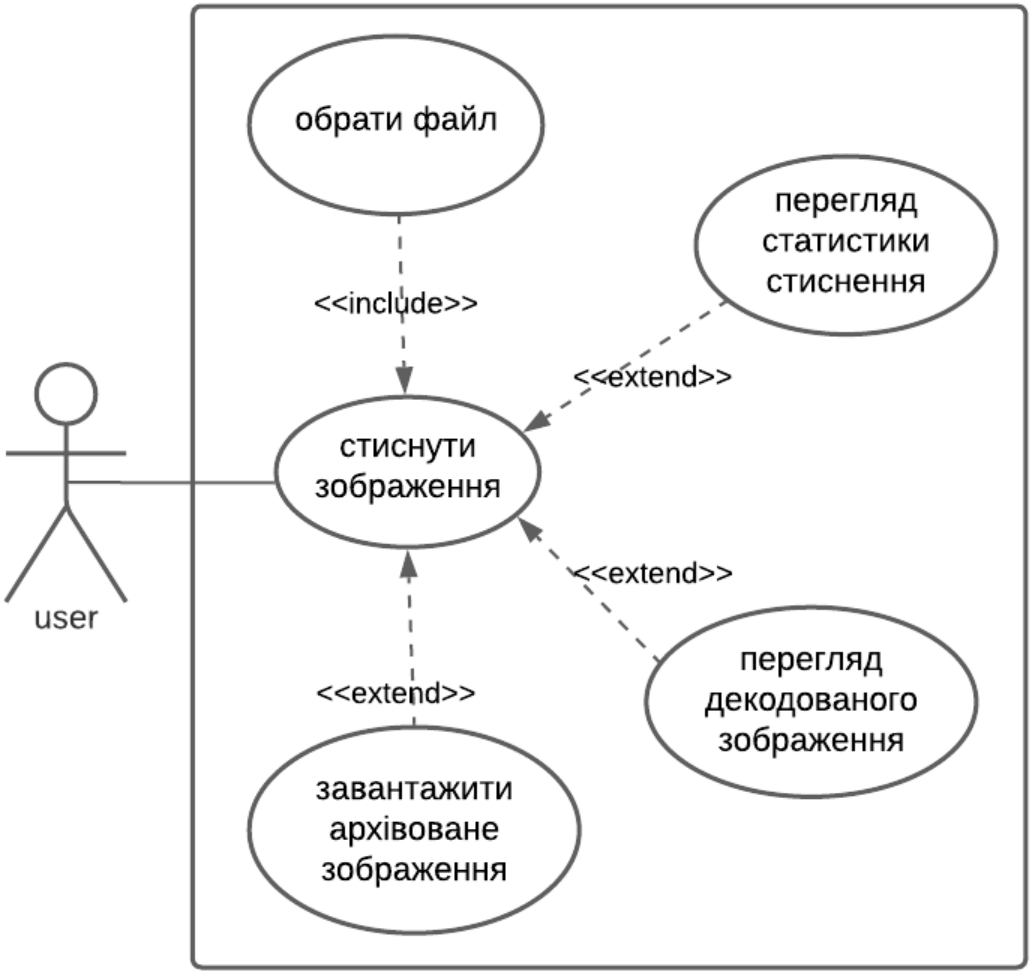


Рисунок 2.1 – Діаграма варіантів використання

В таблицях 2.2 - 2.6 наведені варіанти використання програмного забезпечення.

Таблиця 2.2 - Варіант використання UC-01

Use case name	Стиснути зображення
Use case ID	UC-01
Goals	Стиснути зображення
Actors	Користувач
Trigger	Користувач бажає закодувати зображення
Pre-conditions	-
Flow of Events	Користувач відкриває веб-інтерфейс. Натискає кнопку «Стиснути»
Extension	-
Post-Condition	Відкривається системне вікно для вибору файлу

Таблиця 2.3 - Варіант використання UC-02

Use case name	Обрати файл
Use case ID	UC-02
Goals	Обрати файл
Actors	Користувач
Trigger	Користувач бажає закодувати зображення
Pre-conditions	Користувач натиснув кнопку «Стиснути»
Flow of Events	Користувач вибирає файл з розширенням JPG, JPEG, PNG
Extension	У випадку вибору файлу з іншим розширенням виводиться повідомлення
Post-Condition	Зображення починає кодуватися

Таблиця 2.4 - Варіант використання UC-03

Use case name	Перегляд декодованого зображення
Use case ID	UC-03
Goals	Перегляд декодованого зображення
Actors	Користувач
Trigger	Користувач бажає закодувати зображення

Продовження таблиці 2.4

Pre-conditions	Користувач натиснув кнопку «Стиснути» і обрав файл
Flow of Events	Користувач відкриває веб-інтерфейс
Extension	-
Post-Condition	Користувач бачить декодоване зображення

Таблиця 2.5 - Варіант використання UC-04

Use case name	Перегляд статистики стиснення
Use case ID	UC-04
Goals	Перегляд статистики стиснення
Actors	Користувач
Trigger	Користувач бажає закодувати зображення
Pre-conditions	Користувач натиснув кнопку «Стиснути» і обрав файл
Flow of Events	Користувач відкриває веб-інтерфейс
Extension	-
Post-Condition	Користувач бачить ступінь стиснення, а також індекс структурної подібності оригінального і декодованого зображення

Таблиця 2.6 - Варіант використання UC-05

Use case name	Завантажити архівоване зображення
Use case ID	UC-05
Goals	Завантажити архівоване зображення
Actors	Користувач
Trigger	Користувач бажає завантажити архівоване зображення
Pre-conditions	Користувач вже закодував зображення
Flow of Events	Користувач натиснув кнопку «Завантажити»
Extension	-
Post-Condition	Користувач завантажив архівоване зображення

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.7 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.8.

Таблиця 2.7 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Зчитування зображення
FR-2	Розділення на фрагменти за допомогою оператора зсуву
FR-3	Виділення ознак зображення
FR-4	Збереження зображення і його ознак у базу даних
FR-5	Пошук подібного фрагменту у базі даних
FR-6	Стиснення набору ідентифікаторів фрагментів
FR-7	Відновлення зображення з набору ідентифікаторів фрагментів

На рисунку 2.3 наведена матриця трасування вимог.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7
UC-1	+	+	+	+	+	+	
UC-2	+						
UC-3							+
UC-4		+	+	+	+	+	
UC-5		+	+	+	+	+	

Рисунок 2.3 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Список нефункціональних вимог до програмного забезпечення:

а) підтримка GPU;
б) заповнена база даних фрагментів перед використанням кодування зображень;
в) ПЗ має бути надійним і відповідати вимогам ТЗ;
г) ПЗ має бути зрозумілим у використанні.

Висновки до розділу

Даний розділ описує функціональні та нефункціональні вимоги, а також сценарії використання програми. Головною метою проєкту є розробка та вдосконалення методів стиснення та кодування зображень на основі подібності фрагментів з застосуванням оператора зсуву.

Серед функціональних вимог можна виділити зчитування зображення, розділення на фрагменти, виділення ознак зображення, збереження та відновлення зображення, стиснення та декодування фрагментів.

Невід'ємною частиною розділу є також опис нефункціональних вимог, таких як підтримка GPU, заповнена база даних фрагментів, підтримка оператора зсуву, надійність програмного забезпечення та його зрозумілість для користувача.

Сценарії використання включають стиснення зображення, вибір файлу для кодування, перегляд декодованого зображення, перегляд статистики стиснення та завантаження стисненого зображення. Всі ці сценарії обслуговуються веб-інтерфейсом програми.

Таким чином, другий розділ надає загальне уявлення про можливості та функціональність розробленого програмного продукту. За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Програмне забезпечення представляє собою бібліотеку, яка включає набір класів та методів для кодування і декодування зображень, роботи з базою даних, а також веб-інтерфейсу. При розробці програмного забезпечення використовувалася об’єктно-орієнтована парадигма програмування.

У таблиці 3.1 наведено перелік створених класів та їх опис.

Таблиця 3.1 – Опис створених класів

Клас	**Опис**
Fragment	Клас, який використовується для представлення фрагменту зображення. Він містить зображення фрагменту, його координати та ознаки.
Encoder	Клас для роботи з кодуванням і декодуванням зображення. Він містить методи для розбиття зображення на фрагменти, їх підготовки, кодування та декодування, а також обчислення подібності між фрагментами.
BaseDB	Базовий клас, який використовується для роботи з базою даних. Він визначає основні методи, такі як додавання фрагментів, пошук подібних фрагментів та інші.
DBFactory	Клас для створення та отримання об'єктів бази даних. Залежно від типу бази даних, DBFactory повертає відповідний екземпляр класу для роботи з цією базою даних.
FileDB	Клас, який використовується для роботи з файловою базою даних. Він реалізує методи базового класу BaseDB для зберігання та обробки фрагментів у файлах.
MongoDB	Клас, який використовується для роботи з базою даних MongoDB. Він реалізує методи базового класу BaseDB для зберігання та обробки фрагментів у базі даних MongoDB.
App	Клас, який використовується для побудови веб-інтерфейсу. Він забезпечує завантаження зображень, їх кодування та декодування, а також відображення результатів і обчислення коефіцієнта стиснення та SSIM.

У таблиці 3.2 наведено опис атрибутів класів.

Таблиця 3.2 – Опис атрибутів класів

Клас	**Атрибут**	**Тип**	**Опис**
Fragment	img	np.array	Зображення фрагменту
	feature	np.array	Ознаки фрагменту
	x	int	Координата X верхнього лівого кута фрагменту
	y	int	Координата Y верхнього лівого кута фрагменту
Encoder	db_type	str	Тип бази даних
	db	BaseDB	Об'єкт бази даних
	similarity_threshold	float	Поріг подібності для фрагментів
	model	tf.keras.Model	Модель для витягу ознак зображень
	kernel_size	int	Розмір квадратного фрагменту
	step_size	int	Крок зміщення для фрагментів
BaseDB	target_size	int	Цільовий розмір фрагменту
	tree	AnnoyIndex	Дерево для пошуку найближчих сусідів (подібних фрагментів)
FileDB	save_path	str	Шлях до файлу для зберігання фрагментів
	fragments	dict	Словник для зберігання фрагментів

Продовження таблиці 3.2

Клас	**Атрибут**	**Тип**	**Опис**
MongoDB	client	MongoClient	Об'єкт клієнта MongoDB
	db	Database	Об'єкт бази даних MongoDB
	collection	Collection	Колекція для зберігання фрагментів
DBFactory	db	BaseDB	Об'єкт бази даних, створений залежно від вказаного типу
App	encoder	Encoder	Об'єкт класу Encoder для кодування та декодування зображень

У таблиці 3.3 наведено перелік методів і їх опис.

Таблиця 3.3 – Опис методів класів

Клас	**Метод**	**Опис**
Fragment	-	-
Encoder	__init__	Ініціалізує об'єкт Encoder, встановлює параметри та завантажує модель
	fill_db	Заповнює базу даних фрагментами зображень з вказаного каталогу
	encode	Кодує зображення, розбиваючи його на фрагменти та шукаючи подібні фрагменти у базі даних
	decode	Декодує стиснуті дані для відновлення зображення
	prepare_fragments	Підготовляє фрагменти зображень для кодування, витягаючи їхні ознаки

Продовження таблиці 3.3

Клас	**Метод**	**Опис**
Encoder	is_overlapping	Перевіряє, чи перекриваються два фрагменти
	split_image_into_fragments	Розбиває зображення на фрагменти заданого розміру
	reconstruct_image	Відновлює зображення з фрагментів
	blend_fragments	Змішує фрагменти для покращення якості відновленого зображення
	get_ssim	Обчислює середній SSIM між оригінальним та декодованим зображеннями
BaseDB	is_empty	Перевіряє, чи є база даних порожньою
	build_tree	Будує дерево для пошуку найближчих сусідів
	add_fragment	Додає фрагмент до бази даних
	get_fragment_by_id	Отримує фрагмент з бази даних за його ідентифікатором
	find_similar_fragment_id	Знаходить подібний фрагмент за його ознаками
FileDB	__init__	Ініціалізує файлову базу даних
	save_results	Зберігає фрагменти у файл
MongoDB	__init__	Ініціалізує базу даних MongoDB
	save_results	Зберігає фрагменти у базу даних MongoDB

Продовження таблиці 3.3

Клас	**Метод**	**Опис**
DBFactory	save_results	Ініціалізує фабрику баз даних
	get_db	Повертає об'єкт бази даних відповідно
App	__init__	Ініціалізує об'єкт App, створює екземпляр Encoder
	encode_image	Кодує та декодує зображення, повертає стиснуті дані та відновлене зображення
	run	Запускає веб-інтерфейс для завантаження та обробки зображень

Для відображення зв’язків між описаними вище класами було побудовано діаграму класів, яка зображена на рисунку 3.4. Повний вид діаграми винесено у графічний матеріал дипломного проєкту.

