Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

МЕТОД І ПРОГРАМНИЙ ЗАСІБ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ ІЗ НИЗЬКОЮ РОЗДІЛЬНОЮ ЗДАТНІСТЮ

Виконав: студент 2-го курсу, групи 1КІ-24м
спеціальності 123 — Комп'ютерна інженерія
Галімон Р.С

Керівник: к.пед.н., доц. каф. ОТ
Добровольська Н.В.
«12» 12 2025р.

Опонент: к.т.н., доц. каф. ПЗ
Ткаченко О.М.
«12» 12 2025р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«16» 12 2025р.

ВНТУ 2025

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Галузь знань — Інформаційні технології
Освітній рівень — магістр
Спеціальність — 123 Комп'ютерна інженерія
Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«25» вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Галімону Роману Сергійовичу

1 Тема роботи "Метод і програмний засіб покращення якості зображень із низькою роздільною здатністю" керівник роботи Добровольська Н. В. к.т.н., доц. каф. ОТ, затверджено наказом вищого навчального закладу від 24.09.2025 року №313.

2 Строк подання студентом роботи: 04.12.2025 р.

3 Вихідні дані до роботи: засіб розробки — VS Code, бібліотеки PyTorch, OpenCV і Real-ESRGAN inference framework, об'єкти для покращення (зображення низької роздільної здатності).

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз методів та засобів покращення якості зображень, дослідження методів для покращення якості зображень, розробка засобу для покращення якості зображень, тестування програмного засобу для підвищення якості зображень, економічна частина.

5 Перелік графічного матеріалу: блок-схема алгоритму роботи програми.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Добровольська Н. В. к.пед.н., доц. каф. ОТ		
5	Ратушняк О. Г., к.т.н., доц. каф. ЕПВМ		
Нормоконтроль	Швець С. І., асист. каф. ОТ		

7 Дата видачі завдання: 25.09.2025

8 Календарний план роботи приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів МКР	Строк виконання	Прим
1	Постановка задачі роботи	05.10.2025	викон
2	Аналіз предметної області	17.10.2025	викон
3	Дослідження методів покращення якості зображень із низькою роздільною здатністю	25.10.2025	викон
4	Розробка програмного засобу покращення якості зображень	02.11.2025	викон
5	Розрахунок економічної частини	11.11.2025	викон
6	Оформлення матеріалів до захисту МКР	17.11.2025	викон
7	Перевірка якості виконання МКР та усунення недоліків	25.11.2025	викон
8	Підписи у керівника, опонента, нормоконтролера	03.12.2025	викон
9	Перевірка на антиплагіат та ШІ	04.12.2025	викон

Студент Галімон Р. С.

Керівник роботи Добровольська Н. В.

АНОТАЦІЯ

УДК 004.9

Галімон Р.С. Метод і програмний засіб покращення якості зображень із низькою роздільною здатністю. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп'ютерна інженерія. Вінниця: ВНТУ, 2025, 127 с.

На укр. мові. Бібліогр.: 30 назв; рис.: 11; табл.: 22.

У магістерській кваліфікаційній роботі досліджено методи покращення якості зображень із низькою роздільною здатністю та розроблено програмний засіб, що реалізує гібридний підхід до їх обробки. Розглянуто сучасні алгоритми фільтрації, інтерполяції та нейромережеві моделі надроздільності, на основі яких сформовано комплексну методику підвищення деталізації та зменшення шумових артефактів.

У роботі проведено теоретичний аналіз класичних та глибинних методів обробки, наведено математичні моделі й порівняльні характеристики алгоритмів. Розроблено архітектуру програмного засобу, що включає модулі попередньої обробки, SR-модель Real-ESRGAN, систему плиткової обробки та інтерактивний графічний інтерфейс.

Проведене тестування підтвердило ефективність запропонованого підходу: результати оцінено за метриками PSNR та SSIM, а також шляхом візуального порівняння. Програмний засіб забезпечує суттєве покращення якості зображень, знижує прояви шумів і компресійних артефактів, зберігаючи структуру та контрастність сцен.

Розроблене рішення може бути використане в задачах цифрової реставрації, технічного аналізу та обробки фотографій, де необхідне підвищення деталізації та відновлення інформації.

Ключові слова: покращення зображень, низька роздільна здатність, надроздільність, попередня обробка, інтерполяція, шумозниження, нейромережеві моделі, Real-ESRGAN, фільтрація.

ABSTRACT

УДК 004.9

Halimon R.S. Method and Software Tool for Enhancing the Quality of Low-Resolution Images. Master's Qualification Thesis in the specialty 123 — Computer Engineering. Vinnytsia: VNTU, 2025, 127 p.

In Ukrainian. Bibliography: 30 titles; figures: 11; tables: 22.

This master's thesis investigates methods for enhancing the quality of low-resolution images and presents the development of a software system that implements a hybrid approach to image improvement. The research examines modern filtering techniques, interpolation methods, and deep learning–based super-resolution models, forming a comprehensive methodology for increasing image detail and reducing noise or compression artifacts.

The theoretical part provides an in-depth analysis of classical and neural methods, including mathematical models and comparative characteristics of the corresponding algorithms. A software architecture was designed and implemented, integrating preprocessing modules, the Real-ESRGAN super-resolution model, a tile-based processing system, and an interactive graphical user interface.

Experimental evaluation confirmed the effectiveness of the proposed approach. The quality of processed images was assessed using PSNR and SSIM metrics, supplemented by visual comparison. The developed software significantly improves image clarity, enhances structural details, and suppresses noise while maintaining natural visual characteristics.

The proposed solution can be applied in digital restoration, technical analysis, image processing tasks, and other domains requiring enhanced detail reconstruction and improved visual quality.

Keywords: image enhancement, low resolution, super-resolution, preprocessing, interpolation, denoising, neural network models, Real-ESRGAN, filtering.

ЗМІСТ

ВСТУП

Актуальність теми дослідження полягає в тому, що в умовах стрімкого розвитку інформаційних технологій цифрові зображення відіграють ключову роль у багатьох галузях — від побутової фотографії до медицини, аерокосмічної зйомки, систем безпеки та штучного інтелекту. Проте якість отриманих візуальних даних не завжди відповідає вимогам практичних задач через обмеження сенсорів, низьку роздільну здатність, втрати під час стиснення або наявність шумів. Такі недоліки знижують ефективність систем розпізнавання образів, ускладнюють діагностику та роблять неможливим відновлення важливих деталей на зображенні. Саме тому дослідження методів підвищення якості зображень із низькою роздільною здатністю є надзвичайно актуальним і має як теоретичну, так і прикладну цінність. Зокрема, особливий інтерес викликають сучасні методи, що базуються на глибинному навчанні, які демонструють значно кращі результати порівняно з традиційними алгоритмами інтерполяції та фільтрації.

Метою дослідження є визначення ефективного підходу до покращення зображень із низькою роздільною здатністю на основі поєднання класичних методів цифрової обробки та моделей глибинного навчання. Для досягнення поставленої мети необхідно виконати такі основні **задачі**:

— провести аналіз існуючих методів покращення якості зображень, визначити їх переваги та недоліки;

— дослідити теоретичні основи класичних алгоритмів інтерполяції та принципи роботи нейронних мереж, що застосовуються для задач суперроздільності;

— розробити гібридний підхід до покращення зображень, який поєднує інтерполяційні методи та глибинні нейронні мережі;

— створити програмний засіб для реалізації запропонованого методу, забезпечивши зручний інтерфейс і високу ефективність обробки;

— провести перевірку ефективності запропонованого рішення та порівняти результати з існуючими методами.

Методи дослідження, використані в роботі, включають теоретичний аналіз, математичне моделювання та експериментальну перевірку ефективності алгоритмів. У роботі розглядаються класичні інтерполяційні методи збільшення зображень — такі як nearest-neighbor, бікубічна та Lanczos-інтерполяція, — а також сучасні підходи, засновані на згорткових нейронних мережах (CNN), зокрема моделі SRCNN, ESRGAN і Real-ESRGAN. Для реалізації програмного засобу використовуються середовище Python та бібліотеки OpenCV і PyTorch, що забезпечують можливість як класичної обробки зображень, так і інтеграції попередньо натренованих глибинних моделей.

Об’єктом дослідження є процеси покращення якості цифрових зображень, які охоплюють методи підвищення роздільної здатності, зменшення шумів, відновлення деталей і покращення чіткості.

Предметом дослідження виступають алгоритми цифрової обробки та реконструкції зображень, включаючи традиційні методи інтерполяції і фільтрації, а також сучасні нейронні підходи до задачі суперроздільності, що базуються на глибинному навчанні.

Новизна роботи полягає у визначенні підходу до покращенні зображень із низькою роздільною здатністю, що поєднує переваги класичних алгоритмів та сучасних моделей глибинного навчання. Такий підхід забезпечує підвищення чіткості та деталізації зображень при збереженні природності й мінімальних обчислювальних витратах. Особливістю дослідження є практична інтеграція моделі Real-ESRGAN у власний програмний засіб, що забезпечує отримання результатів, наближених до професійних рішень, при збереженні простоти використання.

Практичне значення отриманих результатів полягає у створенні програмного засобу, який дозволяє автоматично підвищувати якість цифрових зображень різного типу — фотографій, сканів, відеокадрів тощо. Розроблений інструмент може бути застосований для відновлення старих фотоархівів, покращення візуалізації у медичних або технічних системах, підготовки навчальних вибірок для задач комп’ютерного зору та оптимізації зображень у

мобільних додатках. Крім того, результати дослідження можуть бути використані як основа для подальшої роботи у напрямі вдосконалення методів суперроздільності та розробки ефективних нейронних архітектур.

Апробація результатів роботи здійснена на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи – 2026»

За результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1]:

Галімон Р.С., Добровольська Н.В.. Метод покращення якості зображень із низькою роздільною здатністю Конференція ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН-2026). Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/25933

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

1.1 Поняття цифрового зображення та параметри якості

Цифрове зображення — це двовимірна дискретна функція f(x,y), де x та y позначають просторові координати, а значення функції f визначає яскравість (інтенсивність) пікселя в певній точці. Кожен піксель є мінімальним елементом зображення, що містить інформацію про колір або рівень яскравості. У випадку кольорових зображень інтенсивність представлена трьома складовими — червоною (R), зеленою (G) та синьою (B), які утворюють колірну модель RGB. У чорно-білих (градаційних) зображеннях кожен піксель має одне значення яскравості, що описує ступінь освітленості точки.

На практиці цифрові зображення створюються в результаті дискретизації аналогових сигналів, які надходять від сенсорів камер або сканерів. Під час цього процесу неперервне зображення перетворюється у матрицю чисел, які можна обробляти програмно. Якість цього перетворення залежить від двох основних характеристик: просторової роздільної здатності та кількості градацій яскравості (глибини кольору).

Роздільна здатність — це властивість зображення, яка визначає кількість пікселів у зображенні на одиницю довжини. Використовують цю властивість для опису різкості та чіткості зображення. Вона безпосередньо впливає на рівень деталізації — чим більша кількість пікселів, тим точніше відображаються дрібні об’єкти [2, 3]. Проте збільшення роздільної здатності не завжди гарантує поліпшення якості, адже при недостатній інформації або при масштабуванні зображення з низькою щільністю даних можуть з’являтися спотворення, розмиття чи “пікселізація”.

Іншим важливим параметром є глибина кольору, що визначає кількість можливих значень яскравості для одного пікселя. Наприклад, 8-бітне зображення дозволяє відобразити 256 рівнів яскравості, тоді як 24-бітне RGB-зображення може містити понад 16 мільйонів кольорів. Недостатня глибина кольору призводить до

втрати плавності переходів між відтінками та появи так званого “banding-ефекту”.

Одним із критичних факторів, що впливають на якість цифрових зображень, є шум. Під шумом розуміють випадкові відхилення яскравості або кольору пікселів, які не відповідають реальним властивостям сцени. Джерелами шуму можуть бути недосконалості сенсорів, низьке освітлення, високі значення ISO або помилки при передаванні даних. Найпоширенішими типами шуму є адитивний білий шум Гаусса, імпульсний шум (типу “сіль і перець”) та мультиплікативний шум, характерний для радарних і медичних систем. Наявність шуму знижує контрастність і чіткість деталей, ускладнюючи подальшу обробку, тому ефективна фільтрація шумів є важливою складовою покращення якості зображень.

Різкість зображення — це те, наскільки сильно відрізняються між собою пікселі різної яскравості, що знаходяться поруч один з одним. Простіше кажучи, що більше різниця світлого і темного на фотографії, то вища різкість. Недостатня різкість призводить до розмиття контурів, тоді як надмірна різкість може викликати появу артефактів. З математичної точки зору різкість часто оцінюється через похідні зображення або оператори градієнтів (Sobel, Prewitt, Laplacian), які виявляють області з високою зміною яскравості.

Крім зазначених параметрів, на якість зображення також впливають контраст, динамічний діапазон, точність кольоропередачі та ступінь стиснення. При високому ступені стиснення із втратами виникають характерні блокові артефакти та спотворення, які також потребують компенсації під час покращення зображення.

Таким чином, цифрове зображення є складним об’єктом обробки, на якість якого впливає багато параметрів як на етапі формування, так і під час цифрової трансформації. Розуміння цих параметрів є необхідною передумовою для побудови ефективних алгоритмів покращення, здатних підвищити деталізацію та візуальне сприйняття без внесення небажаних спотворень.

1.2 Класичні методи покращення зображень

Класичні методи покращення зображень виникли задовго до появи

нейронних мереж і становлять фундамент цифрової обробки зображень. Вони базуються на використанні математичних і статистичних методів, що дозволяють модифікувати піксельні значення з метою підвищення візуальної якості або покращення характеристик для подальшого аналізу. До таких методів належать інтерполяція, фільтрація шумів, гістограмна корекція, підсилення контрасту, згладжування та різноманітні алгоритми покращення різкості. Незважаючи на стрімкий розвиток штучного інтелекту, ці методи залишаються актуальними завдяки своїй простоті, надійності та відносно низьким обчислювальним вимогам.

Одним із ключових напрямів класичних підходів є інтерполяція, що застосовується для масштабування зображень, відновлення пошкоджених областей або підвищення роздільної здатності. Суть інтерполяції полягає в оцінці інтенсивності нового пікселя на основі відомих сусідніх значень. Коли зображення збільшується, між наявними пікселями виникають “порожні” місця, які необхідно заповнити таким чином, щоб загальна структура зображення залишалася природною та без спотворень.

Інтерполяція найближчого сусіда — метод інтерполяції, при якому за проміжне значення вибирається найближче відоме значення функції [2]. Цей метод має найвищу швидкодію, однак створює ефект “пікселізації” або “східців”, особливо при значному масштабуванні. Такий підхід іноді використовується у випадках, коли пріоритетом є збереження чіткості меж або коли потрібно зберегти точні значення кольорів (наприклад, у растровій графіці чи при обробці картографічних даних).

Подвійна лінійна інтерполяція (білінійна інтерполяція) — лінійна інтерполяція функції двох змінних, тобто інтерполяція по чотирьох точках, що обчислює інтенсивність нового пікселя як зважене середнє значень. Це дозволяє досягти більш плавних переходів між кольорами та зменшує кількість артефактів. Проте водночас білінійна інтерполяція часто призводить до невеликого “змазування” зображення, оскільки відбувається згладжування різких меж.

Бікубічна інтерполяція є вдосконаленим варіантом, у якому для обчислення значення нового пікселя використовуються дані з області 4x4, тобто 16 сусідніх

пікселів [3]. Вона базується на кубічній функції, що забезпечує більш природне згладжування та покращену передачу деталей. Бікубічна інтерполяція дає значно якісніші результати порівняно з білінійною, але потребує більших обчислювальних ресурсів. Саме тому її найчастіше використовують у професійних графічних редакторах (наприклад, Adobe Photoshop) і системах цифрової фотографії.

З розвитком цифрових технологій з’явилися й інші методи, такі як ланцош-інтерполяція (Lanczos Interpolation), що використовує синусоїдальні ядра для згладжування. Вона забезпечує високу чіткість і мінімальні втрати деталей, однак може створювати “дзвінкові” артефакти поблизу різких контрастних меж.

Інтерполяційні методи застосовуються не лише для збільшення зображень, але й для відновлення пошкоджених або відсутніх пікселів, наприклад у випадках втрати частини даних чи при зменшенні шумів. Вони є важливими складовими алгоритмів суперрезолюції — процесу, спрямованого на підвищення роздільної здатності зображення. У таких задачах класичні методи часто комбінуються з сучасними нейронними мережами, виступаючи як попередній або допоміжний етап обробки.

