Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Вдосконалення алгоритмів стиснення та обробки зображень»

Виконав: студент групи ІПЗ23-1зм

Спеціальність

121 Інженерія програмного забезпечення

Калюжний Д.С.
(прізвище та ініціали)

Керівник к.е.н. Яковенко Т. Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровський державний технічний університет
(місце роботи)

доцент кафедри математичного моделювання та системного аналізу
(посада)

к.т.н., доц. Волосова Н.М.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Калюжний Д.С. Вдосконалення алгоритмів стиснення та обробки зображень.

Дипломна робота на здобуття освітнього ступеня магістр за спеціальністю 121 «Інженерія програмного забезпечення» – Університет митної справи та фінансів, Дніпро, 2025.

Кваліфікаційна робота магістра присвячена дослідженню сучасних методів стиснення та обробки зображень, які є одними з ключових аспектів цифрової обробки даних. У роботі виконано глибокий аналіз класичних методів стиснення, таких як безвтратні алгоритми (Huffman Coding, RLE, LZW) та алгоритми з втратами (JPEG, JPEG2000, WebP). Виявлено їх основні переваги, недоліки та області застосування. Особлива увага приділена методам, які використовують математичні перетворення, наприклад, DCT та Wavelet Transform, що забезпечують ефективну компресію для зображень із складною структурою. Розглянуто перспективи використання гібридних методів, які поєднують безвтратні та втратні технології, для досягнення компромісу між високою ефективністю стиснення та якістю відновлених даних. У дослідженні також акцентовано увагу на адаптації алгоритмів до нових форматів, зокрема 3D-зображень та гіперспектральних даних, що є актуальними для багатьох інноваційних галузей.

Практична частина роботи передбачає розробку програмного забезпечення, яке реалізує ефективні методи стиснення та обробки зображень. Проведено тестування системи, яке підтвердило доцільність застосування сучасних підходів для покращення ефективності стиснення та обробки мультимедійних даних.

Ключові слова: стиснення зображень, обробка зображень, безвтратне стиснення, штучний інтелект, нейронні мережі, фільтрація, відновлення зображень, алгоритми з втратами.

ABSTRACT

Kaliuzhnyi D.S. Improving image compression and processing algorithms.

Diploma thesis for the degree of Master's Degree in specialty 121 «Software Engineering» – University of Customs and Finance, Dnipro, 2025.

The master's thesis is devoted to the study of modern methods of image compression and processing, which are one of the key aspects of digital data processing. The work provides an in-depth analysis of classical compression methods, such as lossless algorithms (Huffman Coding, RLE, LZW) and lossy algorithms (JPEG, JPEG2000, WebP). Their main advantages, disadvantages and areas of application are identified. Particular attention is paid to methods that use mathematical transformations, such as DCT and Wavelet Transform, which provide efficient compression for images with complex structures. The prospects of using hybrid methods that combine lossless and lossy technologies to achieve a compromise between high compression efficiency and the quality of recovered data are considered. The study also focuses on the adaptation of algorithms to new formats, such as 3D images and hyperspectral data, which are relevant for many innovative industries.

The practical part of the work involves the development of software that implements effective methods of image compression and processing. The system was tested, which confirmed the feasibility of applying modern approaches to improve the efficiency of compression and processing of multimedia data.

Keywords: image compression, image processing, lossless compression, artificial intelligence, neural networks, filtering, image restoration, lossy algorithms.

ЗМІСТ

ВСТУП

Сучасний світ неможливо уявити без потужного впливу інформаційних технологій, і зокрема без широкого використання зображень у різноманітних сферах життя. Завдяки розвитку цифрових технологій, зокрема зображувальних систем, обробка, збереження та передача зображень набули неабиякої значущості у таких галузях, як медицина, наука, мистецтво, а також в усіх галузях, пов’язаних з обробкою мультимедійних даних [1]. Сучасні методи стиснення зображень дозволяють значно зменшити обсяг даних, що зберігаються та передаються, зберігаючи при цьому необхідну якість. Однак розвиток нових технологій та постійний ріст обсягів інформації ставлять перед дослідниками нові виклики, такі як потреба у високій ефективності стиснення, покращенні якості відновлених зображень, мінімізації помилок при стисненні, а також адаптація методів до нових форматів даних, таких як відео, 3D-об’єкти чи гіперспектральні зображення.

Актуальність дослідження сучасних методів стиснення та обробки зображень обумовлена постійним зростанням обсягів інформації, що обробляється, а також новими вимогами до якості зображень в різноманітних сферах, зокрема в медицині, телекомунікаціях, відеоаналітиці та інших. З кожним роком зростає потреба в ефективних і швидких алгоритмах, які можуть обробляти великі обсяги даних без значних втрат у якості зображень, а також дозволяти здійснювати їх реальне застосування в реальному часі. Саме тому необхідно постійно розвивати нові методи стиснення зображень, покращуючи їх ефективність, знижуючи час обробки та підвищуючи надійність цих методів.

Метою цієї роботи є дослідження та порівняння сучасних методів стиснення та обробки зображень, визначення їхніх переваг і недоліків в залежності від специфіки завдань, а також розробка рекомендацій щодо вибору оптимальних методів для різних типів зображень та умов їх

використання. В рамках цієї роботи розглядатимуться як класичні підходи, так і новітні технології, такі як методи стиснення на основі глибинного навчання, нейронних мереж та адаптивних алгоритмів.

Для досягнення поставленої мети в роботі необхідно виконати ряд завдань:

– описати основні методи стиснення зображень, зокрема їх класифікацію на безвтратні та з втратами, а також характеристику основних підходів для кожного типу стиснення;

– розглянути сучасні методи обробки зображень, такі як фільтрація, відновлення, детекція контурів та інші методи, що використовуються для покращення якості зображень після стиснення;

– оцінити ефективність різних методів стиснення та обробки зображень на основі якості відновлених зображень та часу, необхідного для їх обробки;

– провести порівняння традиційних методів стиснення з новітніми методами, що використовують штучний інтелект та глибинне навчання;

– розробити рекомендації щодо вибору оптимальних методів стиснення для різних задач і типів зображень.

Об’єктом дослідження цієї роботи є процеси стиснення та обробки зображень, що застосовуються для різних типів даних, таких як фотографії, медичні зображення, відеофайли та інші мультимедійні дані. Важливу роль у дослідженні відіграє також вивчення специфічних вимог до методів обробки, що виникають залежно від галузі їх використання.

Предметом дослідження є конкретні алгоритми стиснення та обробки зображень, методи оцінки якості відновлених зображень, а також порівняння ефективності різних підходів, зокрема класичних та інноваційних, на основі теоретичних та практичних результатів.

Методи дослідження, що будуть застосовані в цій роботі, включають аналіз наукових публікацій з даної теми, математичне моделювання, комп’ютерне моделювання та порівняння різних алгоритмів стиснення,

експериментальне дослідження ефективності методів обробки зображень за допомогою програмного забезпечення, а також статистичний аналіз отриманих результатів.

Практична значимість роботи полягає в тому, що результати дослідження можуть бути використані для покращення існуючих систем стиснення зображень та їх обробки в реальних умовах. Це дозволить знизити витрати на зберігання та передачу даних, підвищити якість оброблених зображень, а також розширити можливості застосування даних технологій у різних галузях: від медичних систем діагностики до програм для обробки мультимедійних матеріалів в Інтернеті.

Наукова новизна роботи полягає у комплексному порівнянні класичних та новітніх методів стиснення та обробки зображень, а також в оцінці їх ефективності з точки зору часу обробки та якості результату. Крім того, розглянуті нові підходи до стиснення, що включають використання нейронних мереж та адаптивних алгоритмів, які відкривають нові перспективи для застосування сучасних технологій у галузі обробки зображень.

Кваліфікаційна робота має на меті не лише огляд і порівняння існуючих методів стиснення та обробки зображень, але й розробку нових підходів до покращення їх ефективності та застосування в реальних умовах.

Структура кваліфікаційної роботи магістра. Кваліфікаційна робота складається з трьох розділів. Обсяг кваліфікаційної роботи магістра – 99 сторінок. Робота містить 18 рисунків та 3 таблиці. Список використаних джерел включає 20 посилань.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Класифікація методів стиснення

Методи стиснення даних є важливими інструментами для ефективного зберігання та передавання інформації в різноманітних обчислювальних системах. Вони дозволяють зменшити обсяг даних, що передаються або зберігаються, що є особливо важливим в умовах обмежених ресурсів, таких як пам’ять, пропускна здатність каналів зв’язку або час для обробки [1]. Існують два основних типи методів стиснення: безвтратне стиснення та стиснення з втратами. Обидва типи мають свої характеристики, області застосування та переваги, а також визначаються різними підходами до обробки даних.

Ключові методи стиснення наведено на рисунку 1.1.

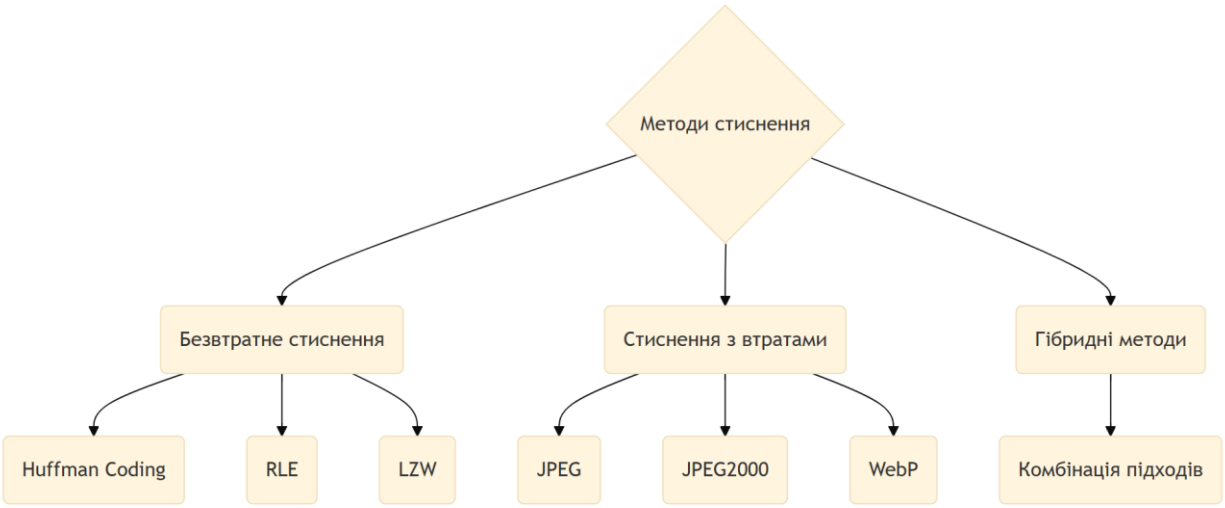

Рисунок 1.1 – Методи стиснення

Безвтратне стиснення є типом стиснення, при якому збереження оригінальної інформації є абсолютним. Це означає, що після процесу

стиснення і наступного відновлення даних (декомпресії) відновлені дані будуть абсолютно ідентичними початковим. Такий підхід особливо важливий для застосувань, де втрата частини даних є неприпустимою, наприклад, у медичних зображеннях, текстових документах або базах даних, де точність і повнота інформації мають критичне значення.

Методи безвтратного стиснення зберігають усю інформацію, що міститься в даних, і використовують спеціальні алгоритми для видалення зайвої або повторюваної інформації. Одним з основних принципів, які використовуються в безвтратному стисненні, є пошук і заміна повторюваних елементів, наприклад, послідовностей символів або блоків даних. Цей підхід застосовується в алгоритмах, таких як Huffman coding (кодування Хаффмана), алгоритм Лемпела-Зіва (LZ77, LZ78) та інші. Алгоритми стиснення можуть використовувати різні техніки, зокрема статистичні методи, для виявлення й ефективного кодування повторюваних структур або шаблонів у даних.

Кодування Хаффмана є одним із класичних і найбільш відомих методів безвтратного стиснення. Це метод кодування, який базується на побудові дерева, де найменші ймовірні елементи отримують більш довгі коди, а найбільш ймовірні – коротші. Завдяки цьому принципу кодування досягається максимальна ефективність у використанні бітів для представлення символів, що зустрічаються найчастіше [1, 2]. Алгоритм Лемпела-Зіва також є основою для багатьох популярних методів стиснення, таких як алгоритм ZIP. Він використовує методи пошуку повторюваних послідовностей символів у потоці даних та заміни їх на коротші позначення, що значно зменшує обсяг даних.

Важливою характеристикою методів безвтратного стиснення є те, що вони можуть застосовуватись до широкого кола типів даних. Це можуть бути як текстові файли, так і графічні зображення, звукові файли, програми чи бази даних. Враховуючи різноманіття типів даних, для кожного з них розроблені специфічні алгоритми безвтратного стиснення, які оптимізують процес стиснення в залежності від особливостей структури даних. Наприклад, для

текстових файлів можуть бути використані алгоритми, засновані на статистичному аналізі символів, а для зображень – методи, що враховують просторові та кольорові характеристики пікселів.

Стиснення з втратами, на відміну від безвтратного стиснення, передбачає певну втрату інформації під час стиснення. Це означає, що після декомпресії даних відновлені файли не будуть ідентичні оригінальним, а в деяких випадках навіть не буде можливо відновити оригінальні дані в повному обсязі. Зазвичай такі методи використовуються в ситуаціях, коли не є критичним збереження всіх деталей оригінальної інформації, і де зменшення обсягу даних важливіше за абсолютну точність відновлення. Прикладом таких ситуацій можуть бути мультимедійні файли, зокрема зображення, аудіо та відео.

Процес стиснення з втратами базується на ідеї, що деякі частини даних є менш помітними для людини або системи, що не мають критичного значення для основного змісту. Таким чином, алгоритми стиснення з втратами намагаються відкинути менш важливу інформацію, щоб зберегти найбільш суттєві частини даних. Це дозволяє значно зменшити обсяг файлів, але за рахунок втрати частини даних. Одним із найбільш поширених прикладів таких методів є стиснення аудіо і відео, де використовуються алгоритми, що знижують точність відтворення звуків чи зображень у діапазоні, який не сприймається людським вухом або оком [1-3].

Для зображень широко застосовуються методи, засновані на стисненні з втратами, такі як JPEG. Цей метод стиснення базується на зменшенні точності кольорових компонентів зображення, зокрема через відкидання високочастотних компонентів, які важко сприймаються людським оком. Технологія стиснення з втратами в JPEG дозволяє досягти значного зменшення розміру зображень, при цьому зберігаючи достатній рівень якості, щоб споживач не помітив різниці між оригіналом і стисненим зображенням. Інші методи, такі як MP3 або AAC для аудіо, також використовують стиснення

з втратами, де знижується точність звуків, які знаходяться поза межами чутливості людини або які не впливають на сприйняття основної інформації.

Однак, незважаючи на численні переваги стиснення з втратами, цей підхід має свої недоліки. Втрата інформації може бути критичною для деяких типів даних, де навіть незначні зміни можуть призвести до зниження якості або спотворення значення. Тому для таких випадків використовують безвтратні методи стиснення, хоча вони можуть бути менш ефективними у зменшенні обсягу даних. Загалом, вибір між безвтратним стисненням і стисненням з втратами залежить від конкретних вимог до зберігання та обробки даних. Якщо важливо зберегти точність і повноту оригінальної інформації, то застосовуються методи безвтратного стиснення. Якщо ж важливо зменшити обсяг даних при збереженні достатньої якості для користувача, то використовуються методи стиснення з втратами.

У деяких випадках також застосовуються комбіновані методи стиснення, які поєднують елементи обох підходів. Такі методи можуть забезпечити кращі результати в специфічних умовах, наприклад, у мультимедійних системах, де частина даних стиснена без втрат, а частина – з втратами. Відповідно, наукові дослідження та розробки в області стиснення даних продовжують еволюціонувати, зокрема завдяки використанню нових алгоритмів, оптимізаційних технік і спеціалізованих підходів, що дозволяють досягати кращих результатів як у безвтратному стисненні, так і в стисненні з втратами.

Таким чином, класифікація методів стиснення на безвтратні та з втратами є основою для розуміння процесів стиснення даних у сучасних обчислювальних системах. Кожен із цих підходів має свої переваги і обмеження, що визначають їх застосування в різних областях, від архівування даних до мультимедійних технологій і передачі інформації. Вибір методу стиснення залежить від балансу між ефективністю зменшення обсягу даних та необхідністю збереження їхньої точності і повноти.

1.2 Класичні методи стиснення

Класичні методи стиснення даних є основою сучасних алгоритмів, що забезпечують ефективне зменшення обсягу даних без значної втрати інформації. Ці методи стали важливими інструментами в області обробки і зберігання даних, особливо в контексті обмежених ресурсів, таких як пам’ять і пропускна здатність мереж [4]. Одними з найбільш відомих і широко використовуваних класичних методів стиснення є кодування Хаффмана (Huffman Coding), кодування з довгими послідовностями (Run-Length Encoding, RLE) та алгоритм Лемпела-Зіва-Вельча (Lempel-Ziv-Welch, LZW). Ці методи різняться за принципами своєї роботи, але всі вони мають на меті зменшити розмір файлів за рахунок використання специфічних структур даних і алгоритмів, що дозволяють ефективно представляти інформацію.

Еволюція класичних методів стиснення наведено на рисунку 1.2.

Кодування Хаффмана є одним із найпоширеніших методів стиснення, що використовуються в різних форматах файлів, таких як ZIP, JPEG, а також в алгоритмах стиснення текстових даних. Метод кодування Хаффмана був розроблений Девідом Хаффманом в 1952 році і базується на принципі, що найбільш часто зустрічаються символи повинні мати короткі коди, а рідше зустрічаються – довші. Це досягається шляхом побудови спеціального двійкового дерева, яке називається деревом Хаффмана. Кожен символ у вхідних даних отримує унікальний код, причому довжина цього коду пропорційна частоті зустрічальності символу. Тобто символи, що з’являються частіше, отримують більш короткий код, а символи, що з’являються рідше – довший. Це дозволяє зменшити середню довжину коду і, відповідно, зменшити загальний обсяг даних [4, 5]. Для побудови дерева Хаффмана використовуються такі кроки, як підрахунок частоти кожного символу, побудова пріоритетної черги для вибору двох найменших частот і об’єднання їх у новий вузол дерева.

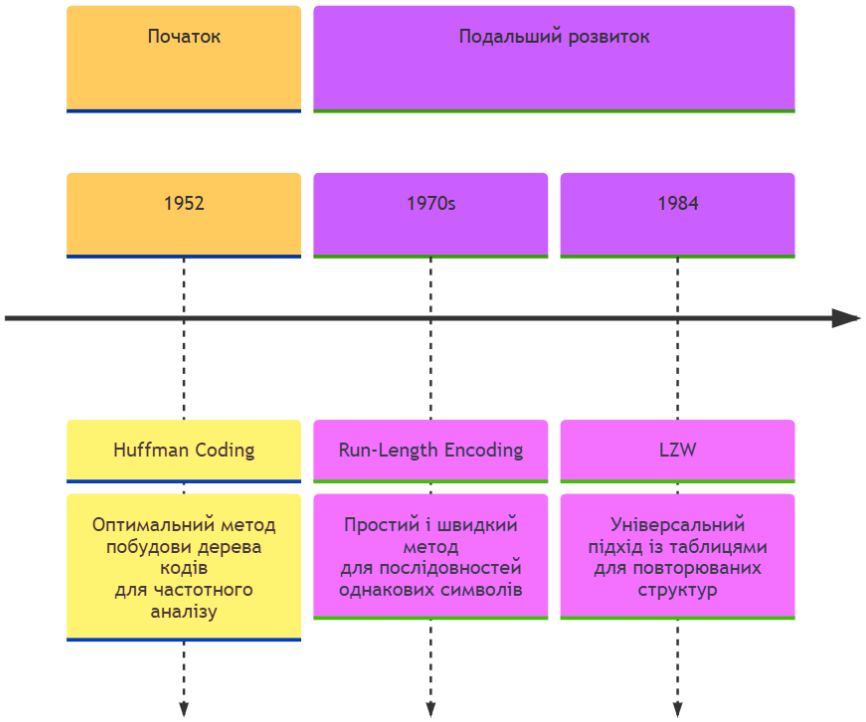

Рисунок 1.2 – Еволюція класичних методів стиснення

Алгоритм Хаффмана має кілька переваг, серед яких – висока ефективність у випадках, коли частота символів значно відрізняється. Однак метод також має свої обмеження. Наприклад, він не є оптимальним для стиснення даних, де всі символи з’являються з однаковою частотою, оскільки в такому випадку стиснення буде менш ефективним. Крім того, цей метод може бути не найкращим для стиснення дуже малих обсягів даних або для даних, що мають багато різних символів з низькою частотою. Для таких випадків були розроблені інші методи, зокрема, кодування з довгими послідовностями та алгоритми Лемпела-Зіва-Вельча.

Кодування з довгими послідовностями, або Run-Length Encoding (RLE), є простим і ефективним методом стиснення, який добре працює в ситуаціях, коли вхідні дані містять великі послідовності однакових елементів. Принцип цього методу полягає в тому, що повторювані символи замінюються на пару «символ-кількість повторень». Наприклад, для послідовності «AAAABBBCCDAAA» її можна закодувати як «4A3B2C1D3A». Це значно

зменшує розмір даних, коли є великі послідовності однакових символів. Однак для даних, що не містять великих послідовностей однакових елементів, ефективність цього методу значно знижується, оскільки результат може бути навіть більшим за вихідний файл. Тому RLE найкраще підходить для стиснення зображень у чорно-білому форматі або для даних, що містять великі порції повторюваної інформації. Метод широко використовується в простих графічних форматах, таких як BMP, і в деяких текстових форматах для стиснення простих даних.