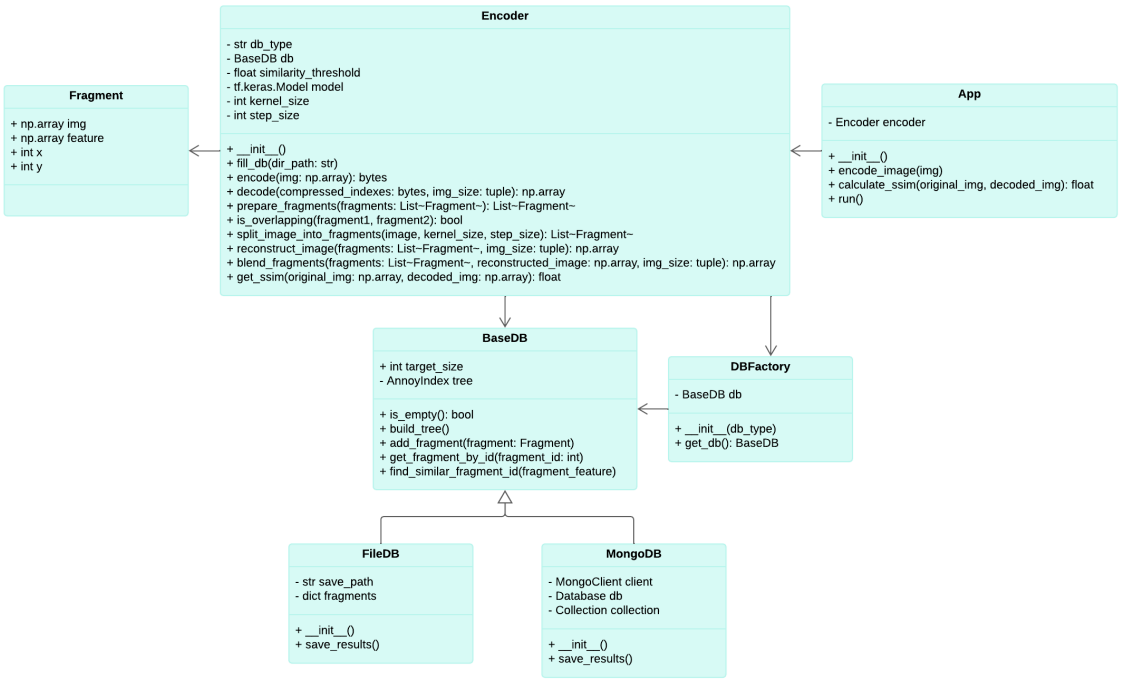


Рисунок 3.4 – Діаграма класів

3.2 Обґрунтування вибору засобів розробки

Для написання дипломного проєкту було використано наступні засоби та технології розробки:

- мова програмування Python;
- редактор коду Visual Studio Code і середовище розробки Google Colab;
- бібліотеки Numpy, OpenCV, Annoy, TensorFlow;
- фреймворк Streamlit;
- база даних Mongo;
- система контролю версій Git.

Нижче розглянуто обрані технології більш детально, описані їх переваги та недоліки, а також обґрунтований їх вибір.

Мова програмування

Python — це високорівнева мова програмування загального призначення, яка підтримує кілька парадигм програмування, включаючи об'єктно-орієнтовану, процедурну та функціональну. Переваги Python включають широкий набір бібліотек для роботи з зображеннями, нейронними

мережами та обробкою даних, велику спільноту розробників та велику кількість ресурсів для навчання, а також простоту та читабельність коду. Недоліком є те, що швидкість виконання може бути нижчою порівняно з компільованими мовами, такими як C++. Python обрано через його зручність у використанні та велику кількість доступних бібліотек для обробки зображень і машинного навчання.

Середовище розробки

Visual Studio Code — це безплатний кросплатформний редактор коду з підтримкою розширень. Його переваги включають широку підтримку розширень для різних мов програмування та інструментів, легку інтеграцію з Git та іншими системами контролю версій, а також вбудовані засоби для налагодження та роботи з терміналом. Недоліком є те, що для ефективної роботи з Python може знадобитися додаткове налаштування. Аналоги включають PyCharm, Sublime Text та Atom.

Google Colab — це безкоштовне хмарне середовище для розробки, яке підтримує Python і дозволяє виконувати код на серверах Google. Переваги Google Colab включають підтримку GPU та TPU для швидкого виконання обчислень, легку інтеграцію з Google Drive для зберігання файлів, а також можливість спільної роботи в режимі реального часу. Недоліком є обмеження на використання ресурсів. Аналоги включають Jupyter Notebook та Kaggle Kernels.

Системи контролю версій

Git — це розподілена система контролю версій, яка дозволяє відстежувати зміни в коді та координувати роботу над проектом між кількома розробниками. Переваги Git включають високу популярність і підтримку в багатьох середовищах розробки, можливість роботи в режимі офлайн та широкий набір інструментів для управління версіями. Аналоги включають Mercurial та Subversion.

База даних

Системи управління базами даних поділяються на дві основні категорії: реляційні (SQL) та нереляційні (NoSQL). Вибір між цими двома типами залежить від специфіки завдань, обсягу даних, потреб у масштабуванні та структури даних.

Реляційні СУБД, такі як MySQL, PostgreSQL та Oracle, використовують строгу табличну структуру з чітко визначеними схемами. Вони ідеально підходять для складних запитів і забезпечують потужні операції з'єднання та транзакції, що забезпечують консистентність і цілісність даних. Проте вони мають обмежену гнучкість схем і їх складно горизонтально масштабувати.

NoSQL бази, такі як MongoDB, Cassandra, Redis, і Neo4j, відходять від традиційної табличної структури і часто пропонують більш гнучкі схеми. Вони підходять для роботи з великими обсягами розподілених даних і часто використовуються в Big Data та веб-застосунках. Перевагами NoSQL баз даних є гнучкі схеми даних, легке горизонтальне масштабування, швидке додавання і зміна нових типів даних. До недоліків можна віднести відсутність стандартизованих запитів подібних до SQL.

Для проекту з кодування растрових зображень на основі подібності фрагментів було обрано NoSQL підхід, зокрема базу даних MongoDB, через кілька ключових переваг. По-перше, гнучкість схеми MongoDB дозволяє зберігати документи в форматі, схожому на JSON, що ідеально підходить для обробки нерегулярних даних і змінних схем, типових для даних зображень. Це уможливлює швидкі зміни у структурі даних без потреби в реорганізації всієї бази даних. По-друге, ефективна підтримка горизонтального масштабування через шардінг дозволяє MongoDB легко розширюватися відповідно до збільшення обсягів даних. По-третє, висока швидкість запису та читання оптимізує обробку великих обсягів даних, що критично для ефективної роботи з зображеннями та їхніми фрагментами. Нарешті, відкритий код та активна спільнота MongoDB забезпечують значну підтримку та ресурси для навчання

та розробки, що сприяє легкості інтеграції та використання цієї технології у проекті.

Вибір MongoDB в порівнянні з іншими NoSQL базами, такими як Cassandra чи Redis, зумовлений її здатністю забезпечувати високу продуктивність при роботі з великими наборами документно-орієнтованих даних, що є ключовим для успішної реалізації проекту.

Бібліотеки та фреймворки

Numpy — це бібліотека для роботи з багатовимірними масивами та матрицями, що надає велику кількість математичних функцій. Переваги Numpy включають високу продуктивність обчислень і широке застосування в наукових дослідженнях та аналізі даних. Аналоги включають MATLAB та SciPy.

OpenCV — це бібліотека з відкритим кодом для комп'ютерного зору та обробки зображень. Переваги OpenCV включають підтримку великої кількості алгоритмів обробки зображень, хорошу продуктивність і підтримку багатьох мов програмування. Аналоги включають PIL (Python Imaging Library) та scikit-image.

Annoy — це бібліотека для побудови і пошуку найближчих сусідів в багатовимірних просторах. Переваги Annoy включають високу швидкість пошуку найближчих сусідів та легкість інтеграції з Python. Однак, бібліотека має обмежену документацію, що може ускладнити використання для новачків або специфічних випадків застосування. Аналоги включають Faiss і Scikit-learn Nearest Neighbors.

TensorFlow — це бібліотека з відкритим кодом для машинного навчання та нейронних мереж, розроблена компанією Google. Вона підтримує широкий спектр моделей машинного навчання і забезпечує високу продуктивність обчислень, особливо при виконанні на GPU, що сприяє значному прискоренню обробки зображень та інших великих датасетів. TensorFlow також вирізняється хорошою документацією та великою спільнотою користувачів, яка сприяє обміну знаннями та досвідом серед розробників.

Основним недоліком є відносно високий поріг входу для новачків, оскільки потребує глибокого розуміння машинного навчання та нейронних мереж. Основними аналогами TensorFlow є PyTorch та Keras.

Streamlit — це фреймворк для швидкого створення веб-додатків на Python, який вирізняється своєю простотою використання та швидкістю розробки. Він забезпечує інтерактивність та можливість візуалізації даних, що робить його ідеальним для демонстрації результатів роботи з даними та моделями машинного навчання. Основним недоліком Streamlit є його обмежена гнучкість порівняно з повноцінними веб-фреймворками, такими як Flask або Django. Проте, для цілей демонстрації результатів обробки зображень та взаємодії з користувачем цей інструмент виявляється оптимальним. Streamlit було обрано за його здатність швидко створювати інтерактивні веб-додатки, які дозволяють користувачам наглядно бачити результати алгоритмів без необхідності глибокого занурення у технічні деталі веб-розробки, що є великою перевагою для швидкої демонстрації та тестування.

3.3 Конструювання програмного забезпечення

3.3.1 Опис алгоритму кодування

У рамках проекту розроблено метод кодування растрових зображень, який базується на пошуку та використанні подібності між фрагментами зображень з використанням оператора зсуву. Нижче наведений більш детальний опис етапів методу.

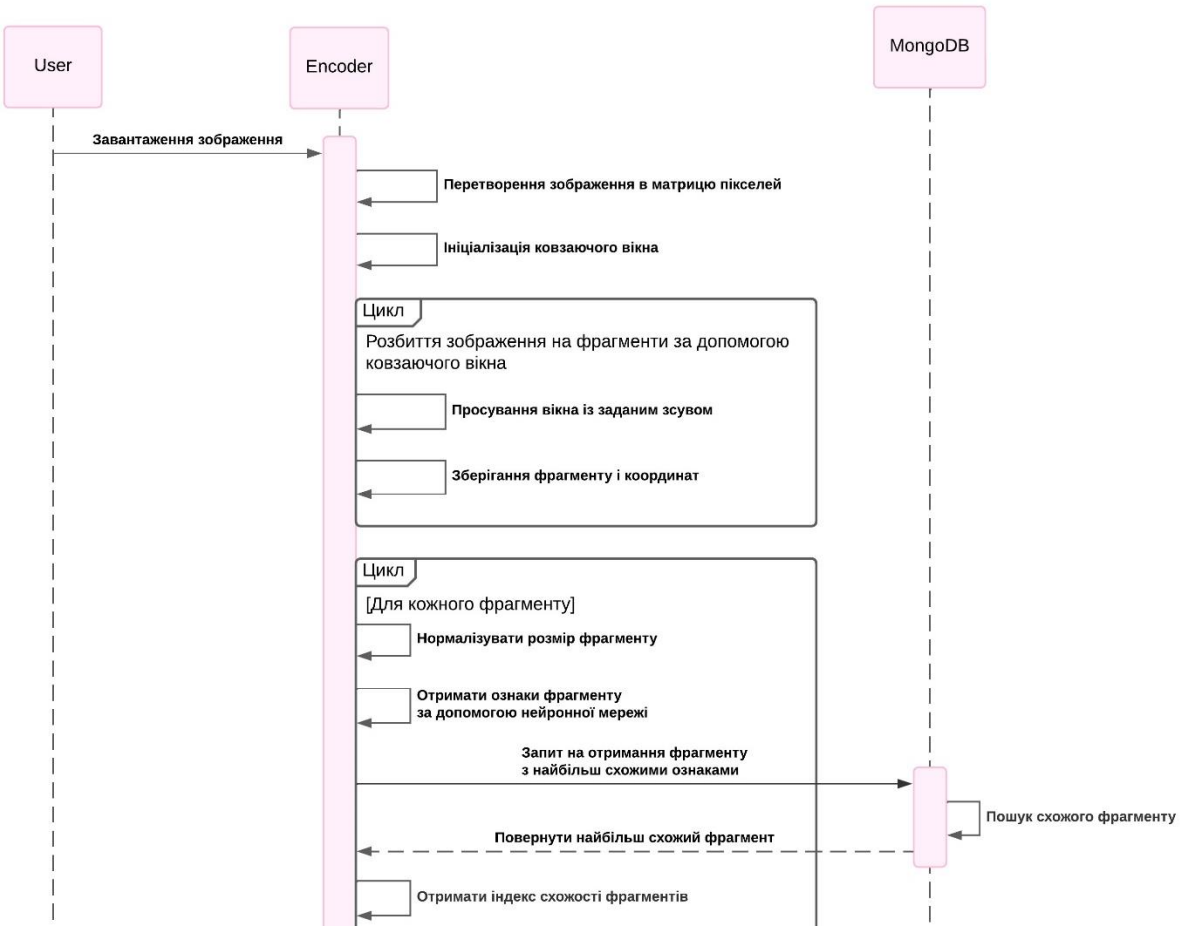


Рисунок 3.5 – Діаграма послідовності. Частина 1

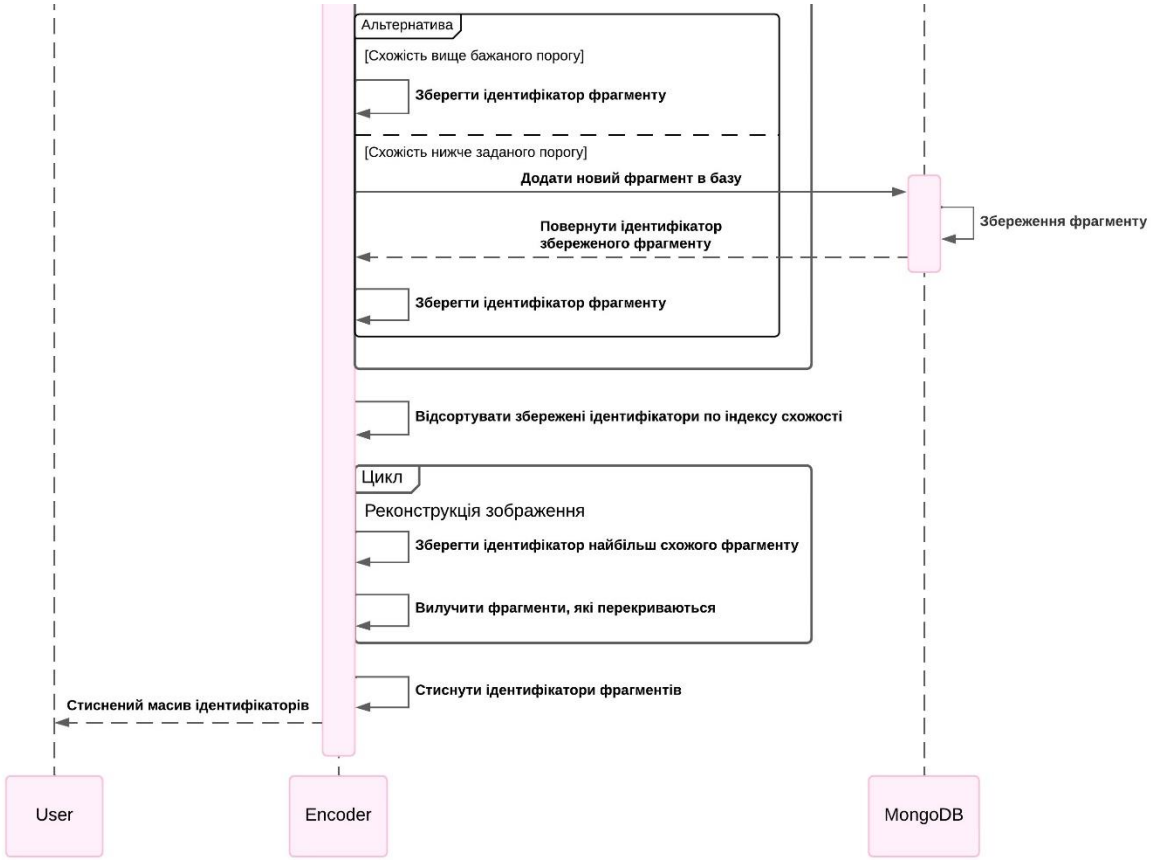


Рисунок 3.6 – Діаграма послідовності. Частина 2

Зчитування зображення

Зображення у форматах JPG, JPEG, PNG зчитується і перетворюється в матрицю пікселів. Це дозволяє далі обробляти зображення на рівні окремих пікселів.