Важливою перевагою класичних методів інтерполяції є їх стабільність і прогнозованість результатів. Вони не залежать від навчальних вибірок або випадкових факторів, тому забезпечують відтворюваність, що особливо важливо в медичній або технічній діагностиці, де будь-яка зміна пікселя може вплинути на кінцеве рішення.

Однією з основних проблем у цифрових зображеннях є наявність шумів, які з’являються в процесі захоплення, передавання або стиснення даних. Шум знижує контраст, розмиття деталей, утворює артефакти та ускладнює подальший аналіз або розпізнавання об’єктів. Класичні методи обробки зображень пропонують різні підходи для зменшення шуму, зокрема лінійні та нелінійні фільтри, кожен з яких має свої переваги та обмеження.

Лінійні фільтри базуються на зваженому середньому значень сусідніх пікселів. Найпоширенішим прикладом є середній (mean) фільтр, який замінює значення кожного пікселя середнім арифметичним із його оточення. Такий підхід

добре згладжує випадкові шуми, але разом із шумом згладжує й дрібні деталі, що може призвести до втрати різкості. Іншим прикладом є гаусовий фільтр, де середнє обчислюється з урахуванням ваги, що зменшується від центру до периферії піксельної області за гаусовим законом. Це дозволяє більш бережливо усувати шуми та зберігати частину важливих деталей, особливо при роботі з фотографіями чи медичними знімками.

Нелінійні фільтри застосовуються для більш ефективного видалення певних типів шуму, таких як імпульсний (“сіль і перець”), який важко усунути лінійними методами. Основним прикладом є медіанний фільтр, який обчислює нове значення пікселя як медіану сусідніх пікселів. Медіанна обробка дозволяє видаляти імпульсні спотворення без значного згладжування контурів, зберігаючи чіткість країв об’єктів [13]. Такий метод особливо ефективний при обробці сканованих або старих зображень, де шум розподілений нерівномірно.

Крім класичних лінійних і медіанних фільтрів, розроблені адаптивні методи, які змінюють інтенсивність згладжування в залежності від локальної структури зображення. Наприклад, адаптивний середній фільтр зменшує вплив шуму у рівномірних областях, але не згладжує різкі переходи на контурі об’єктів [4]. Подібні методи дозволяють досягати кращого балансу між усуненням шуму та збереженням деталей.

Ще одним класичним підходом є підвищення різкості зображень. Для цього використовують оператори першого або другого порядку (градієнтні та лапласіанські), які визначають зміни яскравості пікселів і виділяють контури об’єктів. Застосування таких фільтрів дозволяє підсилити дрібні деталі та зробити зображення більш чітким. Найпоширенішими є оператори Sobel, Prewitt, Laplacian, які використовуються як у класичній комп’ютерній графіці, так і в підготовчих етапах перед нейронними мережами.

Класичні методи фільтрації та інтерполяції часто комбінуються для досягнення більш високої якості зображення. Наприклад, перед масштабуванням зображення може застосовуватися гаусовий або медіанний фільтр для зменшення шумів, після чого здійснюється бікубічна або Lanczos-інтерполяція для підвищення

роздільної здатності. Такий підхід дозволяє отримувати відносно швидкі та передбачувані результати, що особливо важливо при обробці великих обсягів даних або в системах реального часу.

Хоча класичні методи ефективні і прості в реалізації, вони мають обмеження. При значному збільшенні зображення або при високому рівні шуму традиційні інтерполяційні та фільтраційні алгоритми часто призводять до втрати дрібних деталей та появи артефактів. Саме тому в сучасних системах покращення зображень класичні методи часто доповнюються або замінюються алгоритмами на основі глибинного навчання, які здатні відновлювати структури та текстури, недоступні простим математичним операціям.

1.3 Методи на основі машинного та глибинного навчання

Сучасні методи покращення якості зображень на основі машинного та глибинного навчання суттєво перевершують класичні алгоритми завдяки здатності відновлювати деталі, які неможливо отримати простим математичним опрацюванням пікселів. Основним напрямом у цій галузі є суперроздільність, яка передбачає підвищення роздільної здатності зображення та відновлення високочастотних деталей на основі навчання моделей на великих наборах даних.

Одним із перших методів глибинного навчання для задач SR стала SRCNN (Super-Resolution Convolutional Neural Network) [5]. Вона складається з трьох основних шарів: екстракція особливостей із вхідного зображення, нелінійне відображення та реконструкція зображення. SRCNN демонструє значно кращі результати порівняно з класичною бікубічною інтерполяцією за метриками PSNR (Peak Signal-to-Noise Ratio) та SSIM (Structural Similarity Index), особливо для невеликих масштабувань. Проте модель має обмеження: вона зазвичай вимагає попереднього масштабування зображення класичними методами, що може призводити до розмиття деталей і втрати текстур.

Більш сучасною та ефективною є ESRGAN (Enhanced Super-Resolution Generative Adversarial Network). Ця мережа поєднує згорткові шари з генеративною змагальною мережею (GAN), де генератор намагається відновити реалістичне

зображення, а дискримінатор оцінює його природність [6]. Перевага ESRGAN полягає у відновленні текстур та високочастотних деталей, які класичні методи або SRCNN відтворити не можуть. Мережа особливо добре працює для масштабування у 4x та більше, забезпечуючи більш реалістичні результати для фотографій та ілюстрацій. Недоліком є високі обчислювальні витрати та ризик появи генеративних артефактів на деяких типах зображень.

Real-ESRGAN — покращена версія ESRGAN, адаптована для роботи з реальними зображеннями, які можуть містити шуми, стиснення або пошкодження. Модель навчається на комбінації синтетичних і реальних даних, що дозволяє досягати високої якості відновлення без необхідності ручного попереднього налаштування [7]. Реальні зображення часто містять складні текстури, компресію та різні типи шуму, і саме тому Real-ESRGAN є практичною моделлю для інтеграції у програмні засоби.

Ще одним перспективним напрямом є diffusion models (моделі дифузії), що ґрунтуються на ітеративному процесі видалення шуму та поступового відновлення зображення. На відміну від GAN, моделі дифузії формують зображення поетапно, що дозволяє досягти вищої точності при відтворенні деталей та текстур [8]. Вони добре справляються зі складними сценами та багатошаровими текстурами, проте потребують значно більше часу на обчислення і великих обчислювальних ресурсів, що робить їх менш зручними для інтеграції у програми реального часу.

Якщо порівнювати класичні методи та нейронні підходи, можна виділити такі особливості:

— класичні методи (бікубічна інтерполяція, медіанний фільтр, Gaussian згладжування) прості, швидкі та стабільні, не потребують навчання і добре працюють на невеликих зображеннях, проте вони не здатні відновлювати втрачені текстури і деталізацію при значному масштабуванні або високому шумі;

— SRCNN забезпечує базовий рівень навченої суперроздільності, відновлює дрібні деталі краще, ніж класичні методи, але все ще обмежений у високочастотних текстурах та масштабуванні;

— ESRGAN і Real-ESRGAN дають найбільш реалістичні результати для

природних зображень, відновлюють текстури, роблять їх більш чіткими, але вимагають потужного обладнання (GPU) та часу на навчання і обробку;

— Diffusion models здатні відтворювати надзвичайно детальні та точні зображення, однак є “важкими” у плані обчислень і складні для інтеграції в легкі додатки або реальний час.

Крім того, нейронні підходи демонструють краще масштабування при великих коефіцієнтах (від 4x до 8x), тоді як класичні методи за таких умов призводять до значного розмиття і “пікселізації”. Це особливо важливо для завдань реставрації старих фотографій, медичних знімків або низькоякісних відеокадрів, де критично важлива точність деталей.

1.4 Аналіз існуючих програмних засобів

На сучасному ринку існує велика кількість програмних засобів для покращення якості зображень, які реалізують як класичні, так і нейромережеві методи. Серед найбільш відомих і часто використовуваних можна виділити Adobe Photoshop, Topaz Gigapixel AI, Waifu2x, а також кілька open-source рішень на базі глибинного навчання. Аналіз цих засобів дозволяє зрозуміти переваги, обмеження та вимоги до власного програмного забезпечення.

Adobe Photoshop — універсальний графічний редактор, що включає широкий спектр класичних алгоритмів обробки зображень, таких як бікубічна інтерполяція, Gaussian та медіанна фільтрація, підсилення різкості та корекція контрасту (рисунок 1.1).

Недавні версії також інтегрують функції штучного інтелекту для суперроздільності через Adobe Sensei, що дозволяє відновлювати деталі зображень, покращувати різкість та масштабувати фотографії [9]. Переваги Photoshop полягають у високій стабільності та широких функціональних можливостях, проте використання нейронних методів у ньому обмежене обчислювальними ресурсами та може вимагати значного часу для обробки великих зображень.

Topaz Gigapixel AI спеціалізується на покращенні роздільної здатності зображень із застосуванням глибинного навчання (рисунок 1.2). Програма

використовує моделі суперроздільності, схожі на ESRGAN та Real-ESRGAN, що дозволяє відновлювати дрібні деталі, текстури та контури об’єктів, значно перевершуючи класичні методи.

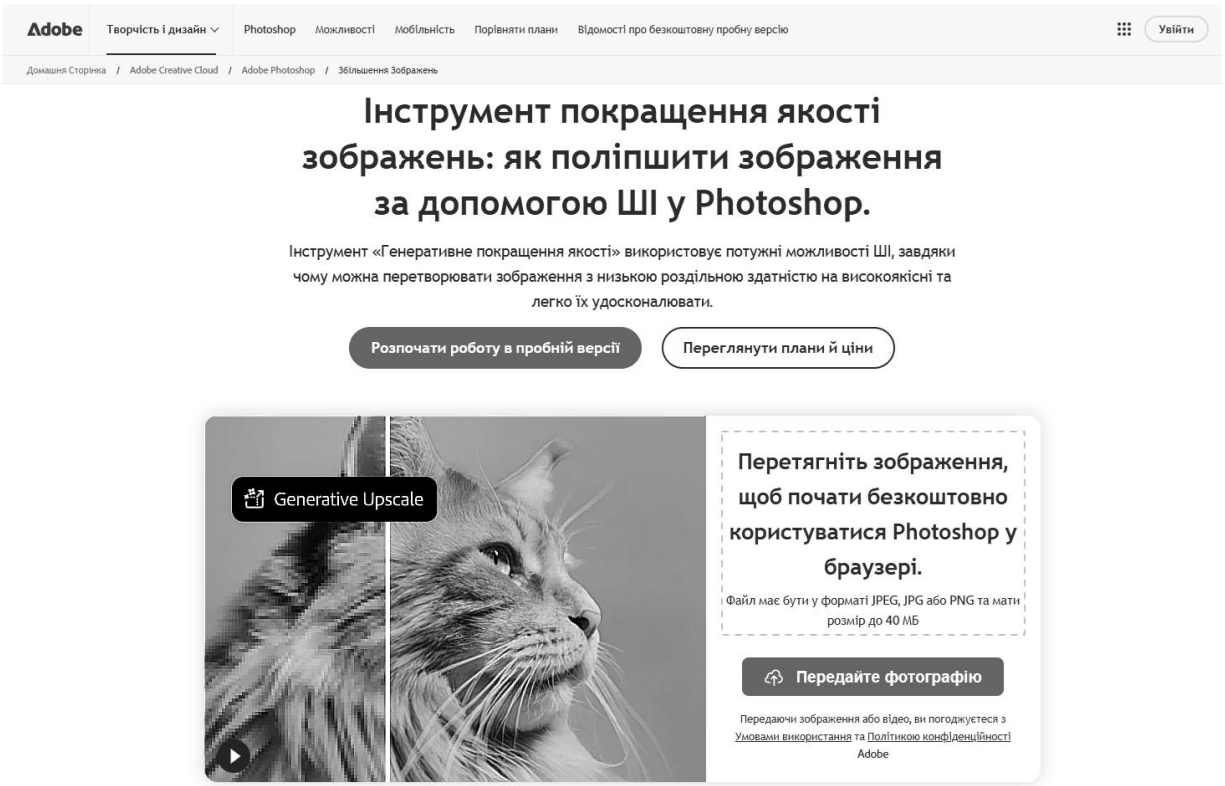

Рисунок 1.1 — Загальний вигляд інтерфейсного вікна Adobe Photoshop Upscaler

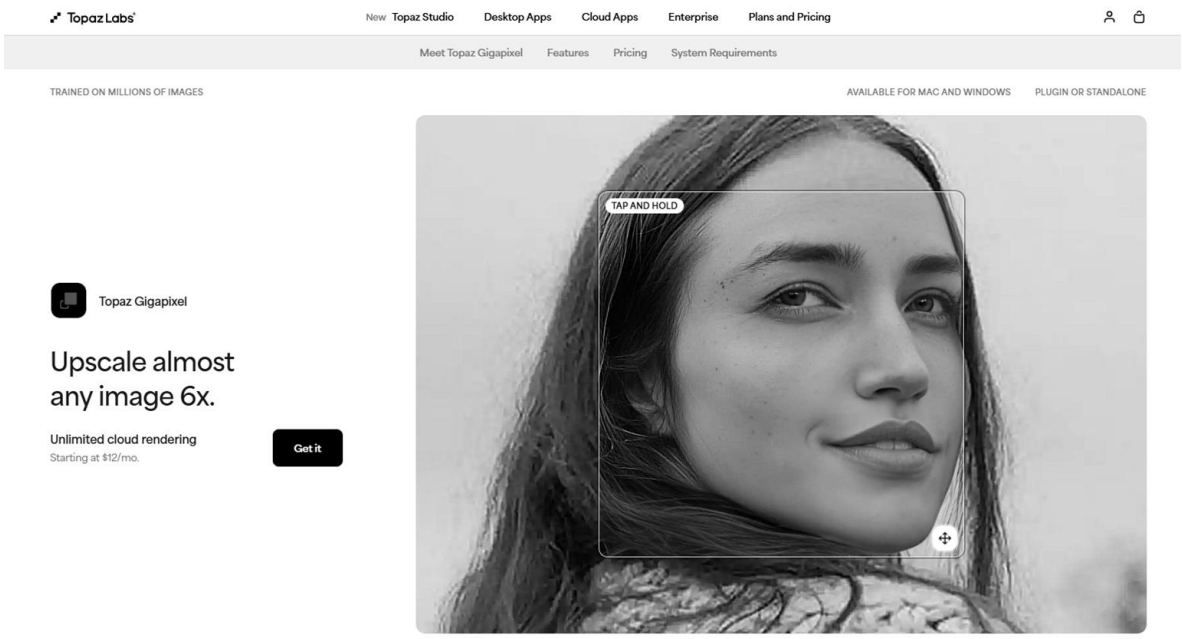

Рисунок 1.2 — Загальний вигляд інтерфейсного вікна Topaz Gigapixel AI

Основна перевага Gigapixel AI — висока якість результату при масштабуванні від 2 до 6 разів та можливість обробки фотографій зі складними текстурами, включаючи старі або розмиті знімки [10]. Недоліком є потреба у потужному обладнанні та відсутність можливості гнучкого редагування алгоритмів користувачем.

Waifu2x — популярний open-source інструмент, орієнтований на масштабування і шумозаглушення зображень, особливо у стилі аніме та ілюстрацій (рисунок 1.3). Він використовує згорткові нейронні мережі для усунення шумів та збільшення роздільної здатності [11]. Основні переваги Waifu2x — простота, безкоштовність та ефективна обробка малюнків і мультфільмів. Обмеження полягають у тому, що для фотографій реального світу модель менш ефективна, і інтерфейс обмежує гнучкість налаштувань.