Алгоритм Лемпела-Зіва-Вельча (LZW) є ще одним класичним методом стиснення, який використовує інший підхід. Він був розроблений в 1984 році Абрахамом Лемпелем, Джейкобом Зівом та Тельмо Вельчем і представляє собою вдосконалений варіант алгоритму Лемпела–Зіва. Алгоритм LZW є дуже популярним і широко використовуваним, зокрема в таких форматах, як GIF та TIFF. Принцип роботи цього алгоритму полягає в тому, що він замінює повторювані послідовності символів на коротші коди, що зберігаються в таблиці [4, 5]. Алгоритм починається з ініціалізації таблиці, в якій зберігаються одиничні символи. Потім він проходить через вхідні дані, шукаючи найбільші можливі послідовності символів, що вже містяться в таблиці, і додає їх у таблицю з новими кодами.

Основною перевагою LZW є те, що він не вимагає попереднього аналізу частоти символів або їх статистичних властивостей, як це робиться в алгоритмі Хаффмана. Це означає, що алгоритм LZW може бути застосований до широкого спектра даних і є більш універсальним у порівнянні з іншими методами. Крім того, LZW ефективно працює з даними, які містять багато повторюваних послідовностей. Однак алгоритм має і свої обмеження. Одним з таких обмежень є те, що в процесі стиснення LZW може призвести до утворення великих таблиць, що потребує значних обчислювальних ресурсів, особливо для великих обсягів даних. Крім того, якщо вхідні дані не містять

багато повторюваних послідовностей, ефективність цього методу може знижуватись.

Всі три методи – кодування Хаффмана, кодування з довгими послідовностями (RLE) та LZW – належать до класичних алгоритмів стиснення, і кожен з них має свої сильні сторони в залежності від типу даних, з якими працює. Кодування Хаффмана є найбільш ефективним для даних, що мають різну частоту символів, і широко використовується для стиснення текстів і деяких графічних форматів. Кодування з довгими послідовностями є простим і швидким методом, що добре працює з графічними зображеннями, що містять великі повтори однакових пікселів [5, 6]. Алгоритм LZW є більш універсальним і здатен адаптуватися до різноманітних типів даних, але має певні недоліки, пов’язані з розміром таблиць і необхідністю обробки великих обсягів інформації.

Загалом, класичні методи стиснення є основою для багатьох сучасних алгоритмів і технологій обробки даних. Вони продовжують використовуватися в ряді реальних застосувань завдяки своїй простоті та ефективності, а також через те, що ці методи можна реалізувати на різних апаратних і програмних платформах. Їхня важливість не зменшується, незважаючи на розвиток нових, більш складних методів стиснення, оскільки вони пропонують оптимальне поєднання швидкості і ефективності для багатьох типів даних і застосувань.

1.3 Алгоритми стиснення з втратами

Перелік існуючих алгоритмів стиснення з втратами [6, 7] наведено в таблиці 1.1.

Таблиця 1.1

Алгоритми стиснення з втратами

Алгоритм	Тип стиснення	Переваги	Недоліки
JPEG	Стиснення з втратами	Висока ефективність стиснення; добре підходить для фотографій; підтримується більшістю пристроїв.	Втрата якості при високому рівні стиснення; артефакти у зображеннях з високою контрастністю.
JPEG2000	Стиснення з втратами (і без втрат)	Краща якість при низькому бітрейті; підтримка безвтратного стиснення; менше артефактів при високому стисненні.	Менша сумісність з програмами та пристроями; великі файли для високої якості; складність алгоритму.
WebP	Стиснення з втратами та без втрат	Кращі показники стиснення порівняно з JPEG; підтримує альфа-канал для прозорості.	Не всі браузери підтримують WebP; може бути менш ефективним для деяких типів зображень.

1.4 Методи стиснення з використанням перетворень

В той же час методи стиснення з використанням перетворень [8, 9] доцільно представити наступним чином (табл. 1.2):

Таблиця 1.2

Методи стиснення з використанням перетворень

Метод стиснення	Тип перетворення	Переваги	Недоліки
DCT (Discrete Cosine Transform)	Перетворення косинусів	Широко використовується в JPEG і MPEG; ефективно стискає зображення з регулярними патернами; швидка обробка.	Втрата інформації при високому рівні стиснення; не підтримує ефективне стиснення для зображень з високим контрастом або шумом.
Wavelet Transform	Перетворення хвильових функцій	Підтримує ефективне стиснення на всіх рівнях; краще працює з зображеннями з високим контрастом і деталями.	Більш складний алгоритм; вимагає більше обчислювальних ресурсів; складність у реалізації для великих зображень.

1.5 Аналіз суміжних досліджень

У дослідженні [1] застосовується фільтр експоненційно зваженого рухомого середнього (EWMA) для перетворення значень інтенсивності пікселів RGB-кольорових зображень. Обробка виконується в колірній системі YC b C r і орієнтована лише на компонент Y, що відповідає за яскравість. Вибір часової фільтрації з експоненціальним зважуванням має на меті створення зображень з нижчим енергетичним споживанням простим і ефективним способом. Цей фільтр надає меншу вагу попереднім інтенсивностям пікселів, водночас більший акцент робиться на вхідному сигналі, який відображає загальну яскравість зображення, а не індивідуальні рівні яскравості окремих пікселів. У порівнянні з алгоритмом лінійного відображення Concurrent Brightness & Contrast Scaling (CBCS), експоненціальний фільтр здатен

генерувати зображення з меншою потужністю, що призводить до більшого стискання гістограми, але також з високою перцептуальною якістю. Для перевірки ефективності фільтра використовуються показники структурної подібності (SSIM) та швидкості зменшення потужності, що дозволяє емпірично визначити оптимальні значення для залучених параметрів.

Стаття [2] стосується методу зіставлення зображень, який широко використовується в системах навігації, що базуються на зображеннях. Більшість методів зіставлення припускають ідеальні умови без врахування реальних проблем, таких як розмиття зображень, що часто виникає в реальному часі. Метод відновлення та зіставлення зображень, зокрема JRM-DSR, є ефективним способом боротьби з погіршенням якості зображень у реальному часі, оскільки використовує рідкісне представлення реального зображення в словнику, побудованому на основі еталонного зображення. Однак, коли розмір еталонного зображення набагато більший за розмір реального зображення, розмір словника стає настільки великим, що це займає багато часу та ускладнює отримання рідкісного представлення. У цій роботі пропонується новий метод відновлення та зіставлення зображень на основі ієрархічного рідкісного представлення (JRM-HSR), який зменшує розмір словника за допомогою кластеризації для виконання грубого зіставлення, а потім виконує точне зіставлення в підмножині оригінального словника. JRM-HSR є практичною моделлю, яка виграє завдяки ієрархічній структурі. У порівнянні з JRM-DSR, швидкість JRM-HSR у рази вища: вона в 16 разів швидша при одному рідкісному представлення і в 2 рази швидша в рамках одного повного алгоритмічного процесу, при цьому зберігаючи таку ж точність.

У статті [3] запропоновано метод стиснення зображень, що базується на виявленні важливих регіонів зображення. Оскільки не вся інформація в зображенні має однакову цінність, можна визначити важливі ділянки для високоякісного стиснення, а менш важливі області можуть бути стиснуті з

більшою втратою якості. Спочатку зображення сегментуються на дві частини: передній план, який є більш важливим, і фон, який має менше значення. Далі застосовується метод оптимального масового транспортування в рамках генеративної змагальної мережі (GAN), щоб збільшити передній план і зменшити фон, зберігаючи форму та загальну площу зображення незмінними. В результаті, у обробленому зображенні співвідношення переднього плану до фону стає більшим, ніж у початковому зображенні, і цей параметр можна контролювати, надаючи користувачам можливість регулювати ступінь стиснення. Після цього зображення, оброблене за допомогою GAN, використовується для стиснення. Для відновлення зображення застосовується модель GAN до стиснутого зображення, і співвідношення переднього плану до фону відновлюється за допомогою оптимальної карти масового транспортування. Результати тестів показують, що цей метод ефективно відновлює деталі важливих компонентів в стиснутих зображеннях, досягаючи при цьому високого коефіцієнта стиснення.

Сьогодні кожна десята людина у світі страждає від серцево-судинних захворювань. Передбачення таких захворювань є важливим завданням для медичних експертів. Існує багато досліджень, присвячених класифікації серцевих захворювань через аналіз ЕКГ сигналів, але лише кілька робіт враховують етап денойзінгу перед класифікацією, щоб зменшити небажані артефакти в ЕКГ сигналах. У роботі [4] реалізовано метод Байєсівського згладжування для видалення шумів з зображень ЕКГ сигналів перед процесом класифікації. Запропонований процес денойзінгу також використовує техніки визначення регіону інтересу (ROI), що знижує обчислювальний час на етапі попередньої обробки та покращує точність класифікації, чітко виділяючи межі сигналу.

У статті [5] досліджується можливість використання колірних гістограм, сформованих у просторі HS, для розпізнавання зображень. Порівняння колірних гістограм здійснюється шляхом обчислення коефіцієнта перехресної

кореляції. Отримано залежності коефіцієнта перехресної кореляції гістограм від ступеня стиснення та методу інтерполяції.

У статті [6] пропонується нова методика стиснення зображень при надзвичайно низьких бітових швидкостях, що є складним завданням у умовах обмеженого каналу зв'язку, як-от в аерокосмічних та глибоководних дослідженнях. Хоча останнім часом глибоке навчання досягло значного прогресу у стисненні зображень, мало хто з існуючих методів орієнтований на наднизькі бітові швидкості. Для вирішення цієї проблеми в роботі запропоновано нову структуру на основі зворотного генерування зображень для стиснення при наднизьких бітових швидкостях. Запропонована структура складається з трьох модулів: модуля зворотного генерування зображень (IIG), модуля стиснення згенерованих зображень (GIC) та модуля коригування стисненого зображення (CIA). Модуль IIG генерує зображення, яке підходить для стиснення, при цьому генерування та відновлення зображень моделюються як взаємно зворотні процеси, щоб уникнути втрати інформації. Після цього модуль GIC стискає згенеровані зображення для збереження бітових швидкостей кодування, а модуль CIA зменшує різницю в якості між стисненим згенерованим зображенням та оригінальним. На завершення зображення з модуля CIA відправляється назад до модуля IIG для відновлення оригінального зображення. Експериментальні результати на трьох різних наборах даних показують, що запропонована структура досягає найкращих результатів у стисненні зображень при наднизьких бітових швидкостях. Крім того, запропоновану структуру було розширено для стиснення ознак при виявленні об'єктів, що дозволяє зекономити 90% бітових швидкостей порівняно з стандартом VVC при збереженні тієї ж точності виявлення.

У статті [7] розглядаються різні методи виявлення дефектів на друкованих платах (ДП), які є майже універсальними компонентами всіх видів електронних пристроїв. Зі швидким розвитком інтегральних схем і напівпровідникових технологій розміри ДП можуть значно зменшуватися, що

вимагає досягнення високої точності та швидкості виявлення дефектів. Робота є оглядом різних методів виявлення дефектів на ДП, в якому проаналізовано більше 100 статей, опублікованих з 1990 по 2022 рік. Спочатку розглядається методологія підготовки цього огляду методів виявлення дефектів на ДП. Далі коротко обговорюються методи ручного виявлення дефектів, після чого детально аналізуються традиційні методи на основі обробки зображень, методи машинного навчання та глибокого навчання для виявлення дефектів. У статті пояснюються та порівнюються їхні алгоритми, процедури, продуктивність, переваги та обмеження. Додаткові огляди в роботі надають глибші погляди, які допоможуть дослідникам зрозуміти сучасні тенденції досліджень і виконувати подальшу роботу, пов’язану з виявленням дефектів.

У статті [8] пропонується підхід, що поєднує низькорангову апроксимацію та тензор на основі другого порядку для суперрозв’язку і денойзінгу зображень оптичної когерентної томографії (ОКТ) сітківки, ефективно використовуючи не локальні просторові кореляції та локальні властивості гладкості. Оскільки зображення ОКТ отримуються за допомогою інтерферометрії, вони часто мають високий рівень шуму. Крім того, під час отримання А-сканування та В-сканування зображень ОКТ здійснюється субдискретизація даних, що робить необхідним використання ефективних алгоритмів суперрозв’язку для відновлення зображень високої роздільної здатності. У статті пропонується підхід регуляризації низького рангу для використання нелокальної самоподібності перед реконструкцією зображень ОКТ. Для використання переваг надмірності мультислайсних даних ОКТ будується тензор третього порядку, що утворюється шляхом виділення нелокальних подібних тривимірних блоків та їх групування за допомогою методу k-ближчих сусідів. Потім нуклеарна норма використовується як регуляризаційний термін для зменшення сингулярних значень побудованого тензора в напрямку нелокальної кореляції. Додатково пропонуються регуляризаційні підходи для тензора першого порядку (FOTTV) та другого

порядку (SOTTV) для кращого збереження шарів сітківки та подавлення артефактів в зображеннях ОКТ. Техніка методу альтернативних напрямків множників (ADMM) використовується для розв’язання отриманої задачі оптимізації. Експериментальні результати показують, що інтеграція SOTTV замість FOTTV в модель низькорангової апроксимації дозволяє досягти значно кращих результатів. Наші результати на денойзінгу та суперрозв’язку зображень ОКТ демонструють, що запропонована модель забезпечує зображення з вищими числовими та візуальними характеристиками порівняно з результатами, отриманими за допомогою передових методів.

У статті [9] запропоновано новий метод шифрування зображень, який використовує два хаотичних псевдонабори. Для покращення процесу хаотичної перестановки багаторазового зменшення (CPMCS) введено поступове видалення вхідного набору (GDIS), що дозволяє дифузувати пікселі зображення та контролювати відстань зсуву при обертаннях рядків та стовпців зображення. Запропонована схема шифрування має кілька переваг: вона забезпечує більший простір ключів, ніж інші методи шифрування зображень, і вимагає меншого часу обробки порівняно з CPMCS. Час обробки запропонованого методу шифрування зображень за допомогою алгоритму GDIS CPMCS у 16,7 разів швидший для чорно-білого зображення і в 43 рази швидший для кольорового зображення порівняно з попередньою роботою. За результатами гістограмного та ентропійного аналізів запропонована схема є стійкою до статистичного аналізу. Крім того, за результатами тестів випадковості на основі стандартів Національного інституту стандартів і технологій (NIST), зашифроване зображення має дуже високий рівень випадковості. У диференційному аналізі зміна одного біта в оригінальному зображенні призводить до значної зміни зашифрованого зображення, що підтверджується показниками середнього коефіцієнта зміни (UACI) та кількості змін пікселів (NPCR) – 33,45% та 99,61% відповідно. Крім того, GDIS CPMCS є стійким до різних типів шуму, таких як сіль і перець, Пуасонів шум,

Гауссов шум і шум спекл, з коефіцієнтами пікових співвідношень сигнал-шум (PSNR) вище за 14. Схема також стійка до втрати даних, оскільки відновлене зображення з втратою 50% даних все ще можна розпізнати, що підтверджується PSNR 11,4.

У статті [10] розглядається обробка п’яти зразків рису з попередньо вареним методом (парбулованим) та трьох зразків без попереднього варіння (непарбулованим) через етапи замочування, варіння та подальшого охолодження, що є частинами гідротермальної обробки для підготовки рису до споживання. Виміряно зміни розмірів під час таких процесів і порівняно для парбулованих та непарбулованих зразків. Було виявлено, що на етапі варіння темп розширення більший, ніж на етапі замочування, для обох типів рису, але желатинізація для непарбулованого рису відбувається швидше, ніж для парбулованого. На етапі охолодження зерна зменшуються до попередніх розмірів. Для двох типів рисових зразків було обчислено два параметри – аспектне співвідношення та форму, а отримані значення були побудовані залежно від часу для порівняння та визначення вигляду зерна, що є важливим показником якості при споживанні як їжі. Всі вимірювання проводились за допомогою аналізу зображень.

У статті [11] запропоновано алгоритм підвищення якості зображень при слабкому освітленні, який базується на не підвищеному перетворенні шерлетів (NSST). Алгоритм дозволяє одночасно досягати покращення контрасту, придушення шуму та підсилення певних напрямлених деталей. Спочатку розробляється структура покращення зображень з низьким рівнем освітлення, після чого зображення розкладається на коефіцієнти низькочастотних підсмуг і напрямлених підсмуг у перетворенні NSST. Потім, у домені NSST, оцінюється карта освітленості на основі яскравого каналу низькочастотних підсмуг, і одночасно придушується шум шляхом зменшення коефіцієнтів напрямлених підсмуг. Нарешті, на основі оціненої карти освітленості, низькочастотних підсмуг та зменшених коефіцієнтів

напрямлених підсмуг проводиться зворотне перетворення NSST для досягнення покращення зображення при слабкому освітленні. Експерименти показують, що LIEST демонструє кращі результати в порівнянні з сімома подібними алгоритмами щодо покращення контрасту, придушення шуму та виділення специфічних деталей.

У статті [12] розглядаються методи стиснення зображень, які використовуються для зменшення витрат на зберігання чи передачу цифрових зображень. Основною технікою, що застосовується для стиснення зображень, є швидке перетворення Фур'є (FFT), яке широко використовується для цього завдання. Цей алгоритм обчислює дискретне перетворення Фур'є (DFT) або його зворотне (IDFT). Завдяки аналізу Фур'є можна змінювати представлення сигналу з часової чи просторової області в частотну, і навпаки. Одним з підходів до стиснення зображень є видалення високочастотних компонентів в частотній області, що дозволяє зменшити обсяг даних. Потім стиснуте зображення можна відновити, застосувавши зворотне перетворення Фур'є. У статті також наводиться огляд літератури з теми стиснення зображень за допомогою швидкого перетворення Фур'є, в якому обговорюються основи перетворення Фур'є та його швидкої версії, а також методи стиснення зображень, що базуються на FFT, такі як JPEG, JPEG2000 та SPIHT. Висновки статті містять обговорення переваг та недоліків використання FFT для стиснення зображень.

У статті [13] пропонується підхід до стиснення медичних зображень, що враховує швидке зростання їх обсягу та потребу в ефективному зберіганні та управлінні ними. Для цього розглядаються методи стиснення, що використовують перетворення в частотній області, зокрема Дискретне Косинусне Перетворення (DCT) та Дискретне Вейвлетне Перетворення (DWT), які мають добре локалізовані коефіцієнти в просторі та частотній області. Оскільки медичні зображення є критично важливими для діагностики, для їх зберігання необхідне безвтратне стиснення. Проте коефіцієнти DWT є

дійсними числами, що ускладнює досягнення безвтратного стиснення. Для подолання цієї проблеми в запропонованій системі використовується Ліфтингова Вейвлетна Трансформація (LWT), яка дозволяє досягти безвтратного стиснення без втрати важливої інформації в медичних зображеннях.

У статті [14] запропоновано метод поліпшення ефективності стиснення та декомпресії зображень на основі генетичних алгоритмів, який спрямований на покращення якості стиснення зображень і зменшення часу та обчислювальної складності декомпресії. Спочатку розглядаються фон та поточний стан технологій стиснення зображень, а також аналізується розвиток існуючих досліджень у цій галузі. Далі детально вводяться основи генетичного алгоритму та його застосування в стисненні зображень. На основі цих знань розробляється алгоритм стиснення зображень, що покращує параметри стиснення через генетичний алгоритм для досягнення більш ефективного стиснення. Одночасно з процесом стиснення вдосконалюється алгоритм декомпресії, щоб зменшити витрати обчислювальних ресурсів. Це дослідження пропонує нові ідеї та методи для покращення технології стиснення та декомпресії зображень, що має важливе теоретичне та практичне значення.

У статті [15] порівнюються два відомі алгоритми безвтратного стиснення зображень: безвтратне перетворення дискретного косинуса (DCT) та безвтратне перетворення вейвлетів Хаара (HWT), з метою визначення ефективності стиснення медичних зображень. Дослідження охоплює три різні набори даних, що містять різні типи медичних зображень, такі як МРТ, КТ та ендоскопічні зображення шлунково-кишкового тракту, з різними форматами зображень: PNG, JPG та TIF. За результатами експериментів, у термінах розміру стисненого зображення та коефіцієнта стиснення, DCT показує кращі результати для форматів PNG та TIF, які відповідають КТ-чорно-білим і МРТ-колірним зображенням. Для формату JPG, що представляє кольорові

зображення ендоскопії, DCT дає хороші результати для чорно-білих зображень, тоді як HWT перевершує DCT для кольорових зображень. Проте, HWT перевершує DCT за часом стиснення для всіх типів та форматів зображень.

Стаття [16] обговорює проблему ефективного стиснення колекцій зображень у зв’язку з вибуховим зростанням цифрових фотографій, що створює труднощі в зберіганні та передачі даних. Автор пропонує підходи до безвтратного стиснення колекцій майже дубльованих зображень, що включають кілька етапів, кожен з яких є обчислювально вимогливим. Основним завданням є прискорення цих процесів без втрати ефективності стиснення. Для цього запропоновані методи ефективної кластеризації, швидкий алгоритм оцінки орієнтованого на напрямок руху та схема перерозподілу зображень з мінімальними витратами на прогнозування для покращення стиснення. Попередні результати запропонованого підходу обіцяють хороші перспективи, і в подальшому планується розширення цього підходу для стиснення гіперспектральних та медичних зображень.

У статті [17] розглядається проблема стиснення об’ємних зображень, що є важливим завданням для ефективної передачі та зберігання зображень, отриманих у біологічних дослідженнях та клінічній практиці. Найбільш поширені методи стиснення об’ємних зображень, такі як JP3D, використовують перетворення вейвлетів, але вони обмежені використанням ідеальної, сепарабельної та фіксованої вейвлет-бази, що знижує їх ефективність. У цій роботі запропоновано нове перетворення на основі тренованих 3D-подібних вейвлетів, яке дозволяє адаптувати перетворення до залежності від сигналу та використовувати нестандартні вейвлети для кращого виявлення локальних кореляцій у різних областях об’ємних зображень. Також введена афінна вейвлет-база для захоплення різних локальних кореляцій, і запропоновано метод адаптивного стиснення на основі цієї нової трансформації в рамках фреймворка aiWave. Крім того, для зменшення

кількості параметрів, було запропоновано стратегії спільного використання ваг для афінних вейвлетів відповідно до характеристик об’ємних даних. Результати експериментів показують, що запропонований метод aiWave перевершує JP3D в плані ефективності стиснення при схожих складностях кодування та декодування, а також досягає значно кращих результатів у порівнянні з HEVC при використанні додаткових контекстних модулів для зменшення надмірності сигналу.