Розбиття на фрагменти

Для розбиття вхідного растрового зображення на фрагменти використовується методика, яка заснована на просуванні вікна фіксованого розміру по всьому зображенню. Розмір цього "вікна" або фрагмента зазвичай встановлюється відповідно до конкретних вимог проекту та може варіюватися для досягнення оптимальної якості та ефективності стиснення.

Вікно заданого розміру (наприклад, 16x16 пікселів) ініціалізується в лівому верхньому куті зображення. Вікно просувається по зображенню зліва направо, ряд за рядом, від верху до низу із заданим зсувом. Після досягнення

кінця ряду, вікно переміщується на наступний ряд і знову просувається зліва направо. Це продовжується до тих пір, поки вікно не пройде весь доступний простір зображення. Кожен фрагмент, вирізаний згідно з розмірами вікна, зберігається як окрема одиниця даних разом з координатами верхнього лівого кута.

Отримання ознак фрагменту

Для ефективного отримання ознак із фрагментів зображень використовується передтренована нейронна мережа, така як ResNet або VGG, адаптована до роботи з малими фрагментами.

Спочатку, кожен фрагмент зображення нормалізується до стандартного розміру, який прийнятний для мережі (наприклад, 224x224 пікселі), та перетворюється в RGB формат, якщо це необхідно. Потім, використовуючи мережу, витягуються ознаки, які характеризують ключові візуальні атрибути фрагментів, такі як текстура, кольоровість, і контури.

Ці ознаки представляються у вигляді векторів і використовуються далі для порівняння фрагментів з даними в базі для визначення ступеня їх схожості. Завдяки цьому підходу можна точно аналізувати та класифікувати частини зображень, значно підвищуючи ефективність процесу кодування зображень.

Перевірка схожості фрагментів та зберігання ідентифікаторів

Після розділення зображення на фрагменти та витягування їх ознак, кожен фрагмент порівнюється з наявними у базі даних фрагментами за допомогою Annoy, що дозволяє швидко знаходити найближчі сусідні фрагменти, що максимально схожі за визначеними ознаками. Коли потенційно схожий фрагмент знайдено, виконується додаткова перевірка рівня схожості між цим фрагментом і поточним фрагментом зображення. Для цього може бути використаний індекс схожості, наприклад, SSIM або MSE.

Якщо індекс схожості перевищує заданий поріг, фрагмент вважається достатньо схожим, і в результуючий масив записується ID цього фрагмента з бази даних. Якщо схожість між фрагментами нижча за порогове значення, новий фрагмент додається до бази даних, і його ID також заноситься в

результуючий масив. Це дозволяє зберегти унікальні властивості зображення, які не можна адекватно представити за допомогою існуючих фрагментів.

Реконструкція зображення

Починаючи з найбільш схожих фрагментів, зображення реконструюється на основі вибраних ID. Під час реконструкції уникнення перекриття фрагментів є критично важливим: якщо запропонований фрагмент перекривається з вже розміщеним, він пропускається, і вибір здійснюється наступний найбільш схожий фрагмент. Це забезпечує цілісність зображення без повторень. Приклад оригінального і реконструйованого зображення наведено на рисунках 3.7 – 3.8.

Рисунок 3.7 – Оригінальне зображення

Рисунок 3.8 – Реконструйоване зображення

Стиснення та збереження результату

Остаточний набір ідентифікаторів фрагментів стискається за допомогою алгоритму LZMA, що призводить до формування компактного набору байтів. Це значно зменшує обсяг даних, необхідних для зберігання або передачі кодованого зображення.

Декодування зображення

Для відновлення зображення з кодованих даних використовуються ідентифікатори, щоб отримати відповідні фрагменти з бази даних. Фрагменти складаються разом, формуючи вихідне зображення. Для покращення якості відновленого зображення можуть використовуватися додаткові методи оптимізації, наприклад, застосування фільтрів для згладжування або покращення контрастності на стадії декодування.

Для процесу кодування зображень на рисунках 3.5-3.6 наведено діаграму послідовностей, яка показує весь процес кодування від вхідного зображення

до отримання масиву ідентифікаторів фрагментів. Повний вид діаграми винесено у графічний матеріал дипломного проєкту.

3.3.2 Вхідні та вихідні дані

Вхідними даними для процесу кодування є растрові зображення у форматі JPG, JPEG, PNG. Вихідними даними є набір байтів. Цей набір байтів представляє собою набір ідентифікаторів фрагментів стиснений методом LZMA, які використовуються для реконструкції зображення.

Для зберігання фрагментів зображень використовується база даних MongoDB. Концептуальна модель бази даних включає основну сутність "Фрагмент", яка описується атрибутами: ідентифікатор, зображення та вектор ознак. Логічна модель бази даних організована таким чином, що кожен документ відповідає одному фрагменту зі своїми атрибутами.

Структуру документу в колекції даних описано в таблиці 3.7.

Таблиця 3.7 – Структура документу в колекції бази даних

Назва поля	**Опис**
id	Унікальний ідентифікатор документа
image	Зображення фрагменту
feature	Ознаки фрагменту

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Оскільки алгоритм кодування растрових зображень працює локально, жодні дані користувачів не збираються, що забезпечує повну безпеку їхньої інформації. Відкритий вихідний код дозволяє будь-кому перевірити його роботу та внести зміни за необхідності. Це також дає змогу розробникам

донавчати модель на власних наборах даних або модифікувати код без ризику того, що їхні зміни будуть доступні конкурентам.

Проте, існує ризик, пов'язаний з використанням заповненої бази даних фрагментів. Якщо база даних містить фрагменти, що включають особисту інформацію або захищені авторськими правами матеріали, це може становити загрозу безпеці даних. У моєму проекті використані зображення з відкритого датасету на платформі Kaggle, що не містять особистої інформації чи даних, захищених авторськими правами, тому ризик компрометації мінімізований.

Висновки до розділу

У даному розділі було систематизовано та детально описано архітектуру розробленого програмного забезпечення, що включає комплексний опис класів та їх взаємодії, представлений через діаграму класів. Це дало змогу чітко визначити структуру програми та її компоненти, що сприяє кращому розумінню та подальшому розвитку проекту.

Було проведено обґрунтування вибору засобів розробки, зокрема мови програмування Python, середовищ розробки як Visual Studio Code та Google Colab, бази даних MongoDB, а також ключових бібліотек і фреймворків, таких як TensorFlow, NumPy, OpenCV та Streamlit. Вибір цих технологій базувався на їх здатності забезпечити необхідний рівень функціональності, продуктивності, а також підтримку спільноти, що є критично важливим для стабільності та масштабування проекту.

У секції конструювання програмного забезпечення було детально розглянуто алгоритм кодування зображень, включно з методикою розбиття зображень на фрагменти, витягуванням та аналізом ознак, порівнянням схожості, і, нарешті, стисненням даних. Використання бази даних для зберігання фрагментів із зазначенням їх ідентифікаторів дозволяє ефективно відновлювати зображення з високим ступенем точності. Діаграма послідовностей, що була створена для візуалізації цього процесу, демонструє

логіку взаємодій між основними компонентами системи, сприяючи кращому розумінню процесів і потоків даних.

Загалом, описані у цьому розділі аспекти лягають в основу побудови надійної та ефективної системи для кодування растрових зображень, що враховує сучасні потреби в обробці великих даних та машинному навчанні.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

У цьому розділі виконується аналіз якості програмного забезпечення, розробленого для кодування растрових зображень на основі подібності фрагментів з оператором зсуву. Метою тестування є забезпечення високої надійності, продуктивності та коректності роботи ПЗ відповідно до встановлених вимог.

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних в базі;
- перевірка сумісності веб-додатку з останніми версіями сучасних браузерів;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

Метриками для оцінки якості ПЗ обрано наступні:

- час кодування зображення;
- відсоток стиснення зображення;
- індекс схожості декодованого і оригінального зображень;
- сумісність програми з різними браузерами та операційними системами;
- стабільність роботи.

Цей комплексний підхід до аналізу якості дозволяє не тільки оцінити поточний стан програмного забезпечення, але й визначити ключові напрямки для його вдосконалення і оптимізації в майбутньому.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.7.

Таблиця 4.1 – Тест 1

Мета тесту	Перевірка завантаження зображення в систему
Початковий стан системи	Відкритий веб-інтерфейс для завантаження файлів
Вхідні дані	Растрове зображення формату JPG, JPEG, PNG
Опис проведення тесту	Натиснути кнопку “Обрати зображення” і завантажити файл
Очікуваний результат	Зображення коректно відображається в інтерфейсі.
Фактичний результат	Зображення коректно відображається в інтерфейсі.

Таблиця 4.2 – Тест 2

Мета тесту	Перевірка кодування зображення
Початковий стан системи	Зображення завантажено і коректно відображається в інтерфейсі
Вхідні дані	Растрове зображення формату JPG, JPEG, PNG
Опис проведення тесту	Натиснути кнопку “Стиснути зображення”
Очікуваний результат	Відображається кнопка для завантаження стисненого зображення, ступінь стиснення, індекс схожості декодованого і оригінального зображення
Фактичний результат	Відображається кнопка для завантаження стисненого зображення, ступінь стиснення, індекс схожості декодованого і оригінального зображення

Таблиця 4.3 – Тест 3

Мета тесту	Перевірка збереження фрагментів у базі
Початковий стан системи	Зображення завантажено і коректно відображається в інтерфейсі
Вхідні дані	Растрове зображення формату JPG, JPEG, PNG
Опис проведення тесту	Натиснути кнопку “Стиснути зображення”
Очікуваний результат	Фрагменти зображення збережені в базі даних
Фактичний результат	Фрагменти зображення збережені в базі даних

Таблиця 4.4 – Тест 4

Мета тесту	Перевірка функції збереження закодованого зображення
Початковий стан системи	Закодоване зображення готове до збереження
Вхідні дані	-
Опис проведення тесту	Натиснути кнопку “Завантажити стиснене зображення”
Очікуваний результат	Файл успішно збережений на диску
Фактичний результат	Файл успішно збережений на диску

Таблиця 4.5 – Тест 5

Мета тесту	Перевірка функції відновлення зображення з закодованих даних
Початковий стан системи	Закодоване зображення зберігається локально у форматі бінарного файлу
Вхідні дані	Бінарний файл
Опис проведення тесту	Натиснути кнопку “Відновити зображення” і обрати файл

Продовження таблиці 4.5

Очікуваний результат	Зображення відновлене та візуально ідентичне оригіналу
Фактичний результат	Зображення відновлене та візуально ідентичне оригіналу

Таблиця 4.6 – Тест 6

Мета тесту	Перевірка роботи веб-інтерфейсу в різних браузерах
Початковий стан системи	Веб-інтерфейс доступний для тестування
Вхідні дані	Стандартні операції в Chrome, Firefox, Opera
Опис проведення тесту	Виконання основних функцій інтерфейсу в кожному браузері
Очікуваний результат	Сумісність з усіма випробуваними браузерами
Фактичний результат	Сумісність з усіма випробуваними браузерами

Таблиця 4.7 – Тест 7

Мета тесту	Тест декодування зображення та вимірювання часу на цю операцію
Початковий стан системи	Закодоване зображення готове до декодування
Вхідні дані	Закодоване зображення
Опис проведення тесту	Вимірювання часу, необхідного для декодування зображення
Очікуваний результат	Час декодування зображення з 1000 фрагментів не більше 10 секунд
Фактичний результат	Час декодування зображення з 1000 фрагментів менше 10 секунд

4.3 Опис контрольного прикладу

Для демонстрації веб-інтерфейсу кодування та декодування растрових зображень, ми розглянемо повний цикл обробки зображення, включаючи вибір режиму роботи, завантаження зображення, його кодування, декодування та оцінку результатів.

Користувач запускає веб-додаток. Користувач може обрати один із двох режимів: "Кодування" або "Декодування". Це перший крок у взаємодії з веб-додатком.