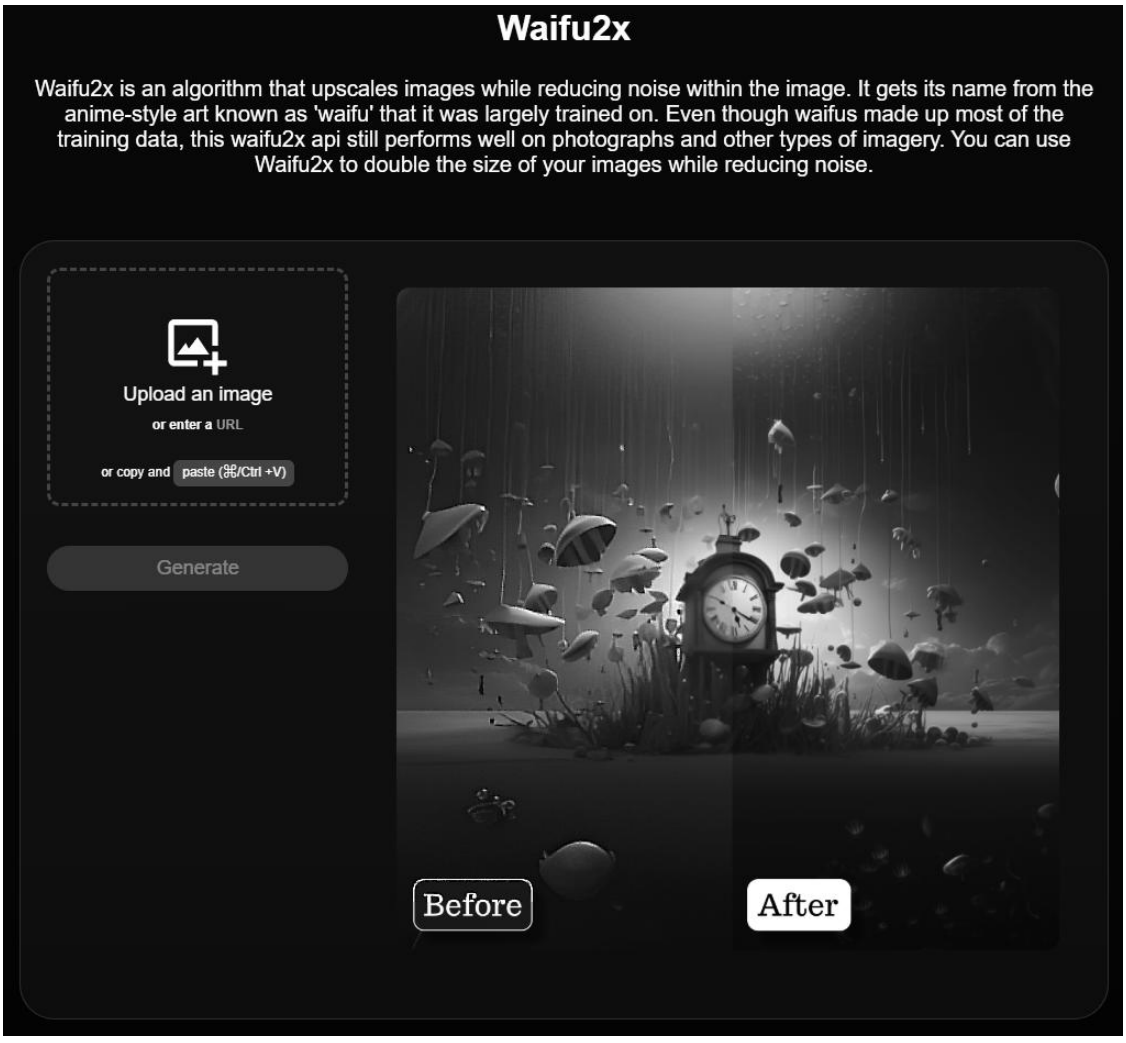

Рисунок 1.3 — Загальний вигляд інтерфейсного вікна Waifu2x

Серед open-source рішень виділяються бібліотеки та проєкти на базі PyTorch та TensorFlow, що реалізують SRCNN, ESRGAN, Real-ESRGAN та diffusion models. Вони дозволяють інтегрувати нейронні мережі безпосередньо у власне програмне забезпечення, налаштовувати параметри навчання і масштабування, а також комбінувати класичні та глибинні методи [7, 12]. Головні переваги — відкритість, можливість адаптації під конкретні задачі, висока точність відновлення деталей. Недоліки — потреба у налаштуванні, високі вимоги до GPU і відносно складна інтеграція в готові програми для кінцевого користувача.

Порівняльно, класичні редактори, такі як Photoshop, добре підходять для швидкої обробки і невеликих проектів, де критична стабільність і контроль. Спеціалізовані нейромережеві рішення (Topaz Gigapixel, Real-ESRGAN) забезпечують максимальну деталізацію та реалістичність, але вимагають більше ресурсів та часу. Open-source інструменти дозволяють поєднувати переваги обох підходів, забезпечуючи гнучкість і точність, але потребують базових навичок програмування та роботи з нейронними мережами.

На основі цього аналізу можна виділити ключові вимоги до власного програмного засобу: необхідність комбінувати класичні методи інтерполяції та фільтрації з нейронними мережами (Real-ESRGAN або подібними моделями), забезпечити зручний інтерфейс користувача, а також адаптивність до різних типів зображень, включаючи старі фотографії, скани, медичні знімки та кадри відео низької роздільної здатності.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

2.1 Основи цифрового зображення

Цифрове зображення визначається як двовимірна дискретна функція:

$$I(x, y): Z^2 \rightarrow R^n, \tag{2.1}$$

де x та y — просторові координати;

n — кількість каналів.

У випадку кольорових зображень n=3, що відповідає компонентам червоного, зеленого та синього кольорів. Яскравість кожного елемента (пікселя) має скінченну кількість градацій, зазвичай у межах:

$$I(x, y) \in [0,255]. \tag{2.2}$$

Піксель є елементарною структурною одиницею зображення та містить дискретне представлення локальної інтенсивності світла [2]. Зображення зі значеннями у діапазоні від 0 до 255 представляють собою 8-бітну глибину на канал, що забезпечує $2^8 = 256$ можливих рівнів інтенсивності. Для 24-бітних RGB-зображень повна кількість можливих кольорів дорівнює:

$$2^{24} = 16\,777\,216. \tag{2.3}$$

Кольорове зображення у просторі RGB задається трійкою функцій:

$$I(x, y) = \big(R(x, y),\ G(x, y),\ B(x, y)\big), \tag{2.4}$$

Кожна компонента так само лежить у дискретному діапазоні від 0 до 255.

Модель RGB є адитивною, тобто результуючий колір формується сумуванням трьох основних світлових компонент.

У низці алгоритмів покращення якості та стиснення використовується перехід у кольоровий простір YCbCr, де яскравісна та хромінансні складові відокремлюються [14]. Перетворення між RGB та YCbCr задається лінійною матрицею:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \quad (2.5)$$

У таблиці 2.1 наведено порівняння найпоширеніших кольорових просторів, що використовуються в системах покращення зображень.

Таблиця 2.1 — Порівняння кольорових просторів

Простір	Кількість каналів	Характеристика	Використання
RGB	3	Адитивна модель кольору	Відображення, нейромережі
YCbCr	3	Яскравість + хромінанс	JPEG, відео, SR-алгоритми
LAB	3	Перцептивно рівномірний	Корекція кольору
HSV	3	Компонентне представлення тону	Сегментація, маскування

Яскравість пікселя у градаціях сірого може бути описана функцією інтенсивності:

$$I(x, y) = L(x, y), \quad (2.6)$$

де L — скалярна величина, що відображає енергетичну складову світлового сигналу.

Для аналізу зображень важливим є розподіл інтенсивностей, який визначається гістограмою. Гістограма є дискретною оцінкою щільності ймовірності:

$$h(k) = \frac{N_k}{N}, \tag{2.7}$$

де N_k — кількість пікселів із рівнем яскравості k;

N — загальна кількість пікселів.

Нормалізована гістограма дозволяє виконувати корекцію контрасту та адаптивні перетворення.

Накопичена функція розподілу (CDF) визначається як:

$$H(k) = \sum_{i=0}^{k} h(i). \tag{2.8}$$

Вона використовується у класичних алгоритмах вирівнювання гістограми. У випадку адаптивних методів (CLAHE) гістограми обчислюються в локальних вікнах, що дозволяє уникати глобальних перенасичень [15].

Гамма-корекція використовується для нелінійного перетворення яскравості:

$$I_{out} = I_{in}^{\gamma}, \tag{2.9}$$

де $\gamma < 1$ — посилює темні області;

$\gamma > 1$ — зменшує контраст у світлих зонах.

Це дозволяє забезпечити кращу відповідність людському сприйняттю, оскільки система зору має логарифмічну чутливість.

Цифрові зображення можуть зберігатися у різних форматах, які відрізняються методами стиснення та збереження метаданих. У таблиці 2.2 наведено основні характеристики найпоширеніших форматів.

Файлові формати також включають метадані, зокрема EXIF, де зберігаються

параметри зйомки, орієнтація кадру, час експозиції, ISO та інші.

Таблиця 2.2 — Порівняння форматів графічних файлів

Формат	Тип стиснення	Втрата якості	Підтримка альфа-каналу	Призначення
JPEG	Дискретне косинусне перетворення (DCT)	Є	Немає	Фото, інтернет
PNG	Логарифмічне безвтратне стиснення	Немає	Є	Прозорість, графіка
TIFF	Втратне/безвтратне	Опціональне	Є	Професійна обробка
BMP	Без стиснення	Немає	Є	Сировинні дані
WEBP	Втратне/безвтратне	Опційно	Є	Веб-оптимізація

Орієнтація є критично важливою для коректної подальшої обробки, оскільки невиправлені EXIF-мітки призводять до неправильного відображення та спотворення результатів суперрезолюції. Позначення орієнтацій визначається стандартом TIFF/EXIF та подається у вигляді значень від 1 до 8 (таблиця 2.3).

Частотне представлення цифрового зображення базується на двовимірному дискретному перетворенні Фур’є:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x,y)\ e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \qquad (2.10)$$

Воно дозволяє аналізувати просторові частоти, визначати рівень деталізації та фільтрувати небажані компоненти. Обернене перетворення визначається як:

$$I(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\ e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}. \quad (2.11)$$

Таблиця 2.3 — Визначення EXIF-орієнтацій

Значення	Опис	Трансформація
1	Нормальна	—
2	Дзеркально горизонтально	Flip horizontally
3	Поворот 180°	Rotate 180°
4	Дзеркально вертикально	Flip vertically
5	Дзеркально по діагоналі	Transpose
6	Поворот 90° CW	Rotate 90° CW
7	Дзеркально по іншій діагоналі	Transverse
8	Поворот 90° CCW	Rotate 90° CCW

У практиці покращення зображень часто застосовується амплітудно-частотна характеристика (АЧХ) фільтрів, що дозволяє формально описати реакцію фільтрів низьких та високих частот. Наприклад, ідеальний низькочастотний фільтр має частотну характеристику:

$$H(u,v) = \begin{cases} 1, & \sqrt{u^2+v^2} \le D_0, \\ 0, & \text{інакше}, \end{cases} \quad (2.12)$$

де D_0 — радіус пропускної смуги.

Проте у реальних задачах використовують апроксимації (Гаусів, Баттерворт), оскільки ідеальний фільтр спричиняє рингінг-артефакти.

Частотний аналіз є важливим, оскільки операції масштабування безпосередньо впливають на спектральний розподіл. Збільшення роздільності призводить до інтерполяції відсутніх частот, а отже до необхідності контролю

високочастотних компонент, що мають вирішальний вплив на сприйняття різкості. У зв’язку з цим у подальших методах значення матимуть як просторове, так і частотне трактування операцій [2, 16].

У таблиці 2.4 представлено порівняння градієнтних операторів за їх властивостями.

Таблиця 2.4 — Порівняння операторів для виділення градієнтів

Оператор	Чутливість до шуму	Точність меж	Обчислювальна складність	Особливості
Робертса	Висока	Низька	Низька	Дуже локальний
Превітта	Середня	Середня	Низька	Аналог Собеля з простішими коефіцієнтами
Собеля	Середня	Висока	Середня	Добре згладжує шум за рахунок вагових коефіцієнтів
Шарра	Низька	Дуже висока	Висока	Краща ізотропність, точне виділення меж

Для оцінювання локальних властивостей використовуються градієнти яскравості. Градієнт визначається як:

$$\nabla I(x, y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right), \tag{2.13}$$

$$|\nabla I(x, y)| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}. \tag{2.14}$$

Градієнтні оператори (Собеля, Превітта, Робертса) використовуються для аналізу різкості, меж та переходів, що важливо для подальших задач збільшення деталізації. Класичні ядра Собеля задаються як:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \tag{2.15}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{2.16}$$

Таким чином, основні властивості цифрових зображень визначаються поєднанням просторових, частотних та статистичних характеристик, що створює фундамент для опису алгоритмів інтерполяції, попередньої обробки та суперрезолюції, які розглядаються в наступних підрозділах.

2.2 Математичні основи інтерполяції

Інтерполяція визначає значення зображення у нових координатах під час масштабування, повороту або афінних перетворень. Нехай вихідне зображення подано у вигляді функції:

$$I: Z^2 \to R. \tag{2.17}$$

Під час масштабування із коефіцієнтом s координати нового зображення задаються як:

$$(x', y') = (sx, sy). \tag{2.18}$$

Оскільки x' і y' у загальному випадку не є цілими числами, значення інтенсивності для нових точок обчислюється за допомогою інтерполяційної функції:

$$I'(x', y') = \mathcal{F}\big(I(x, y)\big), \tag{2.19}$$

де $\mathcal{F}$ — відповідний інтерполяційний оператор.

Найпростішим підходом є інтерполяція найближчого сусіда:

$$I'(x', y') = I\big(\text{round}(x'), \text{round}(y')\big), \tag{2.20}$$

що забезпечує високу швидкість, але призводить до появи ступінчатих переходів та втрати плавності.

Лінійна двовимірна інтерполяція використовує значення чотирьох сусідніх пікселів. Для точки (x′,y′) визначаються координати базового вузла та дробові складові:

$$x = \lfloor x' \rfloor, \quad y = \lfloor y' \rfloor, \tag{2.21}$$

$$a = x' - x, \quad b = y' - y. \tag{2.22}$$

З урахуванням цих значень результат двовимірної інтерполяції задається формулою:

$$\begin{aligned} I'(x', y') = {} & (1-a)(1-b)\, I(x, y) + a(1-b)\, I(x+1, y) + \\ & + (1-a)b\, I(x, y+1) + ab\, I(x+1, y+1). \end{aligned} \tag{2.23}$$

Цей метод забезпечує плавні переходи інтенсивності та значно зменшує артефакти на межах порівняно з найближчим сусідом.

Бікубічна інтерполяція забезпечує значно вищу якість відновлення порівняно з лінійними методами, оскільки враховує не лише значення сусідніх пікселів, але й форму локальної поверхні інтенсивності. Підхід базується на апроксимації функції зображення двовимірним кубічним поліномом, що дозволяє відтворювати плавні градієнти та зберігати більше високочастотних деталей [17].

Метод оперує сіткою 4x4 сусідніх точок, які оточують координати (x′,y′). Основою методу є одновимірна кубічна інтерполяційна функція w(t), яку застосовують окремо вздовж кожної осі. Найпоширеніший її варіант має вигляд:

$$w(t) = \begin{cases} (a+2)|t|^3 - (a+3)|t|^2 + 1, & |t| < 1, \\ a|t|^3 - 5a|t|^2 + 8a|t| - 4a, & 1 \le |t| < 2, \\ 0, & |t| \ge 2, \end{cases} \qquad (2.24)$$

де параметр a зазвичай дорівнює −0.5 (інтерполяція Мітчелла–Нетрвалі) або −0.75 (більш згладжений варіант).

У двовимірному випадку значення в точці (x′,y′) обчислюється як подвійне згортання:

$$I'(x', y') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} I(x+m, y+n)\, w(a-m)\, w(b-n). \qquad (2.25)$$

Завдяки більш плавним ваговим коефіцієнтам бікубічний метод відтворює дрібні деталі помітно точніше, ніж лінійна інтерполяція, та зберігає природність контурів. У той же час використання ширшої області сусідства збільшує обчислювальну складність, що може бути критичним при обробці великих зображень або на пристроях зі слабким процесором.

Наведена нижче таблиця 2.5 узагальнює ключові властивості найбільш уживаних інтерполяційних методів, що дозволяє виконати їх порівняння.

Таблиця 2.5 — Порівняння базових методів інтерполяції

Метод	Розмір області	Якість градієнтів	Різкість контурів	Швидкість
Nearest Neighbor	1x1	Низька	Погана, «сходи»	Дуже висока
Bicubic	4x4	Висока	Добра, плавна	Середня
Bilinear	2x2	Середня	Помірно згладжені	Висока
Lanczos	Залежить від параметра a	Дуже висока	Краща серед класичних	Нижча

Інтерполяція Lanczos ґрунтується на використанні віконної апроксимації функції sinc, яка є ідеальним інтерполювальним ядром у теорії сигналів. На практиці sinc-функцію не можна застосовувати безпосередньо через її нескінченну довжину, тому використовується обмежений варіант, який зберігає основні частотні властивості, але є обчислювально доцільним [3, 18].