У статті [18] запропоновано новий алгоритм стиснення кольорових зображень, який поєднує зменшення бітових площин та сингулярне розкладання матриць (SVD). Метою цього методу є досягнення стиснення зображень з мінімальними втратами при збереженні високої якості. Алгоритм починається з декомпозиції кольорового зображення на три основні кольорові компоненти: червоний, зелений і синій. Після цього ці компоненти об’єднуються в матриці, на яких застосовується зменшення бітових площин. Після цього до оброблених матриць застосовується сингулярне розкладання для покращення показників якості зображення, зокрема для підвищення значень PSNR. У статті представлено два методи: один на основі реального SVD, а інший – на основі квартерніонного SVD. Результати обох методів порівнюються за різними показниками, такими як PSNR, SSIM, MSE та CR.

У статті [19] обговорюється новий гібридний метод безвтратного стиснення зображень, який передбачає використання двох різних підходів для кодування кожної половини зображення. Метод базується на блочному підході: для однієї половини зображення застосовується нове перетворення, що використовує таблицю перетворень для відображення кожного пікселя в однобітні значення, що дозволяє згенерувати більше нулів і досягти стиснення. Після цього застосовується зворотне перетворення для відновлення пікселів без втрат. Для іншої половини зображення використовують диференціальне кодування блоків, що також забезпечує безвтратне стиснення.

Обидва підходи разом дозволяють досягти ефективного безвтратного стиснення зображень.

У статті [20] розглядається новий алгоритм стиснення зображень для покращення ефективності передачі зображень з кораблів, що зумовлено обмеженнями морської комунікаційної технології, зокрема низькою пропускною здатністю, високими витратами на передачу та нестабільністю з'єднання. Запропонований метод стиснення використовує алгоритм ядрового головного компонентного аналізу (HSV-KPCA), який дозволяє ефективно зменшити обсяг даних зображення без значних втрат у якості. Алгоритм спочатку трансформує зображення у високорозмірний простір за допомогою KPCA, що дозволяє виділити основні нелінійні ознаки. Потім, з використанням порогового значення кумулятивної частки внеску, зберігаються лише основні компоненти, що містять важливу інформацію, а інші, менш значущі частини, відкидаються для зменшення обсягу даних. Одержане зображення реконструюється на основі збережених векторів ознак, після чого проводиться перехід від RGB-простору до HSV-простору для покращення компоненту яскравості зображення. Після цього зображення знову перетворюється у RGB-простір для отримання стисненого зображення. Експериментальні результати показали, що запропонований алгоритм HSV-KPCA має кращі показники стиснення у порівнянні з популярними методами JPEG та PCA, зокрема за такими критеріями, як коефіцієнт стиснення, середньоквадратична помилка, пікове співвідношення сигнал-шум та структурна схожість.

1.6 Висновки до першого розділу

Виходячи з проведеного аналізу предметної області, доцільно представити наступні висновки до першого розділу:

1) визначення основних типів методів стиснення. У розділі чітко розмежовуються два основні типи методів стиснення – безвтратні та з втратами. Це дозволяє зрозуміти основні принципи, на яких ґрунтуються різні підходи до стиснення даних, а також виявити їхні сильні та слабкі сторони. Безвтратні методи зберігають оригінальні дані без втрат, що важливо для певних типів інформації (текст, програми, деякі зображення), тоді як методи стиснення з втратами забезпечують високе стиснення за рахунок втрати частини даних, що допустимо для деяких медіафайлів, таких як зображення, відео та аудіо;

2) розгляд класичних методів стиснення. Детальний аналіз класичних методів стиснення, таких як кодування Хаффмана, RLE та LZW, надає уявлення про основні техніки, що застосовуються в безвтратному стисненні. Ці методи використовують різні стратегії для досягнення зменшення обсягу даних, зокрема через використання статистичних властивостей даних (наприклад, частоти символів) і ефективне кодування повторюваних або схожих блоків даних;

3) алгоритми стиснення з втратами. Вивчення методів стиснення з втратами, таких як JPEG, JPEG2000 і WebP, допомагає з’ясувати, як ці методи використовують специфічні математичні та алгоритмічні підходи для досягнення високого рівня стиснення. Однак вони також мають свої обмеження, зокрема у вигляді втрати якості даних при сильному стисненні. Розгляд цих алгоритмів дозволяє зрозуміти, як можна зберегти високу якість зображень при значному зменшенні їх розміру, що важливо для таких застосувань, як веб-дизайн або зберігання мультимедійних файлів;

4) методи стиснення на основі перетворень. Розгляд методів стиснення, які використовують перетворення, таких як DCT та Wavelet, вказує на важливість математичних моделей для досягнення ефективного стиснення даних, зокрема для зображень і відео. Ці методи дозволяють більш ефективно

обробляти частини даних, які менш помітні для людського сприйняття, таким чином зменшуючи обсяг даних без суттєвих втрат у якості;

5) комплексність і гнучкість методів стиснення. В результаті аналізу різних методів стиснення можна зробити висновок, що кожен з них має свої переваги і недоліки в залежності від типу даних, з якими працює система, та вимог до якості і розміру файлу. Це дає підстави для подальших досліджень та розвитку нових підходів, які дозволяють поєднувати різні методи для досягнення оптимальних результатів.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ СУЧАСНИХ МЕТОДІВ СТИСНЕННЯ ТА ОБРОБКИ ЗОБРАЖЕНЬ

2.1 Основи обробки зображень

Техніки обробки зображень [10, 11] представлено в таблиці 2.1.

Таблиця 2.1

Техніки обробки зображень

Техніка	Опис	Приклад використання
Фільтрація	Фільтрація зображень – це процес згладжування або різкості зображення для видалення шуму або розмиття деталей.	Фільтрація для зменшення шуму, наприклад, за допомогою середнього фільтру чи гаусівського фільтру.
Покращення контрасту	Покращення контрасту – це збільшення різниці між яскравими та темними областями зображення, що підвищує його візуальну чіткість.	Застосування гістограмної рівномірної нормалізації або корекції гамма-значень.
Корекція кольору	Корекція кольору використовується для коригування кольорової балансу, насиченості або відтінків зображення.	Застосування корекції балансу білого або коригування відтінку та насиченості для виправлення кольору.

2.2 Алгоритми відновлення зображень

Алгоритми відновлення зображень (рис. 2.1) є важливою частиною комп’ютерної графіки та обробки зображень, адже вони дозволяють відновлювати зображення, що зазнали спотворень або пошкоджень [11]. Пошкодження можуть бути викликані різними факторами, такими як шум,

часткова втрата даних або погіршення якості зображення під час зберігання чи передачі. Відновлення зображень є складним завданням, яке вимагає використання різноманітних математичних методів, алгоритмів і підходів для досягнення максимальної точності та реалістичності відновлених зображень. У цьому контексті важливими є два основних напрямки: видалення шумів і відновлення з часткових або пошкоджених даних.

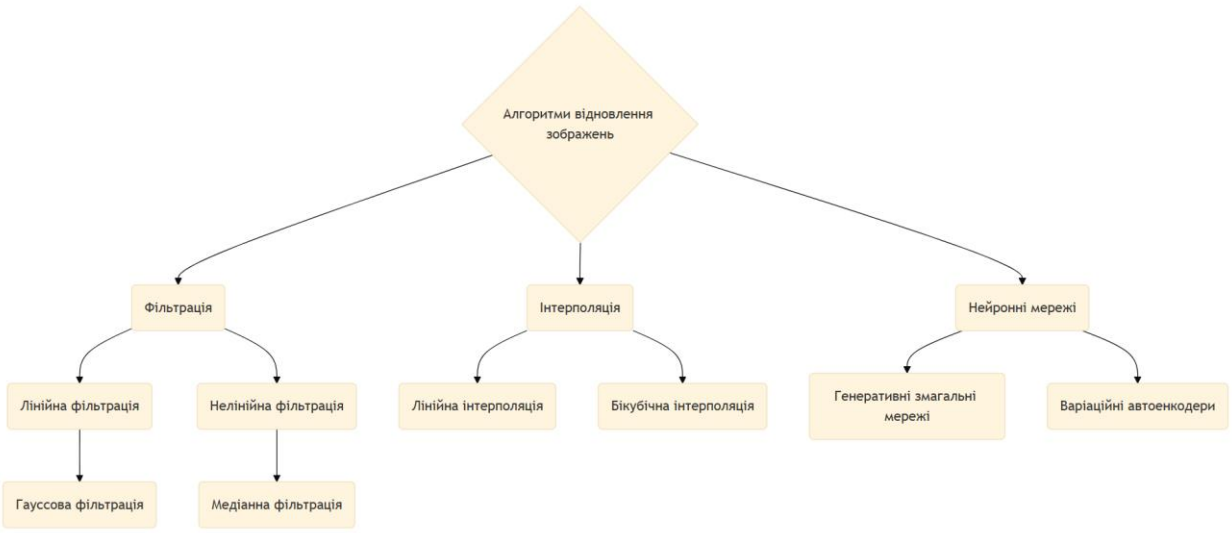

Рисунок 2.1 – Алгоритми відновлення зображень

Шум є одним з найпоширеніших типів спотворень, з якими стикаються при обробці зображень. Він може з’являтися через різноманітні причини, такі як помилки при зйомці зображень, перешкоди в передачі даних або технічні обмеження в сенсорах зображення. Шум може бути різного типу, серед яких найпоширенішими є гаусівський, сольовий і перцевий шум, а також імпульсні шуми, що проявляються у вигляді окремих пікселів, значення яких значно відрізняються від нормальних.

Для видалення шумів існує кілька підходів, що ґрунтуються на математичних моделях. Один з найбільш відомих і широко застосовуваних методів – це фільтрація. Основним принципом фільтрації є використання

математичних фільтрів для згладжування або видалення шуму з зображення. Існують різні типи фільтрів, зокрема лінійні та нелінійні.

Лінійні фільтри, наприклад, фільтри Гауса, використовують зважене середнє значення сусідніх пікселів для заміни значення пікселя [12]. Це дозволяє згладжувати зображення і зменшувати ефект шуму, однак для сильного шуму, особливо імпульсного, цей метод може бути неефективним, оскільки він розмиває деталі зображення. Нелінійні фільтри, такі як медіанний фільтр, виявляються більш ефективними при роботі з імпульсним шумом, оскільки вони замінюють піксель на медіану значень його сусідів, що дозволяє зберігати структуру зображення і при цьому ефективно усувати шум.

Ще одним ефективним методом видалення шуму є використання вейвлет-перетворення. Вейвлети дозволяють розкласти зображення на різні частоти, що дозволяє виділяти низькочастотні компоненти, які є основними для відновлення основної структури зображення, і фільтрувати високочастотні компоненти, що зазвичай відповідають за шум. Це дає змогу ефективно зберігати важливі деталі зображення при зменшенні шуму.

Крім того, для видалення шуму часто застосовують методи машинного навчання та глибокого навчання, які можуть бути більш адаптивними і точними. Наприклад, нейронні мережі, спеціально натреновані на великих наборах даних зображень, здатні вивчати закономірності між шумом і справжніми пікселями, що дозволяє значно покращити результат відновлення зображення. Ці методи включають використання автоенкодерів або сверточних нейронних мереж, які демонструють відмінні результати на різних типах шуму.

Другим важливим завданням є відновлення зображень з частковими або пошкодженими даними. Така ситуація виникає, коли частина зображення стає недоступною через різні фактори, наприклад, через фізичне пошкодження носія інформації або втрату частини даних під час зберігання або передачі.

Відновлення таких зображень є складним завданням, яке потребує застосування більш складних методів інтерполяції та відновлення [12, 13].

Один з основних підходів до відновлення зображень з пошкодженими даними – це методи інтерполяції. Інтерполяція полягає в тому, щоб відновити значення пікселів, яких не вистачає, на основі сусідніх пікселів. Існує кілька видів інтерполяції, включаючи лінійну, біквадратичну та бікубічну. Лінійна інтерполяція проста і швидка, але вона може давати не зовсім точні результати, особливо при значних втратах даних. Бікубічна інтерполяція дозволяє досягти більш точних результатів за рахунок використання більшої кількості сусідніх пікселів для відновлення відсутніх значень.

Іншою технікою, що використовується для відновлення частково пошкоджених зображень, є методи на основі фреймів і патернів. Вони передбачають побудову математичних моделей, що відображають взаємозв’язки між пікселями та їх структурами в зображенні. Одним з таких методів є використання частотного аналізу, де зображення розкладається на частоти, і відновлення зображення відбувається шляхом заповнення відсутніх частот.

Більш складні методи включають застосування методів багатовимірної статистики, таких як матричні факторизації або низькорівнева аппроксимація, які намагаються відновити відсутні дані на основі статистичних закономірностей у зображеннях. Ці методи вимагають використання значної кількості даних для навчання моделей і є набагато більш ресурсозатратними, але вони можуть забезпечити високу якість відновлення.

Один з найбільш потужних підходів до відновлення пошкоджених зображень – це використання глибокого навчання. Нейронні мережі, зокрема генеративні змагальні мережі (GANs) та варіаційні автоенкодери (VAEs), здатні навчатися на великих наборах даних і можуть генерувати відсутні частини зображення на основі статистичних закономірностей, знайдених у тренувальних даних. Генеративні моделі використовують дві мережі –

генератор та дискримінатор, де генератор створює відновлені частини зображення, а дискримінатор перевіряє їх якість. Це дозволяє отримати дуже реалістичні результати при відновленні частково пошкоджених або втрачених даних.

Методи відновлення зображень, засновані на глибоких нейронних мережах, також використовують трансформери та інші передові методи, що дозволяють працювати з великими зображеннями та точніше відновлювати їх навіть в умовах серйозних пошкоджень [13, 14].

Алгоритми відновлення зображень є ключовими для покращення якості зображень та відновлення їх після різних пошкоджень. Процес відновлення може включати як видалення шуму, так і відновлення пошкоджених частин зображення. Застосування методів фільтрації, інтерполяції, статистичних моделей та глибокого навчання дозволяє досягти значних успіхів у відновленні зображень при різних типах пошкоджень. Завдяки розвитку технологій машинного навчання та нейронних мереж, які можуть адаптуватися до різних умов і навчатися на великих обсягах даних, сучасні методи відновлення зображень демонструють високу ефективність та точність у порівнянні з традиційними підходами.

2.3 Методи детекції контурів та меж

Методи детекції контурів та меж зображень дозволяють виділити основні структури і об’єкти, що є візуально важливими, що значно полегшує подальші етапи аналізу зображень, такі як сегментація, розпізнавання об’єктів та класифікація. Детермінація меж об’єктів дозволяє підвищити точність розпізнавання та виконувати більш складні завдання, наприклад, реконструкцію тривимірних об’єктів за допомогою 2D зображень. Існують різноманітні підходи та алгоритми для детекції контурів, серед яких методи Собеля, Кенні та Лапласа Гауса є одними з найбільш популярних і ефективних.

Кожен з них має свої особливості та застосовується в залежності від типу зображень, а також від поставленої задачі [15].

Метод Собеля є класичним методом, який використовує фільтрацію зображення для виявлення контурів. Цей метод зазвичай застосовується для виявлення градієнтів у зображенні, що дозволяє виявити різкі зміни інтенсивності, які відповідають за контури об’єктів. Основою методу є використання двох фільтрів для обчислення градієнтів зображення: один для горизонтальних змін (фільтр для X-осі), а інший – для вертикальних змін (фільтр для Y-осі). Операція з фільтрацією дозволяє виявити напрямок і величину зміни інтенсивності пікселів. Фільтри Собеля є симетричними і мають розмір 3x3, що дозволяє їм захоплювати як локальні, так і глобальні зміни в зображенні, одночасно зберігаючи оброблену картину досить чіткою і без сильних спотворень. Результатом застосування фільтра Собеля є зображення, яке відображає контури в напрямках вертикальних і горизонтальних ліній. Процес полягає у двох етапах: спочатку обчислюються градієнти по обох осях, потім комбінуються результати для отримання загальної величини градієнта та його напрямку. Хоча метод Собеля є простим і швидким, він може бути недостатньо точним для виявлення тонких контурів або в разі сильного шуму, оскільки його фільтри можуть не враховувати всі зміни інтенсивності в зображенні [16].

Метод Кенні є більш складним і вдосконаленим варіантом детекції контурів, який також базується на аналізі градієнтів, але додає етапи, що покращують точність виявлення. Цей метод є одним з найбільш популярних для виявлення контурів через його здатність ефективно обробляти зображення з різними рівнями шуму та забезпечувати точніші результати в порівнянні з іншими методами. Алгоритм Кенні включає декілька основних етапів: фільтрацію для виявлення градієнтів зображення, визначення напрямку і величини градієнтів, здійснення процесу пригладжування, використовуючи техніку порогового значення для усунення слабких контурів, а також

застосування подвійного порогового значення для подальшого відбору значущих контурів. Перший етап методу передбачає використання фільтру, подібного до фільтру Собеля, для отримання вертикальних і горизонтальних градієнтів. Після цього проводиться процес non-maximum suppression, в результаті якого зберігаються тільки ті пікселі, які є найбільш значущими для контурів, тобто пікселі, у яких величина градієнта є більшою, ніж у сусідніх. На наступному етапі алгоритм застосовує подвійне порогове значення для виділення сильних та слабких контурів. Пікселі, що мають значення градієнта вище за верхній поріг, вважаються сильними контурами, тоді як пікселі між верхнім і нижнім порогами можуть бути частково контурними. В кінці відбираються тільки ті слабкі контури, які мають зв’язок з сильними контурами, що дозволяє забезпечити більш точні і стійкі результати. Перевагою методу Кенні є його здатність ефективно виділяти тонкі контури і зменшувати кількість помилкових контурів при наявності шуму, що робить його одним з найкращих виборів для застосувань, де необхідна висока точність.

Метод Лапласа Гауса (LoG) є іншим ефективним підходом до детекції контурів, який ґрунтується на комбінуванні операцій згладжування зображення та застосування оператора Лапласа. Оператор Лапласа є другим похідним від функції, що дозволяє визначити точки, де змінюється кривизна інтенсивності зображення, а це саме ті точки, які часто відповідають за контури. Однак, оскільки оператор Лапласа сам по собі чутливий до шуму, його зазвичай комбінують з фільтром Гауса, щоб згладити зображення перед застосуванням оператора [17]. Цей комбінований підхід дозволяє отримати результат, який зберігає контури зображення, одночасно зменшуючи вплив шуму на кінцевий результат. Процес обчислення контурів за допомогою Лапласа Гауса зазвичай складається з кількох етапів. Спочатку зображення згладжується за допомогою фільтра Гауса, що дозволяє усунути високочастотні компоненти і зменшити ефект шуму. Потім застосовується

оператор Лапласа, який знаходить точки, де інтенсивність зображення змінюється найбільш різко, що відповідає за контури. У результаті зображення мають дуже чіткі контури, при цьому шум зменшується завдяки попередньому згладжуванню. Однак метод Лапласа Гауса може бути менш точним для виявлення контурів при слабкому контрасті між об’єктами та фоном, оскільки він чутливий до зміни інтенсивності і може пропускати деякі тонкі контури, особливо на зображеннях з низьким контрастом. Крім того, при налаштуванні фільтра Гауса важливо знайти оптимальне значення для розміру фільтра, оскільки занадто великі значення можуть привести до втрати дрібних деталей, а занадто малі – до недостатнього згладжування шуму.

Кожен з розглянутих методів має свої переваги та недоліки, і їх вибір залежить від конкретних умов задачі. Метод Собеля є простим і ефективним для швидкої детекції контурів, особливо на зображеннях з високим контрастом, але він може бути менш точним на зображеннях з великим рівнем шуму або тонкими деталями [18]. Метод Кенні, завдяки своїй здатності до оптимізації контурів і усунення слабких контурів, є більш точним і надійним у складних умовах, коли зображення містить шум або має невеликі деталі, що необхідно виявити. Метод Лапласа Гауса, в свою чергу, забезпечує чудове згладжування перед виявленням контурів і є ефективним при роботі з висококонтрастними зображеннями, однак його чутливість до змін інтенсивності може бути обмеженням у випадках, коли контрасти на зображеннях є мізерними.

У підсумку, вибір методу для детекції контурів залежить від конкретних умов задачі, таких як тип зображення, рівень шуму, контрастність і необхідна точність виявлення контурів. Кожен з методів – Собель, Кенні і Лаплас Гауса – має своє застосування і може бути адаптований для досягнення оптимальних результатів при вирішенні різноманітних задач обробки зображень.

2.4 Алгоритми підвищення якості зображень після стиснення

Алгоритми підвищення якості зображень після стиснення є важливою галуззю обробки цифрових зображень, що спрямована на поліпшення візуальної якості зображень, які зазнали втрат через застосування методів стиснення. Стиснення зображень є критично важливим у сучасному світі, де необхідно оптимізувати використання дискового простору і швидкість передачі даних. Однак стиснення часто супроводжується втратами інформації, що може призводити до значного погіршення якості зображень, особливо при високому рівні стиснення. Алгоритми підвищення якості зображень після стиснення розроблені для того, щоб мінімізувати ці негативні ефекти і відновити важливі деталі, які були втрачені через компресію. Ці методи варіюються від традиційних підходів, що ґрунтуються на статистичних моделях і фільтрації, до сучасних методів, заснованих на використанні глибокого навчання і нейронних мережах [19].

Процес стиснення зображень включає перетворення зображення у формат, що займає менший обсяг пам’яті, зазвичай через усунення надлишкової або невидимої для людського ока інформації. Один з найбільш поширених методів стиснення – це методи стиснення з втратами, такі як JPEG, які зменшують розмір файлу за рахунок усунення деталей, які вважаються неістотними для візуального сприйняття. Однак цей процес веде до втрати деяких характеристик зображення, таких як різкість, текстури та кольорові градації, що в свою чергу призводить до погіршення якості після відновлення. Після стиснення зображення часто можна спостерігати артефакти, такі як блоки, шуми, розмиття та інші дефекти, які знижують його візуальну привабливість. Алгоритми, призначені для підвищення якості після стиснення, фокусуються на відновленні втрачених деталей та усуненні цих дефектів.