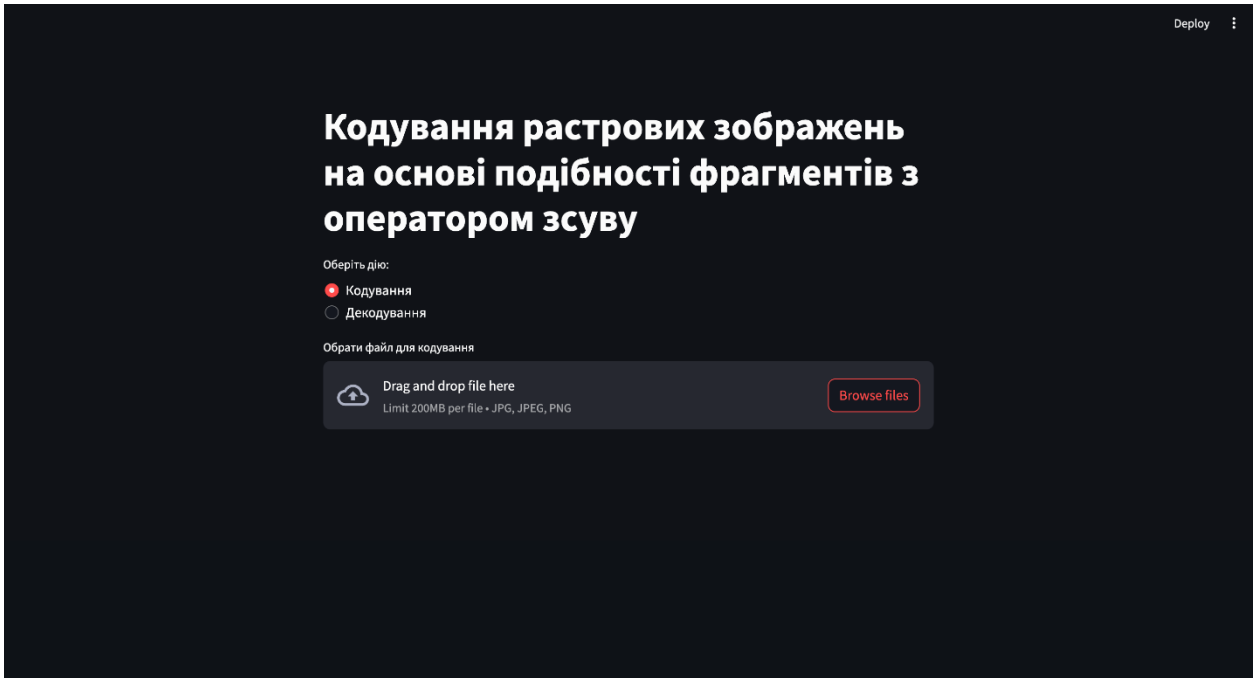


Рисунок 4.8 – Вигляд веб-додатку після запуску

У режимі "Кодування", користувач завантажує зображення та ініціює процес кодування. Користувач обирає "Кодування". Завантажується зображення через віджет файлового завантажувача. Зображення відображається на екрані перед кодуванням.

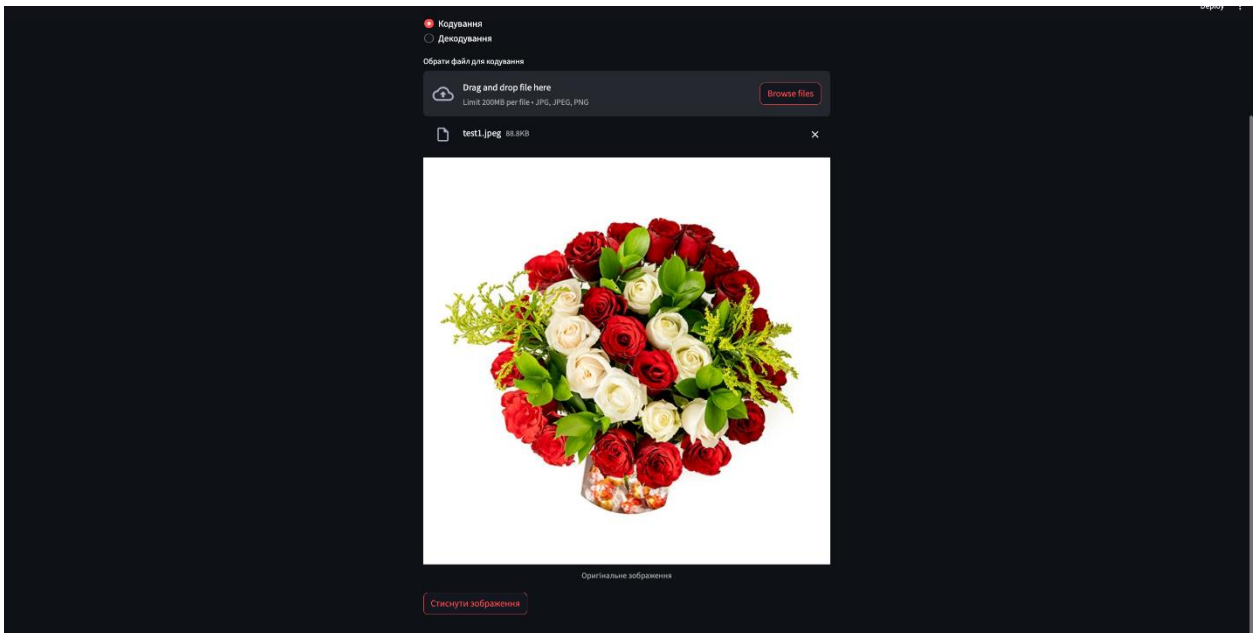


Рисунок 4.9 – Вигляд веб-додатку після завантаження файлу

Після натискання кнопки "Стиснути зображення", воно кодується, і результати кодування відображаються у вигляді відновленого зображення. Користувач може оцінити якість відновлення за допомогою індексу схожості структур (SSIM) та відсотка стиснення. Окрім цього, можливо завантажити стиснуте зображення натиснувши кнопку “Завантажити”.

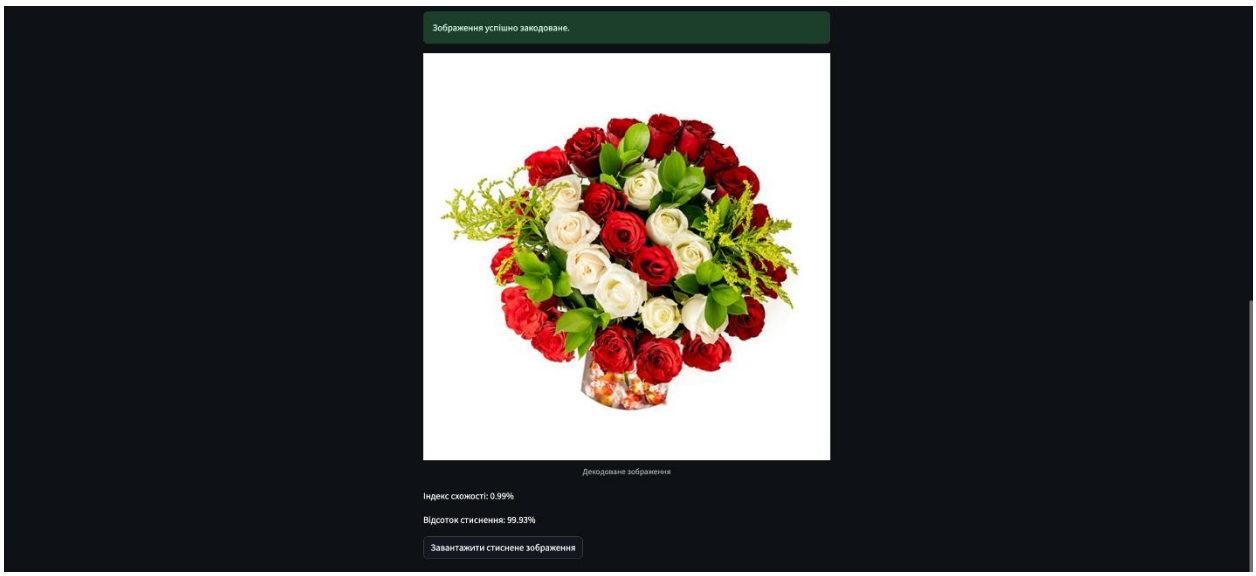


Рисунок 4.10 – Вигляд веб-додатку після кодування зображення

У режимі "Декодування", користувач може завантажити раніше закодовані дані і відновити оригінальне зображення. Користувач обирає "Декодування". Завантажується файл з закодованими даними. Користувач вводить розміри оригінального зображення. Після натискання на кнопку "Відновити зображення", здійснюється декодування і зображення відображається. Якщо користувач бажає, він може зберегти відновлене зображення у форматі BMP.

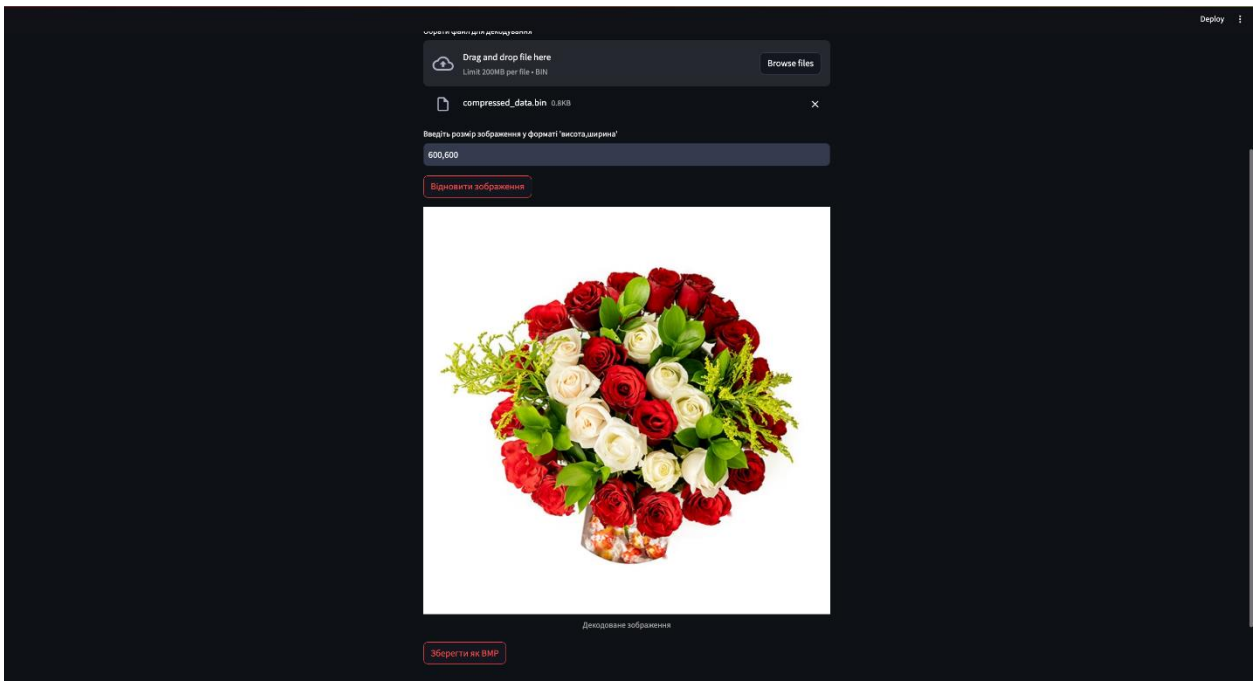


Рисунок 4.11 – Вигляд веб-додатку після декодування зображення

Цей контрольний приклад включає всі можливі розгалуження веб-інтерфейсу, з якими може взаємодіяти користувач. Він забезпечує зрозуміле і повне представлення процесу кодування та декодування растрових зображень у додатку.

Для порівняння продуктивності кодування зображень на CPU та GPU спробуємо закодувати зображення без наповненої бази даних. Для цього використаємо одну і ту ж вихідну картинку і виміряємо час кодування як на CPU, так і на GPU. Процес кодування включав розбиття зображення на 550 фрагментів розміром 25 на 25 пікселей та обробку кожного фрагменту для отримання його ознак. Зазначимо, що цей процес є достатньо ресурсоємним і потребує значного обсягу обчислювальних ресурсів.

Час кодування на CPU склав приблизно 112 секунд. Завдяки значно більшій обчислювальній потужності GPU порівняно з CPU, час кодування суттєво зменшився і склав приблизно 18 секунд. Отримані результати демонструють, що використання GPU для кодування зображень є значно ефективнішим, ніж використання CPU. З огляду на це, всі подальші експерименти було вирішено проводити на GPU.

На наступному етапі експерименту було заповнено базу даних фрагментами зображень. Для цього було використано набір зображень, які були розбиті на фрагменти і збережені у базу даних. Кожен фрагмент був оброблений для отримання його ознак, які зберігалися разом з самим зображенням фрагменту. Всього було збережено 160 000 фрагментів розміром 10 на 10 пікселей.

Після заповнення бази даних було виконано кодування вихідного зображення, використовуючи наповнену базу даних. Процес кодування включав пошук схожих фрагментів у базі даних та заміну фрагментів вихідного зображення на знайдені схожі фрагменти. Цей підхід дозволив суттєво зменшити розмір закодованого зображення.

Для оцінки ефективності кодування було обчислено ступінь стиснення та індекс структурної подібності (SSIM) між вихідним та декодованим зображенням. За результатами експерименту отримано ступінь стиснення 98%, що означає, що розмір закодованого зображення був на 98% менший за розмір вихідного зображення. Індекс SSIM становив 0.995, що вказує на високу якість відновленого зображення.

Приклад оригінального і відновленого зображення наведено на рисунках 4.12 - 4.13.

Рисунок 4.12 – Відновлене зображення

Рисунок 4.13 – Оригінальне зображення

Висновки до розділу

У даному розділі було проведено детальний аналіз процесів та методів, які забезпечують високу якість розробленого програмного забезпечення для кодування растрових зображень. Основна увага була приділена розробці стратегії тестування, вибору ключових метрик для оцінки продуктивності та функціональності додатку, а також практичному виконанню тестів для верифікації роботи системи.

Було виконане мануальне тестування програмного забезпечення, щоб переконатися, що система працює коректно та відповідає зазначеним вимогам. Для оцінки ефективності програмного забезпечення були обрані метрики, такі як швидкість обробки зображень, індекс схожості оригінального і відновленого зображення, рівень стиснення даних, а також загальна стабільність роботи системи. Такий підхід дозволив глибше зрозуміти ефективність розробленого рішення. Аналіз результатів тестування показав, що розроблене програмне забезпечення відповідає поставленим вимогам. Таким чином, можна стверджувати, що розроблене програмне забезпечення має високу якість і готове до розгортання та експлуатації.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Розгортання включає налаштування серверного середовища, встановлення необхідних залежностей, налаштування бази даних та запуск веб-додатку.

Першочергово, процес розгортання програмного забезпечення включає налаштування серверного середовища та встановлення необхідних залежностей. Після вибору сервера та операційної системи, необхідно встановити Python та менеджер пакетів Python PIP. Це дозволяє встановлювати додаткові бібліотеки, такі як Numpy, OpenCV, TensorFlow для обробки зображень і машинного навчання.