Базове ядро методу визначається функцією:

$$L(x) = \begin{cases} \mathrm{sinc}(x)\mathrm{sinc}\left(\frac{x}{a}\right), & |x| < a \\ 0, & |x| \geq a \end{cases}, \tag{2.26}$$

де — a параметр, що задає ширину вікна та визначає якість згладжування.

Типові значення:

— a = 2 — Lanczos-2 (баланс між якістю та швидкістю);

— a = 3 — Lanczos-3 (один із найпоширеніших варіантів у графічних редакторах).

Функція sinc(x) записується як:

$$\mathrm{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x}, & x \neq 0 \\ 1, & x = 0 \end{cases}. \tag{2.27}$$

Двовимірна інтерполяція виконується як добуток незалежних одновимірних ядер:

$$I'(x', y') = \sum_{m=-a+1}^{a} \sum_{n=-a+1}^{a} I(x + m, y + n)\ L(a - m)\, L(b - n). \tag{2.28}$$

Вона забезпечує симетричну і плавну інтерполяцію завдяки властивостям sinc-функції. Використання множників sinc(x/a) створює вікно, яке пригнічує високочастотні компоненти, що могли б викликати рингінг-артефакти.

Порівняно з бікубічним методом, Lanczos точніше відтворює дрібні деталі та забезпечує вищу різкість контурів. Завдяки цьому він часто застосовується у

професійних програмах для масштабування фотографій та у високоякісних відеоплеєрах. Водночас збільшена складність обчислень робить метод менш ефективним для реального часу на слабких пристроях.

Ефективність інтерполяційного методу залежить не лише від математичного ядра, а й від того, наскільки точно він передає локальну структуру зображення. Під час збільшення масштабу важливим є баланс між збереженням різкості, пригніченням шуму та уникненням артефактів типу рингінгу. У цьому контексті інтерполяція Lanczos завдяки своїм частотним характеристикам забезпечує одну з найкращих апроксимацій, однак її обчислювальна вартість є значно вищою, ніж у лінійних методів.

У частотному аналізі різниця між методами особливо помітна: бікубічна інтерполяція відносно м’яко пригнічує високі частоти, у той час як Lanczos зберігає їх значно краще, що робить контури чіткішими. Однак ця властивість також підвищує ймовірність появи слабких осциляцій навколо різких переходів, особливо якщо вхідне зображення містить шум або компресійні артефакти.

Важливою характеристикою інтерполяційних ядер є їхня просторово-частотна локалізація. Методи, засновані на вузькій області сусідства (nearest, bilinear), демонструють високу швидкість, але втрачають дрібні деталі. Ширші ядра (bicubic, Lanczos) забезпечують точнішу реконструкцію, проте потребують більше обчислень і є більш чутливими до якості вихідних даних.

Комплексний огляд наведених методів дозволяє обґрунтовано вибирати інтерполяційну схему залежно від поставленої задачі та якості початкового зображення. У подальших розділах глибша увага приділятиметься підходам, які передбачають поєднання класичних та нейромережевих методів збільшення роздільності, що дозволяє досягти вищої якості реконструкції у практичних застосуваннях.

Для наочного узагальнення властивостей основних інтерполяційних підходів доцільно представити їх характерні ознаки у порівняльній таблиці 2.6.

Таблиця 2.6 — Порівняння властивостей інтерполяційних методів

Метод	Збереження деталей	Ризик рингінгу	Згладженість	Потреба в обчисленнях	Типові застосування
Nearest	Дуже низьке	Нульовий	Немає	Мінімальна	Піксель-арт, прев’ю
Bilinear	Низьке	Нульовий	Висока	Низька	Швидке збільшення зображення
Bicubic	Високе	Низький	Помірна	Середня	Фото, графіка, поліграфія
Lanczos-2	Високе	Середній	Низька	Середня–висока	Відео, професійне масштабування
Lanczos-3	Дуже високе	Помітний при шумі	Низька	Висока	Високоточне масштабування деталей

2.3 Класичні методи попередньої обробки зображень

Класичні методи попередньої обробки забезпечують первинне покращення структури зображення перед застосуванням складніших алгоритмів. Їх метою є зменшення шуму, згладжування артефактів, корекція контрасту та підготовка даних до подальших перетворень. Ці методи ґрунтуються на математичних операторах згортки, локальних статистичних оцінках та фільтрації у просторі чи частотній області.

Фільтри згладжування є одним із базових класів операцій попередньої обробки. Вони зменшують вплив випадкового шуму шляхом усереднення локальних значень інтенсивності. Найпростішим різновидом є лінійний фільтр з маскою ядра:

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} h(i,j)\ I(x+i,\ y+j), \qquad (2.29)$$

де h(i,j) — коефіцієнти фільтра;

k — радіус вікна.

Властивості фільтра залежать від вибору ядра, що дозволяє регулювати ступінь згладжування і форму імпульсної характеристики.

Одним із найпоширеніших варіантів є гаусів фільтр, ядро якого визначається функцією нормального розподілу. Двовимірний гаусів фільтр задається формулою:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \qquad (2.30)$$

Збільшення σ призводить до сильнішого розмивання, завдяки чому зменшується шум, проте втрачаються дрібні деталі.

На відміну від лінійних фільтрів, медіанний фільтр базується на порядковій статистиці та особливо ефективний проти імпульсного шуму (“сіль і перець”) [19]. Значення пікселя після обробки визначається як медіана інтенсивностей у локальному вікні:

$$I'(x,y) = \text{median}\{I(x+i,\ y+j)\}, \qquad (2.31)$$

де i,j ∈ [−k;k].

Метод практично не розмиває контури, оскільки замінює значення не середнім, а центральним за порядком.

Ще одним важливим підходом до фільтрації є білінійний (bilateral) фільтр, який враховує як просторову близькість пікселів, так і подібність їхніх значень. Його дія описується як подвійне зважування:

$$I'(x,y) = \frac{1}{W} \sum_{i,j} I(x+i, y+j)\ f_s(i,j)\, f_r\big(I(x+i,y+j) - I(x,y)\big), \qquad (2.32)$$

де f_s — просторове згасання (зазвичай гаусівське);

f_r — радіальне згасання за інтенсивністю;

W — нормуючий коефіцієнт.

Такий фільтр добре пригнічує шум, зберігаючи при цьому краї.

Для аналізу ефективності фільтрів корисним є спектральне представлення їхніх передавальних функцій. Наприклад, гаусів фільтр у частотній області має вигляд:

$$H(u,v) = \exp\left(-2\pi^2\sigma^2(u^2+v^2)\right). \quad (2.33)$$

Окремим класом задач попередньої обробки є усунення JPEG-артефактів, що виникають унаслідок блокового дискретного косинусного перетворення. До таких артефактів належать помітні 8x8 блоки, рингінг на межах контрасту та втрати локальних деталей. Для їх зменшення застосовуються згладжувальні та адаптивні фільтри, які зменшують різкі зміни між блоками. Поширеним підходом є локальне згладжування хромінансних компонент або застосування слабкого гаусівського фільтра з малим σ, що зменшує блоковість, але не надто розмиває зображення.

Ефективність розглянутих фільтрів значною мірою залежить від типу шуму, який присутній у зображенні. Для випадкового (гаусівського) шуму найкраще працюють лінійні згладжувальні фільтри, оскільки вони знижують дисперсію інтенсивностей, зберігаючи загальну структуру сигналу. Для імпульсного шуму медіанний фільтр залишається найбільш ефективним, тому що повністю усуває поодинокі яскраві або темні викиди. У ситуаціях, коли важливо зберегти контури, доцільним є застосування білінійного фільтра, який поєднує згладжування та збереження різкості [20].

Для систематизації основних властивостей класичних фільтрів наведено порівняльну таблицю 2.7, в якій зазначено їх ефективність щодо різних типів шуму та поведінку на краях зображень.

Крім фільтрації, важливим етапом попередньої обробки є корекція контрасту. Одним із найпоширеніших методів є глобальне вирівнювання гістограми, яке

посилює контраст шляхом розтягування динамічного діапазону яскравостей.

Таблиця 2.7 — Порівняння фільтрів попередньої обробки

Фільтр	Тип шуму	Збереження країв	Можливі артефакти	Обчислювальна складність
Гаусів	Гаусівський	Низьке	Розмивання	Низька
Медіанний	Імпульсний	Високе	Можливе спотворення тонких ліній	Середня
Bilateral	Будь-який з локальною структурою	Високе	Нелінійні спотворення при сильному шумі	Висока
Box filter	Будь-який слабкий шум	Низьке	Висока ступінь розмивання	Дуже низька

Перетворення визначається функцією накопиченої гістограми:

$$I'(x,y) = H\big(I(x,y)\big) \cdot (L-1), \tag{2.34}$$

де H(k) — нормалізована CDF;

L — кількість рівнів інтенсивності (зазвичай 256).

Цей метод забезпечує рівномірний розподіл яскравостей, але іноді спричиняє перенасичення світлих або темних областей.

Адаптивний варіант — локальне вирівнювання гістограми (CLAHE), у якому зображення поділяється на блоки фіксованого розміру. Для кожного блока обчислюється локальна CDF, а переповнені значення обмежуються для запобігання надмірного підсилення шуму. Отже, CLAHE поєднує посилення контрасту з обмеженням артефактів, що часто робить його більш придатним для попередньої обробки перед алгоритмами підвищення роздільності.

У задачах попереднього згладжування важливо враховувати і частотні властивості операцій. Наприклад, фільтр низьких частот для згладжування визначається функцією:

$$H(u,v) = \exp\left(-\frac{u^2+v^2}{2D_0^2}\right), \quad (2.35)$$

де D_0 — параметр, що регулює силу пригнічення високих частот.

Така форма дозволяє контролювати баланс між зменшенням шуму та збереженням деталей.

Окрему увагу слід приділити зменшенню артефактів JPEG, що виникають унаслідок блокової структури DCT. Одним із ефективних способів приглушення блокових переходів є застосування слабкого гаусівського згладжування до хромінансних компонент після перетворення до YCbCr. Враховуючи те, що людське око менш чутливе до кольорових змін, така операція зменшує блоковість без значної втрати деталізації в яскравісній складовій [21].

Для формального опису адаптивної антиблокувальної обробки використовується комбінована фільтрація з урахуванням різниці між сусідніми блоками:

$$I'(x,y) = I(x,y) - \alpha \cdot \Delta_B(x,y), \quad (2.36)$$

де ΔB(x,y)) — різниця інтенсивностей на межі блоків;

α — коефіцієнт згладжування.

Такий підхід усуває різкі перепади між блоками, не впливаючи суттєво на внутрішню структуру зображення.

Серед задач попередньої обробки важливим є не лише приглушення шуму чи згладжування блокових артефактів, але й відновлення різкості після фільтрації. Одним із найпоширеніших методів підсилення деталей є різницевий фільтр підвищення різкості. Він ґрунтується на виділенні високочастотної складової та її

повторному додаванні до вихідного зображення. Основні кроки включають згладжування зображення, обчислення різниці та масштабоване підсилення:

$$I'(x,y) = I(x,y) + k\big(I(x,y) - I_s(x,y)\big), \tag{2.37}$$

де $I_s(x,y)$ — згладжене зображення;

k — коефіцієнт підсилення.

При достатньо малому k метод робить текстури чіткішими та покращує сприйняття контурів. При надмірному підсиленні можуть з’являтися яскраві ореоли навколо меж, тому вибір параметрів має суттєвий вплив на кінцевий результат.

Іншим способом підвищення різкості є застосування лапласіанового фільтра. Оператор Лапласа визначається другою похідною за обома координатами і виділяє області різких змін інтенсивності:

$$\nabla^2 I(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}. \tag{2.38}$$

Це дає можливість підсилити локальну структуру, однак у чистому вигляді лапласіанське підсилення може збільшувати шум. Тому на практиці використовується комбінація фільтра Лапласа та гаусівського згладжування, що подається як оператор LoG (Laplacian of Gaussian) [22]. Його імпульсна характеристика визначається формулою:

$$LoG(x,y) = \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right), \tag{2.39}$$

де σ — регулює масштаб структури, що підсилюється.

Такий підхід дозволяє виділяти контури різних рівнів деталізації та водночас зменшувати вплив випадкового шуму.

До задач попередньої обробки також належить корекція освітленості та локального контрасту. Часто вхідні зображення містять нерівномірні градієнти

освітлення, що можуть спотворювати локальну різкість або призводити до втрати деталей у затемнених областях. Для корекції можна використовувати модель розділення зображення на освітленість та відбиття:

$$I(x,y) = L(x,y) \cdot R(x,y), \tag{2.40}$$

де L(x,y) — повільно змінна складова освітлення;

R(x,y) — відбита текстурна структура.

Локальне згладжування L(x,y) дозволяє оцінити освітлення, після чого отримується скориговане зображення:

$$R(x,y) = \frac{I(x,y)}{L_s(x,y)}, \tag{2.41}$$

де $L_s(x,y)$ — згладжена оцінка освітленості.

Такий метод покращує баланс світлих і темних областей, підвищуючи результативність подальшої суперрезолюції.

Для повноти аналізу класичних методів доцільно оцінити їх у контексті попередньої обробки для моделей підвищення роздільності. Оскільки нейромережеві моделі особливо чутливі до шуму, структурних дефектів і невідповідності освітлення, класичні підходи виконують роль стабілізатора вхідних даних, зменшуючи випадкові варіації та покращуючи узгодженість текстур (таблиця 2.8).

Комбінація класичних методів дає можливість створити адаптивний ланцюг попередньої обробки, у якому кожен елемент спрямований на усунення конкретних недоліків сигналу.

Застосування таких методів формує основу стабільного подальшого підсилення роздільності та покращення структури, особливо в умовах низькоякісних, зашумлених або нерівномірно освітлених зображень.

Таблиця 2.8 — Роль класичних фільтрів у підготовці зображень до суперрезолюції

Метод	Вплив на SR-моделі	Переваги	Недоліки
Гаусів фільтр	Зменшує шум	Стабілізує вхід; покращує збіжність	Розмиває дрібні деталі
Медіанний фільтр	Усуває імпульсний шум	Зберігає краї	Може викривляти текстури
Bilateral фільтр	Зберігає структуру	Чіткі краї + згладжування	Висока вартість
CLAHE	Підвищує контраст	Покращує виявлення дрібних деталей	Може підсилювати шум
Анти-JPEG фільтри	Зменшують блоковість	Вирівнюють структуру	Не усувають DCT-шум повністю

2.4 Нейромережеві методи покращення зображень

Нейромережеві методи покращення зображень ґрунтуються на здатності глибоких моделей відновлювати високочастотні компоненти, які були втрачені під час зменшення роздільності, шуму чи компресії. На відміну від класичних інтерполяційних методів, які відтворюють значення пікселів на основі локальної структури, нейронні мережі використовують статистичні залежності, отримані з великих наборів даних, що дозволяє реконструювати текстури, недоступні традиційним підходам [23].

У більшості моделей підвищення роздільності вихідне низькороздільне зображення позначається як I_{LR}, а цільове високороздільне — як I_{HR}. Мета моделі полягає у наближенні функції відображення:

$$I_{SR} = F_\theta(I_{LR}), \tag{2.42}$$

де F_θ — нейронна мережа з параметрами θ;

I_{SR} — реконструйоване зображення.

Базою більшості SR-моделей є згортка, яка для кожного каналу застосовує

ядро K до локальної області. Операція згортки визначається формулою:

$$(\mathrm{F} * \mathrm{K})(\mathrm{x}, \mathrm{y}) = \sum_{\mathrm{i}=-\mathrm{k}}^{\mathrm{k}} \sum_{\mathrm{j}=-\mathrm{k}}^{\mathrm{k}} \mathrm{F}(\mathrm{x}+\mathrm{i}, \mathrm{y}+\mathrm{j})\ \mathrm{K}(\mathrm{i}, \mathrm{j}). \quad (2.43)$$

Згортки дозволяють моделі виділяти структури різної складності: від простих горизонтальних чи вертикальних граней до складних текстур, характерних для певних типів зображень.

CNN-архітектури зазвичай складаються з послідовностей згорткових блоків з нелінійністю. Однією з найпоширеніших активацій є ReLU:

$$\mathrm{ReLU}(\mathrm{x}) = \max(0, \mathrm{x}). \quad (2.44)$$

Вона покращує збіжність моделі та зменшує ефект затухання градієнтів.