Одним із класичних підходів до підвищення якості зображень після стиснення є застосування фільтрації для згладжування артефактів. Суть цього

методу полягає в тому, щоб застосувати до зображення фільтри, які дозволяють видалити шум і усунути блокові артефакти, характерні для стиснення. Наприклад, один з таких фільтрів – це медіанний фільтр, який замінює піксель на медіану значень пікселів його сусідів. Медіанний фільтр є ефективним для усунення імпульсного шуму, який може виникати після стиснення, і дозволяє зберегти основні контури та структуру зображення. Проте він може бути менш ефективним при відновленні більш складних деталей зображення або при дуже сильному стисненні.

Іншим методом є фільтрація Гауса, яка згладжує зображення шляхом застосування фільтра Гауса для кожного пікселя, зважуючи його сусіди відповідно до їх відстані. Фільтр Гауса добре підходить для усунення шумів, оскільки він м’яко згладжує зображення, не знижуючи значною мірою його деталізацію. Проте цей метод також не здатний повністю відновити деталі, що були втрачені під час стиснення, і може розмити тонкі структури. Як наслідок, фільтрація Гауса використовується здебільшого в поєднанні з іншими методами, щоб досягти кращих результатів [20].

Одним з більш складних підходів є використання багатовимірних статистичних моделей, таких як методи машинного навчання, для підвищення якості зображень. Ці методи орієнтовані на побудову моделей, які вивчають статистичні залежності між пікселями в зображеннях і можуть передбачати значення втрачених або пошкоджених пікселів після стиснення. Такі методи, як регресія, методи головних компонент та інші статистичні техніки, дозволяють відновити деякі втрачені деталі і підвищити контрастність зображення, що робить його більш чітким і деталізованим.

З розвитком машинного навчання та нейронних мереж з’явилися більш просунуті підходи до підвищення якості зображень після стиснення. Особливо варто відзначити використання глибоких нейронних мереж, таких як автоенкодери або генеративні змагальні мережі (GAN), для відновлення якості зображень після стиснення. Автоенкодери – це нейронні мережі, які

навчаються стискати і відновлювати зображення, зберігаючи основні риси зображення при мінімізації інформаційних втрат. Вони складаються з двох основних частин: енкодера, який стискає вхідні дані в компактне представлення, і декодера, який відновлює зображення з цього компактного коду. При застосуванні автоенкодерів до відновлення зображень після стиснення, енкодер стискає зображення до низькорозмірного простору, що дозволяє виділити найважливіші ознаки, а декодер відновлює зображення, мінімізуючи артефакти та втрати.

Генеративні змагальні мережі (GAN), в свою чергу, є потужним інструментом для відновлення зображень. GAN складаються з двох нейронних мереж: генератора і дискримінатора. Генератор створює нові зображення або відновлює пошкоджені частини, а дискримінатор намагається визначити, чи є ці зображення реалістичними [7, 18]. Під час тренування генератор навчається створювати зображення, що максимально наближені до оригіналу, а дискримінатор вчиться виявляти помилки. Така структура дозволяє досягти дуже реалістичних результатів, оскільки генератор отримує зворотний зв’язок від дискримінатора і може поступово покращувати якість відновлення. Використання GAN для підвищення якості зображень після стиснення дозволяє відновлювати навіть дуже дрібні деталі, що були втрачені під час стиснення, і мінімізувати такі артефакти, як блокові артефакти або розмиття.

Крім того, для підвищення якості зображень після стиснення використовуються також різноманітні методи суперрезолюції, які спрямовані на збільшення роздільної здатності зображення. Суперрезолюція може бути корисною при відновленні зображень, що були стиснуті до низької роздільної здатності. Вона включає використання алгоритмів, які можуть генерувати додаткові деталі, що дозволяє збільшити чіткість зображення. Сучасні підходи до суперрезолюції базуються на глибокому навчанні, де нейронні мережі навчаються відновлювати високоякісні деталі з низькорозмірних зображень.

Загалом, методи підвищення якості зображень після стиснення за допомогою глибокого навчання показують значні переваги порівняно з традиційними методами, такими як фільтрація та статистичні моделі, завдяки своїй здатності до адаптації та навчанню на великих наборах даних. Вони дозволяють значно покращити візуальну якість зображень після стиснення, усувати артефакти і відновлювати важливі деталі, які можуть бути втрачені при застосуванні традиційних методів стиснення з втратами.

Таким чином, алгоритми підвищення якості зображень після стиснення займають важливе місце в обробці зображень, оскільки вони дозволяють мінімізувати втрати інформації, покращити візуальну якість і підвищити чіткість. З розвитком методів глибокого навчання і нейронних мереж, ці алгоритми стають дедалі точнішими і ефективнішими, здатними обробляти складні зображення та досягати високих результатів навіть при сильному стисненні.

2.5 Використання штучного інтелекту та глибинного навчання в обробці зображень

Використання штучного інтелекту (ШІ) та глибинного навчання (ГН) в обробці зображень стало однією з найбільш революційних технологій останніх десятиліть. Зокрема, згорткові нейронні мережі (CNN, Convolutional Neural Networks) являють собою потужний інструмент для аналізу, обробки та інтерпретації зображень. CNN є основною архітектурою для глибинного навчання, яка дозволяє здійснювати складні операції, такі як класифікація зображень, детекція об’єктів, сегментація зображень і навіть генерація нових зображень [12, 13]. Штучний інтелект, заснований на CNN, дозволяє автоматично виявляти суттєві ознаки в зображеннях, що раніше вимагало значних зусиль від людини або традиційних алгоритмів обробки зображень. Враховуючи великий обсяг і складність сучасних візуальних даних,

використання ШІ та ГН в обробці зображень стало важливою складовою багатьох галузей, включаючи медицину, автомобільну промисловість, безпеку, робототехніку, сільське господарство та багато інших.

Згорткові нейронні мережі, або CNN, являють собою архітектуру, яка включає декілька ключових компонентів, що дозволяють ефективно обробляти візуальну інформацію. Одна з основних характеристик CNN полягає в тому, що вона використовує згорткові операції для аналізу зображень, що дає змогу ефективно виявляти просторові взаємозв'язки між пікселями. Це особливо важливо для обробки зображень, оскільки багато зображень мають високу просторову кореляцію, тобто пікселі в одній частині зображення часто пов'язані з пікселями в іншій його частині. CNN допомагає виявити ці зв'язки та витягти корисні ознаки, які можуть бути використані для класифікації, сегментації або інших завдань.

Архітектура згорткової нейронної мережі зазвичай складається з кількох основних шарів, кожен з яких виконує певну функцію в процесі обробки зображень. Перший шар, що є згортковим, здійснює операцію згортки з використанням фільтру, або ядра, яке проходить через зображення, аналізуючи локальні патерни і витягуючи ознаки, такі як краї, текстури, кольори та інші деталі. Ці фільтри, що складаються з набору параметрів, можуть бути навчені під час тренування мережі, що дозволяє мережі автоматично оптимізувати свої фільтри для виявлення найважливіших особливостей зображень.

Після операції згортки вихід з кожного фільтру передається на наступний шар, який є підсумовуючим (або пулінговим) шаром. Пулінг (зазвичай максимальний пулінг або середній пулінг) зменшує просторові розміри зображення, зберігаючи при цьому важливі ознаки, такі як наявність контурів чи текстур. Пулінг дозволяє зменшити розмір даних, що обробляються, що в свою чергу знижує вимоги до обчислювальних ресурсів і

запобігає перенавчанню моделі, дозволяючи мережі фокусуватися на важливих ознаках.

Далі на наступних шарах нейронної мережі здійснюється кілька операцій згортки та пулінгу, що дозволяє мережі поступово виділяти все більш складні і абстрактні ознаки зображення. Це може бути, наприклад, виявлення різних частин об’єктів, таких як очі та ніс на зображеннях осіб, або окремі елементи архітектурних структур на фотографіях будівель [17-19]. Чим більше шарів має мережа, тим більшу глибину абстракції вона здатна досягати, дозволяючи більш точно класифікувати або сегментувати зображення на основі складних патернів.

Після кількох згорткових та пулінгових шарів, нейронна мережа переходить до повнозв’язних шарів, які виконують остаточну класифікацію або регресію. Ці шари відповідають за прийняття рішення на основі всіх попередньо витягнутих ознак і зазвичай використовують функцію активації, таку як ReLU або сигмоїду, для визначення ймовірності того, що зображення належить до певного класу чи категорії. Для класифікаційних завдань, як правило, використовують функцію активації softmax на виході, що дозволяє отримати ймовірності для кожного можливого класу.

Однією з ключових переваг CNN є її здатність до автоматичного навчання ознак, що дозволяє уникнути необхідності вручну визначати та виділяти важливі характеристики зображень. Це значно знижує складність і дозволяє нейронним мережам працювати з набагато складнішими та різноманітнішими зображеннями, ніж це було б можливо за допомогою традиційних методів обробки зображень, які потребують ручної попередньої обробки або налаштування параметрів. Крім того, глибинне навчання дозволяє побудувати мережі, які можуть працювати з великими обсягами даних і поступово покращувати свої результати через тренування на численних прикладах.

CNN показали свою ефективність в ряді завдань обробки зображень, таких як класифікація, детекція об’єктів, сегментація зображень, відновлення зображень і навіть генерація нових зображень [20]. Наприклад, у задачі класифікації зображень CNN може бути використана для того, щоб автоматично визначити, чи містить зображення певний об’єкт, наприклад, кота чи собаку, або для класифікації медичних зображень на наявність патологій, таких як пухлини. Згорткові нейронні мережі можуть також використовуватися для задач детекції об’єктів, де мережа повинна виявити й локалізувати об’єкти на зображенні, а також для сегментації, де потрібно визначити межі об’єктів і сегменти зображення.

Застосування CNN в медичній діагностиці є однією з найбільш перспективних областей. Глибинні нейронні мережі можуть автоматично аналізувати медичні зображення, такі як рентгенівські знімки, МРТ або КТ-сканування, для виявлення різних аномалій. Це може включати виявлення пухлин, пошкоджень або інших патологій з високою точністю, що допомагає лікарям приймати швидкі та обґрунтовані рішення. Окрім цього, CNN активно використовуються в автоматизованих системах безпеки, таких як розпізнавання осіб, відеоаналітика та моніторинг, а також в автомобільних системах, зокрема в технології автономного водіння, де мережі допомагають аналізувати зображення навколишнього середовища і приймати рішення щодо руху автомобіля.

Для досягнення високих результатів у застосуванні CNN необхідно враховувати кілька важливих аспектів. По-перше, потрібно мати великий обсяг навчальних даних, що дозволяють мережі навчитися виділяти ознаки зображень. По-друге, налаштування архітектури мережі та гіперпараметрів, таких як кількість шарів, розмір фільтрів і кількість нейронів у кожному шарі, має велике значення для досягнення високої точності. По-третє, для запобігання перенавчанню (overfitting), коли модель запам’ятовує специфіку навчальних даних і не здатна узагальнювати на нових даних, застосовуються

різні техніки регуляризації, такі як Dropout, або використовується додаткове збільшення навчальних даних.

Загалом, застосування згорткових нейронних мереж у обробці зображень дозволяє значно покращити ефективність і точність розв’язання численних завдань, що стосуються аналізу та інтерпретації візуальної інформації. Ці технології, використовуючи потужні обчислювальні ресурси та глибокі мережі, дають змогу отримувати результати, які були б неможливими за допомогою традиційних методів обробки зображень, і стають основою для нових досягнень у багатьох важливих сферах діяльності.

2.6 Адаптація методів до нових форматів зображень

Адаптація методів обробки зображень до нових форматів, таких як 3D зображення та гіперспектральні зображення, є однією з найважливіших тем сучасної науки та технологій, що обумовлена необхідністю ефективного аналізу складних даних у різних галузях. З розвитком технологій та вдосконаленням методів збирання даних виникає потреба в ефективних підходах для обробки і аналізу зображень, що виходять за межі традиційних двовимірних зображень [18-20]. Зокрема, 3D зображення та гіперспектральні зображення є такими форматами, які мають значний потенціал для застосування в різноманітних областях, таких як медицина, дистанційне зондування, промисловість, а також в області наукових досліджень. Враховуючи їх складність, ці формати вимагають нових підходів та методів обробки, адаптованих до специфічних особливостей таких зображень.

3D зображення відрізняються від традиційних 2D зображень тим, що вони містять інформацію не лише про плоске розташування пікселів, а й про просторові відносини між об’єктами в трьох вимірах. Це надає зображенням значно більшої кількості інформації, що дозволяє створювати точніші моделі реальних об’єктів і середовищ. Одним з найбільш розповсюджених форматів

3D зображень є зображення, що описуються тривимірними вокселями (пікселями в трьох вимірах), подібно до того, як 2D зображення складаються з двовимірної решітки пікселів. Вокселі, що є аналогами пікселів у 3D просторі, зберігають інформацію не тільки про колір, але й про глибину, відстань, текстуру та інші параметри, що є важливими для відтворення об'ємних об'єктів;

Адаптація традиційних методів обробки зображень до 3D форматів потребує вирішення ряду складних завдань. Однією з основних проблем є необхідність врахування просторових взаємозв'язків між різними шарами в 3D зображеннях. Методи, що використовуються для обробки 2D зображень, такі як фільтрація, детекція контурів або сегментація, не можуть бути безпосередньо застосовані до 3D зображень без змін, оскільки вони не враховують багатовимірну структуру зображення. Тому для обробки 3D зображень необхідно використовувати більш складні методи, зокрема, згорткові нейронні мережі (CNN), які здатні працювати з тривимірними даними. Вони застосовуються для таких завдань, як автоматична сегментація органів на МРТ або КТ зображеннях, відновлення 3D моделей з серії 2D зображень, виявлення об'єктів у тривимірних просторах, а також для інших застосувань, де необхідна точна і швидка обробка складної просторової інформації.

Один з підходів до адаптації методів обробки зображень до 3D формату полягає в використанні 3D згорткових нейронних мереж (3D CNN), що дозволяють ефективно обробляти об'ємні дані. Основною ідеєю 3D CNN є застосування тривимірних фільтрів, які заміняють стандартні двовимірні фільтри зображень, застосовувані в традиційних CNN [11]. Ці фільтри здатні захоплювати просторові залежності не тільки в межах одного шару зображення, а й між різними шарами, що дозволяє більш точно виявляти структури та деталі у тривимірному просторі. Завдяки цьому 3D CNN можуть застосовуватися в різноманітних сферах, таких як медична діагностика, де

вони дозволяють здійснювати точну сегментацію органів і тканин, а також в таких областях, як робототехніка, де необхідно отримувати об’ємні карти навколишнього середовища для орієнтації роботів у просторі.

Гіперспектральні зображення є ще одним важливим типом зображень, що відрізняються від традиційних зображень тим, що містять значно більшу кількість спектральних каналів. Традиційні зображення зазвичай мають три канали (червоний, зелений і синій), але гіперспектральні зображення можуть мати сотні або навіть тисячі спектральних каналів, кожен з яких відповідає за певну частину електромагнітного спектра. Це дає змогу отримати набагато більше інформації про об’єкти на зображенні, наприклад, про їх хімічний склад, температуру, вологість і інші фізичні характеристики. Гіперспектральні зображення активно використовуються в таких сферах, як дистанційне зондування Землі, екологія, сільське господарство, а також у медицині для аналізу тканин і органів.

Однак для ефективної обробки гіперспектральних зображень необхідно адаптувати існуючі методи, оскільки традиційні алгоритми для 2D зображень не можуть бути безпосередньо застосовані до даних з високою спектральною роздільною здатністю. Однією з основних проблем є те, що гіперспектральні зображення часто містять багато корельованих каналів, що може призводити до надмірної складності даних і, як наслідок, до погіршення ефективності алгоритмів обробки. Для вирішення цієї проблеми застосовуються методи редукції розмірності, такі як аналіз головних компонент (PCA) або латентне просторове моделювання, які дозволяють зменшити кількість каналів, зберігаючи при цьому найбільш важливу інформацію.

Ще однією проблемою є те, що гіперспектральні зображення можуть містити велику кількість шумів, зокрема, через обмеження в сенсорах або умовах знімання. Це вимагає використання спеціалізованих методів фільтрації та обробки сигналу, таких як вейвлет-трансформація або алгоритми на основі глибоких нейронних мереж, здатних ефективно виділяти важливі ознаки в

даних з високою спектральною та просторовою складністю. Наприклад, використання глибинних нейронних мереж для аналізу гіперспектральних зображень дозволяє автоматично витягувати корисні ознаки, такі як види рослин або типи ґрунтів, що значно покращує точність та ефективність аналізу [12-14].

Іншим підходом до адаптації методів обробки зображень до гіперспектральних даних є використання методів класифікації, що ґрунтуються на аналізі спектральних профілів. Для кожного пікселя гіперспектрального зображення можна побудувати спектральний профіль, що містить інформацію про його колір у різних спектральних діапазонах. Використовуючи ці профілі, можна здійснювати класифікацію пікселів або об’єктів на зображенні за допомогою таких методів, як підтримка векторних машин (SVM), деревоподібні класифікатори або нейронні мережі. Ці методи дозволяють автоматично розпізнавати типи об’єктів і матеріалів на зображеннях, що є особливо корисним для застосувань в дистанційному зондуванні, екології, сільському господарстві та інших областях.

Адаптація методів обробки зображень до нових форматів, таких як 3D зображення і гіперспектральні зображення, є складним, але необхідним кроком для розвитку сучасних технологій аналізу даних. Оскільки ці формати зображень дають значно більше інформації, ніж традиційні 2D зображення, вони відкривають нові можливості для наукових досліджень і практичних застосувань. Застосування нових методів, таких як згорткові нейронні мережі для 3D зображень і глибоке навчання для гіперспектральних зображень, дозволяє вирішувати складні задачі і досягати високої точності в таких сферах, як медицина, екологія, робототехніка та багато інших. Враховуючи швидкий розвиток технологій, в найближчому майбутньому можна очікувати нові досягнення в цій галузі, що дозволять значно покращити ефективність обробки та аналізу зображень у різних сферах діяльності.

2.7 Висновки до другого розділу

Другий розділ був присвячений сучасним методам стиснення та обробки зображень. Він охоплює широкий спектр актуальних тем, що дозволяє зробити кілька важливих висновків.

По-перше, значна увага приділяється основним методам обробки зображень, таким як фільтрація, покращення контрасту та корекція кольору, що є базовими етапами підготовки зображень для подальшої обробки та аналізу. Ці методи є важливими для забезпечення високої якості зображень перед використанням більш складних алгоритмів відновлення або детекції. Важливість таких технік полягає в тому, що вони створюють умови для ефективнішого аналізу, зменшення шумів та підвищення інформативності зображень.

По-друге, алгоритми відновлення зображень, які розглядаються в розділі, займають важливе місце, оскільки багато зображень, що використовуються в різних сферах, можуть бути пошкоджені або містити шуми. Методи відновлення дозволяють відновити цінну інформацію та покращити якість зображень, що є критично важливим для таких галузей, як медицина, аерокосмічні дослідження та промисловість.

По-третє, методи детекції контурів та меж, зокрема алгоритми Sobel, Canny та Laplacian of Gaussian (LoG), є важливими інструментами для аналізу структур та об’єктів на зображеннях. Детекція контурів є основою для подальших етапів обробки, таких як сегментація зображень або виявлення об’єктів, що дозволяє здійснювати автоматичний аналіз зображень з високою точністю.

По-четверте, алгоритми підвищення якості зображень після стиснення є важливим напрямом для роботи з зображеннями, що пройшли етап стиснення. Стиснення зображень, яке зменшує їх розмір, може призвести до втрати якості, тому необхідність у методах відновлення якості є актуальною. Ці алгоритми

допомагають відновити важливі деталі та зменшити артефакти, що виникають внаслідок стиснення, зберігаючи баланс між розміром файлу і якістю зображення.

По-п’яте, використання штучного інтелекту та глибинного навчання, зокрема згорткових нейронних мереж (CNN), в обробці зображень є дуже перспективним напрямом, який дозволяє досягати високої точності та ефективності в задачах класифікації, сегментації, відновлення та покращення якості зображень. Ці методи відкривають нові можливості в обробці великих обсягів даних і є необхідними для складних задач, таких як обробка медичних зображень або дистанційне зондування.

По-шосте, адаптація методів обробки зображень до нових форматів, таких як 3D зображення та гіперспектральні зображення, є важливим кроком у розвитку технологій обробки даних. Ці формати вимагають нових підходів, оскільки вони містять набагато більше інформації, ніж традиційні 2D зображення. Це дозволяє отримувати більш точні результати в таких галузях, як медична діагностика, географічне картографування, а також для аналізу матеріалів та екологічних даних.

Отже, можна зробити висновок, що сучасні методи обробки зображень є багатоаспектними та інтегрують традиційні підходи з новітніми технологіями, такими як штучний інтелект та глибинне навчання, що дозволяє значно підвищити ефективність та точність обробки зображень у різноманітних сферах застосування. Розвиток і адаптація методів до нових форматів зображень, таких як 3D та гіперспектральні, сприяє відкриттю нових можливостей для наукових досліджень і промислових застосувань.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Мета розробки, постановка задачі та функціональні вимоги

Постановка задачі: розробка програмного забезпечення для інтерактивної обробки цифрових зображень з широким набором інструментів і можливостей.

Мета проєкту: створити настільну програму з графічним інтерфейсом користувача (GUI), що дозволяє завантажувати, переглядати, змінювати, стискати та зберігати цифрові зображення. Інструмент повинен забезпечувати простоту використання, швидку обробку зображень і надавати користувачу можливість застосовувати різноманітні трансформації, зберігаючи при цьому якість файлів.

Основні функціональні вимоги передбачають:

1) завантаження та управління зображеннями:

– надати користувачам можливість завантажувати одночасно кілька файлів;

– забезпечити список для перегляду завантажених зображень із можливістю вибору для подальшої роботи;

– реалізувати функцію очищення списку завантажених файлів;

2) попередній перегляд:

– відображення оригінальних та оброблених зображень із підтримкою режимів перегляду (оригінал, оброблене зображення, порівняння);

– масштабування для зручного перегляду;

3) інструменти обробки зображень:

– застосування фільтрів і трансформацій;

– налаштування стискання: контроль якості, розміру зображення та зменшення роздільної здатності;

4) масове застосування фільтрів до завантажених файлів із налаштованими параметрами;

5) оцінка якості обробки:

– визначення схожості між оригінальним і обробленим зображенням за допомогою метрики SSIM;

– підрахунок обсягів стиснення зображення;

6) експорт результатів:

– збереження оброблених зображень у зазначену користувачем папку;

– генерація текстового звіту про оброблені файли.