Окрім програмних залежностей, важливим аспектом є налаштування і встановлення бази даних. Вона використовується для зберігання даних і також потребує налаштування. MongoDB, популярний вибір для додатків, що працюють з великими обсягами даних, встановлюється та конфігурується для прийому з'єднань від програми. Після встановлення та конфігурації, програмне забезпечення запускається через термінал, і його доступність перевіряється за допомогою браузера.

Запуск програмного забезпечення виконується через термінал за допомогою команд Python. Перед запуском необхідно переконатися, що всі компоненти системи правильно взаємодіють між собою, і що база даних правильно налаштована для прийому з'єднань від програми.

Для забезпечення більшої універсальності та легкості в розгортанні, можна використовувати Docker, що дозволяє контейнеризувати програмне забезпечення. Docker обгортає програму в контейнер, ізольований від хост-системи, що значно спрощує розгортання на різних платформах і зменшує "проблеми із сумісністю". Цей підхід гарантує, що програмне забезпечення може бути легко розгорнуте будь-де, де є Docker, без необхідності додаткового налаштування середовища.

Останнім кроком є тестування розгорнутої системи на предмет її функціональності та продуктивності. Це включає перевірку коректності кодування та декодування зображень, а також перевірку загальної стабільності та продуктивності програми на сервері. Після тестування та верифікації система готова до використання кінцевими користувачами.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі.

Підтримка програмного забезпечення є ключовим аспектом, що забезпечує його тривалу та ефективну експлуатацію. Користувачі програми мають можливість регулярно отримувати оновлення, які включають в себе виправлення помилок, покращення функціональності та оновлення безпеки. Кожна нова версія програмного забезпечення публікується у відкритому репозиторії GitHub.

Підтримка користувачів забезпечується через систему керування запитами на підтримку, де користувачі можуть надсилати звіти про помилки, запити на нові функції, а також отримувати консультації. Регулярне оновлення документації допомагає користувачам ефективно використовувати програмне забезпечення.

Завдяки впровадженню систематичного підходу до супроводу програмного забезпечення, забезпечується його надійність, безпека та актуальність, що в кінцевому підсумку підвищує загальне задоволення користувачів і довіру до продукту.

Висновки до розділу

В даному розділі надається вичерпна інформація про процес розгортання програмного забезпечення, починаючи з налаштування серверного середовища до випуску і підтримки продукту. Первинне розгортання програми передбачає налаштування сервера, що включає встановлення операційної системи, конфігурацію мережі, і встановлення всіх

необхідних залежностей за допомогою менеджера пакетів Python PIP. Це забезпечує підтримку різних бібліотек, необхідних для обробки зображень і виконання машинного навчання, таких як Numpy, OpenCV, і TensorFlow.

Ключовим елементом розгортання є налаштування бази даних MongoDB, яка використовується для збереження даних, важлива для роботи з великими обсягами інформації. Після налаштування бази даних програмне забезпечення запускається через термінал, а його функціональність перевіряється за допомогою браузера, що забезпечує легкість перевірки доступності та правильності роботи програми.

Окрім традиційних методів, використання Docker забезпечує додаткові можливості по спрощенню розгортання, ізоляції додатку від середовища, і легкості переносимості на різні платформи. Контейнеризація дозволяє стандартизувати робоче середовище і відокремити від специфіки інфраструктури, що важливо для реплікації середовища в різних умовах. Розгортання з Docker знижує ймовірність помилок, пов'язаних з різницею середовищ, і дозволяє використовувати автоматизовані інструменти для випуску та оновлення програми.

Фінальним кроком процесу є тестування і верифікація розгорнутого рішення, що дозволяє переконатися у його працездатності, стабільності та ефективності. Підтримка програми забезпечується через систематичне оновлення, обробку запитів на підтримку від користувачів і регулярне оновлення документації. Такий підхід до супроводу програми гарантує її надійне та довгострокове використання, підвищує задоволення користувачів і підтримує довіру до продукту.

ВИСНОВКИ

В результаті виконання дипломного проєкту було спроєктовано та розроблено програмне забезпечення для кодування растрових зображень на основі подібності фрагментів з використанням оператора зсуву. Це програмне забезпечення дозволяє значно зменшити обсяг даних, необхідний для зберігання та передачі зображень, що є актуальною проблемою в сучасних інформаційних системах.

У якості середовища розробки було обрано Python завдяки його простоті, великій кількості доступних бібліотек для роботи з зображеннями та машинного навчання, а також широкій спільноті розробників, що забезпечує швидкий пошук рішень для виникаючих проблем. В якості бази даних використано MongoDB, яка забезпечує гнучке зберігання даних і високу продуктивність при роботі з великими обсягами інформації. Це дозволило ефективно реалізувати зберігання та пошук подібних фрагментів зображень.

Наукова новизна роботи полягає у модифікації методу кодування растрових зображень шляхом використанням оператора зсуву, що дозволяє знаходити більше подібних фрагментів, враховуючи можливі зміщення об'єктів на зображенні. Це дозволило досягти більш високої якості стиснення, зберігаючи при цьому інформаційну цінність зображень.

Проведений аналіз якості та тестування програмного забезпечення включав перевірку правильності роботи програмного забезпечення, збереження даних, сумісності з різними операційними системами та браузерами, а також зручність графічного інтерфейсу. Всі ці тести були успішно пройдені, що свідчить про високу якість розробленого програмного забезпечення.

Науково-технічна значущість роботи полягає у впровадженні нових методів кодування та стиснення зображень, що дозволяють зменшити витрати на зберігання та передачу даних, підвищити швидкість обробки інформації та знизити вимоги до пропускної здатності мереж. Це має безпосередній вплив

на ефективність роботи інформаційних систем та знижує експлуатаційні витрати.

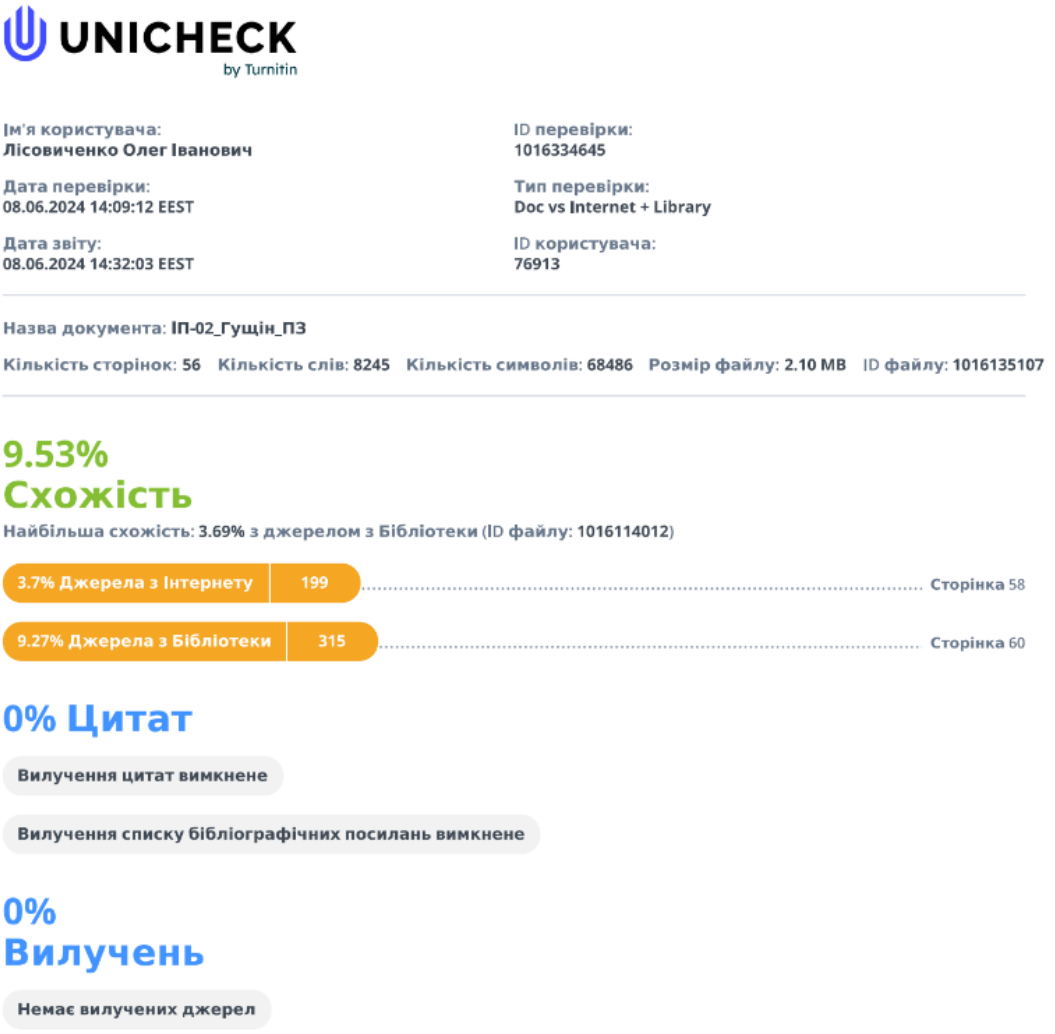
Подальші дослідження можуть включати покращення алгоритмів кодування та декодування, врахування додаткових параметрів зображень, а також оптимізацію програмного забезпечення для роботи з ще більшими обсягами даних. Це дозволить зробити процес стиску ще більш ефективним та адаптованим до нових вимог та умов експлуатації.

В результаті виконання дипломного проекту всі поставлені задачі були успішно вирішені. Було проведено передпроєктне обстеження предметної області, розроблено вимоги до програмного забезпечення, сконструйовано та розроблено програмне забезпечення, проведено аналіз його якості та тестування, а також описано процеси розгортання та супроводу програмного забезпечення. Цей комплексний підхід забезпечив створення програмного продукту, готового до практичного використання та подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Huffman Encoding and Data Compression [Електронний ресурс] / J. Zelenski, K. Schwarz, M. Stepp – Режим доступу до ресурсу: https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1176/assnFiles/assign6/huffman-encoding-supplement.pdf.
2) Encoding Techniques for Image Compression: A Survey [Електронний ресурс] / A. Nitesh, V. Ashutosh – Режим доступу до ресурсу: https://www.researchgate.net/profile/Nitesh-Agarwal-3/publication/344786311_Encoding_Techniques_for_Image_Compression_A_Survey/links/5f904d1b299bf1b53e37c20f/Encoding-Techniques-for-Image-Compression-A-Survey.pdf.
3) Compression Algorithms: Huffman and Lempel-Ziv-Welch [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://web.mit.edu/6.02/www/s2012/handouts/3.pdf.
4) The Discrete Cosine Transform [Електронний ресурс] / Strang Gilbert – Режим доступу до ресурсу: https://math.mit.edu/~gs/papers/dct.pdf.
5) Transform Techniques for Image Compression [Електронний ресурс] / S. S., K. Rajesh, V. R. K. // MECS. – 2014. – Режим доступу до ресурсу: https://www.mecs-press.org/ijigsp/ijigsp-v6-n2/IJIGSP-V6-N2-7.pdf.
6) Портяний І. С., Поспєлова К. І., Олійник Ю. О. (2021). КОДУВАННЯ РАСТРОВИХ ЗОБРАЖЕНЬ НА ОСНОВІ ПОДІБНОСТІ ФРАГМЕНТІВ. Вісник Хмельницького національного університету, (6), 73-80.

ДОДАТОК А. ЗВІТ ПОДІБНОСТІ



UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016334645

Дата перевірки:
08.06.2024 14:09:12 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 14:32:03 EEST

ID користувача:
76913

Назва документа: ІП-02_Гущін_ПЗ

Кількість сторінок: 56 Кількість слів: 8245 Кількість символів: 68486 Розмір файлу: 2.10 MB ID файлу: 1016135107

9.53%
Схожість

Найбільша схожість: 3.69% з джерелом з Бібліотеки (ID файлу: 1016114012)

3.7% Джерела з Інтернету 199 ..... Сторінка 58

9.27% Джерела з Бібліотеки 315 ..... Сторінка 60

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

Текст програми

КПІ.ІП-0215.045480.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Дмитро ГУЩІН

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/DHushchin/fragment-image-encoding

Файл fragment.py

```
class Fragment:
    def __init__(self, img = None, x = None, y = None, feature = None):
        self.img = img
        self.feature = feature
        self.x = x
        self.y = y
```