Процес навчання SR-моделі зводиться до мінімізації різниці між результатом мережі та справжнім високороздільним зображенням. Найпростішою є евклідова функція втрат (L2-loss):

$$\mathcal{L}_{\mathcal{L}2} = |\mathrm{I}_{\mathrm{SR}} - \mathrm{I}_{\mathrm{HR}}|_2^2. \quad (2.45)$$

Вона забезпечує стабільне навчання, але може приводити до надмірного згладжування.

Альтернативою є абсолютна похибка (L1-loss):

$$\mathcal{L}_{\mathcal{L}1} = |\mathrm{I}_{\mathrm{SR}} - \mathrm{I}_{\mathrm{HR}}|_1. \quad (2.46)$$

Вона краще зберігає різкі переходи та текстурні елементи.

Для підвищення природності структури зображень використовують перцептивні функції втрат, що оцінюють різницю у просторі ознак, виділених попередньо навченою мережею (наприклад, VGG):

$$\mathcal{L}_{\text{pr}} = |\phi(\text{I}_{\text{SR}}) - \phi(\text{I}_{\text{HR}})|_2^2, \quad (2.47)$$

де $\phi(\cdot)$ — вихід певного шару згорткової мережі.

Цей метод дає можливість зберігати природність текстур навіть при слабкій початковій структурі.

У моделях, подібних до SRGAN або ESRGAN, використовується додатковий дискримінатор, який навчається відрізняти реконструйовані зображення від справжніх [24]. Функція втрат генератора включає змагальний термін:

$$\mathcal{L}_{\text{adv}} = -\log \text{D}\,(\text{I}_{\text{SR}}). \quad (2.48)$$

Вона стимулює утворення реалістичних високочастотних деталей.

Методи на основі GAN забезпечують найбільш виразні результати, але можуть спричиняти появу штучних текстур, якщо навчання відбувається нерівномірно або вхідні дані містять шум.

Одним із перших успішних підходів у суперрезолюції стала модель SRCNN, яка складалася лише з трьох згорткових шарів і демонструвала різке покращення якості порівняно з класичними методами. Її архітектура включала операції витягування ознак, нелінійного перетворення та реконструкції. Формально перший шар моделі обчислював карту ознак:

$$\text{F}_1 = \sigma(\text{W}_1 * \text{I}_{\text{LR}} + \text{b}_1). \quad (2.49)$$

де W_1 та b_1 — ваги та зсув першого шару;

σ — функція активації.

Незважаючи на простоту, модель демонструвала суттєве покращення якості, проте мала обмежену здатність до відтворення високочастотних текстур.

Еволюцією цієї ідеї стала архітектура EDSR, яка усуває нормалізаційні шари та використовує глибокі залишкові блоки з широкими фільтрами. Залишкове з’єднання задається рівнянням:

$$F_{out} = F_{in} + H(F_{in}), \tag{2.50}$$

де $H(\cdot)$ — послідовність згорткових шарів усередині блоку.

Така конструкція допомагає уникати зникання градієнтів та стабілізує навчання дуже глибоких мереж.

У сучасних SR-моделях важливу роль відіграє операція піксельного піднесення, яка дозволяє ефективно збільшувати роздільність без втрат інформації. Нехай у моделі формується тензор TTT розмірності $H \times W \times r^2C$, де r — коефіцієнт масштабування. Операція піксельного рішафлінгу визначає перетворення:

$$I_{SR} = PS(T), \tag{2.51}$$

де PS — перестановка елементів тензора, що збирає субпікселі в нову, збільшену решітку.

Великий вплив на якість SR зробили генеративні моделі, зокрема ESRGAN, яка переглянула структуру залишкового блоку та замінила прямі залишкові зв’язки на щільні [24]. Один із ключових компонентів моделі — residual-in-residual dense block (RRDB) — описується як вкладення трьох залишкових блоків, кожен із яких містить щільні з’єднання між шарами. Для одного такого блоку вихідний сигнал обчислюється як:

$$F_{RRDB} = F_{in} + \beta \cdot H_{RRDB}(F_{in}), \tag{2.52}$$

де $\beta<1$ — коефіцієнт стабілізації, що зменшує амплітуду градієнтів.

У новітніх підходах, таких як SwinIR, на перший план виходять трансформерні архітектури. Вони використовують віконовану самоувагу для обробки зображення блоками, що істотно зменшує обчислювальну складність. Механізм самоуваги визначається як:

$$\text{Attention}(\text{Q},\text{K},\text{V}) = \text{softmax}\left(\frac{\text{QK}^{\text{T}}}{\sqrt{\text{d}}}\right)\text{V}, \quad (2.53)$$

де Q,K,V — запити, ключі та значення;

d — розмірність ознак.

Такий підхід дає змогу моделі враховувати залежності на великих відстанях, що особливо корисно для структур з повторюваними або регулярними текстурами.

Сучасні SR-моделі поєднують різні типи втрат — L1/L2, перцептивні та адвесаріальні — щоб забезпечити як точне наближення оригіналу, так і природність сприйняття. Комбінована функція втрат може мати вигляд:

$$\mathcal{L} = \lambda_1\mathcal{L}_{\mathcal{L}1} + \lambda_2\mathcal{L}_{perc} + \lambda_3\mathcal{L}_{adv}, \quad (2.54)$$

де λ1,λ2,λ3 регулюють внесок окремих компонентів.

2.5 Порівняльний аналіз сучасних моделей покращення зображень

Сучасні моделі покращення зображень розвивалися у кількох напрямках, що зумовило появу різноманітних архітектурних рішень. Хоча більшість з них належать до класу згорткових або трансформерних мереж, відмінності у структурі блоків, глибині мережі та способах реконструкції деталей суттєво впливають на характер отриманого результату. Тому порівняльний аналіз цих моделей є важливою складовою для розуміння їх переваг та обмежень перед переходом до розробки власного підходу.

Першим кроком еволюції стали базові згорткові архітектури, такі як SRCNN та її наступники [23]. Вони використовують послідовності згорткових операцій, де кожен шар обробляє локальні фрагменти зображення. Типовий вихід згорткового шару описують:

$$\text{F}_{\text{out}}(\text{x},\text{y}) = \sigma\big((\text{F} * \text{W})(\text{x},\text{y}) + \text{b}\big). \quad (2.55)$$

Ці методи показали, що навіть невелика кількість згорткових шарів здатна

значно покращувати якість, однак існували обмеження, пов’язані з недостатньою здатністю моделювати складні текстури або далекі залежності.

Подальший розвиток привів до появи залишкових мереж. Залишковий зв’язок компенсує деградацію якості при збільшенні глибини та покращує стабільність навчання. Формально його подають як:

$$F_{res} = F_{in} + H(F_{in}). \tag{2.56}$$

Ця ідея стала фундаментальною у моделях EDSR, ESRGAN та їх численних модифікаціях.

Окремим напрямком стали моделі на основі щільних зв’язків. Вони забезпечують надходження інформації від кожного шару до всіх наступних, що підсилює збереження локального контексту та дозволяє отримати багатші представлення ознак. У таких моделях баланс між багатством ознак та обчислювальною складністю стає ключовим фактором.

Значний прорив у якості реконструкції відбувся після впровадження трансформерних архітектур. Вони використовують механізм самоуваги, що дозволяє враховувати глобальні залежності між різними частинами зображення, незалежно від просторової відстані. На практиці це суттєво покращує роботу з регулярними структурами та дрібними текстурами, які важко моделювати класичним CNN. Хоча трансформерні моделі потребують більше ресурсів, їх здатність адаптуватися до складних високочастотних деталей значно підвищує якість кінцевого результату [25].

Сучасні підходи також відрізняються способами збільшення роздільності. Серед найефективніших — sub-pixel операції та транспоновані згортки. Вони дозволяють моделі відновлювати зображення у високій роздільності без значних втрат інформації. Водночас такі операції часто вимагають додаткових згорткових блоків для компенсації небажаних артефактів, що виникають на етапі масштабування.

Порівнюючи сучасні архітектури покращення зображень, важливо

враховувати не лише їх структурні особливості, але й поведінку моделей у різних режимах деградації зображень. Різні підходи по-різному реагують на шуми, артефакти компресії та втрати деталей. Наприклад, класичні CNN-моделі добре справляються зі згладженням однорідних областей, але іноді демонструють недостатню здатність відтворювати складні текстури. GAN-архітектури навпаки, здатні формувати багаті високочастотні структури, проте можуть створювати неприродні деталі за умов неправильного підбору параметрів або надто агресивного навчання.

Важливою характеристикою моделей є їхня стійкість до шумів та артефактів, що часто зустрічаються у реальних даних. Моделі, адаптовані до реальних деградацій, такі як Real-ESRGAN, демонструють високу стабільність, оскільки враховують складні шуми, зміни кольору, локальні перекручення та артефакти, спричинені стисненням. Це робить їх ефективними у прикладних задачах, де вхідні дані є нерегулярними та сильно деградованими.

Архітектури, що використовують механізми самоуваги, показують кращу здатність моделювати глобальні залежності між елементами зображення. Це забезпечує точніше відтворення регулярних структур, наприклад фактур тканини, повторюваних геометричних патернів чи природних структур. У таких моделях взаємодія між елементами ознак відбувається не через фіксовані вікна згортки, а через зважену комбінацію інформації з різних областей.

Сучасні моделі відрізняються також стратегіями масштабування, використанням нормалізаційних шарів, кількістю параметрів та доступною швидкістю обробки. Наприклад, EDSR уникає нормалізації, роблячи акцент на стабільність глибоких моделей, тоді як новітні трансформерні архітектури використовують віконовану нормалізацію та ефективні механізми обробки локальних блоків. Моделі з великою кількістю параметрів часто демонструють кращу якість, але потребують суттєвої обчислювальної потужності, що може бути важливо при розгортанні у реальних системах.

Узагальнюючи ключові властивості основних архітектур, доцільно представити їх у таблиці 2.9 з акцентом на перевагах, недоліках та сферах

застосування.

Таблиця 2.9 — Порівняльні характеристики сучасних моделей покращення зображень

Архітектура	Сильні сторони	Недоліки	Найкраще підходить для
SRCNN	Простота, низькі вимоги до ресурсів	Обмежена деталізація	Мобільні та легкі системи
EDSR	Висока точність реконструкції	Великий обсяг параметрів	Професійна ретуш та обробка
ESRGAN	Реалістичні високочастотні деталі	Може створювати штучні артефакти	Реставрація зображень
SwinIR	Відтворення складних текстур, глобальні зв’язки	Висока обчислювальна вартість	Високоякісна SR
Real-ESRGAN	Стійкість до реальних деградацій	Менша “художня” деталізація	Побутові та промислові застосування

Таким чином, сучасні моделі покращення зображень відрізняються не лише архітектурою та принципом роботи, але й спеціалізацією. Одні підходи забезпечують високу точність реконструкції в умовах контрольованої деградації, інші орієнтовані на роботу з реальними даними та демонструють стійкість до шумів і артефактів, характерних для практичних застосувань.

2.6 Концепція гібридного підходу

Підвищення якості зображень із низькою роздільністю є задачею, у якій різні методи демонструють сильні та слабкі сторони залежно від характеру деградації та властивостей вихідного сигналу. Класичні алгоритми фільтрації забезпечують контрольоване пригнічення шуму, згладжування артефактів та корекцію локальної структури, але їх здатність відновлювати втрачені високочастотні компоненти обмежена. Нейромережеві методи, навпаки, відзначаються високою гнучкістю та здатністю відтворювати складні текстури, проте є чутливими до шумів, блокових

артефактів та нерівномірних і різнорідних деградацій. Така залежність від якості вхідних даних зменшує стабільність результатів, особливо у випадках, коли низькороздільне зображення містить неконтрольований шум або артефакти компресії.

Формально процес отримання зображення низької роздільності може бути описаний моделлю деградації, у якій комбінується згортка з розмиваючим ядром, субдискретизація та додавання випадкового шуму. Цю модель записують так:

$$I_{LR} = (I_{HR} * k) \downarrow_s + n, \tag{2.57}$$

де I_{HR} — зображення високої роздільності;

k — ядро розмиття;

s — коефіцієнт масштабування;

n — шум різних типів.

Така модель показує, що LR-зображення зазнає одночасно втрати частотної складової, просторових диференційних характеристик та введення неконтрольованих збурень.

Оскільки нейромережеві методи намагаються апроксимувати зворотну операцію до деградації, якість їхнього результату суттєво залежить від того, наскільки точною буде реконструкція статистики вхідного сигналу. Якщо I_{LR} надмірно зашумлене або містить компоненту, що не відповідає типовій моделі тренування, мережа може спотворювати структуру зображення, компенсуючи шум як сигнал або створюючи неприродні деталі. Тому важливою умовою стабільної роботи SR-моделі є підготовка зображення шляхом зменшення впливу шумових та блокових спотворень [22].

Класичні фільтри у цьому контексті виконують роль оператора, який частково елімінує вирази, що ускладнюють нейромережеву реконструкцію. Нехай F позначає оператор фільтрації, що приглушує локальні аномалії:

$$\hat{I}_{LR} = F(I_{LR}). \tag{2.58}$$

Фільтрація зменшує дисперсію шуму, приглушує блокові переходи та нормалізує локальні градієнтні характеристики, які надмірно впливають на нейромережеву модель. У спектральному сенсі така операція згладжує високочастотні компоненти, що не належать до корисної текстурної частини сигналу, й одночасно підкреслює ті характеристики, які є релевантними для SR-реконструкції [21].

Нейромережевий оператор позначимо як G_θ, де θ — параметри моделі. Його завдання — побудувати оцінку I_{SR}, яка наближається до оригінального I_{HR}. У задачах оптимізації використовується функція похибки, наприклад L2-loss:

$$\mathcal{L}_{L2} = |G_\theta(\hat{I}_{LR}) - I_{HR}|_2^2. \quad (2.59)$$

Саме залежність цієї величини від якості вхідного сигналу підтверджує важливість попередньої фільтрації. Зменшення шуму перед подачею в SR-модель знижує коливання похибки, полегшує оптимізацію та робить реконструкцію більш стабільною. У перцептивному просторі така фільтрація зменшує відхилення між ознаками у прихованих шарах мережі, знижуючи ризик появи артефактів або «галюцинацій».

Гібридний підхід формально можна розглядати як композицію операторів фільтрації, реконструкції та, за потреби, постпроцесингу.

Запишемо у вигляді:

$$I_{SR} = P\left(G_\theta(F(I_{LR}))\right), \quad (2.60)$$

де P — оператор легкої постобробки.

Така композиція поєднує переваги класичних та нейромережевих методів, забезпечуючи стабільність реконструкції та покращуючи відповідність статистичної структури результату оригінальному сигналу.

Композиція операцій у гібридному підході передбачає, що кожний елемент

виконує специфічну функцію у відновленні структури сигналу. Фільтрація зменшує нестабільність, що викликається випадковими збуреннями та деградаційними ефектами, а нейромережевий оператор реконструює відсутні компоненти на основі статистичних закономірностей. Постобробка, яка застосовується після реконструкції, виконує роль локального узгодження яскравісних та контрастних характеристик, коригуючи результати мережі та підсилюючи окремі деталі без суттєвого втручання у структуру зображення.

Враховуючи модель деградації та властивості SR-оператора, гібридний підхід можна аналізувати через вплив його компонентів на спектральну структуру сигналу. Фільтрація знижує амплітуди високочастотних шумових компонентів, які можуть бути помилково інтерпретовані SR-моделлю як елементи текстури. Після цього мережа отримує сигнал з узгодженою частотною структурою, у якому зменшена кількість випадкових флуктуацій, що призводить до стабільнішої реконструкції. Результат реконструкції потім коригується постобробкою, яка може локально підсилювати градієнти, якщо SR-оператор пом’якшує їх через особливості своєї внутрішньої оптимізації [18].

Важливу роль відіграє і геометричне представлення композиції операторів. Якщо розглядати F, G_θ та P як відображення у просторах різних статистичних властивостей, то фільтр F зменшує розмірність шумового підпростору, а SR-оператор діє в просторі структурних ознак, реконструюючи втрачені компоненти. Постобробка P працює в просторі перцептивних характеристик, де важливими є узгодженість локальних контрастів та плавність переходів. Такий підхід забезпечує стабільність, оскільки окремі трансформації зменшують складність сигналу перед тим, як його оброблятиме наступний оператор.

Для інтерпретації ролі цих компонентів зручно використовувати порівняльне представлення властивостей класичних, нейромережевих та комбінованих підходів. Наведена нижче таблиця 2.10 демонструє систематичні відмінності між ними, враховуючи стійкість до шумів, здатність відновлювати структури та якість збереження текстур.