3.2 Вимоги до системи та її переваги

Особливі вимоги до системи доцільно зазначити наступним чином:

– програма повинна працювати в середовищі ОС Windows із використанням бібліотек Python (Tkinter, PIL, NumPy, OpenCV тощо);

– інтерфейс має бути інтуїтивно зрозумілим, а всі операції повинні виконуватись у межах одного додатку;

– забезпечити багатопоточну обробку для підвищення швидкодії.

Ключові переваги:

– зручність у використанні для кінцевих користувачів;

– широкий вибір інструментів для редагування зображень;

– контроль якості й ефективне стиснення для економії місця;

– режим пакетної обробки для роботи з великими обсягами файлів;

– надання звітності та оцінки обробки.

3.3 Мова програмування та бібліотеки реалізації

У представленому проєкті застосовуються різноманітні сучасні технології, інструменти та бібліотеки, які разом утворюють комплексну

систему для обробки та аналізу зображень з інтерактивним графічним інтерфейсом. Програмний продукт розроблено мовою програмування Python, яка є однією з найпопулярніших у світі завдяки своїй зрозумілості, високому рівню абстракції та потужній екосистемі бібліотек і фреймворків для різних напрямів розробки. Саме поєднання переваг Python і спеціалізованих модулів дозволило створити ефективний інтерфейс, що підтримує взаємодію користувача з файлами, їх обробку, аналіз та перетворення.

Однією з ключових складових системи є бібліотека Tkinter, яка є стандартним модулем у Python для побудови графічного інтерфейсу користувача (GUI). Tkinter забезпечує взаємодію між користувачем і програмою завдяки створенню віконних компонентів, кнопок, слайдерів, списків, рамок тощо. Бібліотека дозволяє організувати зручну та інтуїтивно зрозумілу структуру інтерфейсу, завдяки якій користувач може завантажувати файли, вибирати параметри для обробки зображень і спостерігати за результатами роботи програми в реальному часі. Важливою особливістю використання Tkinter є можливість організації зручного простору для відображення попереднього перегляду оброблених зображень та інтерактивного контролю за всіма стадіями роботи.

Для обробки зображень застосовується бібліотека Pillow (PIL), яка є розширенням класичної бібліотеки для роботи із растровими зображеннями в Python. Pillow дозволяє читати, зберігати, змінювати та обробляти зображення у різних форматах, таких як JPEG, PNG, BMP, GIF тощо. У контексті цього проєкту Pillow використовується для виконання операцій, пов’язаних із масштабуванням, зміною яскравості, контрастності, розмиттям, перетворенням у чорно-білий формат та іншими фільтрами. Крім того, бібліотека надає можливість працювати з мініатюрами для ефективного відображення зображень у графічному інтерфейсі. Оброблені зображення можна зберігати у стисненому вигляді з оптимізацією якості, що дозволяє зменшувати їхній розмір без значної втрати візуальних характеристик. Окрім

цього, програма підтримує налаштування параметрів якості та максимального розміру зображень, що робить її корисною для оптимізації графічного контенту.

Ще однією важливою технологією, що застосовується у цьому проєкті, є OpenCV – потужна бібліотека для комп'ютерного зору та обробки зображень. OpenCV є однією з найефективніших і найшвидших бібліотек для роботи з відео- та фотоконтентом завдяки її глибокій інтеграції з мовами низького рівня, зокрема C++ і оптимізованими інструкціями процесора. У контексті реалізації програми OpenCV використовується для фільтрації зображень із застосуванням алгоритмів шумозаглушення, таких як fastNlMeansDenoising, bilateralFilter і медіанне згладжування. Кожен з цих методів має свої переваги у боротьбі зі специфічними типами шумів. Наприклад, алгоритм FastNlMeans є ефективним для усунення випадкових шумів із мінімальною втратою деталей, тоді як бічний фільтр підходить для збереження країв об'єктів під час згладжування.

Окрім OpenCV, для оцінки якості зображень та порівняння результатів застосовується функція обчислення структурної подібності (SSIM) із бібліотеки scikit-image. Показник SSIM дозволяє визначити ступінь схожості між двома зображеннями, що особливо корисно при стисненні або інших перетвореннях, які можуть впливати на візуальну якість. Завдяки цій метриці можна не лише оцінювати втрати, а й порівнювати ефективність різних алгоритмів обробки.

Для зберігання та роботи із зображеннями у форматі масивів даних застосовується NumPy – одна з основних бібліотек для наукових обчислень у Python. NumPy дозволяє швидко й ефективно виконувати операції над багатовимірними масивами, що є необхідним для роботи зі зображеннями, які представляються у вигляді числових матриць. Завдяки можливостям NumPy проводяться операції перетворення зображень між форматами Pillow та

OpenCV, обчислення хеш-сум для унікальної ідентифікації оброблених файлів, а також реалізація різних алгоритмів обробки.

Важливою складовою програми є механізм багатопотоковості, який реалізовано за допомогою модуля threading. Багатопотоковість дозволяє виконувати обробку великої кількості зображень у фоновому режимі без блокування основного графічного інтерфейсу, що забезпечує зручність і швидкодію програми. Це особливо важливо при пакетній обробці файлів, оскільки дозволяє користувачу продовжувати взаємодію з інтерфейсом під час виконання складних обчислень у фонових потоках.

Окрім цього, у проєкті використовується бібліотека hashlib, що дозволяє обчислювати унікальні хеш-суми для оброблених зображень із метою їхньої ідентифікації та виявлення дублюючих файлів. Обчислення хеш-сум є важливою частиною процесу валідації результатів обробки, оскільки дозволяє підтвердити, що зміни в зображенні були збережені коректно.

Проєкт також забезпечує можливість генерації звітів про обробку зображень у текстовому форматі, для чого використовується базовий функціонал роботи з файлами у Python. Генерація звітів включає детальний опис оброблених файлів, їхніх параметрів, розмірів і метрик якості, що робить програму корисною для аналізу та документування результатів обробки.

Таким чином, використання широкого спектру технологій і бібліотек дозволяє забезпечити повноцінний функціонал для завантаження, обробки, аналізу та збереження зображень. Поєднання можливостей Tkinter, Pillow, OpenCV, NumPy, scikit-image, threading та hashlib дозволило створити потужну програму з інтуїтивно зрозумілим графічним інтерфейсом, яка підтримує інтерактивну взаємодію з користувачем та високопродуктивну обробку графічного контенту.

3.4 Складові програмного забезпечення

Архітектура програмного коду, розробленого в межах проєкту, є багаторівневою та модульною, що забезпечує високу структурованість, гнучкість та ефективність роботи. Основна структура коду базується на об’єктно-орієнтованій парадигмі програмування, що сприяє інкапсуляції даних, повторному використанню коду та спрощенню процесу його підтримки та масштабування. Код організовано у вигляді кількох взаємодіючих компонентів, де кожен відповідає за окремий набір функціональних можливостей. Така модульна архітектура дозволяє легко вносити зміни до окремих частин програми, не впливаючи на загальну структуру, що є особливо важливим при розробці масштабних проєктів.

Основу архітектури складає центральний клас, який відповідає за ініціалізацію програми, організацію головного циклу подій і координацію всіх внутрішніх модулів. Цей клас об’єднує роботу графічного інтерфейсу, модуля обробки зображень, підсистеми збереження даних і системи логування, що дозволяє забезпечити узгоджену роботу програми як єдиного цілого. Під час ініціалізації програми створюється об’єкт центрального класу, який виконує послідовну інстанціацію усіх необхідних компонентів, таких як інтерфейс користувача, модулі для завантаження, обробки та збереження файлів, а також допоміжні сервіси. Важливо відзначити, що структура проєкту передбачає ізоляцію логіки роботи інтерфейсу від бізнес-логіки та низькорівневих операцій, що досягається шляхом використання окремих класів і методів для кожного функціонального напрямку.

Графічний інтерфейс програми побудовано з використанням бібліотеки Tkinter, де головне вікно програми служить контейнером для різноманітних віджетів, які реалізують функції взаємодії з користувачем. Основний клас інтерфейсу ініціалізує усі графічні компоненти та відповідає за їх розташування за допомогою геометричних менеджерів, що забезпечують

зручний та логічно структурований дизайн. Кожен віджет, такий як кнопки, поля для введення тексту, списки та області для відображення зображень, має власні методи обробки подій, які викликають відповідні функції у бізнес-логіці програми. Це дозволяє розділити інтерфейсний код від основної логіки, що сприяє підтримуваності та зрозумілості архітектури. Важливо зазначити, що система обробки подій у графічному інтерфейсі побудована на основі механізму callback-функцій, які забезпечують асинхронну взаємодію між користувачем та програмою.

Модуль обробки зображень є одним із ключових компонентів архітектури і відповідає за виконання операцій над завантаженими графічними файлами. Цей модуль організовано у вигляді класу, який включає в себе методи для виконання базових і складних операцій над зображеннями, таких як зміна розмірів, застосування фільтрів, перетворення форматів і оптимізація якості. Обробка зображень реалізована за допомогою бібліотек Pillow та OpenCV, які забезпечують швидку і ефективну роботу з растровими зображеннями. Кожен метод модуля обробки є незалежним і може викликатися окремо для виконання конкретного завдання, що забезпечує високу модульність та повторне використання коду. Крім того, завдяки використанню класів у цьому модулі, зображення представлено як об’єкт, що зберігає свої властивості, такі як розмір, формат, шляхи до вихідного та обробленого файлів, а також інші метадані, що полегшує подальший аналіз та управління ними.

Механізм збереження оброблених файлів реалізовано у вигляді окремого підмодуля, який відповідає за збереження графічного контенту на диск у відповідному форматі з урахуванням заданих параметрів, таких як якість і оптимізація розміру. Цей підмодуль забезпечує можливість збереження зображень у кількох популярних форматах, а також виконує перевірку на унікальність файлів завдяки обчисленню хеш-сум за допомогою hashlib. Такий підхід дозволяє уникнути дублювання даних та підвищує

ефективність використання дискового простору. Крім того, підсистема збереження підтримує можливість збереження результатів у заданих користувачем директоріях з автоматичним створенням нових підпапок при необхідності.

Для забезпечення швидкодії програми і підтримки паралельного виконання операцій обробки зображень використано механізм багатопотоковості, реалізований через модуль threading. Основна бізнес-логіка програми виконується у фонових потоках, які запускаються під час виконання ресурсномістких операцій, таких як пакетна обробка зображень або застосування складних фільтрів. Це дозволяє уникнути блокування основного потоку, що відповідає за роботу графічного інтерфейсу, і забезпечує плавну взаємодію користувача з програмою навіть під час виконання інтенсивних обчислень. Кожен фоновий потік є окремим об’єктом класу, який виконує конкретне завдання, а результати його роботи передаються у головний потік для оновлення інтерфейсу та відображення результатів.

Система логування та моніторингу стану програми реалізована за допомогою стандартного модуля logging, що дозволяє зберігати інформацію про всі події, помилки та виняткові ситуації під час виконання програми. Логи зберігаються у текстовому форматі у спеціальних файлах, що дозволяє розробникам аналізувати роботу програми, виявляти проблеми та оптимізувати її продуктивність. Логування включає записи про ініціалізацію компонентів, завантаження файлів, виконання операцій над зображеннями, результати обробки та помилки, що виникають у процесі роботи.

Таким чином, архітектура розробленого програмного коду є чітко структурованою, модульною та об’єктно-орієнтованою. Її ключовими особливостями є розділення логіки інтерфейсу користувача, бізнес-логіки та низькорівневих операцій, використання багатопотоковості для підвищення продуктивності та застосування сучасних бібліотек для роботи з графічними даними. Завдяки такій архітектурі програма є гнучкою, масштабованою та

легко підтримуваною, що забезпечує можливість її подальшого розвитку і вдосконалення відповідно до нових вимог користувачів.

3.5 Опис роботи системи обробки зображень

Розроблене програмне забезпечення функціонує як цілісна система з чітко визначеним алгоритмом роботи, що базується на логічно структурованих етапах обробки даних, взаємодії з користувачем та забезпеченні оптимальної продуктивності. Весь процес починається з ініціалізації програми, яка полягає у створенні всіх необхідних об’єктів, ініціалізації внутрішніх компонентів і налаштуванні параметрів середовища виконання. У цей момент відбувається завантаження графічного інтерфейсу користувача, налаштування внутрішніх структур даних і встановлення зв’язків між модулями, що забезпечують функціональну інтеграцію різних частин системи. Цей етап передбачає перевірку наявності необхідних ресурсів, таких як файли, бібліотеки чи інші зовнішні залежності, а також встановлення первинних значень параметрів для стабільної роботи програмного забезпечення. Після завершення фази ініціалізації програма переходить до очікування взаємодії з користувачем.

Ключовим елементом у процесі роботи розробленого програмного забезпечення є взаємодія з користувачем через графічний інтерфейс, побудований таким чином, щоб забезпечити інтуїтивно зрозумілу і зручну навігацію. Користувач має змогу завантажити вхідні дані у вигляді зображень або інших графічних файлів, які стають основою для подальшої обробки. Після завантаження файлу графічний інтерфейс передає відповідну команду внутрішньому модулю обробки зображень, що функціонує як окремий об’єктний клас у межах програмної архітектури. Програма виконує попередню перевірку завантажених файлів на відповідність формату, розміру та цілісності. Для цього використовуються вбудовані механізми валідації, які аналізують метадані файлів і виконують контрольні операції, такі як

обчислення контрольних сум або аналіз заголовків. Цей процес необхідний для уникнення помилок у подальших етапах роботи і забезпечує надійність функціонування системи.

Після успішного завантаження та валідації вхідного файлу система переходить до фази обробки даних. У межах цієї фази програмне забезпечення виконує комплекс операцій над завантаженим зображенням, використовуючи методи та функції, реалізовані у відповідному модулі обробки. Основні операції включають зміни розмірів, застосування фільтрів, оптимізацію якості, конвертацію форматів і створення проміжних копій файлів. Кожна з цих операцій виконується як окрема функція, яка приймає вхідний файл і необхідні параметри, а також повертає результат у вигляді нового об’єкта зображення або відповідного набору даних. Для реалізації функціональності використовується комбінація бібліотек, зокрема Pillow для базових операцій над зображеннями і OpenCV для виконання складніших алгоритмів обробки, таких як фільтрація, сегментація чи трансформація зображень. Важливим елементом є те, що всі операції виконуються у пам’яті, що дозволяє мінімізувати кількість операцій запису на диск і забезпечує високу продуктивність.

Щоб забезпечити ефективну роботу під час виконання ресурсномістких операцій, програмне забезпечення використовує багатопотоковість і паралельну обробку даних. Це означає, що основний потік програми відповідає за управління інтерфейсом користувача та реагування на події, тоді як фонові потоки виконують обчислення і обробку зображень. Механізм багатопотоковості дозволяє уникнути блокування інтерфейсу, що є критичним для забезпечення комфортного користувацького досвіду. Кожен потік створюється як об’єкт класу Thread і отримує конкретне завдання, після чого починає свою роботу незалежно від інших потоків. Результати виконання операцій передаються головному потоку, який оновлює стан інтерфейсу, відображаючи результати або повідомлення про завершення завдання.

Коли обробка зображень завершена, результати передаються модулю збереження файлів, який відповідає за їх запис у задане користувачем місце. Цей процес включає перевірку прав доступу до цільової директорії, створення нових папок за потреби та запис файлів у необхідному форматі. Завдяки використанню оптимізованих методів збереження даних програма забезпечує мінімізацію втрат якості зображень та ефективну компресію для зменшення обсягу вихідних файлів. Під час збереження оброблених зображень програма генерує лог-файли, які містять інформацію про виконані операції, параметри обробки та шляхи до вихідних файлів. Це дозволяє користувачеві відстежувати процес роботи програми та при необхідності виконувати повторні операції.

Особливістю процесу роботи є також система логування, що функціонує у фоновому режимі і фіксує всі ключові події, включаючи помилки, виклики функцій, результати обробки та інші важливі моменти. Логи зберігаються у вигляді текстових файлів, що дозволяє розробникам і користувачам проводити аналіз роботи системи, діагностувати проблеми та оптимізувати її подальшу роботу. У разі виникнення виняткових ситуацій або помилок користувач отримує відповідне повідомлення через графічний інтерфейс, а сама програма продовжує роботу, завдяки механізмам обробки виключень, реалізованим у ключових місцях коду.

Завершальний етап процесу роботи програмного забезпечення полягає у завершенні роботи програми та очищенні ресурсів. Під час закриття програми центральний модуль ініціює послідовне завершення роботи всіх потоків, збереження поточного стану програми та вивільнення ресурсів, що були виділені під час виконання. Це включає закриття відкритих файлів, видалення тимчасових даних з пам’яті та фіксацію завершального стану у лог-файлах. Завдяки такому підходу забезпечується стабільність програми та запобігання можливим витокам пам’яті або некоректному завершенню роботи.

Таким чином, процес роботи розробленого програмного забезпечення є складним і добре структурованим механізмом, що базується на етапах ініціалізації, взаємодії з користувачем, обробки даних, збереження результатів і завершення роботи. Завдяки використанню сучасних технологій, багатопотоковості та ефективних алгоритмів обробки зображень програма забезпечує високу продуктивність, надійність і зручність користування, що є основними вимогами до програмного забезпечення такого типу.

3.6 Оцінка ефективності програмної реалізації

Оцінка ефективності програмної реалізації є комплексним процесом, що включає аналіз продуктивності, ресурсомісткості, надійності та оптимізації розробленого програмного забезпечення з метою визначення його здатності відповідати вимогам до продуктивності та функціональності у реальних умовах експлуатації. Цей процес передбачає проведення низки емпіричних тестувань, вимірювань продуктивності, аналізу використання обчислювальних ресурсів та обґрунтування доцільності застосованих алгоритмічних і архітектурних рішень. Оцінка ефективності включає якісні та кількісні характеристики програмного коду, визначення часу виконання операцій, аналіз обсягу використаної пам’яті, а також визначення витривалості системи під високими навантаженнями, що є критичним фактором для сучасних програмних продуктів.

Одним із перших аспектів оцінки є визначення часу виконання основних функціональних компонентів системи, що здійснюється шляхом вимірювання тривалості виконання ключових операцій у різних умовах навантаження. Для цього застосовуються інструменти профілювання, які дозволяють деталізовано визначити місця у програмному коді, що споживають найбільшу кількість часу процесора. У розробленому програмному продукті вимірювання часу виконання відбувається для операцій завантаження даних,

їхньої валідації, обробки зображень та запису результатів у зовнішню пам’ять. Зокрема, на етапі обробки зображень оцінка часу стосується як простих операцій, наприклад, зміни розміру чи конвертації формату, так і складних алгоритмічних задач, таких як фільтрація, сегментація чи застосування перетворень. Тестування показало, що використання оптимізованих бібліотек, таких як OpenCV і Pillow, дозволило значно зменшити час виконання порівняно з ручною реалізацією алгоритмів. Результати вимірювань демонструють, що для обробки зображень середнього розміру у межах 1920x1080 пікселів програма виконує більшість операцій у межах сотень мілісекунд, що є прийнятним показником продуктивності для такого типу завдань.

Другим важливим критерієм оцінки є використання пам’яті програмним забезпеченням. Оскільки робота з графічними файлами передбачає маніпуляцію великими обсягами даних, зокрема матрицями пікселів, оцінка споживання оперативної пам’яті є критично важливою для загальної продуктивності системи. У ході тестувань було встановлено, що використання внутрішніх структур даних, оптимізованих для обробки зображень, а також зменшення кількості проміжних копій дозволяє мінімізувати споживання пам’яті під час виконання ресурсомістких операцій. Результати профілювання продемонстрували, що для завдань середнього рівня складності програма використовує обсяг пам’яті, що не перевищує кількох сотень мегабайт, навіть за умови обробки декількох файлів одночасно. Більше того, завдяки механізмам автоматичного збирання сміття, реалізованим у середовищі програмування Python, непотрібні об’єкти вивільняються з пам’яті відразу після завершення їх використання, що забезпечує ефективне управління ресурсами.

Наступним етапом оцінки ефективності є аналіз стабільності та надійності програмного забезпечення. Надійність системи визначається її здатністю функціонувати коректно за різних умов, включаючи стресові

ситуації, такі як обробка великих обсягів даних чи одночасне виконання багатьох завдань. Для перевірки стабільності проводилися стрес-тести, які полягали у послідовному завантаженні, обробці та збереженні значної кількості файлів протягом тривалого часу. У результаті тестувань виявлено, що програма здатна функціонувати без збоїв упродовж тривалого періоду часу, а всі виняткові ситуації, такі як спроби обробки некоректних файлів чи нестача пам’яті, обробляються за допомогою механізмів виключень. Це забезпечує стійкість програмного забезпечення до непередбачуваних ситуацій і дозволяє користувачам отримувати належний рівень функціональності навіть за наявності помилок чи нетипових вхідних даних.

Оптимізація алгоритмів і коду є ще одним важливим компонентом оцінки ефективності, оскільки саме цей фактор безпосередньо впливає на продуктивність програмного забезпечення. Було здійснено аналіз алгоритмічної складності основних функцій програми, що дозволило визначити найкритичніші місця, які потребували оптимізації. Зокрема, заміна деяких ітеративних процесів на векторизовані операції, реалізовані з використанням бібліотеки NumPy, значно скоротила час виконання задач обробки зображень. Подібна оптимізація дозволяє досягти лінійної залежності часу виконання від обсягу вхідних даних, що є ключовим фактором для масштабованості програми. Крім того, використання багатопотоковості дозволило виконувати операції у фоновому режимі паралельно, що зменшило загальний час обробки великих пакетів файлів.