Файл encoder.py

```
import cv2
import os
import lzma
from typing import List

import numpy as np
import tensorflow as tf
from numpy.lib.stride_tricks import as_strided
from skimage.metrics import structural_similarity as ssim_metric
from scipy.spatial import KDTree
import streamlit as st

from db.factory import DBFactory
from fragment import Fragment
from time import time


if tf.test.gpu_device_name():
    print("Default GPU Device: {}".format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")


class Encoder:
    def __init__(self):
        self.db_type = 'mongo'
        self.db = DBFactory(self.db_type).get_db()
        self.similarity_threshold = 0.95
        self.model = tf.keras.applications.ResNet50(include_top=False,
weights='imagenet', input_shape=(224, 224, 3))
        self.kernel_size = 32
        self.step_size = 16


    def fill_db(self, dir_path: str):
        for file_path in os.listdir(dir_path):
            if not file_path.endswith('.png') and not
file_path.endswith('.jpeg') and not file_path.endswith('.jpg'):
                continue

            print(f'Processing image {file_path}')
            img = cv2.imread(os.path.join(dir_path, file_path))
```

```
            fragments = self.split_image_into_fragments(img,
self.kernel_size, self.step_size)

            print(f'Fragments count: {len(fragments)}')

            prep_fragments = self.prepare_fragments(fragments)

            start_time = time()

            for fragment in prep_fragments:
                self.db.add_fragment(fragment)

            print(f'Fragments adding time: {time() - start_time}')

        if self.db_type == 'file':
            self.db.save_results()


    def encode(self, img: np.array) -> bytes:
        start_time = time()
        fragments = self.split_image_into_fragments(img, self.kernel_size,
self.step_size)
        prep_fragments = self.prepare_fragments(fragments)
        print(f'Fragments count: {len(fragments)}')

        encoded_data = [] # List of tuples (fragment_id, x, y)
        similarity_data = [] # List of tuples (fragment, fragment_id,
similarity)

        for fragment in prep_fragments:
            if self.db.is_empty():
                new_fragment_id = self.db.add_fragment(fragment)
                similarity_data.append((fragment, new_fragment_id, 1))
                continue

            similar_fragment_id =
self.db.find_similar_fragment_id(fragment.feature)
            similar_fragment_img =
self.db.get_fragment_by_id(similar_fragment_id)['image']
            similarity = self.get_ssim(fragment.img, similar_fragment_img)

            if similarity > self.similarity_threshold:
                similarity_data.append((fragment, similar_fragment_id,
similarity))
            else:
                new_fragment_id = self.db.add_fragment(fragment)
                similarity_data.append((fragment, new_fragment_id, 1))

        if self.db_type == 'file':
            self.db.save_results()

        # Filter and sort similarity data
        similarity_data.sort(key=lambda x: x[2], reverse=True)

        # Create a mask to track occupied areas
        mask = np.zeros(img.shape[:2], dtype=bool)

        # Add fragments to encoded_data
        for fragment, fragment_id, _ in similarity_data:
            x, y = fragment.x, fragment.y
            h, w = self.kernel_size, self.kernel_size

            # Clip fragment coordinates to fit image boundaries
```

```
            x_min = max(0, x)
            y_min = max(0, y)
            x_max = min(img.shape[1], x + w)
            y_max = min(img.shape[0], y + h)

            # Check if the fragment overlaps with the occupied area
            if np.any(mask[y_min:y_max, x_min:x_max]):
                continue

            # Update the mask
            mask[y_min:y_max, x_min:x_max] = True

            # Add the fragment to the encoded data
            encoded_data.extend([fragment_id, x, y])

        # Convert encoded_data to bytes and return
        encoded_bytes = np.array(encoded_data, dtype=np.uint64).tobytes()
        compressed_data = lzma.compress(encoded_bytes)
        print(f'Encoded data size: {len(compressed_data)}')
        print(f'Encoding time: {time() - start_time}')

        return compressed_data


    def decode(self, compressed_indexes: bytes, img_size: tuple) -> np.array:
        decoded_data = lzma.decompress(compressed_indexes)
        decoded_array = np.frombuffer(decoded_data, dtype=np.uint64)
        decoded_info = [tuple(decoded_array[i:i+3]) for i in range(0,
len(decoded_array), 3)]
        decoded_fragments = []

        for fragment in decoded_info:
            fragment_img = self.db.get_fragment_by_id(fragment[0])['image']
            fragment_img = cv2.resize(fragment_img, (self.kernel_size,
self.kernel_size))
            decoded_fragments.append(Fragment(img=fragment_img,
x=fragment[1], y=fragment[2]))

        reconstructed_image = self.reconstruct_image(decoded_fragments,
img_size)
        return reconstructed_image


    @tf.function
    def resize_and_predict(self, images):
        with tf.device('/GPU:0'):
            resized_images = tf.image.resize(images, [224, 224])
            return self.model(resized_images)

    def prepare_fragments(self, fragments):
        start_time = time()
        batch_size = 64

        # Transform fragments into a tensor
        fragment_images = np.array([fragment.img for fragment in fragments])
        fragment_images = tf.convert_to_tensor(fragment_images,
dtype=tf.float32)

        # Prepare fragments in batches
        num_batches = (len(fragments) + batch_size - 1) // batch_size
        prep_fragments = []
        for i in range(num_batches):
            batch_images = fragment_images[i*batch_size:(i+1)*batch_size]
            features = self.resize_and_predict(batch_images)
```

```
            features = features.numpy().reshape(features.shape[0], -1)

            # Добавляем обработанные фрагменты
            for j, feature in enumerate(features):
                fragment_index = i*batch_size + j
                if fragment_index < len(fragments):
                    fragment = fragments[fragment_index]
                    prep_fragments.append(Fragment(img=fragment.img,
feature=feature, x=fragment.x, y=fragment.y))

        print(f'Fragments preparation time: {time() - start_time}')
        return prep_fragments


    def is_overlapping(self, fragment1, fragment2):
        """
        Checks if two fragments overlap.

        Args:
            fragment1 (tuple): First fragment in the form (fragment, x, y).
            fragment2 (tuple): Second fragment in the form (fragment, x, y).

        Returns:
            bool: True if the fragments overlap, False otherwise.
        """
        x1, y1, w1, h1 = fragment1[1], fragment1[2], self.kernel_size,
self.kernel_size
        x2, y2, w2, h2 = fragment2[1], fragment2[2], self.kernel_size,
self.kernel_size

        if x1 >= x2 + w2 or x2 >= x1 + w1 or y1 >= y2 + h2 or y2 >= y1 + h1:
            return False

        return True


    @staticmethod
    def split_image_into_fragments(image, kernel_size, step_size) ->
List[Fragment]:
        """
        Splits an image into fragments.

        Args:
            image (numpy.array): The input image.
            kernel_size (int): Size of the square fragment.
            step_size (int): Step size for moving the window.

        Returns:
            list: A list of tuples (fragment, x, y), where fragment is a
fragment of the image,
                (x, y) is the coordinate of the top-left corner of the
fragment.
        """
        fragments = []
        height, width = image.shape[:2]

        # Check for valid parameters
        if kernel_size < 2 or kernel_size > min(height, width):
            raise ValueError("Invalid kernel size")
        if step_size < 1 or step_size > kernel_size:
            raise ValueError("Invalid step size")

        # Create a view of the image with the desired fragment shape and
strides
```

```
        fragment_shape = (kernel_size, kernel_size, 3)
        strides = (image.strides[0]*step_size, image.strides[1]*step_size,
image.strides[0], image.strides[1], image.strides[2])
        fragments_view = as_strided(image, shape=((height-
kernel_size)//step_size+1, (width-kernel_size)//step_size+1,
*fragment_shape), strides=strides)

        # Create a list of fragments
        for y in range(fragments_view.shape[0]):
            for x in range(fragments_view.shape[1]):
                fragment = fragments_view[y, x]
                fragments.append(Fragment(img=fragment, x=x*step_size,
y=y*step_size))

        return fragments


    def reconstruct_image(self, fragments: List[Fragment], img_size: tuple) -
> np.array:
        """
        Reconstructs the image from non-overlapping fragments.

        Args:
            fragments (list): List of tuples (fragment, x, y) where fragment
is a fragment of the image,
                              and (x, y) is the coordinate of the top-left
corner of the fragment.
            img_size (tuple): Size of the original image (height, width).

        Returns:
            numpy.array: Reconstructed image.
        """
        # Create an empty image
        reconstructed_image = np.zeros((*img_size, 3), dtype=np.uint8)

        # Create a mask to mark occupied pixels
        mask = np.zeros(img_size, dtype=bool)
        for fragment in fragments:
            # Convert coordinates to integers
            x = int(fragment.x)
            y = int(fragment.y)
            h, w, _ = fragment.img.shape
            # Assign the fragment to the corresponding region in the
reconstructed image
            reconstructed_image[y:y+h, x:x+w] = fragment.img
            mask[y:y+h, x:x+w] = True

        # Get the coordinates of non-black pixels
        fragment_indices = np.argwhere(mask)

        # Get the coordinates of empty pixels outside the filled fragments
        empty_indices = np.argwhere(~mask)

        # Create a KDTree from non-black pixel indices
        non_black_tree = KDTree(fragment_indices)

        # Define the number of nearest neighbors to consider
        num_neighbors = 16

        # For each empty pixel, find the nearest non-empty pixels and assign
their color
        for y, x in empty_indices:
            # Find the nearest non-empty pixels using KDTree
```

```
            dist, nearest_indices = non_black_tree.query([(y, x)],
k=num_neighbors)  # Find the nearest non-black pixels
            nearest_yx = fragment_indices[nearest_indices[0]]  # Get the
coordinates of the nearest non-black pixels
            # Calculate the mean color of the nearest non-black pixels
            mean_color = np.mean(reconstructed_image[nearest_yx[:, 0],
nearest_yx[:, 1]], axis=0)
            # Assign the mean color to the empty pixels
            reconstructed_image[y, x] = mean_color

        #reconstructed_image = self.blend_fragments(fragments,
reconstructed_image, img_size)

        return reconstructed_image


    def blend_fragments(self, fragments: List[Fragment], reconstructed_image:
np.array, img_size: tuple) -> np.array:
        for fragment in fragments:
            # Convert coordinates to integers
            x = int(fragment.x)
            y = int(fragment.y)
            h, w, _ = fragment.img.shape

            # Clip fragment coordinates to fit image boundaries
            x_min = max(0, x)
            y_min = max(0, y)
            x_max = min(img_size[1], x + w)
            y_max = min(img_size[0], y + h)

            # Get overlapping region
            overlapping_region = reconstructed_image[y_min:y_max,
x_min:x_max]
            # Get corresponding region in the fragment
            fragment_region = fragment.img[y_min - y:y_max - y, x_min -
x:x_max - x]

            # Detect edges between the overlapping region and the fragment
region
            edges = cv2.Canny(overlapping_region, 100, 200)

            # Apply morphological closing to get a wider border
            size = max(self.kernel_size // 2, 5)
            kernel = np.ones((size, size), np.uint8)
            edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

            # Check if there is a significant color change along the border
            if np.mean(edges) > 0:
                # Perform blending only on the border
                blended_region = cv2.addWeighted(overlapping_region, 0.5,
fragment_region, 0.5, 0)
                reconstructed_image[y_min:y_max, x_min:x_max] =
blended_region
            else:
                # No significant color change detected, just copy the
fragment
                reconstructed_image[y_min:y_max, x_min:x_max] =
fragment_region

        return reconstructed_image


    def get_ssim(self, original_img: np.array, decoded_img: np.array) ->
float:
```

```
        """
        Computes the Structural Similarity Index (SSIM) between two images.
        """
        similarity = ssim_metric(original_img, decoded_img,
multichannel=True, win_size=min(self.db.target_size, 7), channel_axis=2)
        return similarity
```

Файл base.py

```
from fragment import Fragment


class BaseDB:
    def __init__(self):
        pass


    def is_empty(self):
        pass


    def build_tree(self):
        pass


    def add_fragment(self, fragment: Fragment):
        pass


    def get_fragment_by_id(self, fragment_id: int):
        pass


    def find_similar_fragment_id(self, fragment_feature):
        pass
```

Файл factory.py

```
from .base import BaseDB
from .file_db import FileDB
from .mongo_db import MongoDB


class DBFactory:
    def __init__(self, db_type) -> None:
        if db_type == 'file':
            self.db = FileDB()
        elif db_type == 'mongo':
            self.db = MongoDB()
        else:
            raise Exception("Invalid database type")


    def get_db(self) -> BaseDB:
        return self.db
```

Файл file_db.py

```
import os
import numpy as np
from annoy import AnnoyIndex
from .base import BaseDB
from fragment import Fragment

class FileDB(BaseDB):
    def __init__(self):
        self.save_path = '/db/fragments.npy'
        self.fragments = {}
        self.target_size = 64
        self.tree = None
        if os.path.exists(self.save_path):
            self.fragments = np.load(self.save_path,
allow_pickle=True).item()

    def is_empty(self):
        return len(self.fragments) == 0

    def build_tree(self):
        dims = self.fragments[0]['feature'].shape[0]
        self.tree = AnnoyIndex(dims, 'euclidean')
        for i, fragment in self.fragments.items():
            feature = fragment['feature']
            self.tree.add_item(i, feature)
        self.tree.build(100, n_jobs=-1)

    def add_fragment(self, fragment: Fragment):
        fragment_elem = {
            'image': fragment.img,
            'feature': fragment.feature
        }
        fragment_id = len(self.fragments)
        self.fragments[fragment_id] = fragment_elem
        return fragment_id


    def get_fragment_by_id(self, fragment_id: int):
        fragment = self.fragments[fragment_id]
        return fragment

    def find_similar_fragment_id(self, fragment_feature):
```

```
        if self.is_empty() or self.tree is None:
            self.build_tree()

        similar_fragment_id = self.tree.get_nns_by_vector(fragment_feature,
1)[0]
        return similar_fragment_id

    def save_results(self):
        np.save(self.save_path, self.fragments)
```

Файл mongo_db.py

```
import base64
import numpy as np
from annoy import AnnoyIndex
from pymongo import MongoClient
from .base import BaseDB
from fragment import Fragment

class MongoDB(BaseDB):
    def __init__(self, url='mongodb://mongo:27017/', db_name='fragments',
collection_name='fragments'):
        print('Connecting to MongoDB...')
        self.client = MongoClient(url)
        self.db = self.client[db_name]
        # self.db.drop_collection(collection_name) # Clear collection
        self.collection = self.db[collection_name]
        self.target_size = 32
        self.tree = None

    def is_empty(self):
        return self.collection.count_documents({}) == 0

    def build_tree(self):
        dims = np.frombuffer(self.collection.find_one({})['feature'],
dtype=np.float32).shape[0]
        self.tree = AnnoyIndex(dims, 'angular')
        for fragment in self.collection.find():
            fragment_id = int(fragment['_id'])
            feature = np.frombuffer(fragment['feature'], dtype=np.float32)
            self.tree.add_item(fragment_id, feature)
        self.tree.build(10, n_jobs=-1)
```

```
    def add_fragment(self, fragment: Fragment):
        image_base64 = base64.b64encode(fragment.img.tobytes()).decode('utf-
8')
        feature_bytes = fragment.feature.tobytes()

        fragment_doc = {
            '_id': self.collection.count_documents({}),
            'image': image_base64,
            'feature': feature_bytes
        }

        result = self.collection.insert_one(fragment_doc)

        return result.inserted_id

    def get_fragment_by_id(self, fragment_id: int):
        fragment = self.collection.find_one({'_id': int(fragment_id)})
        fragment['image'] =
np.frombuffer(base64.b64decode(fragment['image']),
dtype=np.uint8).reshape(self.target_size, self.target_size, 3)
        fragment['feature'] = np.frombuffer(fragment['feature'],
dtype=np.float32)
        return fragment

    def find_similar_fragment_id(self, fragment_feature):
        if self.is_empty() or self.tree is None:
            self.build_tree()

        similar_fragment_id = self.tree.get_nns_by_vector(fragment_feature,
1)[0]
        return similar_fragment_id
```