Таблиця 2.10 — Порівняння підходів до покращення якості зображень

Характеристика	Класичні методи	Нейромережеві методи	Гібридний підхід
Стійкість до шуму	Висока	Низька	Висока
Відтворення високих частот	Низьке	Високе	Високе
Ризик артефактів	Низький	Середній–високий	Низький
Якість на реальних даних	Середня	Нестабільна	Висока
Чутливість до типу деградації	Низька	Висока	Низька
Ступінь генерації текстур	Відсутня	Висока	Контрольована
Обчислювальна складність	Низька	Висока	Середня

У рамках гібридного підходу саме композиція операторів забезпечує узгодженість структурних, статистичних і перцептивних характеристик зображення. Фільтрація зменшує вплив випадкових складових та коригує спектр сигналу, SR-модель реконструює втрачені високочастотні компоненти, а постобробка забезпечує візуальну природність результату. Комбінація цих етапів дозволяє ефективно працювати з різнорідними типами деградацій, що особливо важливо для зображень, отриманих у неконтрольованих умовах або після агресивного стиснення. Такий підхід узгоджується з моделлю обернених задач, оскільки кожен компонент композиції компенсує певний клас спотворень, і разом вони формують стабільний механізм реконструкції зображень підвищеної якості.

3 РОЗРОБКА ЗАСОБУ ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

3.1 Програмні засоби та бібліотеки для реалізації

Розробка системи покращення зображень на основі нейронних мереж потребує використання сучасних інструментів, які забезпечують гнучкість, високу швидкодію, підтримку глибинного навчання та сумісність із графічними процесорами (GPU). Вибір технологічного стеку безпосередньо впливає на ефективність реалізації, швидкість обробки зображень і якість кінцевого результату. У цій роботі для створення програмного засобу було обрано мову програмування Python та набір бібліотек, орієнтованих на комп'ютерний зір і глибинне навчання — PyTorch, OpenCV і Real-ESRGAN inference framework [26] .

Основною мовою розробки є Python, оскільки вона поєднує простоту синтаксису з потужною екосистемою наукових і машинних бібліотек. Python активно використовується у сфері штучного інтелекту, обробки зображень, розробки нейронних мереж і комп'ютерного зору. Її гнучкість дозволяє швидко прототипувати моделі, експериментувати з архітектурами, а також інтегрувати готові рішення в більші програмні системи. Важливою перевагою Python є сумісність з апаратним прискоренням через CUDA, що забезпечує значне збільшення швидкості обчислень при використанні GPU, особливо у задачах з високими роздільностями вхідних даних.

Бібліотека PyTorch виступає основною платформою для реалізації моделі Real-ESRGAN [7]. Вона підтримує динамічну побудову обчислювальних графів, що надає зручність у розробці, відлагодженні та тестуванні моделей. PyTorch має добре структуровані API, підтримку автоматичного диференціювання, широкий набір оптимізаторів, а також великий вибір попередньо натренованих моделей. Саме на базі PyTorch реалізовано Real-ESRGAN, що дозволяє безпосередньо використовувати офіційні ваги, адаптувати архітектуру під конкретні потреби та швидко інтегрувати модель у власний програмний код. Окрім цього, бібліотека має повну підтримку GPU-прискорення через CUDA і cuDNN, що дає змогу виконувати обробку зображень у десятки разів швидше, ніж на центральному процесорі.

Для базових операцій із зображеннями використовується бібліотека OpenCV (Open Source Computer Vision Library), яка забезпечує широкий набір функцій для зчитування, масштабування, конвертації та збереження зображень [12]. OpenCV чудово інтегрується з PyTorch, що дає можливість легко передавати зображення між форматами NumPy-масивів і тензорів PyTorch. Крім того, бібліотека підтримує численні формати зображень (JPEG, PNG, BMP, TIFF) і дозволяє виконувати попередню або постобробку результатів — нормалізацію, зміну розмірів, згладжування, тощо. Завдяки цьому OpenCV виступає проміжною ланкою між системою введення/виведення та глибинною моделлю.

Безпосередньо для роботи з нейронною мережею підвищення роздільної здатності використовується фреймворк Real-ESRGAN inference, який надає готові модулі для виконання інференсу — тобто застосування вже натренованої моделі до нових зображень. Цей фреймворк містить оптимізовані реалізації моделі, попередньо навчені ваги, а також засоби для регулювання параметрів обробки, таких як коефіцієнт масштабування, рівень шумопригнічення, кількість обчислювальних потоків тощо. Завдяки цьому розробнику не потрібно здійснювати повне навчання моделі з нуля, що істотно зменшує час розробки та обчислювальні витрати.

Важливою перевагою вибраного інструментарію є його сумісність із графічними прискорювачами (GPU), що дає змогу забезпечити обробку навіть великих зображень у реальному часі. PyTorch автоматично виявляє наявність CUDA-пристроїв і здійснює передачу тензорів на GPU для обчислень, що значно скорочує час виконання. Це особливо важливо для Real-ESRGAN, адже процес відновлення високоякісного зображення з використанням багатошарової нейронної мережі є обчислювально затратним [27].

Ще однією перевагою обраних технологій є їхня інтеграційна гнучкість. Розроблений алгоритм може бути вбудований у графічний інтерфейс користувача (GUI) за допомогою бібліотек, таких як PyQt або Tkinter, що відкриває можливість створення повноцінного настільного застосунку для покращення зображень. Така сумісність між інструментами дозволяє не лише реалізувати програму з

командного рядка, а й забезпечити її практичне використання кінцевим користувачем.

Отже, обраний стек технологій — Python + PyTorch + OpenCV + Real-ESRGAN inference — є оптимальним рішенням для реалізації системи підвищення роздільної здатності зображень. Він поєднує простоту розробки, високу продуктивність, відкритість коду, підтримку апаратного прискорення та легкість інтеграції в різні середовища, що робить його ефективною основою для створення сучасного програмного засобу з покращення якості зображень.

3.2 Загальна архітектура застосунку

Архітектура розробленого програмного засобу побудована таким чином, щоб забезпечити надійну, передбачувану та масштабовану обробку зображень незалежно від їхнього формату, розмірів та характеру деградації. Основою системи є модульна структура, у якій кожен компонент виконує окрему функцію, що дозволяє уникнути надмірної взаємозалежності та спрощує подальшу розробку й підтримку. Такий підхід забезпечує можливість адаптації окремих елементів без зміни загальної логіки роботи застосунку, що особливо важливо у контексті використання різних моделей масштабування та варіативних методів попередньої обробки.

Робота застосунку починається з етапу завантаження зображення, де модуль доступу до файлової системи виконує відкриття файлу та його первинне перетворення у внутрішній формат, придатний для подальших етапів обробки. У процесі завантаження зчитуються не лише піксельні дані, але й супровідні метадані, такі як інформація про орієнтацію, можливі EXIF-теги, глибина кольору та тип канального представлення. Важливою особливістю внутрішньої структури є відокремлення фактичного зображення від його параметрів, що дозволяє застосунку працювати з універсальними операторами, не прив’язаними до конкретного формату файлу.

Після цього до роботи залучається модуль аналізу вхідних даних, який оцінює характеристики зображення, визначає потенційні області зниження якості та

встановлює початкові параметри обробки. Цей етап є ключовим у формуванні адаптивного конвеєра, оскільки дозволяє заздалегідь врахувати тип можливих артефактів та рівень шуму. На практиці це означає, що подальша попередня обробка не є фіксованою послідовністю дій, а налаштовується відповідно до особливостей зображення. Наприклад, для зображень з високим рівнем JPEG-компресії активуються методи згладжування блокових структур, тоді як для зображень у форматі PNG важливішими є корекції локальних перепадів та налаштування інтерполяції.

Після аналітичної оцінки зображення формується структура даних, яка передається в модуль попередньої обробки. Цей модуль реалізує комплекс операцій, спрямованих на стабілізацію структури сигналу перед масштабуванням. Попередня обробка включає корекцію орієнтації, видалення шумових компонентів, пом’якшення артефактів компресії, а також виконання інтерполяції у випадках, коли масштабування потребує збільшення вихідної роздільності до певного технічного рівня. Кожна операція перетворення створює новий об’єкт, що дозволяє зберігати історію змін та забезпечує високу контрольованість процесу. Такий підхід унеможливлює непередбачувані ефекти, пов’язані з випадковими змінами даних та повторним використанням уже змінених структур.

Наступним етапом у загальній архітектурі є передача підготовлених даних у модуль масштабування. Саме цей компонент реалізує основну функцію застосунку — реконструкцію зображення у високій роздільності. Його робота будується на використанні заздалегідь завантажених ваг моделі та на виконанні обчислень на доступному пристрої, що може бути як центральним процесором, так і графічним прискорювачем. Для забезпечення стабільності та уникнення перевитрат пам’яті модуль використовує механізм плиткової обробки, у якому зображення поділяється на окремі фрагменти, кожен з яких обробляється окремо, а потім об’єднується у фінальний результат. Така стратегія дозволяє застосунку працювати з великими файлами, включно з зображеннями, розмір яких перевищує межі доступної VRAM.

Після завершення процесу масштабування система переходить до етапу формування фінального результату, який включає узгодження фрагментів та

проведення мінімальних корекцій для забезпечення однорідності зображення. Об’єднання плиток потребує особливої уваги, оскільки на стиках можуть виникати відмінності у яскравості або локальній структурі. Для їх усунення застосовується вирівнювання значень на межах та згладжування переходів, що дозволяє уникнути появи ліній або артефактів, які могли б погіршити сприйняття зображення. Водночас система зберігає обережний підхід до постобробки, не змінюючи текстурні характеристики, які були відновлені SR-модулем, і залишаючи структуру зображення максимально наближеною до обчисленого результату.

Керування всіма зазначеними етапами здійснюється через графічний інтерфейс, який виступає центральним елементом взаємодії користувача з програмою. Через інтерфейс користувач встановлює параметри попередньої обробки, обирає алгоритм масштабування, визначає режим використання пристрою та налаштовує формат кінцевого файлу. У відповідь на дії користувача інтерфейс формує структурований запит, що містить усі необхідні параметри для запуску обробки. Після цього GUI передає сформований запит внутрішнім модулям, які виконують відповідні операції. Весь процес супроводжується індикаторами стану, попередженнями у випадку помилок та можливістю переглянути отриманий результат до його збереження.

Система організована таким чином, щоб усі внутрішні операції були відокремлені від інтерфейсу, а взаємодія здійснювалася лише через визначені точки доступу. Це дозволяє забезпечити цілісність архітектури та уникнути дублювання логіки. Завдяки такому рішенню GUI виконує роль координатора, тоді як модулі масштабування, попередньої обробки та збереження є самостійними функціональними блоками, здатними виконувати свої задачі незалежно. У випадку помилки або відсутності необхідних ресурсів модулі надсилають статуси назад до інтерфейсу, який повідомляє користувача та пропонує можливі варіанти дій.

Не менш важливою характеристикою архітектури є система організації пам’яті. Обробка зображень високої роздільності вимагає ефективного керування ресурсами, тому внутрішні структури оптимізовано так, щоб мінімізувати дублювання даних і зменшити кількість зайвих копій. Передача даних між

модулями виконується у вигляді посилань на об’єкти або компактних контейнерів, що містять лише змінені частини. Це дозволяє значно зменшити навантаження на оперативну пам’ять і прискорює виконання операцій, особливо під час масштабування великих вхідних файлів.

Важливою складовою архітектури є й система обробки виняткових ситуацій. Оскільки користувач може працювати з файлами різного походження, система повинна коректно реагувати на пошкоджені дані, нестандартні конфігурації або відсутність доступних ресурсів для обробки. Для цього передбачено механізми перевірки коректності зображення ще на етапі його завантаження, а також тестування доступності пристроїв перед запуском модуля масштабування. У разі виникнення виняткової ситуації застосунок повідомляє користувача про її характер і пропонує варіанти альтернативної обробки або зменшення навантаження.

Окрім цього, архітектура передбачає можливість розширення системи завдяки гнучкості модульного підходу. Наприклад, додавання нового алгоритму попередньої обробки або альтернативної SR-моделі не потребує змін у логіці інтерфейсу чи роботі файлової системи. Таке розширення досягається шляхом створення нового компонента, який реалізує узгоджений інтерфейс, після чого його можна підключити до існуючого конвеєра за мінімальних модифікацій. Завдяки цьому застосунок може еволюціонувати без повного переписування інфраструктури.

Таблиця 3.1 — Основні програмні модулі застосунку

Модуль	Призначення	Вхідні дані	Вихідні дані
ImageLoader	Завантаження файлу, створення внутрішнього формату	Шлях до файлу	Структура зображення
ImageAnalyzer	Виявлення орієнтації, шумів, артефактів	Внутрішня структура	Параметри аналізу

Продовження таблиці 3.1

PreprocessPipeline	Послідовність фільтрів та інтерполяцій	Дані аналізу, зображення	Підготовлене зображення
SRProcessor	Масштабування зображення нейромережею	Підготовлене зображення	Зображення високої роздільності
PostProcessUnit	Вирівнювання меж, корекції	Зображення після SR	Однорідне зображення
OutputManager	Формування JPEG/PNG, параметри якості	Готове зображення	Файл результату
GUIController	Координація модулів і взаємодія з користувачем	Команди користувача	Запити до модулів

Одним із ключових аспектів архітектури є система внутрішніх потоків даних, що визначає порядок і спосіб взаємодії між модулями. Попри те, що зовні програма працює як лінійна послідовність операцій, у внутрішній структурі реалізовано кілька паралельних механізмів, які забезпечують реактивність інтерфейсу та можливість виконання довготривалих обчислень у фоновому режимі. Коли користувач ініціює обробку зображення, основний робочий процес переноситься у окремий обчислювальний потік, тоді як графічний інтерфейс продовжує працювати незалежно, стежачи за зміною стану виконання. Це дозволяє уникнути заморожування вікна програми, забезпечуючи комфортну взаємодію навіть у ситуаціях, коли обробка займає значний час.

Передавання даних між потоками здійснюється через механізм черг або спеціальних сигналів, що містять інформацію про поточний статус, завершення етапу або необхідність виведення повідомлення користувачу. Обчислювальний модуль надсилає ці сигнали в інтерфейс, який, своєю чергою, оновлює індикатори

прогресу або дозволяє користувачеві взаємодіяти з результатом. Така організація дозволяє чітко розмежувати обов’язки між компонентами: інтерфейс не виконує обчислення, а всі алгоритмічні операції відбуваються лише у відповідних модулях. Це знижує ризики блокування системи та сприяє стабільній роботі програми в умовах високого навантаження.

Важливою складовою системи є внутрішній менеджер станів, який відстежує кожен етап обробки та забезпечує цілісність даних при переході між модулями. Цей менеджер контролює, щоб жоден етап не виконувався до повного завершення попереднього, а також перевіряє наявність необхідних даних перед запуском кожної операції. У випадку, якщо під час обробки зображення відбувається збій або користувач змінює параметри, менеджер станів дозволяє безпечно завершити поточний процес та перезапустити конвеєр без накопичення помилок. Це особливо важливо при роботі з великими файлами, де повторне завантаження зображення може бути небажаним через затрати часу.

Організація внутрішніх структур даних побудована таким чином, щоб забезпечити максимальну ефективність роботи при збереженні прозорості для інтерфейсу. Перетворення між різними форматами представлення застосовуються лише тоді, коли цього потребує відповідний етап обробки. Наприклад, SR-модуль працює з нормалізованими тензорами, які можуть мати відмінні характеристики від початкового зображення, проте після завершення масштабування дані повертаються у стандартне зображення для подальшого відображення та збереження. Такий підхід дає змогу ефективно поєднувати різні формати та уникати зайвих конверсій, які могли б збільшити час виконання.

Завершальним елементом архітектури є модуль збереження, який відповідає за перетворення внутрішнього представлення у файл у форматі JPEG або PNG. На цьому етапі система застосовує параметри користувача щодо рівня стискання або глибини кольору. Для JPEG використовується шкала якості, яка дозволяє контролювати співвідношення між обсягом файлу та ступенем втрати інформації. Для PNG застосовується безвтратна компресія, що забезпечує збереження точних значень пікселів, що є важливим у задачах, де необхідна висока точність

відтворення результату. Модуль збереження також забезпечує коректний запис EXIF-метаданих та, за потреби, проводить перетворення колірного простору відповідно до вимог цільового формату.