Під час оцінки ефективності було також проаналізовано користувацький досвід, що включає швидкість відгуку програми на дії користувача, інтуїтивність інтерфейсу та стабільність роботи графічного середовища. Тестування показало, що завдяки розподіленню навантаження між основним і фоновими потоками програма забезпечує миттєвий відгук на команди користувача, що є важливим фактором для забезпечення зручності використання. Інтерфейс розроблений таким чином, щоб мінімізувати

кількість зайвих дій та надавати чіткі вказівки на кожному етапі взаємодії з програмою.

На завершення, результати оцінки ефективності показали, що розроблене програмне забезпечення демонструє високі показники продуктивності, надійності та оптимізації. Завдяки застосуванню сучасних технологій, таких як багатопотоковість, оптимізовані бібліотеки для обробки зображень та ефективне управління пам’яттю, програма здатна виконувати завдання обробки графічних даних швидко та стабільно. Показники часу виконання, споживання пам’яті та стійкості до стресових навантажень відповідають вимогам до сучасного програмного забезпечення, що дозволяє стверджувати про високу якість програмної реалізації та її готовність до використання у реальних умовах.

3.7 Тестування системи стиснення та обробки зображень

На рисунку 3.1 наведено програмний застосунок одразу після його запуску. Програмний застосунок містить необхідні можливості для того, аби обрати зображення (одне або декілька), очистити список обраних зображень, налаштувати параметри стиснення (якість, максимальний розмір, зменшення роздільної здатності). Також користувач може застосувати додаткові перетворення та взаємодію із зображеннями: (чорно-біле, блюр, різкість, додати контрастність, додати яскравість). Після того, як усі параметри виставлені, користувач може встановити, який варіант він хоче переглядати: оригінальний, оброблений, або у режимі порівняння: зліва – оригінал зображення, праворуч – оброблене зображення.

Якщо одразу натиснути на обробку, то виникне помилка (рис. 3.2).

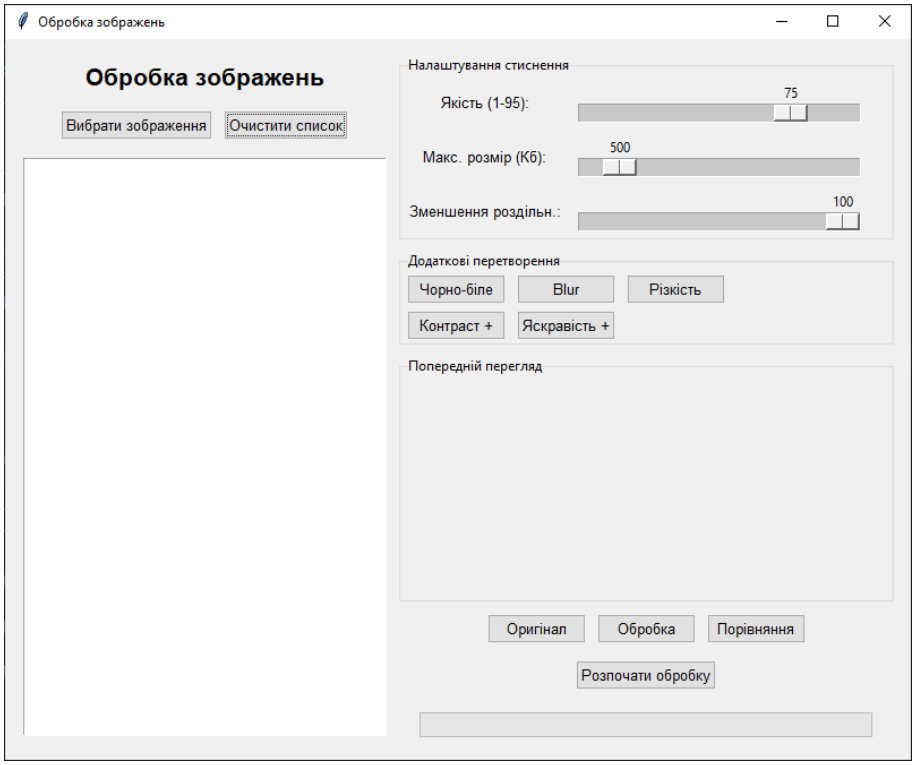

Рисунок 3.1 – Інтерфейс застосунку

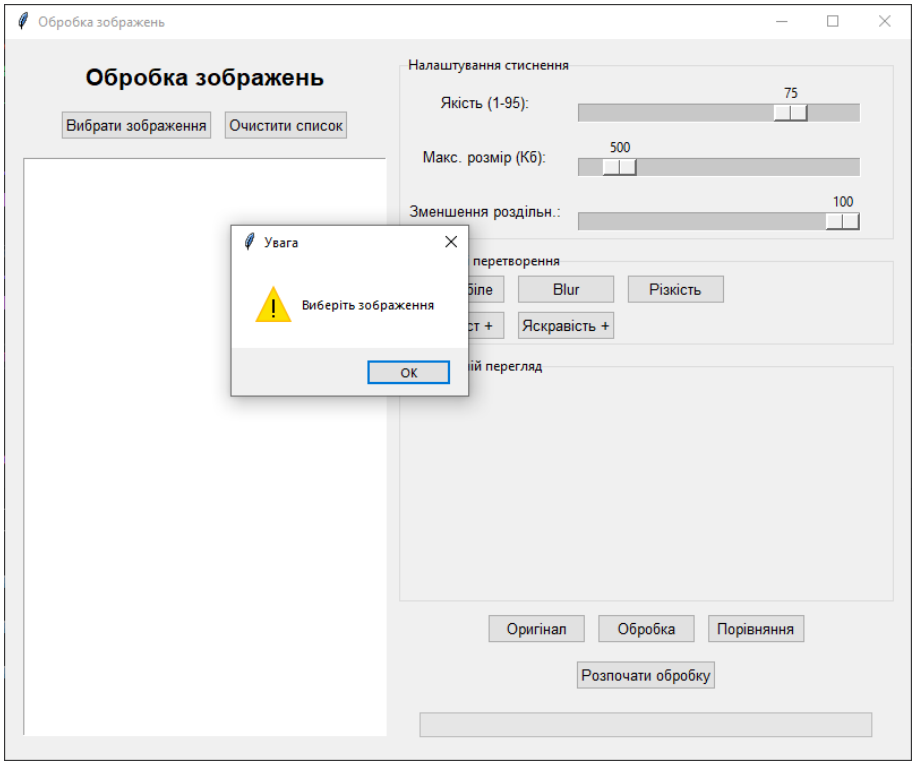

Рисунок 3.2 – Помилка у випадку відсутності зображення

Коли користувач успішно вибере будь-яке зображення зі свого комп’ютера, воно буде відображено у списку зліва (рис. 3.3).

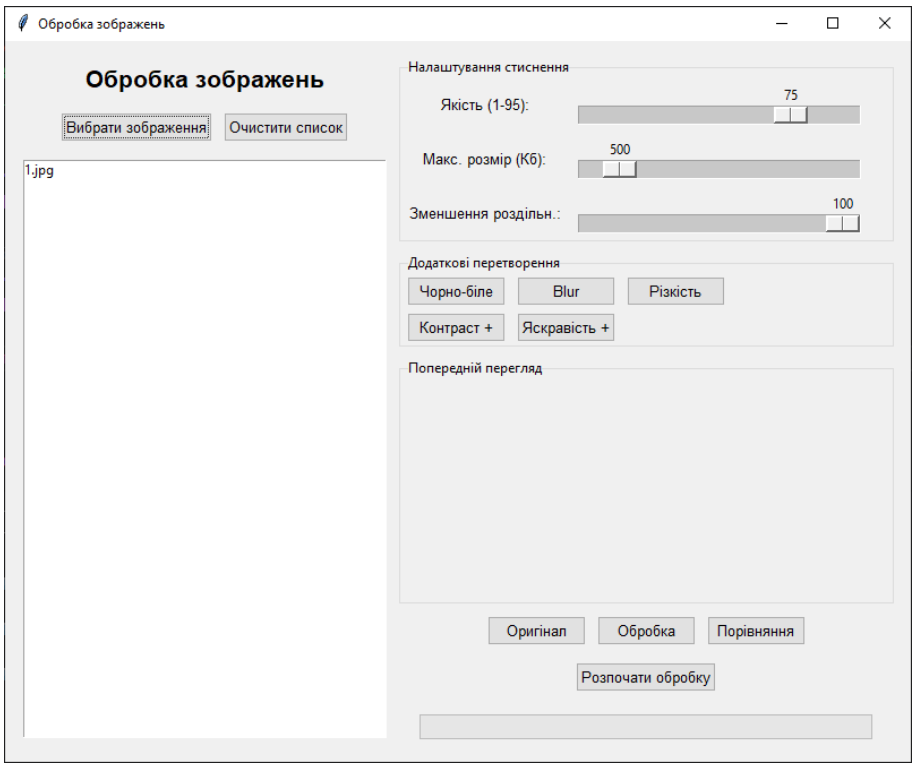

Рисунок 3.3 – Обране зображення відображається у спису зліва

Для того, аби почати роботу з одним із завантажених у програму зображень, необхідно натиснути на нього у списку (рис. 3.4). Тоді подальша робота буде відбуватись саме з обраним зображенням.

На рисунку 3.5 наведено використання фільтра чорно-біле.

На рисунку 3.6 було увімкнуто режим перегляду «порівняння», коли зліва знаходиться оригінал зображення, а праворуч – його оброблена версія.

На рисунку 3.7 наведено використання фільтра блюр.

Збільшена різкість зображення продемонстрована на рисунку 3.8

Збільшення контрастності та яскравості наведено на рисунках 3.9 та 3.10 відповідно.

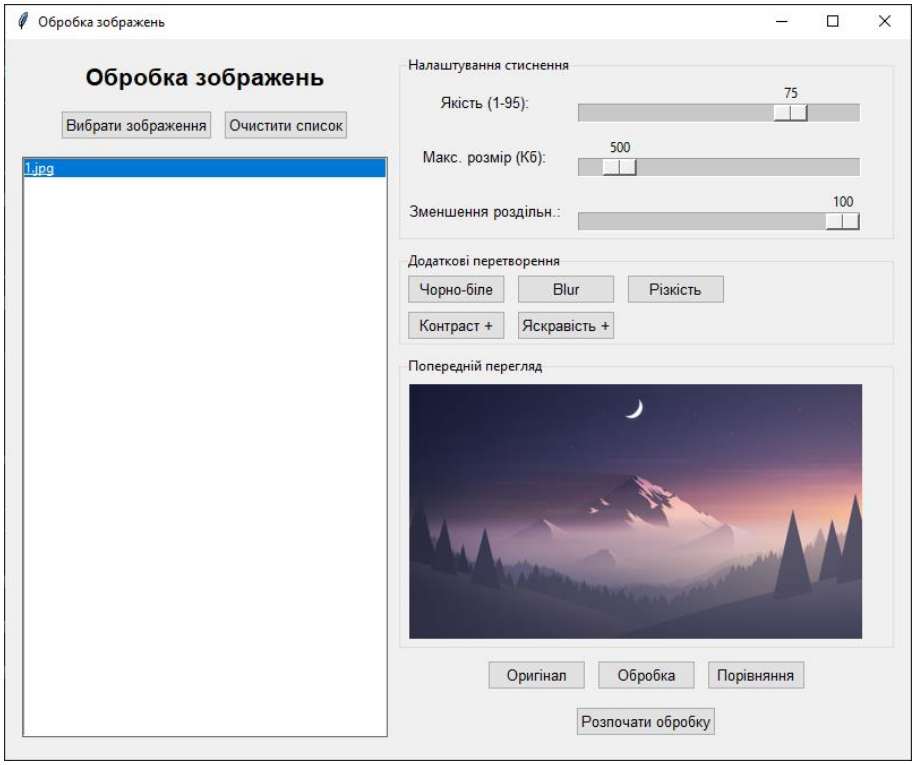

Рисунок 3.4 – Обране зображення зі списку

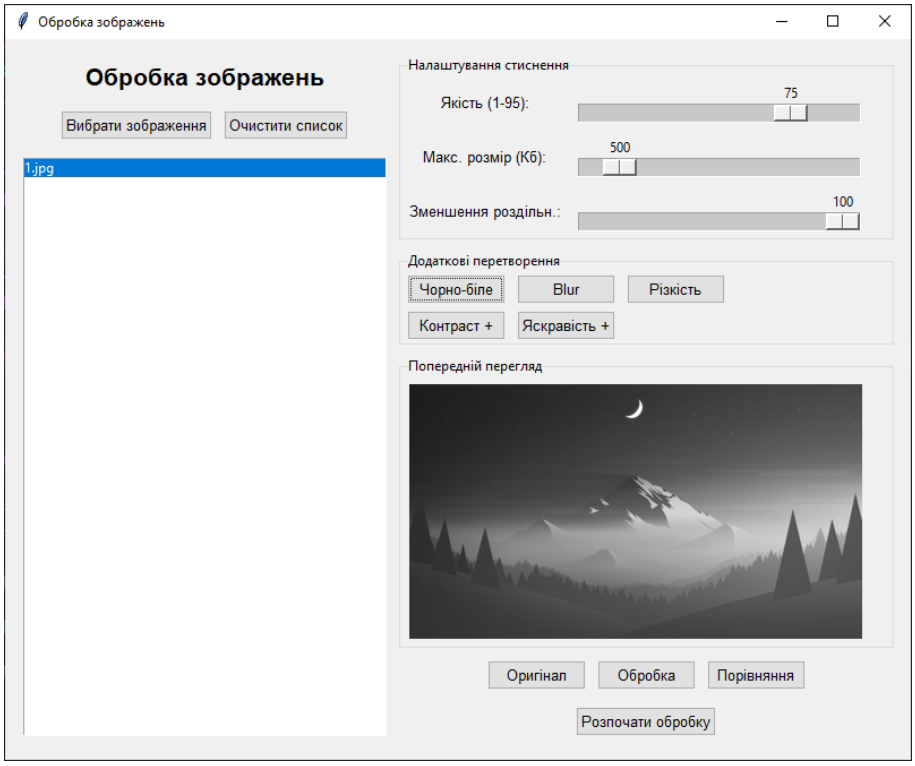

Рисунок 3.5 – Використання фільтра чорно-біле

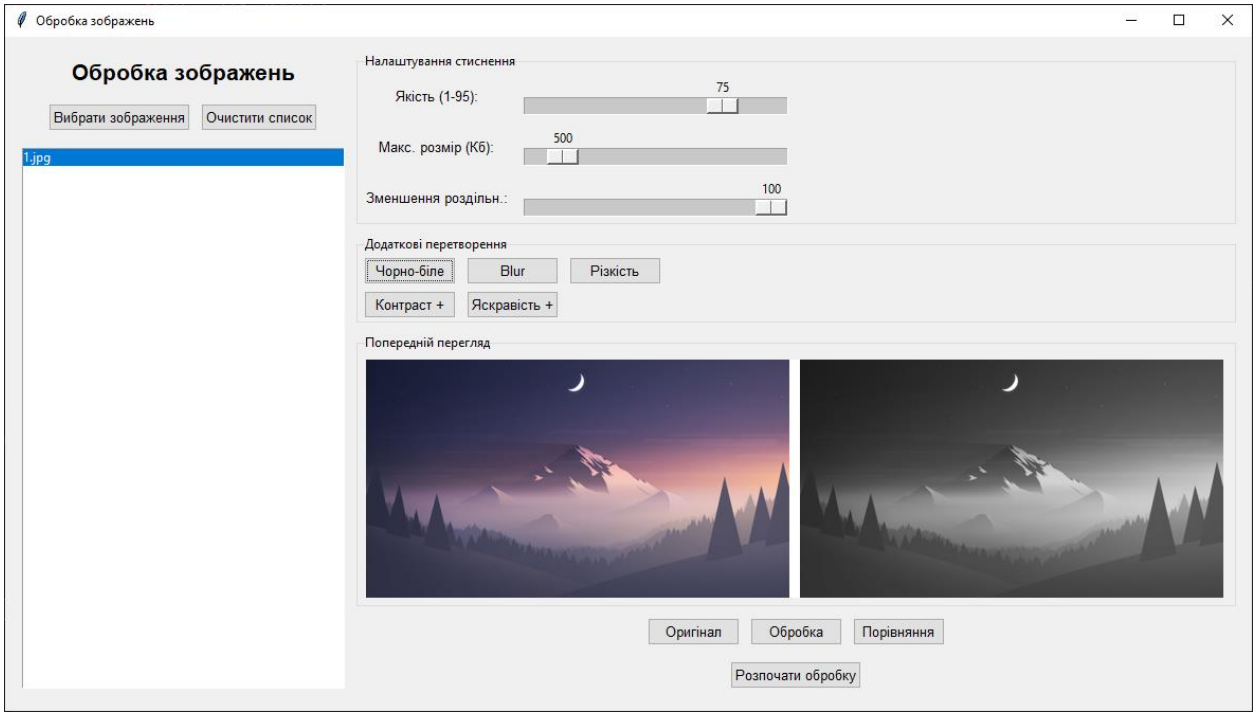

Рисунок 3.6 – Демонстрація режиму перегляду «порівняння»