Файл app.py

```
import streamlit as st
import numpy as np
import cv2
from encoder import Encoder

@st.cache_resource
def load_encoder():
    return Encoder()
```

```
class App:
    def __init__(self):
        self.encoder = load_encoder()

    def encode_image(self, img):
        print("Encoding image...")
        compressed_data = self.encoder.encode(img)
        print("Image encoded.")
        return compressed_data

    def decode_image(self, compressed_data, img_shape):
        print("Decoding image...")
        reconstructed_img = self.encoder.decode(compressed_data, img_shape)
        print("Image decoded.")
        return reconstructed_img

    def get_ssim(self, img1, img2):
        """Обчислює індекс схожості структури (SSIM) між двома
зображеннями."""
        return self.encoder.get_ssim(img1, img2)

    def get_compression_ratio(self, original_img: np.ndarray,
compressed_data: bytes) -> float:
        """Обчислює відсоток стиснення."""
        original_size = original_img.size
        compressed_size = len(compressed_data)
        return 100 * (1 - compressed_size / original_size)

    def save_image_as_bmp(self, img, filename):
        """Зберігає зображення у форматі BMP."""
        cv2.imwrite(filename + '.bmp', img)
        print("Image saved as BMP.")

    def run(self):
        st.title("Кодування растрових зображень на основі подібності
фрагментів з оператором зсуву")

        mode = st.radio("Оберіть дію:", ("Кодування", "Декодування"))

        if mode == "Кодування":
            uploaded_img = st.file_uploader("Обрати файл для кодування",
type=["jpg", "jpeg", "png"])
            if uploaded_img is not None:
```

```
                img_array = np.array(bytearray(uploaded_img.read()),
dtype=np.uint8)
                img = cv2.imdecode(img_array, 1)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                st.image(img, caption="Оригінальне зображення",
use_column_width=True)

                if st.button("Стиснути зображення"):
                    compressed_data = self.encode_image(img)
                    st.success("Зображення успішно закодоване.")

                    reconstructed_img = self.decode_image(compressed_data,
img.shape[:2])
                    st.image(reconstructed_img, caption="Декодоване
зображення", use_column_width=True)

                    st.write("Індекс схожості:
{:.2f}%".format(self.get_ssim(img, reconstructed_img)))
                    st.write("Відсоток стиснення:
{:.2f}%".format(self.get_compression_ratio(img, compressed_data)))
                    st.download_button(label="Завантажити стиснене
зображення", data=compressed_data, file_name="compressed_data.bin",
mime="application/octet-stream")

        elif mode == "Декодування":
            uploaded_file = st.file_uploader("Обрати файл для декодування",
type=["bin"])
            if uploaded_file is not None:
                compressed_data = uploaded_file.getvalue()
                img_shape = st.text_input("Введіть розмір зображення у
форматі 'висота,ширина'", "1024,768")
                img_shape = tuple(map(int, img_shape.split(',')))

                if st.button("Відновити зображення"):
                    reconstructed_img = self.decode_image(compressed_data,
img_shape)
                    st.image(reconstructed_img, caption="Декодоване
зображення", use_column_width=True)

                    if st.button("Зберегти як BMP"):
                        self.save_image_as_bmp(reconstructed_img,
'decoded_image')
                        st.success("Зображення збережено у форматі BMP.")
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

Програма та методика тестування

КПІ.ІХ-0215.045480.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Дмитро ГУЩІН

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є кодоване представлення растрового зображення.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних в базі;
- перевірка сумісності веб-додатку з останніми версіями сучасних браузерів;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на відповідність функціональним вимогам;
- тестування продуктивності;
- тестування сумісності з різними операційними системами;
- тестування сумісності з різними браузерами;
- тестування інтерфейсу користувача;
- тестування зручності використання.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

Керівництво користувача

КПІ.ІХ-0215.045480.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Дмитро ГУЩІН

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Програмне забезпечення "Кодування растрових зображень на основі подібності фрагментів з оператором зсуву" призначене для ефективного стиснення та відновлення растрових зображень. Воно розроблено для зменшення розміру зображень при збереженні високої якості та деталізації.

Основним призначенням програми є зменшення обсягу пам'яті, необхідної для зберігання зображень, а також підвищення ефективності їх передачі через мережі зв'язку.

Кінцева збірка програмного забезпечення включає веб-інтерфейс, який дозволяє користувачам легко взаємодіяти з додатком. Інсталяційна версія програмного забезпечення представлена у вигляді Docker-контейнера, що забезпечує зручне розгортання та налаштування на різних платформах.

Вміст репозиторію містить:

- вихідний код програми, включаючи основні модулі для кодування та декодування зображень;
- сценарії для налаштування середовища та запуску програми, включаючи Dockerfile та docker-compose.yml;
- документацію, що описує процес налаштування та використання програмного забезпечення;
- приклади тестових зображень та файлів для демонстрації функціональності програми.

Програмне забезпечення надає можливість виконання основних функцій через зручний користувацький інтерфейс, який включає можливості завантаження зображень, виконання операцій кодування та декодування, а також оцінки якості відновлених зображень.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;

– об‘єм ОЗП: 4 Гб;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів:

– тип процесору: Intel Core i7;

– об‘єм ОЗП: 8 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

2.2 Завантаження застосунку

На даний момент для отримання доступу до збірки програмного забезпечення користувачі можуть скористатися репозиторієм на платформі GitHub. Для завантаження необхідно склонувати репозиторій на свій локальний комп'ютер за допомогою команди «git clone». Для зручного розгортання програми на будь-якій платформі можна скористатися Docker, переконавшись, що Docker встановлено на вашій системі, і запустити команду «docker-compose up –build» для створення та запуску контейнерів.

2.3 Перевірка коректної роботи

Після завершення процесу розгортання програмного забезпечення необхідно відкрити веб-браузер і перейти за адресою, яка вказана у конфігурації. У вікні інтерфейсу треба переконатися, що всі елементи користувацького інтерфейсу завантажилися коректно і відображаються правильно.

3 ВИКОНАННЯ ПРОГРАМИ

При відкритті веб-інтерфейсу з’являється головний екран з двома опціями: "Кодування" та "Декодування". Для початку роботи необхідно обрати одну з них, натиснувши відповідне радіо-поле.

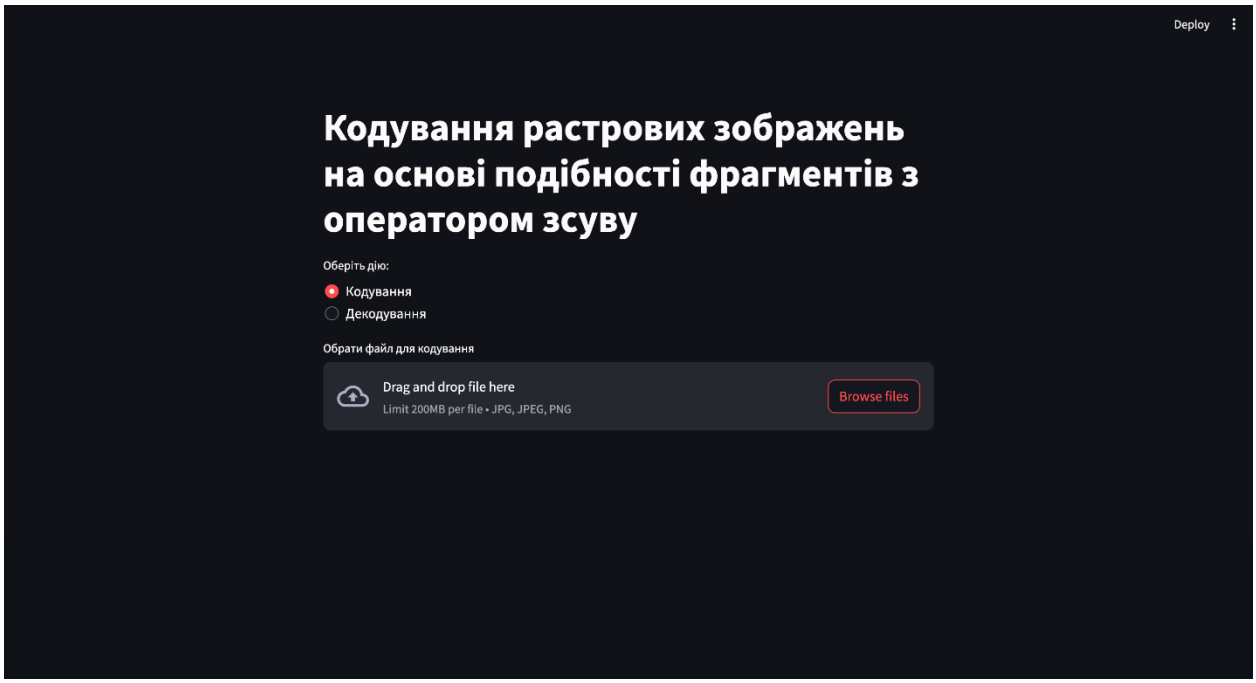


Рисунок 3.1 – Вигляд веб-додатку після запуску

У режимі "Кодування" користувач має змогу натиснути кнопку "Обрати файл для кодування" та завантажити зображення у форматі JPG, JPEG або PNG. Після завантаження зображення буде відображено на екрані. Для того щоб розпочати процес кодування необхідно натиснути кнопку "Стиснути зображення". Після завершення процесу закодоване зображення з'явиться на екрані, а також будуть показані індекс схожості (SSIM) та відсоток стиснення. Для збереження закодованого файлу на пристрій необхідно натиснути кнопку "Завантажити стиснене зображення".

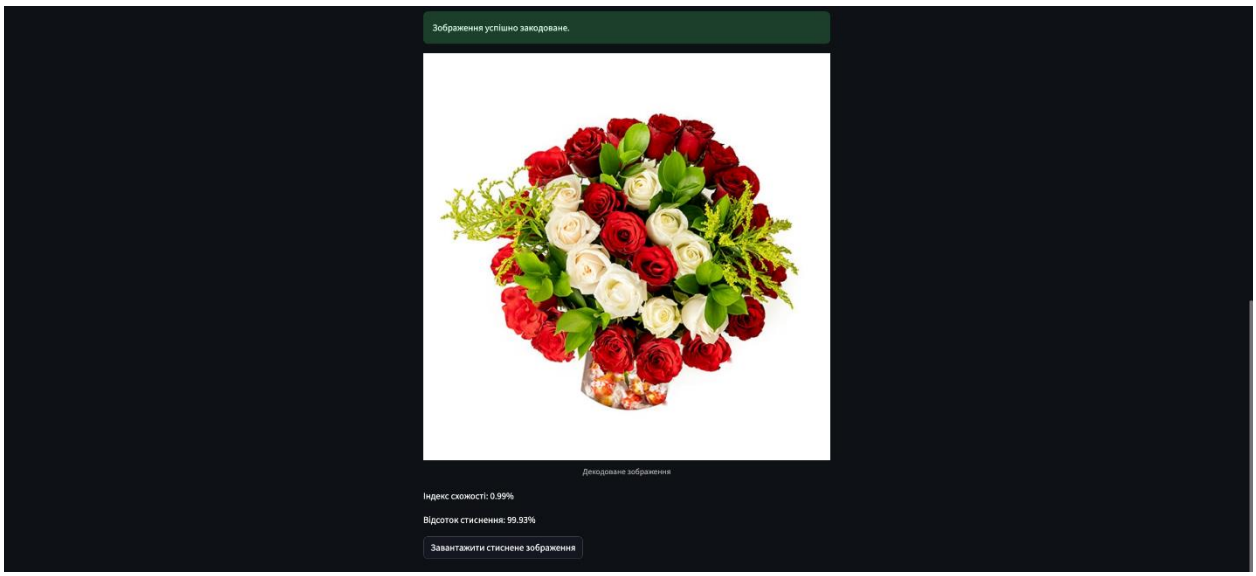


Рисунок 4.10 – Вигляд веб-додатку після кодування зображення

У режимі "Декодування" користувач має змогу натиснути кнопку "Обрати файл для декодування" та завантажити раніше збережений закодований файл у форматі BIN. Далі треба ввести розміри оригінального зображення у форматі та натиснути кнопку "Відновити зображення". Після завершення процесу декодування відображене зображення з'явиться на екрані. Для того щоб зберегти декодоване зображення у форматі BMP на ваш пристрій необхідно натиснути кнопку "Зберегти як BMP".

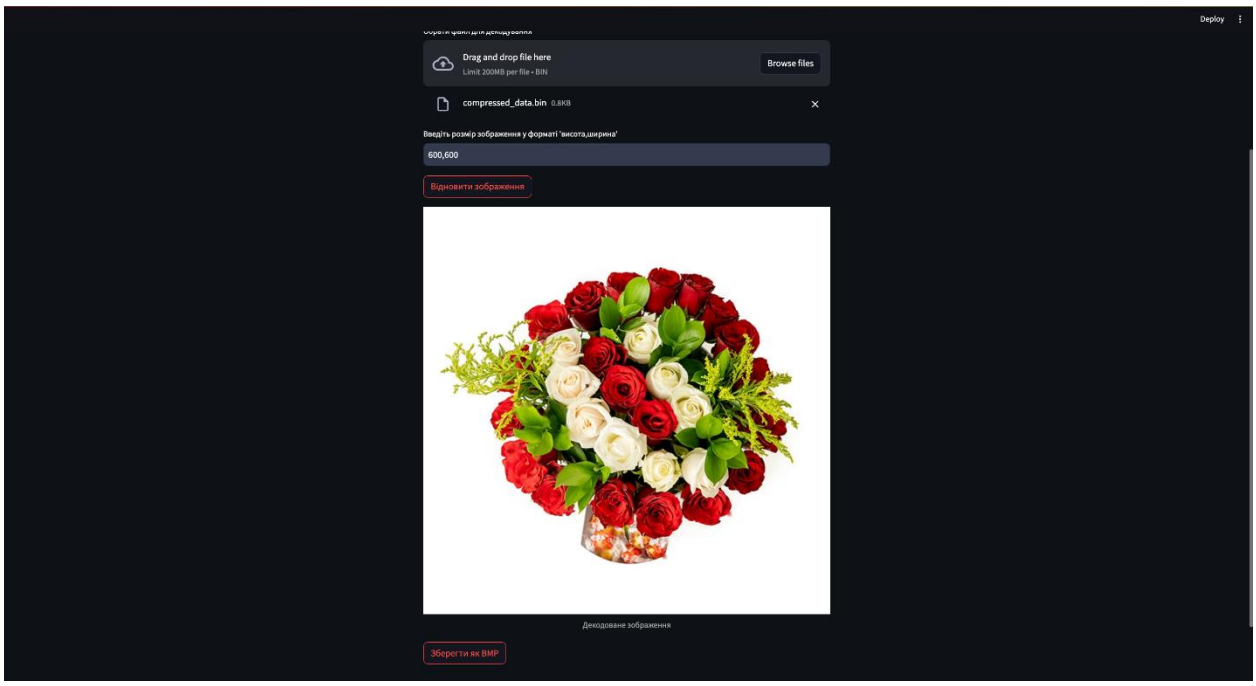


Рисунок 4.11 – Вигляд веб-додатку після декодування зображення

Під час роботи з програмою важливо слідкувати за повідомленнями про успішність виконання операцій та можливими помилками, що можуть з'являтися на екрані. У разі виникнення помилок необхідно перевірити коректність завантажуваних файлів та правильність введених даних. Для завершення роботи достатньо закрити веб-браузер або зупинити сервер, на якому запущено програму.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення кодування растрових зображень на основі подібності фрагментів з оператором зсуву

Графічний матеріал

КПІ.ІХ-0215.045480.06.99

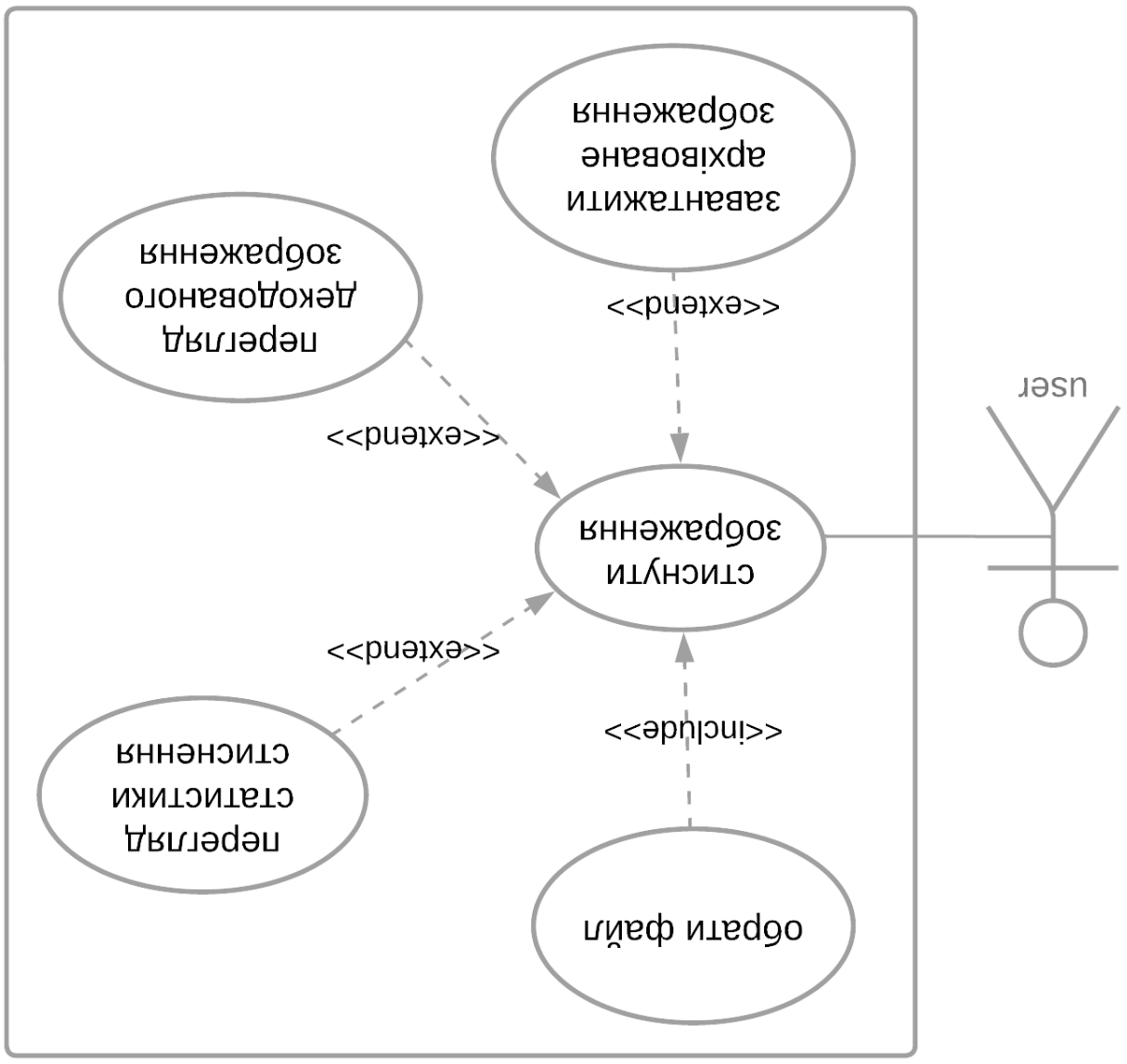
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юрій ОЛІЙНИК

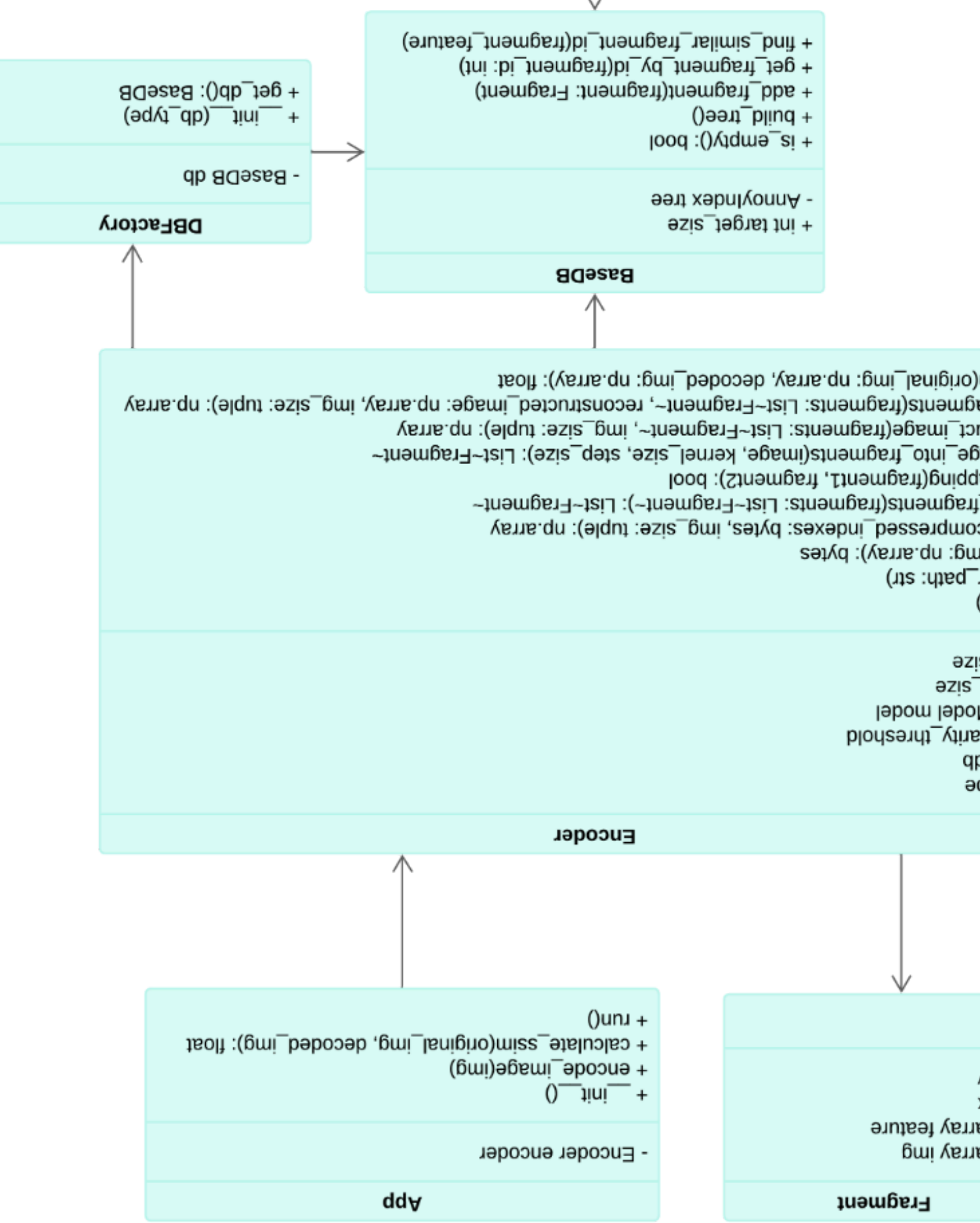
Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Дмитро ГУЩІН

Київ – 2024

								КПІ.ІП-0215.045480.06.99.СВВ	
Зм.	Арк.	№ докум.	Підп.	Дата					
Розробив		Гуцій Д.О.			Схема структура варіантів використання			Літ.	Маса	Масштаб
Перевірив		Олійник Ю.О.								
Т. контр.								Аркуш 1		Аркушів 1
								КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02	
Н. контр.		Шулькевич Т.В.			Кодування растрових зображень на основі подібності фрагментів з оператором зсуву				
Затвердив		Жарікова Е.В.							





Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Гуцій Д.О.		
Перевірив		Олійник Ю.О.		
Т. контр.				
Н. контр.		Шупькевич Т.В.		
Затвердив		Жаріков Е.В.		

КПІ.ІП-0215.045480.06.99.ССК

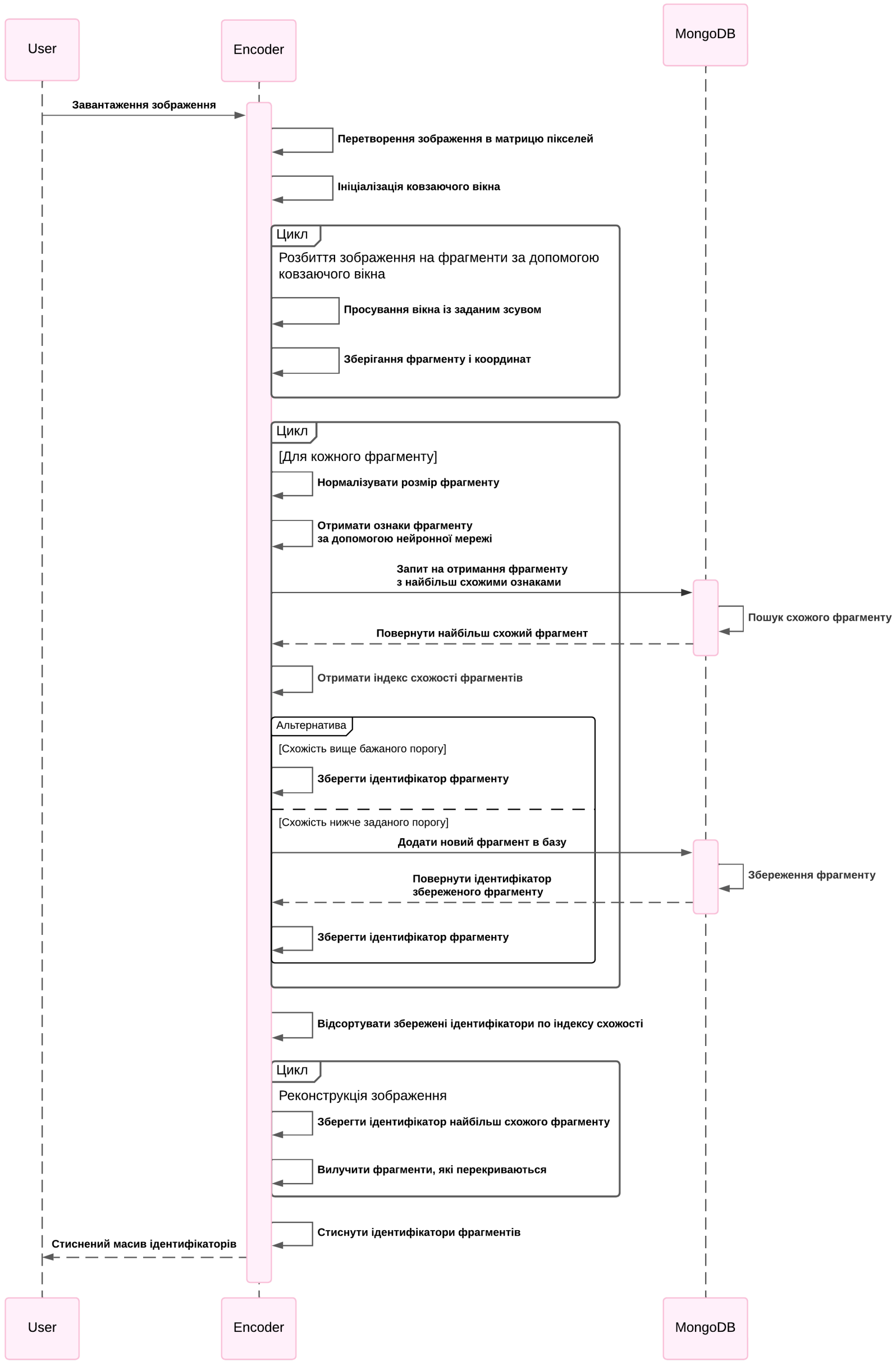
Схема структура класів програмного забезпечення

Літ.	Маса	Масштаб

Аркуш 1 | Аркушів 1

Кодування растрових зображень на основі подібності фрагментів з оператором зсуву

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-02



					КПІ.ІП-0215.045480.06.99.ССП
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна послідовності
Розробив		Гущін Д.О.			Лит. / Маса / Масштаб
Перевірив		Олійник Ю.О.			Аркуш 1 / Аркушів 1
Т. контр.					Кодування растрових зображень на основі подібності фрагментів з оператором зсуву
					КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02
Н. контр.		Шулькевич Т.В.			
Затвердив		Жаріков Е.В.			