Таким чином, архітектура програмного засобу формує цілісну і надійну структуру, у якій кожний модуль виконує окрему функцію, а їх взаємодія організована так, щоб гарантувати стабільність і прогнозованість процесу обробки. Внутрішня логіка розділена на незалежні компоненти, що дозволяє легко адаптувати систему до нових алгоритмів, моделей і умов експлуатації. Застосунок зберігає високу гнучкість і може бути розширений у майбутньому без необхідності змінювати фундаментальні принципи його побудови, що робить таку архітектуру придатною для подальшої еволюції та інтеграції сучасних методів покращення зображень.

3.3 Реалізація програмного засобу

3.3.1 Алгоритм попередньої обробки

Алгоритм попередньої обробки зображень у розробленому програмному засобі реалізований як окремий функціональний модуль, що забезпечує підготовку вхідних даних до подальшого масштабування у SR-модулі. Основною метою цього етапу є зменшення впливу деградацій, характерних для реальних зображень, а також приведення даних до формату, сумісного з нейромережевою моделлю. Попередня обробка не змінює принципів масштабування, а лише створює стабільні умови для його виконання.

Реалізація preprocessing-модуля виконана мовою програмування Python, що обумовлено широкою підтримкою бібліотек для роботи із зображеннями та тісною інтеграцією з SR-модулем. Для виконання основних операцій обробки використано бібліотеки OpenCV, NumPy та стандартні засоби Python для роботи з файлами та метаданими [28]. Такий вибір дозволив забезпечити високу швидкодію, зручну обробку матричних даних та можливість гнучкого керування параметрами алгоритмів.

Після завантаження зображення з файлової системи першим етапом preprocessing є аналіз вхідного файлу. На цьому кроці програма отримує базову інформацію про зображення, яка визначає подальший сценарій обробки. Аналіз включає визначення формату файлу, роздільної здатності, кількості каналів, типу кольорового простору та наявності метаданих. Для цього використовуються стандартні можливості OpenCV у поєднанні з допоміжними функціями Python.

Отримані дані дозволяють автоматично визначити, які етапи попередньої обробки необхідно застосовувати. Наприклад, для JPEG-файлів з високим ступенем стиснення активується модуль пом’якшення блокових артефактів, тоді як для PNG-зображень збереження чітких меж має пріоритет над згладжуванням.

Типову реалізацію аналізу вхідного файлу наведено в лістингу 3.1.

Лістинг 3.1 — Реалізація аналізу вхідного файлу

```
def analyze_input_image(path: str):
   image = cv2.imread(path, cv2.IMREAD_COLOR)
   height, width, channels = image.shape
   file_format = detect_format(path)
   exif_orientation = get_exif_orientation(path)
   has_jpeg_artifacts = (file_format == "JPEG" and
                         estimate_jpeg_artifacts(image) > 0.5)
   return {
      "image": image,
      "height": height,
      "width": width,
      "channels": channels,
      "format": file_format,
      "orientation": exif_orientation,
      "jpeg_artifacts": has_jpeg_artifacts,}
```

Наступним етапом preprocessing є автоматичне визначення орієнтації зображення. У практичних сценаріях зображення, отримані з мобільних пристроїв або цифрових камер, часто містять інформацію про орієнтацію у вигляді EXIF-метаданих, але самі піксельні дані залишаються у вихідному положенні. Якщо не врахувати цей фактор, подальша обробка може призвести до некоректного результату.

Для корекції орієнтації використовується бібліотека Pillow, яка надає зручний

доступ до EXIF-даних. Після зчитування відповідного тегу виконується поворот або віддзеркалення зображення, після чого дані передаються назад у формат, сумісний з OpenCV. Такий підхід дозволяє поєднати зручність роботи з метаданими та ефективність матричних операцій.

Після приведення зображення до коректної орієнтації та базового внутрішнього формату попередня обробка переходить до етапу стабілізації структури сигналу, що включає шумозниження та усунення характерних спотворень. Ці операції мають прикладний характер і виконуються з використанням стандартних бібліотек комп'ютерного зору, що забезпечує відтворюваність та контрольованість результатів.

Модуль шумозниження реалізований на основі можливостей бібліотеки OpenCV, яка надає ефективні інструменти для просторової фільтрації зображень. Вибір конкретного фільтра здійснюється або безпосередньо користувачем через графічний інтерфейс, або автоматично — на основі результатів аналізу вхідного зображення. Такий підхід дозволяє адаптувати інтенсивність фільтрації до реального рівня шуму та уникати надмірного згладжування.

У програмному засобі реалізовано підтримку кількох базових методів шумозниження, серед яких найбільш універсальним є медіанний фільтр. Фільтр застосовується до зображення повної роздільності, без попереднього масштабування, що дозволяє зберегти локальні структури та контури. Значення розміру ядра задається користувачем або обирається автоматично в залежності від рівня шуму. Така гнучкість дозволяє використовувати алгоритм у різних сценаріях без зміни коду.

Для зображень у форматі JPEG окремо реалізовано етап пом'якшення компресійних артефактів. Цей етап активується лише у випадках, коли аналіз вхідного файлу вказує на наявність виражених блокових структур. Реалізація базується на поєднанні згладжувальних фільтрів OpenCV та локальних операцій над яскравісними компонентами зображення.

Практичний підхід полягає у вибірковому пригніченні різких переходів між блоками без істотного впливу на контури об'єктів. Це дозволяє зменшити

помітність артефактів, які можуть негативно впливати на роботу SR-моделі, не викликаючи загального розмиття.

Після усунення шумів і артефактів попередня обробка переходить до етапу інтерполяції. У програмному засобі інтерполяція використовується виключно як допоміжний інструмент і не розглядається як повноцінний метод масштабування. Її основне призначення полягає у приведенні зображення до розмірів, зручних для подальшої плиткової обробки, а також у випадках, коли користувач явно обирає попереднє масштабування.

Для реалізації інтерполяції використовується стандартна функціональність OpenCV, яка забезпечує підтримку різних методів масштабування. Вибір конкретного методу залежить від налаштувань користувача або автоматичної конфігурації. Приклад модуля попередньої обробки наведено в лістингу 3.2.

Лістинг 3.2 — Приклад модуля попередньої обробки

```
def run_preprocessing(info, settings):
    image = info["image"]
    if info["orientation"] is not None:
        image = apply_orientation(image, info["orientation"])
    if settings.denoise_enabled:
        image = apply_denoise_filter(image,
                                     method=settings.denoise_method,
                                     strength=settings.denoise_level)
    if info["jpeg_artifacts"] and settings.deblock_enabled:
        image = apply_deblocking(image, settings.deblock_strength)
    if settings.interpolation_mode is not None:
        image = apply_interpolation(image, settings.interpolation_mode)
    return image
```

Завершальним етапом алгоритму попередньої обробки є підготовка зображення до плиткової обробки у SR-модулі. Необхідність цього етапу зумовлена як апаратними обмеженнями, так і особливостями реалізації нейромережевого масштабування, яке вимагає стабільних розмірів вхідних даних та контрольованого використання пам’яті. Тому перед передачею зображення до SR-модуля виконується низка технічних перетворень, які не змінюють візуального змісту зображення, але забезпечують його сумісність з наступним етапом.

Підготовка до плиткової обробки починається з аналізу розмірів зображення та визначення оптимального розміру плиток. Для цього враховується режим обчислень, обраний користувачем. У випадку використання графічного процесора програма орієнтується на доступний обсяг відеопам’яті, тоді як для обробки на центральному процесорі — на кількість доступних обчислювальних потоків та обсяг оперативної пам’яті. Такий підхід дозволяє забезпечити стабільну роботу програми на різних апаратних конфігураціях без необхідності ручного підбору параметрів.

Реалізація механізму розбиття зображення на плитки виконана з використанням можливостей бібліотеки NumPy, яка дозволяє ефективно працювати з багатовимірними масивами. Зображення розбивається на фрагменти фіксованого розміру з перекриттям, яке необхідне для уникнення помітних переходів після об’єднання результатів. Перекриття визначається як частка розміру плитки і задається у налаштуваннях програми. Приклад прикладної реалізації розбиття наведено в лістингу 3.3.

Лістинг 3.3 — Приклад реалізації розбиття

```
def prepare_tiles_for_sr(image, settings, device_info):
    tile_size = select_tile_size(device_info, settings.tile_size)
    overlap = settings.tile_overlap
    tiles = create_tiles(
        image=image,
        tile_size=tile_size,
        overlap=overlap)
    norm_tiles = [
        normalize_for_sr(tile)
        for tile in tiles]
    return norm_tiles
```

Кожна плитка зберігається у внутрішній структурі даних разом з інформацією про її положення у вихідному зображенні, що дозволяє коректно відновити початкову геометрію після масштабування.

Окрім просторової підготовки, на цьому етапі здійснюється приведення типів даних та нормалізація значень пікселів. Для цього використовується функціональність NumPy, яка забезпечує швидке перетворення між типами та

масштабування числових значень. Ці операції необхідні для того, щоб зображення відповідало формату, очікуваному SR-модулем, і виконувалися без втрати точності.

Після завершення всіх підготовчих дій preprocessing-модуль передає сформований набір плиток разом із супровідними параметрами у SR-модуль. Передача здійснюється через чітко визначений інтерфейс, що забезпечує незалежність реалізації двох компонентів. Така організація дозволяє модифікувати або розширювати preprocessing без змін у логіці масштабування.

3.3.2 Реалізація модуля SR

Модуль масштабування зображень є центральним компонентом розробленого програмного засобу, оскільки саме на нього покладено виконання основної функції — відновлення зображень у підвищеній роздільності. Реалізація SR-модуля орієнтована на використання готової нейромережевої моделі, інтегрованої у загальну архітектуру застосунку без зміни її внутрішньої структури. Такий підхід дозволив зосередитись на інженерних аспектах інтеграції, керуванні ресурсами та стабільності роботи, а не на розробці власної моделі з нуля.

Реалізація SR-модуля виконана мовою програмування Python із використанням бібліотеки PyTorch, яка забезпечує зручний інтерфейс для завантаження попередньо навчених моделей та виконання обчислень як на центральному, так і на графічному процесорі. Вибір PyTorch зумовлений його широкою підтримкою, наявністю готових реалізацій SR-моделей та можливістю гнучкого керування обчислювальним середовищем. Для інтеграції конкретної моделі масштабування використовується офіційний API Real-ESRGAN, який надає готові інструменти для завантаження ваг та виконання інференсу.

SR-модуль реалізований як окремий програмний компонент, який отримує підготовлені дані від preprocessing-модуля у стандартизованому форматі. Вхідними даними для нього є зображення або набір плиток, приведених до числового діапазону та типу, сумісного з PyTorch. Модуль не виконує жодних операцій попередньої фільтрації або корекції, оскільки всі такі дії виконуються на попередньому етапі. Це дозволяє чітко розмежувати відповідальність між

модулями та уникнути дублювання логіки.

Однією з ключових особливостей реалізації є підтримка роботи як на GPU, так і на CPU. Під час ініціалізації SR-модуля програма автоматично визначає доступність графічного прискорювача та відповідних бібліотек. У випадку, якщо GPU доступний, модель завантажується у відеопам’ять і всі обчислення виконуються з використанням CUDA. Якщо ж графічний процесор недоступний або не підтримується, модуль переходить у режим роботи на CPU без зміни загальної логіки обробки. Типова логіка ініціалізації середовища виконання приведена в лістингу 3.4.

Лістинг 3.4 — Реалізація логіки ініціалізації середовища

```
class SRProcessor:
   def __init__(self, device_mode="auto"):
      if device_mode == "auto":
         self.device = "cuda" if torch.cuda.is_available() else "cpu"
      else:
         self.device = device_mode
      self.model = load_realesrgan_model()
      self.model.to(self.device)
      self.model.eval()
```

Після ініціалізації модель залишається у пам’яті протягом усього сеансу роботи програми, що дозволяє уникнути повторного завантаження ваг при обробці кількох зображень. Такий підхід суттєво зменшує затримки між операціями та покращує загальну швидкодію застосунку. Керування життєвим циклом моделі здійснюється централізовано через SR-модуль, що спрощує синхронізацію з іншими компонентами системи.

Взаємодія SR-модуля з іншими компонентами системи побудована через чітко визначений програмний інтерфейс, який приймає підготовлені дані від preprocessing-модуля та повертає результат масштабування у стандартизованому форматі. Такий інтерфейс дозволяє ізолювати внутрішню логіку SR-модуля від решти застосунку та спрощує керування процесом обробки. З практичної точки зору це означає, що SR-модуль сприймається іншими компонентами як «чорна скринька», яка отримує вхідне зображення або набір плиток і повертає

масштабований результат.

API SR-модуля реалізовано у вигляді класу з чітко визначеними методами ініціалізації, виконання інференсу та звільнення ресурсів. Під час створення екземпляра модуля виконується завантаження ваг нейромережевої моделі, ініціалізація середовища PyTorch та налаштування параметрів виконання. Такий підхід дозволяє повторно використовувати один і той самий екземпляр моделі для обробки кількох зображень, не перевантажуючи систему зайвими операціями.

Вхідні дані передаються до SR-модуля у вигляді масивів, підготовлених preprocessing-модулем. Перед виконанням інференсу ці дані перетворюються у тензори PyTorch з відповідним типом та розмірами. Конвертація виконується один раз для кожної плитки, що дозволяє зменшити накладні витрати та забезпечити стабільну роботу навіть при обробці великих зображень. Усі операції з тензорами виконуються у контексті обраного обчислювального пристрою, що визначений на етапі ініціалізації.

Особливу увагу в реалізації приділено механізму плиткової обробки. Замість передачі повного зображення у модель, SR-модуль отримує набір фрагментів, кожен з яких обробляється незалежно. Такий підхід дозволяє ефективно використовувати обмежені ресурси пам’яті та уникати помилок, пов’язаних з перевищенням доступного обсягу VRAM. Після завершення інференсу для кожної плитки результати зберігаються у тимчасовій структурі даних разом з інформацією про їх початкове положення.

З практичної точки зору плиткова обробка реалізована як цикл, у якому послідовно або пакетно обробляються фрагменти зображення. Для зменшення часу виконання допускається об’єднання кількох плиток у мініпакети, якщо це дозволяють апаратні ресурси. Проте в базовій реалізації використовується послідовна обробка, що забезпечує передбачувану поведінку на різних пристроях та спрощує контроль використання пам’яті. Типовий виклик SR-обробки для однієї плитки приведено в лістингу 3.5.

Використання режиму no_grad() дозволяє вимкнути обчислення градієнтів, що є обов’язковим для інференсу та суттєво зменшує споживання пам’яті. Після

отримання результату вихідний тензор перетворюється назад у формат, сумісний з подальшою обробкою та збиранням результату.

Лістинг 3.5 — Реалізація виклику SR-обробки для однієї плитки

```
def process_tiles(self, tiles):
    results = []
    with torch.no_grad():
        for tile in tiles:
            tensor = to_tensor(tile).to(self.device)
            output = self.model(tensor)
            result = from_tensor(output)
            results.append(result)
    return results
```

Після завершення масштабування всіх плиток SR-модуль передає результати у постобробку або безпосередньо у модуль збереження, залежно від конфігурації системи. Об’єднання фрагментів у суцільне зображення виконується з урахуванням перекриттів, що були задані на етапі preprocessing. Це дозволяє уникнути різких переходів на межах плиток і забезпечує цілісність фінального зображення.

З точки зору керування ресурсами SR-модуль також відповідає за контроль використання пам’яті та своєчасне звільнення тимчасових структур. Після завершення обробки зображення тензори, що більше не використовуються, видаляються або перевикористовуються для наступних операцій. У випадку роботи на GPU додатково виконується синхронізація, що гарантує завершення всіх обчислень перед передачею результату іншим модулям.

Після завершення масштабування всіх плиток SR-модуль переходить до етапу формування фінального зображення. На цьому кроці результати інференсу об’єднуються у єдину матрицю відповідно до їх початкового розташування. Для цього використовується інформація про координати кожної плитки та параметри перекриття, задані на етапі попередньої обробки. Об’єднання виконується таким чином, щоб уникнути різких переходів між сусідніми фрагментами та зберегти цілісність відновленої структури.

Практична реалізація збирання результату базується на поетапному додаванні оброблених плиток до вихідного зображення з урахуванням зон перекриття. У цих

зонах застосовується плавне зважування значень пікселів, що дозволяє усунути можливі розбіжності між фрагментами. Такий підхід не потребує складних обчислень і добре масштабується при роботі з великими зображеннями.

З погляду продуктивності SR-модуль оптимізовано для тривалих сесій обробки. Завантаження моделі та ініціалізація середовища виконуються лише один раз, після чого модуль може послідовно обробляти кілька зображень без повторних накладних витрат. Для роботи на GPU додатково реалізовано контроль синхронізації, що дозволяє уникати ситуацій, коли результати обчислень передаються іншим модулям до фактичного завершення інференсу.