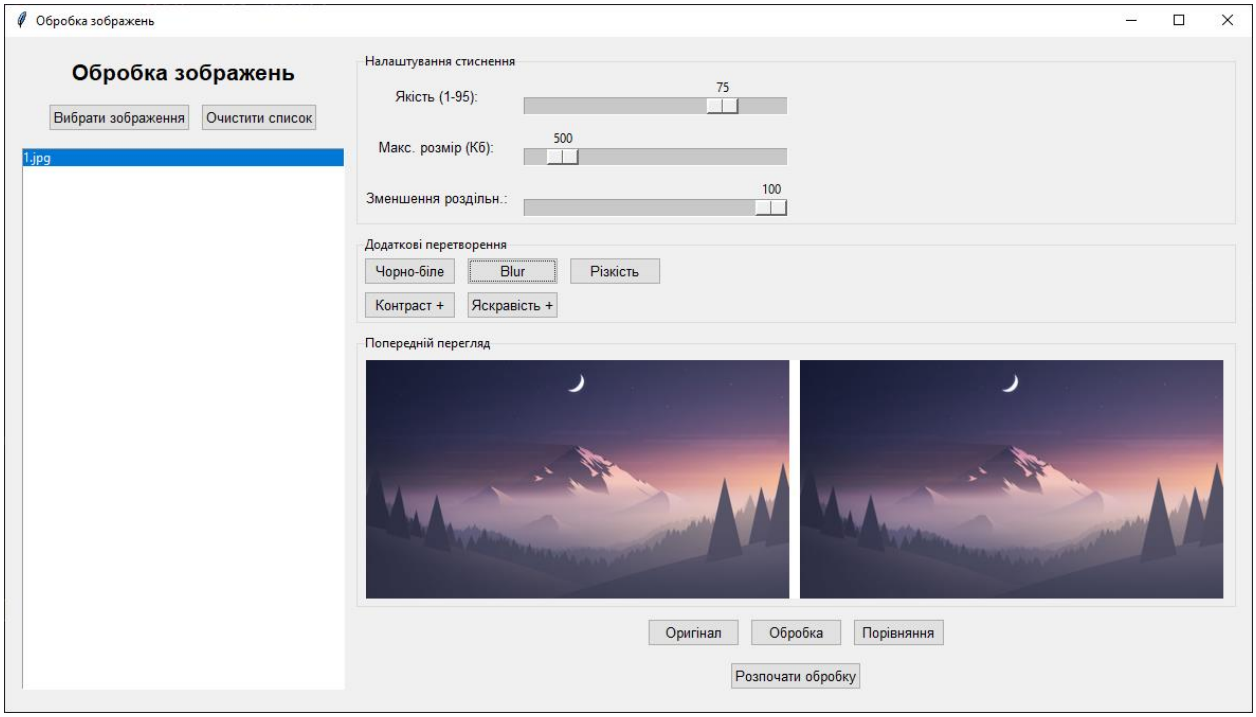

Рисунок 3.7 – Використання фільтра блюр

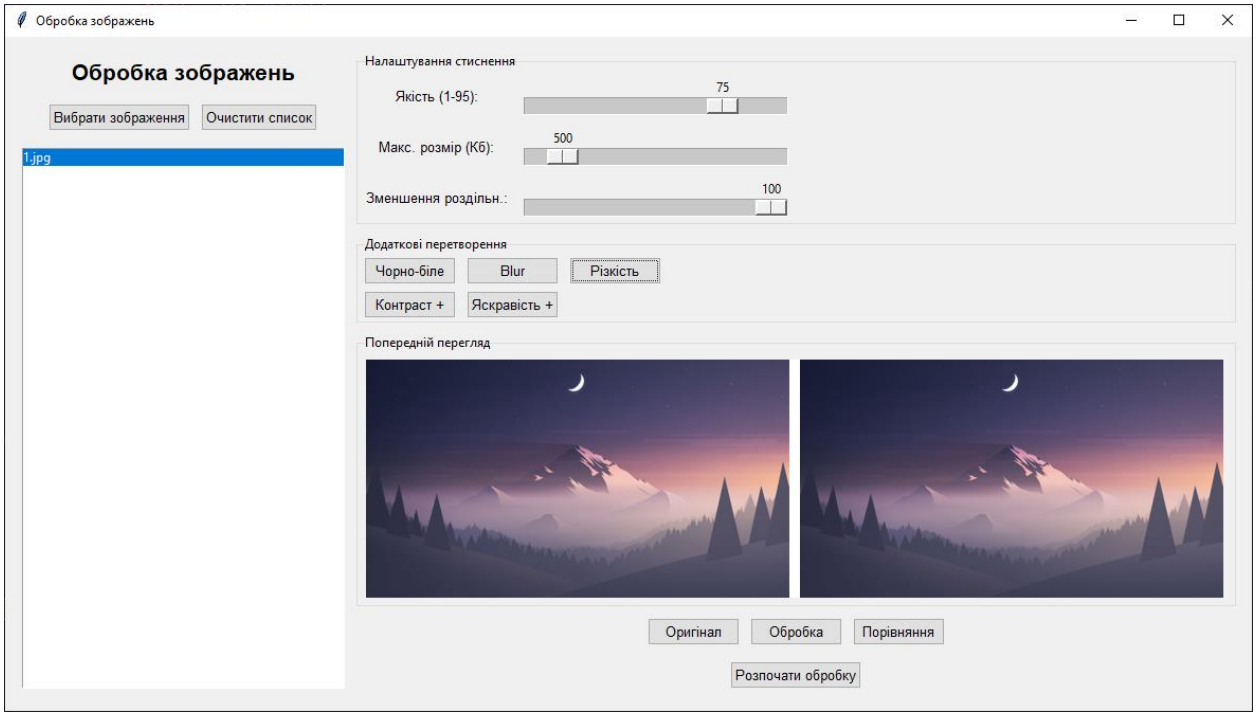

Рисунок 3.8 – Збільшення різкості

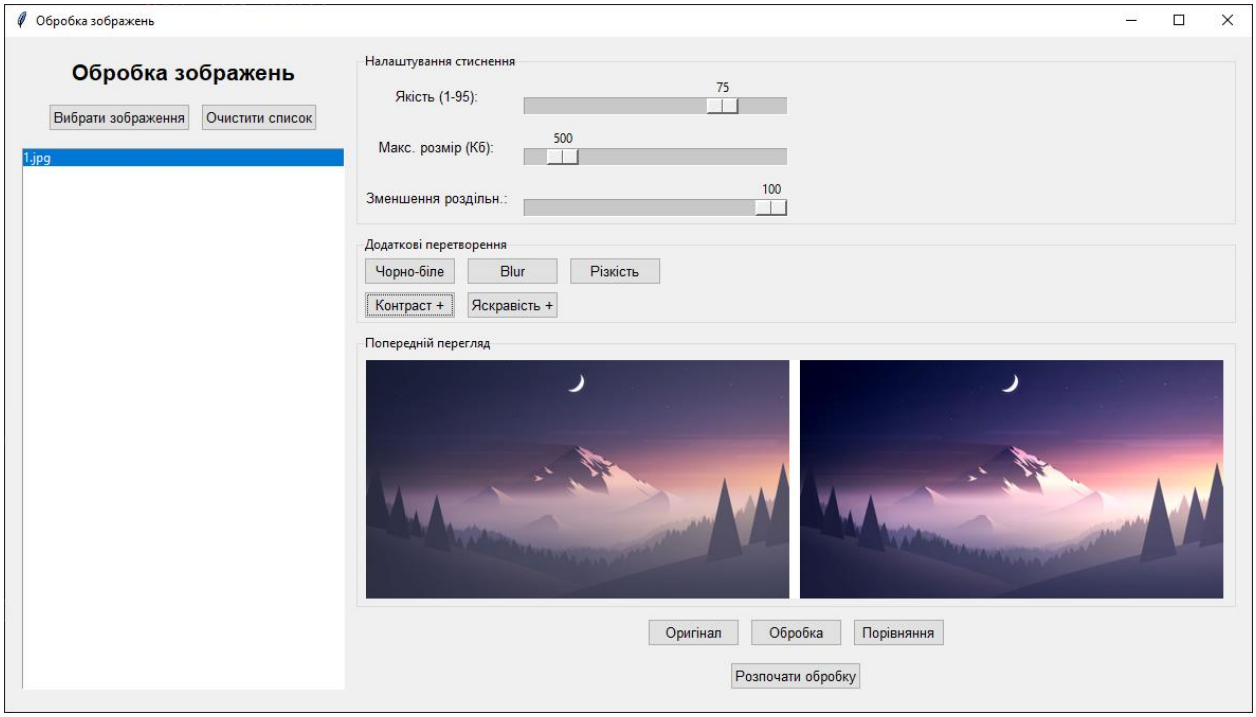

Рисунок 3.9 – Збільшення контрастності зображення

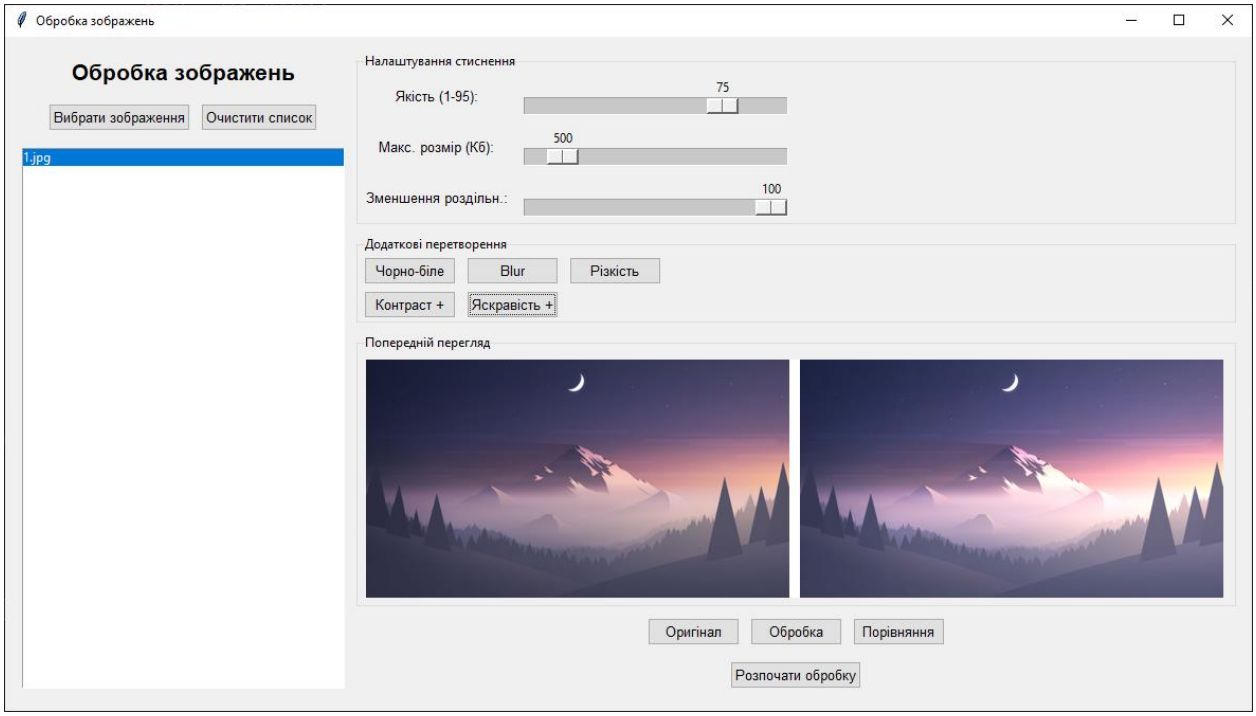

Рисунок 3.10 – Збільшення яскравості зображення

Перед тим, як розпочати обробку та зберегти оброблене зображення, встановимо необхідні параметри за допомогою налаштувань стиснення зверху (рис. 3.11).

Після цього необхідно натиснути на кнопку «Розпочати обробку» і обрати шлях для збереження обробленого зображення (рис. 3.12).

Після того, як шлях було обрано, зображення почне оброблюватися, після чого користувач побачить відповідне інформаційне повідомлення (рис. 3.13), що його зображення було оброблено та збережено за вказаним шляхом.

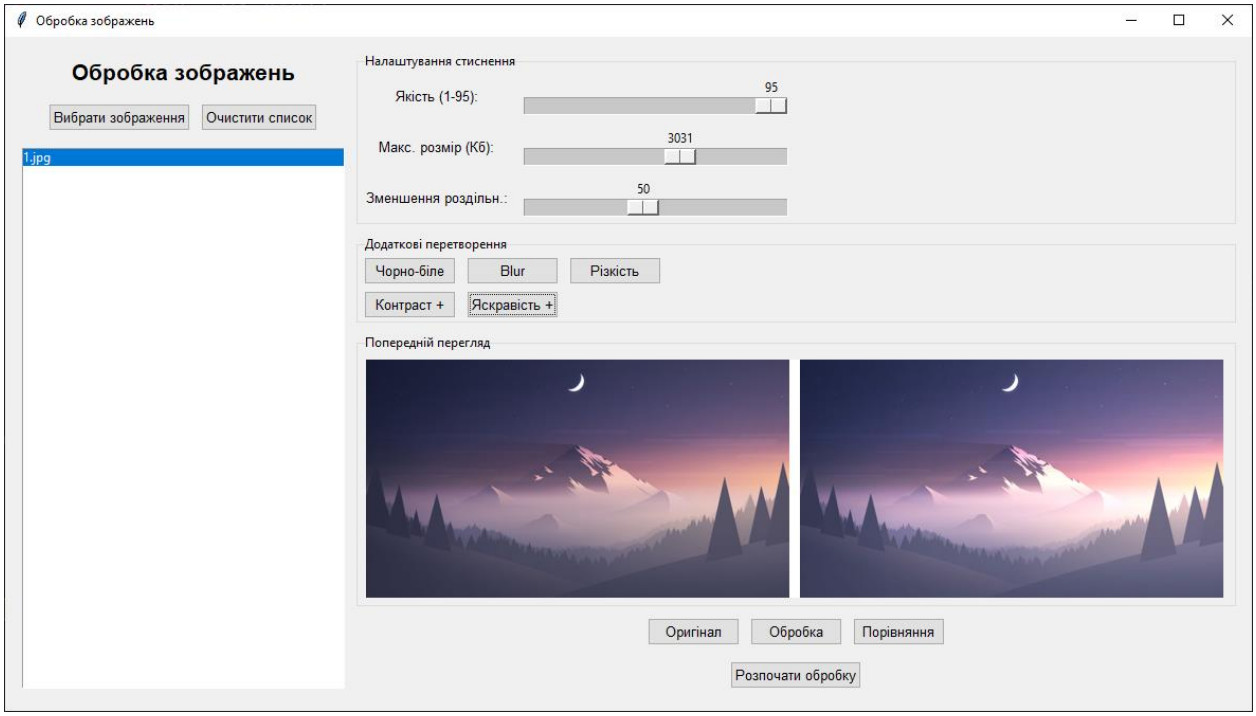

Рисунок 3.11 – Встановлення необхідних параметрів стиснення

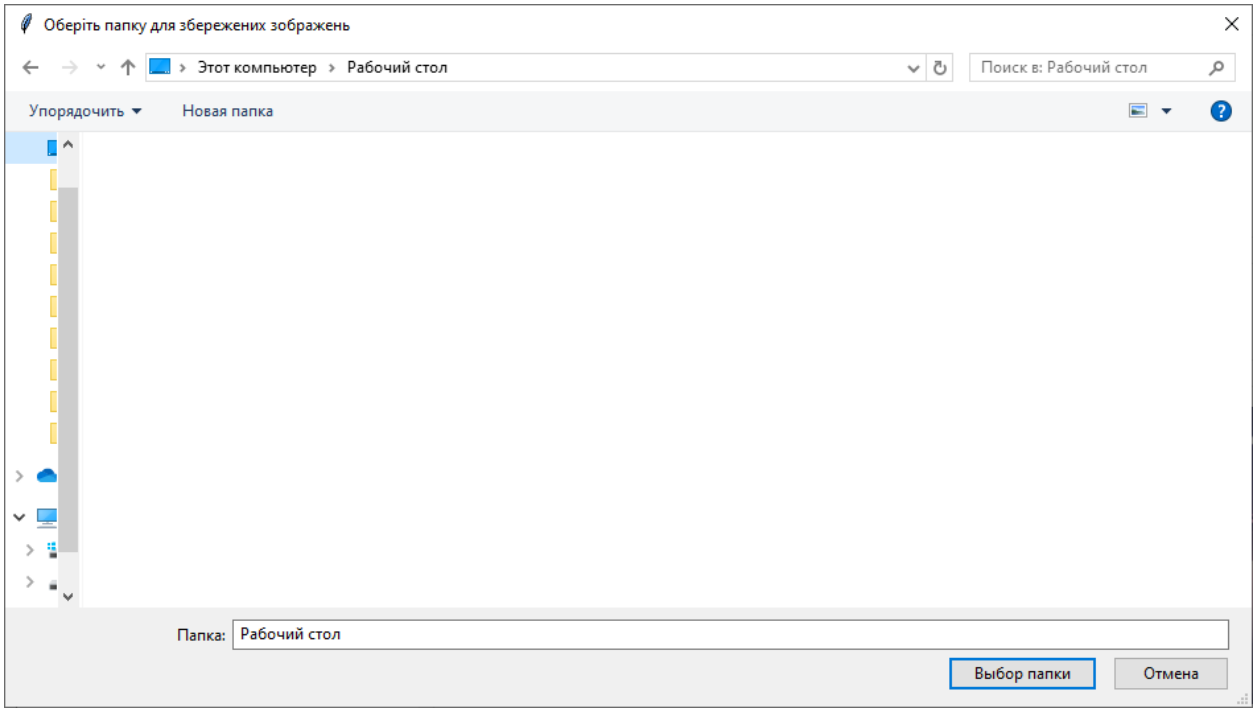

Рисунок 3.12 – Вибір шляху для збереження обробленого зображення

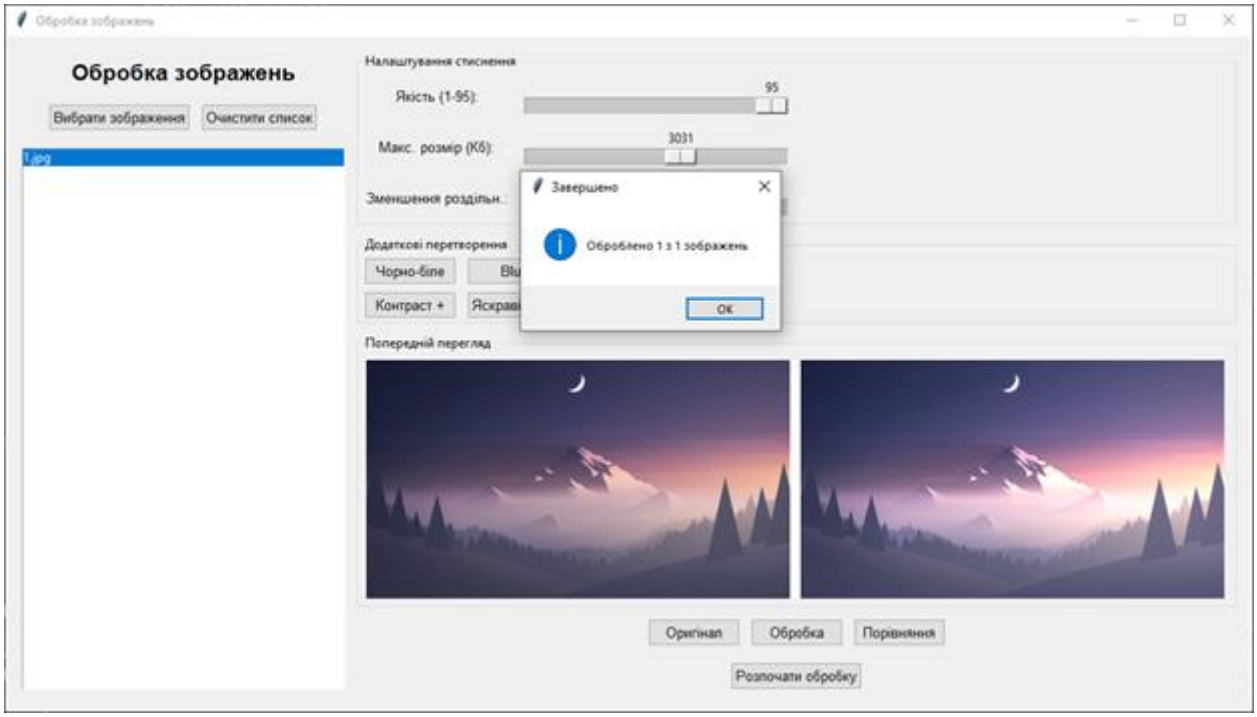

Рисунок 3.13 – Підтвердження завершення обробки зображення

Тепер можна порівняти отриманий розмір файлу та його роздільну здатність з оригіналом.

Отже, на рисунку 3.14 наведено роздільну здатність та розмір для оригінального зображення, тоді як на рисунку 3.15 наведено роздільну здатність та розмір обробленого та стиснутого зображення.

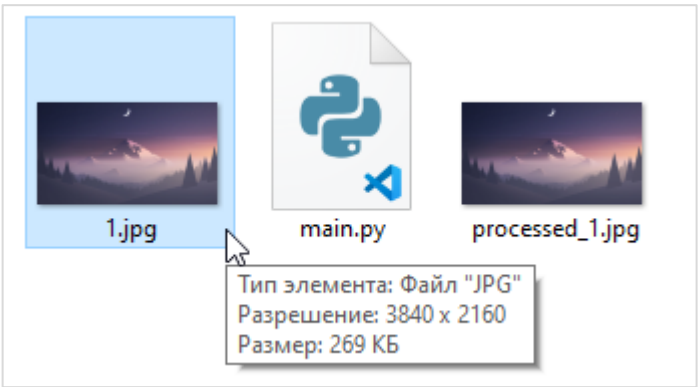

Рисунок 3.14 – Параметри оригінального зображення

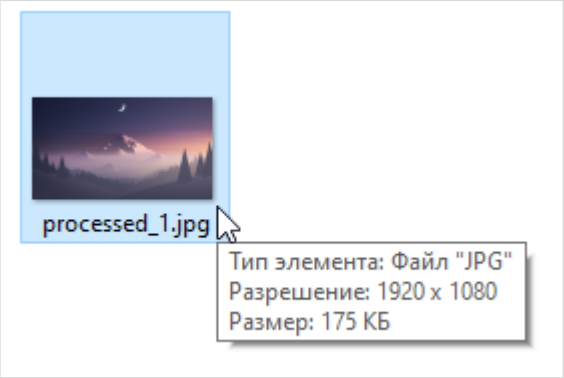

Рисунок 3.15 – Параметри обробленого та стиснутого зображення

В цілому після проведених експериментів можна зробити висновок про те, що вдалося досягти мети розробки та отримати більш компактне та менше за розміром зображення. Воно було оброблено відповідно до вказаних налаштувань та параметрів стискання.

3.8 Висновки до третього розділу

Основною метою розробки було створення програмного забезпечення для обробки та стиснення зображень, яке забезпечувало б високу продуктивність, надійність і ефективність виконання поставлених задач. У ході реалізації було обґрунтовано вибір мови програмування та бібліотек для розробки. Використання мови Python та потужних бібліотек, таких як OpenCV, Pillow та NumPy, дозволило забезпечити оптимальне поєднання швидкодії та функціональності для обробки графічних даних.

Розглянуто архітектуру програмного забезпечення, яка складається з чітко визначених компонентів, зокрема модулів для завантаження зображень, їхньої обробки, оптимізації, стиснення та збереження результатів. Кожен компонент був розроблений таким чином, щоб забезпечити модульність, гнучкість та можливість подальшого розширення системи.

Важливим етапом стало детальне описання роботи системи обробки зображень, включаючи алгоритмічні підходи, застосовані для виконання

базових і складних операцій. Програма підтримує широкий спектр функціональних можливостей, таких як зміна розмірів, конвертація форматів, фільтрація, сегментація та інші перетворення, що відповідають потребам користувача.

Ефективність програмної реалізації була оцінена на основі продуктивності, споживання ресурсів та надійності системи. Застосування інструментів для профілювання коду показало, що система працює ефективно навіть із великими обсягами даних. Оптимізація алгоритмів та використання багатопотоковості дозволили зменшити час виконання основних операцій та мінімізувати використання оперативної пам’яті. Тестування підтвердило стабільну роботу програми в умовах високих навантажень та за наявності нестандартних вхідних даних.

У процесі тестування системи стиснення та обробки зображень були отримані результати, що підтвердили коректність роботи всіх функціональних компонентів. Програма забезпечує швидке та надійне виконання всіх основних операцій, демонструючи високу якість реалізації та відповідність поставленим вимогам.

Таким чином, розроблене програмне забезпечення є ефективним, продуктивним і готовим до використання у реальних умовах. Обґрунтований вибір технологій та архітектурних рішень забезпечив виконання усіх поставлених задач і можливість подальшого розвитку системи.

ВИСНОВКИ

Загальний аналіз методів стиснення та обробки зображень, проведений у роботі, демонструє суттєві досягнення та виклики, що стоять перед сучасними дослідниками в галузі інформаційних технологій. У світі, де цифрові зображення стали невід'ємною частиною життя, ефективність зберігання, обробки та передачі таких даних є критично важливою. У роботі вдалося охопити широкий спектр тем: від класичних методів стиснення, таких як алгоритми Хаффмана, RLE та LZW, до сучасних підходів, включаючи алгоритми, засновані на глибинному навчанні, адаптивних методах та гібридних рішеннях.

Важливе місце відведено аналізу безвтратного стиснення, яке гарантує збереження даних, що є критично важливим у таких галузях, як медицина та наука. Паралельно, стиснення з втратами, хоч і жертвує частиною інформації, забезпечує високу ефективність для мультимедійних файлів, зокрема у форматах JPEG, WebP і JPEG2000, та дозволяє значно скоротити обсяги даних без помітної втрати якості. Гібридні методи, що поєднують переваги обох підходів, пропонують перспективні рішення для задач із високими вимогами до якості та обмеженнями на розмір даних.

Окрему увагу приділено сучасним методам обробки зображень, які включають фільтрацію, детекцію контурів, корекцію кольору та підвищення контрасту. Ці техніки не лише покращують якість візуального представлення даних, а й є основою для більш складних застосувань, таких як автоматична сегментація та аналіз зображень у медичній діагностиці або системах відеоспостереження. Методи обробки після стиснення, зокрема усунення шумів і відновлення втрачених деталей, спрямовані на мінімізацію негативного впливу компресії, відкривають нові можливості для покращення кінцевої якості відновлених зображень.

Особливий акцент зроблено на використанні штучного інтелекту у задачах обробки зображень. Глибинне навчання, зокрема згорткові нейронні мережі, автоенкодери та генеративні змагальні мережі, довели свою ефективність у задачах відновлення, підвищення роздільної здатності та усунення артефактів. Завдяки адаптивності та здатності до самонавчання, ці методи стають потужним інструментом для роботи із зображеннями у форматах високої складності, таких як 3D-моделі чи гіперспектральні дані.

Практична значимість роботи полягає у тому, що отримані результати та розроблені рекомендації можуть бути застосовані в різних галузях, від оптимізації зберігання та передачі даних у телекомунікаціях до медичних систем діагностики, автоматизації процесів обробки зображень у промисловості та створення програмного забезпечення для роботи з мультимедійними даними. Сучасний підхід до порівняння ефективності класичних і новітніх методів дозволив чітко ідентифікувати їхні переваги, обмеження та області застосування.

Загалом, кваліфікаційна робота сприяє розвитку знань у сфері обробки та стиснення зображень, пропонуючи комплексний погляд на існуючі методи та перспективи їх подальшого вдосконалення. Це дозволяє зробити висновок, що подальші дослідження повинні зосереджуватись на інтеграції нових технологій, таких як квантові обчислення або гібридні моделі штучного інтелекту, для створення ще більш ефективних і адаптивних методів роботи з візуальними даними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. Trigka, E. Dritsas, and K. Moustakas, “Joint Power and Contrast Shrinking in RGB Images with Exponential Smoothing”, in *2022 IEEE 14th Image, Video, Multidimensional Signal Process. Workshop (IVMSP)*, Nafplio, Greece, Jun. 26–29, 2022. IEEE, 2022. DOI: https://doi.org/10.1109/ivmsp54334.2022.9816299

2. W. Li, N. Sang, C. Gao, and Y. Shao, “Joint Image Restoration and Matching Based on Hierarchical Sparse Representation”, in *2019 IEEE Int. Conf. Image Process. (ICIP)*, Taipei, Taiwan, Sep. 22–25, 2019. IEEE, 2019. DOI: https://doi.org/10.1109/icip.2019.8803466

3. Z. Li, D. An, Y. Feng, X. Gu, X. Xu, and M. Zhang, “Image Compression Based on Importance Using Optimal Mass Transportation Map”, in *2022 IEEE Int. Conf. Image Process. (ICIP)*, Bordeaux, France, Oct. 16–19, 2022. IEEE, 2022. DOI: https://doi.org/10.1109/icip46576.2022.9897380

4. A. Subashini, G. Raghuraman, and L. SaiRamesh, “Enhancing the Classification Accuracy of Cardiac Diseases using Image Denoising Technique from ECG signal”, in *2019 Int. Conf. Comput. Intell. Data Sci. (ICCIDS)*, Chennai, India, Feb. 21–23, 2019. IEEE, 2019. DOI: https://doi.org/10.1109/iccids.2019.8862168

5. Y. Furgala, A. Velhosh, S. Velhosh, and B. Rusyn, “Using Color Histograms for Shrunk Images Comparison”, in *2021 IEEE 12th Int. Conf. Electron. Inf. Technol. (ELIT)*, Lviv, Ukraine, May 19–21, 2021. IEEE, 2021. DOI: https://doi.org/10.1109/elit53502.2021.9501117

6. F. Gao, X. Deng, J. Jing, X. Zou, and M. Xu, “Extremely Low Bit-rate Image Compression via Invertible Image Generation”, *IEEE Trans. Circuits Syst. Video Technol.*, p. 1, 2023. DOI: https://doi.org/10.1109/tcsvt.2023.3317424

7. Q. Ling and N. A. M. Isa, “Printed Circuit Board Defect Detection Methods Based on Image Processing, Machine Learning and Deep Learning: A

Survey", *IEEE Access*, p. 1, 2023. DOI: https://doi.org/10.1109/access.2023.3245093

8. P. G. Daneshmand, A. Mehridehnavi, and H. Rabbani, "Reconstruction of Optical Coherence Tomography Images Using Mixed Low Rank Approximation and Second Order Tensor Based Total Variation Method", *IEEE Trans. Med. Imag.*, p. 1, 2020. DOI: https://doi.org/10.1109/tmi.2020.3040270

9. K. Ramli, Y. Suryanto, Magfirawaty, and N. Hayati, "Novel Image Encryption Using a Pseudoset Generated by Chaotic Permutation Multicircular Shrinking With a Gradual Deletion of the Input Set", *IEEE Access*, vol. 8, pp. 110351–110361, 2020. DOI: https://doi.org/10.1109/access.2020.3001949

10. S. K. Bhattacharyya and S. Pal, "Measurement of Parboiled and Non-parboiled Rice Grain Dimension during Hydro Thermal Treatment Using Image Processing", in *2020 Nat. Conf. Emerg. Trends Sustain. Technol. Eng. Appl. (NCETSTEA)*, Durgapur, India, Feb. 7–8, 2020. IEEE, 2020. DOI: https://doi.org/10.1109/ncetstea48365.2020.9119920

11. M. Wang, Z. Tian, W. Gui, X. Zhang, and W. Wang, "Low-Light Image Enhancement Based on Nonsubsampled Shearlet Transform", *IEEE Access*, vol. 8, pp. 63162–63174, 2020. DOI: https://doi.org/10.1109/access.2020.2983457

12. S. Karmakar *et al.*, "Image Compression and Decompression based on FFT algorithm", in *2023 7th Int. Conf. Electron., Mater. Eng. Nano-Technol. (IEMENTech)*, Kolkata, India, Dec. 18–20, 2023. IEEE, 2023. DOI: https://doi.org/10.1109/iementech60402.2023.10423491

13. R. Krishnaswamy and S. NirmalaDevi, "EFFICIENT MEDICAL IMAGE COMPRESSION BASED ON INTEGER WAVELET TRANSFORM", in *2020 Sixth Int. Conf. Bio Signals, Images, Instrum. (ICBSII)*, Chennai, India, Feb. 27–28, 2020. IEEE, 2020. DOI: https://doi.org/10.1109/icbsii49132.2020.9167597

14. W. Li and C. Yin, "Optimization of Image Compression and Decompression Performance Based on Genetic Algorithm", in *2024 Int. Conf.*

Interact. Intell. Syst. Techn. (IIST), Bhubaneswar, India, Mar. 4–5, 2024. IEEE, 2024, pp. 702–707. DOI: https://doi.org/10.1109/iist62526.2024.00138

15. M. Alzahrani and M. Albinali, "Comparative Analysis of Lossless Image Compression Algorithms based on Different Types of Medical Images", in *2021 Int. Conf. Women Data Sci. Taif Univ. (WiDSTaif)*, Taif, Saudi Arabia, Mar. 30–31, 2021. IEEE, 2021. DOI: https://doi.org/10.1109/widstaif52235.2021.9430242

16. T. Shinde, "Efficient Image Set Compression", in *2019 IEEE Int. Conf. Image Process. (ICIP)*, Taipei, Taiwan, Sep. 22–25, 2019. IEEE, 2019. DOI: https://doi.org/10.1109/icip.2019.8803230

17. D. Xue, H. Ma, L. Li, D. Liu, and Z. Xiong, "aiWave: Volumetric Image Compression with 3-D Trained Affine Wavelet-like Transform", *IEEE Trans. Med. Imag.*, p. 1, 2022. DOI: https://doi.org/10.1109/tmi.2022.3212780

18. M. Dabass, S. Vashisth, and R. Vig, "Lossy Color Image Compression Technique using Reduced Bit Plane-Quaternion SVD", in *2019 9th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Noida, India, Jan. 10–11, 2019. IEEE, 2019. DOI: https://doi.org/10.1109/confluence.2019.8776990

19. V. S. Kamatar and V. P. Baligar, "A Novel Hybrid Compression Scheme for Lossless Gray Scale Image Compression", in *2023 4th IEEE Global Conf. Advancement Technol. (GCAT)*, Bangalore, India, Oct. 6–8, 2023. IEEE, 2023. DOI: https://doi.org/10.1109/gcat59970.2023.10353248

20. X. Liu and X. Guo, "Ship Image Compression Method Based on HSV Color Space and Kernel Principal Component Analysis", in *2022 Int. Symp. Sens. Instrum. 5G IoT Era (ISSI)*, Shanghai, China, Nov. 17–18, 2022. IEEE, 2022. DOI: https://doi.org/10.1109/issi55442.2022.9963454

ДОДАТКИ

Додаток А

Лістинг програмного коду

```
import os
import tkinter as tk
from tkinter import filedialog, messagebox, ttk, simpledialog
from PIL import Image, ImageEnhance, ImageFilter, ImageTk
import threading
import numpy as np
import cv2
from skimage.metrics import structural_similarity as ssim
import hashlib

class AdvancedImageProcessingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Обробка зображень")
        self.root.geometry("800x600")
        self.root.configure(bg="#f0f0f0")

        self.selected_files = []
        self.processed_images = {}
        self.current_preview_index = 0
        self.preview_mode = "original"

        self.create_ui()
```

```
    def create_ui(self):

        style = ttk.Style()
        style.configure("TButton", font=("Arial", 10))
        style.configure("TLabel", font=("Arial", 10), background="#f0f0f0")


        main_frame = ttk.Frame(self.root)
        main_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)


        left_frame = ttk.Frame(main_frame)
        left_frame.pack(side=tk.LEFT, padx=5, fill=tk.BOTH, expand=True)

        ttk.Label(left_frame, text="Обробка зображень", font=("Arial", 16,
"bold")).pack(pady=10)


        button_frame = ttk.Frame(left_frame)
        button_frame.pack(pady=5)

        ttk.Button(button_frame, text="Вибрати зображення",
command=self.select_images).pack(side=tk.LEFT, padx=5)
        ttk.Button(button_frame, text="Очистити список",
command=self.clear_files).pack(side=tk.LEFT, padx=5)


        self.file_listbox = tk.Listbox(left_frame, width=50, height=15)
        self.file_listbox.pack(pady=10, fill=tk.BOTH, expand=True)
```

```
        self.file_listbox.bind('<<ListboxSelect>>', self.show_preview)


        right_frame = ttk.Frame(main_frame)
        right_frame.pack(side=tk.RIGHT, padx=5, fill=tk.BOTH, expand=True)


        compression_frame = ttk.LabelFrame(right_frame, text="Налаштування стиснення")
        compression_frame.pack(pady=5, fill=tk.X)

        settings = [
            ("Якість (1-95):", "quality", 1, 95, 75),
            ("Макс. розмір (Кб):", "max_size", 10, 5000, 500),
            ("Зменшення роздільн.:", "resize_percent", 10, 100, 100)
        ]

        self.settings_vars = {}
        for i, (label, key, min_val, max_val, default) in enumerate(settings):
            ttk.Label(compression_frame, text=label).grid(row=i, column=0, padx=5, pady=3)
            var = tk.Scale(compression_frame, from_=min_val, to=max_val, orient=tk.HORIZONTAL, length=250)
            var.set(default)
            var.grid(row=i, column=1, padx=5, pady=3)
            self.settings_vars[key] = var
```

```
        transform_frame = ttk.LabelFrame(right_frame, text="Додаткові перетворення")
        transform_frame.pack(pady=5, fill=tk.X)

        transforms = [
            ("Чорно-біле", self.apply_bw),
            ("Blur", self.apply_blur),
            ("Різкість", self.apply_sharpen),
            ("Контраст +", self.increase_contrast),
            ("Яскравість +", self.increase_brightness)
        ]

        for i, (name, func) in enumerate(transforms):
            ttk.Button(transform_frame, text=name, command=func).grid(row=i//3, column=i%3, padx=5, pady=3)


        preview_frame = ttk.LabelFrame(right_frame, text="Попередній перегляд")
        preview_frame.pack(pady=5, fill=tk.BOTH, expand=True)

        self.preview_label = ttk.Label(preview_frame)
        self.preview_label.pack(padx=5, pady=5, fill=tk.BOTH, expand=True)


        preview_control_frame = ttk.Frame(right_frame)
        preview_control_frame.pack(pady=5)

        preview_modes = [
            ("Оригінал", "original"),
```

```
            ("Обробка", "processed"),
            ("Порівняння", "compare")
        ]

        for name, mode in preview_modes:
            ttk.Button(preview_control_frame, text=name,
                       command=lambda m=mode:
self.set_preview_mode(m)).pack(side=tk.LEFT, padx=5)



        ttk.Button(right_frame, text="Розпочати обробку",
command=self.start_processing).pack(pady=10)



        self.progress = ttk.Progressbar(right_frame, length=400, mode='determinate')
        self.progress.pack(pady=10)

    def select_images(self):
        files = filedialog.askopenfilenames(
            filetypes=[
                ("Зображення", "*.jpg *.jpeg *.png *.bmp *.gif"),
                ("Всі файли", "*.*")
            ]
        )

        if files:
            self.selected_files = list(files)
            self.file_listbox.delete(0, tk.END)
            for file in self.selected_files:
```

```
            self.file_listbox.insert(tk.END, os.path.basename(file))


        self.processed_images = {}

    def clear_files(self):
        self.selected_files.clear()
        self.file_listbox.delete(0, tk.END)
        self.processed_images.clear()
        self.preview_label.config(image='')

    def show_preview(self, event=None):
        if not self.selected_files:
            return

        selected_indices = self.file_listbox.curselection()
        if not selected_indices:
            return

        index = selected_indices[0]
        file_path = self.selected_files[index]
        self.current_preview_index = index

        img = Image.open(file_path)
        img.thumbnail((400, 400))

        if self.current_preview_index in self.processed_images:
            processed_img = self.processed_images[self.current_preview_index]
            processed_img.thumbnail((400, 400))
```

```
            if self.preview_mode == "original":
                display_img = img
            elif self.preview_mode == "processed":
                display_img = processed_img
            else:
                display_img = self.create_comparison_image(img, processed_img)
        else:
            display_img = img

        photo = ImageTk.PhotoImage(display_img)
        self.preview_label.config(image=photo)
        self.preview_label.image = photo

    def set_preview_mode(self, mode):
        self.preview_mode = mode
        self.show_preview()

    def create_comparison_image(self, img1, img2):

        width = img1.width + img2.width + 10
        height = max(img1.height, img2.height)

        comparison_img = Image.new('RGB', (width, height), color='white')
        comparison_img.paste(img1, (0, 0))
        comparison_img.paste(img2, (img1.width + 10, 0))

        return comparison_img
```

```
    def start_processing(self):
        if not self.selected_files:
            messagebox.showwarning("Увага", "Виберіть зображення")
            return


        output_dir = filedialog.askdirectory(title="Оберіть папку для збережених зображень")
        if not output_dir:
            return


        quality = int(self.settings_vars['quality'].get())
        max_size_kb = float(self.settings_vars['max_size'].get())
        resize_percent = int(self.settings_vars['resize_percent'].get())


        threading.Thread(
            target=self.process_images_thread,
            args=(output_dir, quality, max_size_kb, resize_percent),
            daemon=True
        ).start()

    def process_images_thread(self, output_dir, quality, max_size_kb, resize_percent):
        total_files = len(self.selected_files)


        self.root.after(0, self.progress.config, {"maximum": total_files})
```

```
        successful_processed = 0
        for index, file_path in enumerate(self.selected_files, 1):

            self.root.after(0, self.progress.configure, {"value": index})


            result = self.process_image(file_path, output_dir, quality, max_size_kb,
resize_percent)

            if result:
                successful_processed += 1

                self.processed_images[index-1] = result['processed_image']


        self.root.after(0, messagebox.showinfo,
            "Завершено",
            f"Оброблено {successful_processed} з {total_files} зображень"
        )

        self.root.after(0, self.show_preview)

    def process_image(self, file_path, output_dir, quality, max_size_kb,
resize_percent):
        try:

            img = Image.open(file_path)
            orig_filename = os.path.basename(file_path)
```

```
        orig_size = os.path.getsize(file_path) / 1024


        if resize_percent < 100:
            new_width = int(img.width * resize_percent / 100)
            new_height = int(img.height * resize_percent / 100)
            img = img.resize((new_width, new_height), Image.LANCZOS)


        processed_img = self.apply_additional_processing(img)


        current_quality = quality
        while current_quality > 1:
            output_path = os.path.join(output_dir, f"processed_{orig_filename}")
            processed_img.save(output_path, optimize=True,
quality=current_quality)


            file_size_kb = os.path.getsize(output_path) / 1024
            if file_size_kb <= max_size_kb:
                break

            current_quality -= 5


        orig_array = np.array(img)
        processed_array = np.array(processed_img)
        ssim_score = ssim(orig_array, processed_array, multichannel=True)
```

```
            file_hash = self.calculate_image_hash(processed_img)

            return {
                'output_path': output_path,
                'original_size_kb': orig_size,
                'processed_size_kb': file_size_kb,
                'ssim_score': ssim_score,
                'file_hash': file_hash,
                'processed_image': processed_img
            }
        except Exception as e:
            print(f"Помилка обробки {orig_filename}: {e}")
            return None

    def apply_additional_processing(self, img):

        return img

    def apply_bw(self):
        if not self.selected_files:
            return

        file_path = self.selected_files[self.current_preview_index]
        img = Image.open(file_path)
        bw_img = img.convert('L')

        self.processed_images[self.current_preview_index] = bw_img
```

```
        self.show_preview()

    def apply_blur(self):
        if not self.selected_files:
            return

        file_path = self.selected_files[self.current_preview_index]
        img = Image.open(file_path)
        blurred = img.filter(ImageFilter.GaussianBlur(radius=2))

        self.processed_images[self.current_preview_index] = blurred
        self.show_preview()

    def apply_sharpen(self):
        if not self.selected_files:
            return

        file_path = self.selected_files[self.current_preview_index]
        img = Image.open(file_path)
        sharpened = img.filter(ImageFilter.SHARPEN)

        self.processed_images[self.current_preview_index] = sharpened
        self.show_preview()

    def increase_contrast(self):
        if not self.selected_files:
            return

        file_path = self.selected_files[self.current_preview_index]
```

```
    img = Image.open(file_path)
    enhancer = ImageEnhance.Contrast(img)
    contrasted = enhancer.enhance(1.5)

    self.processed_images[self.current_preview_index] = contrasted
    self.show_preview()

def increase_brightness(self):
    if not self.selected_files:
        return

    file_path = self.selected_files[self.current_preview_index]
    img = Image.open(file_path)
    enhancer = ImageEnhance.Brightness(img)
    brightened = enhancer.enhance(1.3)

    self.processed_images[self.current_preview_index] = brightened
    self.show_preview()

def calculate_image_hash(self, image):

    img_array = np.array(image)
    img_hash = hashlib.md5(img_array.tobytes()).hexdigest()
    return img_hash

def generate_detailed_report(self):

    if not self.processed_images:
        messagebox.showwarning("Увага", "Немає оброблених зображень")
```

```
            return

        report_path = filedialog.asksaveasfilename(
            defaultextension=".txt",
            filetypes=[("Текстові файли", "*.txt")]
        )

        if not report_path:
            return

        with open(report_path, 'w', encoding='utf-8') as report_file:
            report_file.write("Звіт про обробку зображень\n")
            report_file.write("=" * 40 + "\n\n")

            total_original_size = 0
            total_processed_size = 0

            for index, (orig_path, processed_data) in enumerate(zip(self.selected_files,
self.processed_images.values()), 1):
                orig_filename = os.path.basename(orig_path)
                orig_img = Image.open(orig_path)

                report_file.write(f"Зображення #{index}: {orig_filename}\n")
                report_file.write(f"Оригінальний розмір:
{orig_img.width}x{orig_img.height}\n")

                if processed_data:
                    report_file.write(f"Оброблений розмір:
{processed_data.width}x{processed_data.height}\n")
```

```
                report_file.write(f"Співвідношення сторін:
{processed_data.width/processed_data.height:.2f}\n")

            report_file.write("\n")

        report_file.write("Статистика стиснення\n")
        report_file.write("-" * 20 + "\n")

    messagebox.showinfo("Звіт", f"Звіт збережено: {report_path}")

def batch_process_images(self):

    batch_settings = tk.Toplevel(self.root)
    batch_settings.title("Пакетна обробка")
    batch_settings.geometry("400x500")


    settings = [
        ("Стиснення", [
            ("Якість", 1, 95, 75),
            ("Макс. розмір (Кб)", 10, 5000, 500)
        ]),
        ("Трансформації", [
            ("Зменшення", 10, 100, 100),
            ("Blur", 0, 5, 0),
            ("Контраст", 0.5, 2.0, 1.0)
        ])
    ]
```

```
        batch_vars = {}
        for category, options in settings:
            frame = ttk.LabelFrame(batch_settings, text=category)
            frame.pack(pady=10, padx=10, fill=tk.X)

            for label, min_val, max_val, default in options:
                ttk.Label(frame, text=label).pack(side=tk.TOP)
                var = tk.Scale(frame, from_=min_val, to=max_val,
orient=tk.HORIZONTAL, length=300)
                var.set(default)
                var.pack(side=tk.TOP)
                batch_vars[label] = var

        def start_batch_process():

            batch_settings.destroy()

        ttk.Button(batch_settings, text="Почати обробку",
command=start_batch_process).pack(pady=10)

    def denoising_filter(self):
        if not self.selected_files:
            return

        file_path = self.selected_files[self.current_preview_index]

        try:

            img = cv2.imread(file_path)
```

```
        if img is None or img.size == 0:
            messagebox.showerror("Помилка", "Не вдалося прочитати
зображення")
            return

        height, width = img.shape[:2]


        if height < 7 or width < 7:
            messagebox.showwarning("Увага", "Зображення занадто мале для
фільтрації")
            return


        denoising_methods = {
            'fastNlMeans': lambda x: cv2.fastNlMeansDenoising(x, None, 3, 7, 21),
            'bilateral': lambda x: cv2.bilateralFilter(x, 5, 75, 75),
            'median': lambda x: cv2.medianBlur(x, 3)
        }


        method_name = simpledialog.askstring(
            "Метод видалення шуму",
            "Виберіть метод (fastNlMeans/bilateral/median):",
            initialvalue="fastNlMeans"
        )
```

```
            if method_name not in denoising_methods:
                method_name = 'fastNlMeans'

            denoised = denoising_methods[method_name](img)


            denoised_pil = Image.fromarray(cv2.cvtColor(denoised,
cv2.COLOR_BGR2RGB))

            self.processed_images[self.current_preview_index] = denoised_pil
            self.show_preview()

        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося обробити зображення:
{str(e)}")

def main():
    root = tk.Tk()
    app = AdvancedImageProcessingApp(root)
    root.mainloop()

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