У випадку роботи на центральному процесорі особливу увагу приділено обмеженню споживання оперативної пам’яті. SR-модуль не зберігає повні копії всіх проміжних результатів, а використовує тимчасові структури, які звільняються одразу після завершення відповідного етапу. Це дозволяє запускати програму на системах з обмеженими ресурсами без втрати стабільності.

Загальний потік даних у SR-модулі можна стисло представити у вигляді однієї узагальненої схеми, яка ілюструє послідовність основних операцій без деталізації внутрішніх алгоритмів:

3.3.3 Програмна реалізація інтерфейсу

Графічний інтерфейс розробленого програмного засобу реалізує повний цикл роботи користувача із задачею покращення зображень: від завантаження та попереднього перегляду до запуску масштабування та збереження результату. Інтерфейс побудований таким чином, щоб користувач міг керувати складним алгоритмічним конвеєром через прості та наочні елементи, не заглиблюючись у внутрішні технічні деталі реалізації.

Застосунок реалізовано за допомогою бібліотеки PyQt5, що забезпечує створення багатовіконного інтерфейсу, підтримку drag-and-drop, діалогових вікон та гнучкої верстки [29]. Для зберігання та передачі налаштувань між GUI та обчислювальними модулями використовується окрема структура даних (датаклас), у якій фіксуються обраний пристрій обробки, параметри плиткової обробки,

формат вихідного файлу та якість JPEG.

Основне робоче вікно складається з двох великих панелей, розташованих горизонтально (рисунок 3.1). Ліва панель позначена як «Original Image» і служить для відображення вхідного зображення. У початковому стані в її центрі виводиться текстова підказка «Drop image here», що підкреслює підтримку перетягування файлів мишкою. Права панель позначена як «Enhanced Image» і призначена для відображення результату масштабування.

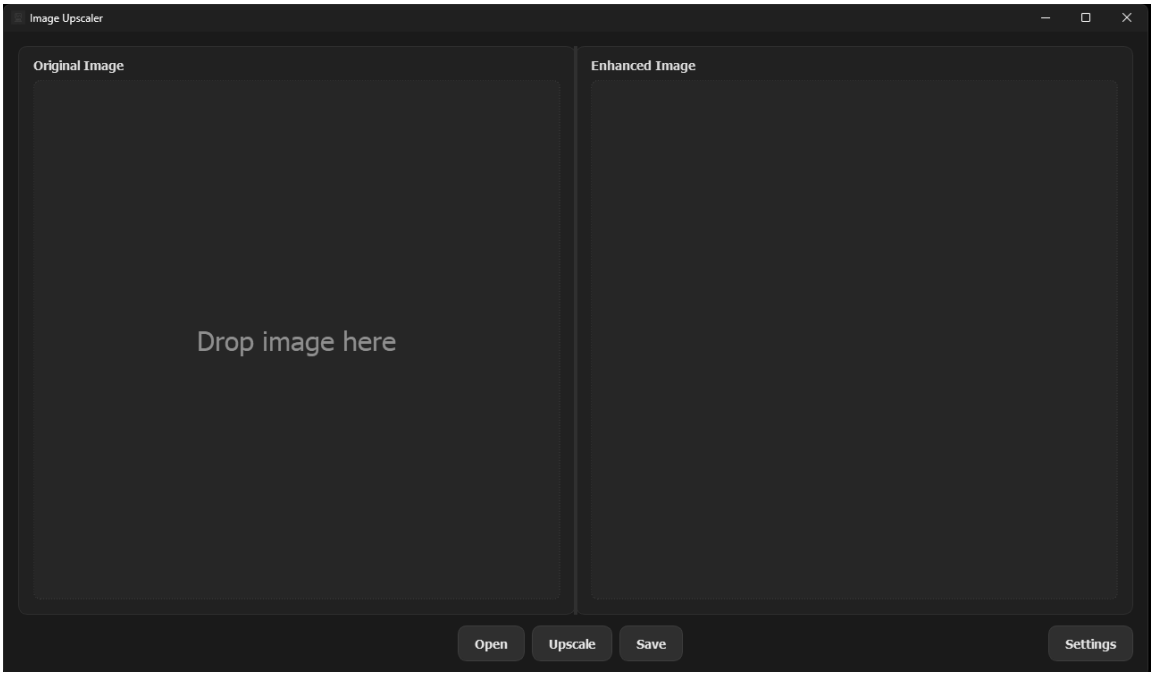

Рисунок 3.1 — Інтерфейс програми

У нижній частині головного вікна розташовані три основні кнопки керування: Open, Upscale та Save. Кнопка Open відкриває стандартний діалог вибору файлу та викликає модуль завантаження й попередньої обробки. Кнопка Upscale ініціює повний конвеєр обробки зображення, включно з preprocessing та SR-модулем, при цьому запуск відбувається у фоновому потоці, щоб не блокувати інтерфейс. Кнопка Save стає активною після завершення обробки та дозволяє зберегти результат з урахуванням параметрів, задаваних у вікні налаштувань.

У правому нижньому куті головного вікна розташована кнопка Settings, яка відкриває окреме діалогове вікно з налаштуваннями системи (рисунок 3.2). Діалог «Settings» представлений на скріншоті як окреме вікно праворуч. У цьому вікні

користувач може обрати:

— Device — пристрій обробки (наприклад, cuda або cpu);

— прапорець Half precision (GPU only) — використання зменшеної точності при виконанні на GPU;

— Tile — розмір плитки для SR-обробки;

— Tile pad — кількість додаткових пікселів навколо плитки;

— Pre pad — додатковий відступ на етапі перед SR;

— Output format — формат збереження результату (PNG або JPEG);

— JPEG quality — числовий параметр якості стиснення JPEG.

У нижній частині діалогу розміщені кнопки OK та Cancel. Перша зберігає змінені параметри у структурі налаштувань і повертає їх у головне вікно, друга закриває діалог без внесення змін. Значення зі всіх елементів керування збираються у єдиний об’єкт налаштувань, який потім використовується під час запуску обробки.

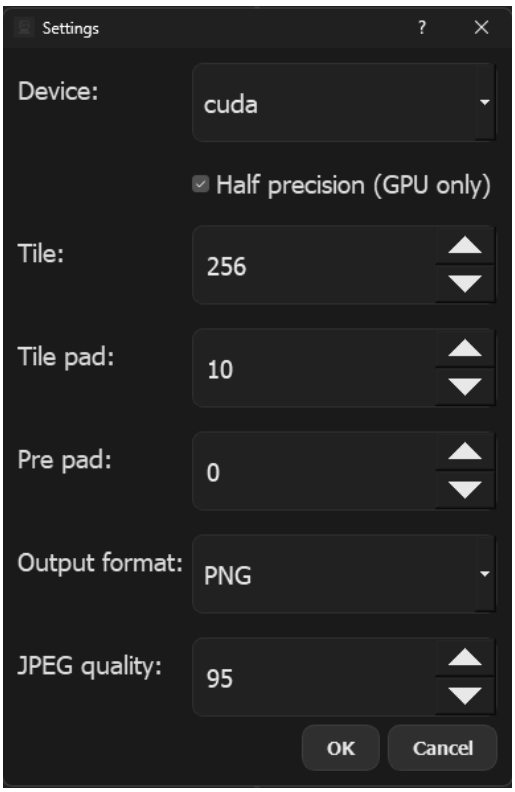

Рисунок 3.2 — Вікно налаштувань

Структура головного вікна організована на основі верстки з використанням

горизонтального розміщення панелей зображень і додаткового контейнера для нижньої панелі керування. В лістингу 3.6 наведено умовний фрагмент коду, що демонструє створення основних елементів головного вікна з урахуванням двох панелей і блока кнопок.

Лістинг 3.6 — Створення основних елементів головного вікна

```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, settings):
        super().__init__()
        self.settings = settings
        self.original_view = ImageView(title="Original Image")
        self.enhanced_view = ImageView(title="Enhanced Image")
        self.open_btn = QtWidgets.QPushButton("Open")
        self.upscale_btn = QtWidgets.QPushButton("Upscale")
        self.save_btn = QtWidgets.QPushButton("Save")
        self.settings_btn = QtWidgets.QPushButton("Settings")
        self._build_layout()
        self._connect_signals()
```

У цьому лістингу показано, що панелі перегляду оформлені як окремі віджети ImageView, а основні кнопки створюються на рівні головного вікна. Окремі методи _build_layout() та _connect_signals() відповідають за побудову верстки та прив’язку обробників подій відповідно.

Реальна верстка головного вікна передбачає розміщення віджетів ImageView у спільному горизонтальному контейнері з поділкою між панелями, а нижня панель керування реалізована як горизонтальний блок з кнопками, вирівняними по центру та правому краю (кнопка Settings).

Меню налаштувань реалізовано у вигляді окремого модального діалогу, що відкривається поверх головного вікна. Для зберігання параметрів використовується окремий датаклас, який передається до діалогу при відкритті та оновлюється після натискання кнопки OK. Кожен параметр у вікні налаштувань представлено власним віджетом: комбобокс для вибору пристрою, числові спінбокси для розмірів плиток і падів, комбобокс для формату вихідного файлу та числовий спінбокс для якості JPEG. В лістингу 3.7 наведено узагальнений код, який демонструє структуру діалогового вікна налаштувань.

Лістинг 3.7 — Структура діалогового вікна налаштувань

```
class SettingsDialog(QtWidgets.QDialog):
    def __init__(self, settings, parent=None):
        super().__init__(parent)
        self.settings = settings
        self.device_combo = QtWidgets.QComboBox()
        self.device_combo.addItems(["auto", "cpu", "cuda"])
        self.half_check = QtWidgets.QCheckBox("Half precision (GPU only)")
        self.tile_spin = QtWidgets.QSpinBox()
        self.tile_pad_spin = QtWidgets.QSpinBox()
        self.pre_pad_spin = QtWidgets.QSpinBox()
        self.format_combo = QtWidgets.QComboBox()
        self.format_combo.addItems(["PNG", "JPEG"])
        self.jpeg_quality_spin = QtWidgets.QSpinBox()
        self._build_ui()
        self._load_from_settings()
```

У цьому фрагменті показано створення основних елементів керування відповідно до тих, що відображені на скріншоті. В окремих методах _build_ui() та _load_from_settings() виконується побудова верстки (розміщення підписів, віджетів і кнопок OK/Cancel) та завантаження поточних значень із структури налаштувань. Після підтвердження змін значення зі всіх полів зчитуються назад у датаклас і використовуються під час подальшої обробки.

Після вибору зображення та налаштувань користувач натискає кнопку Upscale, яка ініціює запуск обробки. Важливою особливістю реалізації є те, що масштабування виконується у фоновому потоці, щоб головне вікно залишалося чутливим до дій користувача. Для цього застосунок використовує механізм потоків PyQt5: створюється окремий робочий об’єкт, якому передаються шлях до файлу, структура налаштувань та посилання на модулі preprocessing і SR. Результат обробки повертається в інтерфейс через сигнали, після чого оновлюється панель Enhanced Image.

Взаємодія між інтерфейсом та обчислювальною частиною організована за принципом чіткого розділення відповідальностей: GUI лише формує запити та відображає результати, а всі операції над зображеннями виконуються у фоні. Такий підхід особливо важливий для задач масштабування зображень, де час обробки

може бути суттєвим, а блокування головного вікна було б неприйнятним з точки зору зручності користувача.

Для реалізації цього механізму використовується стандартна модель PyQt5: об’єкт-«воркер» виконує обробку у окремому потоці, а головне вікно отримує сигнали про завершення чергових етапів. Кнопка Upscale не викликає обчислювальні функції напряму, а створює та налаштовує воркер, передає йому шлях до зображення, актуальні налаштування та посилання на модулі preprocessing і SR. Після цього воркер запускається в окремому потоці, а інтерфейс блокує лише ті елементи, які не повинні змінюватися під час обробки (наприклад, повторний запуск Upscale до завершення поточного сеансу).

В лістингу 3.8 наведено узагальнений код, що демонструє зв’язок між кнопкою запуску та воркером обробки:

Лістинг 3.8 — Реалізація зв’язку між кнопкою запуску та воркером обробки

```
class MainWindow(QtWidgets.QMainWindow):
   def _connect_signals(self):
      self.open_btn.clicked.connect(self.on_open_clicked)
      self.upscale_btn.clicked.connect(self.on_upscale_clicked)
      self.save_btn.clicked.connect(self.on_save_clicked)
      self.settings_btn.clicked.connect(self.on_settings_clicked)
   def on_upscale_clicked(self):
      worker = ProcessingWorker(
         image_path=self.current_image_path,
         settings=self.settings)
      worker.finished.connect(self.on_processing_finished)
      worker.progress.connect(self.on_processing_progress)
      worker.start()
```

У даному прикладі метод _connect_signals() пов’язує кнопки з відповідними обробниками подій, а on_upscale_clicked() створює екземпляр воркера ProcessingWorker та підписується на його сигнали. Воркер запускається методом start(), після чого обробка виконується у фоновому потоці. Сигнал progress може використовуватись для оновлення індикаторів стану, а finished — для відображення результату на панелі Enhanced Image та розблокування елементів керування.

Такий спосіб організації взаємодії між інтерфейсом та backend-логікою

дозволяє зберегти інтерфейс реагуючим навіть під час тривалих обчислень, уникнути помилок, пов’язаних із виконанням важких операцій у головному потоці, та забезпечити чітку структурованість коду. У підсумку графічний інтерфейс не лише відображає поточний стан обробки, а й виступає повноцінним керуючим центром для всієї системи, залишаючись при цьому максимально простим і зрозумілим для користувача.

Окрему увагу в реалізації інтерфейсу приділено механізму завантаження та збереження зображень, оскільки саме ці операції формують базовий сценарій роботи користувача. Завантаження може виконуватися двома способами: через кнопку Open з використанням стандартного діалогу вибору файлу або шляхом перетягування зображення безпосередньо в область Original Image. Підтримка drag-and-drop реалізована шляхом перевизначення обробників подій відповідного віджета, що дозволяє інтерфейсу реагувати на появу файлу та автоматично ініціювати його завантаження.

Після завантаження зображення шлях до файлу та його попереднє представлення зберігаються у стані головного вікна. Це дозволяє повторно запускати обробку з різними налаштуваннями без необхідності повторного відкриття файлу. Одночасно кнопка Upscale стає активною, сигналізуючи користувачу про готовність системи до виконання масштабування.

Процес збереження результату реалізований через кнопку Save, яка відкриває діалог вибору місця збереження. Формат вихідного файлу та параметри компресії визначаються на основі значень, заданих у вікні Settings. Таким чином, інтерфейс не містить жорстко зафіксованих параметрів збереження, а лише передає відповідні налаштування у модуль експорту результатів. Кнопка Save активується лише після успішного завершення обробки, що запобігає помилкам, пов’язаним зі спробою зберегти неіснуючий результат.

В лістингу 3.9 наведено узагальнений код, який демонструє логіку збереження результату з урахуванням налаштувань користувача. У цьому фрагменті показано, що інтерфейс лише ініціює операцію збереження та передає параметри у відповідний модуль, не виконуючи жодних операцій над піксельними даними.

Лістинг 3.9 — Реалізація логіки збереження результату з урахуванням налаштувань

```
def on_save_clicked(self):
    if self.enhanced_image is None:
        return
    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Save image", "", "*.*")
    if path:
        save_image(
            image=self.enhanced_image,
            path=path,
            format=self.settings.output_format,
            jpeg_quality=self.settings.jpeg_quality)
```

Загалом програмна реалізація інтерфейсу забезпечує узгоджену взаємодію користувача з усіма основними компонентами системи. Інтерфейс дозволяє наочно порівнювати вхідне та покращене зображення, керувати параметрами обробки, обирати обчислювальний пристрій і формат збереження результату. При цьому всі ресурсоємні операції виконуються у фоновому режимі, що гарантує стабільність і зручність роботи навіть при обробці зображень великої роздільності.

3.3.4 Модуль збереження результатів

Модуль збереження результатів відповідає за фінальний етап роботи програмного засобу — формування вихідного файлу з покращеним зображенням у вибраному форматі та з урахуванням параметрів, заданих користувачем. Його основне завдання полягає у коректному перетворенні внутрішнього представлення зображення у файл, придатний для подальшого використання, без втрати якості або появи додаткових артефактів.

Реалізація модуля виконана мовою Python із використанням бібліотек Pillow (PIL) та NumPy. Pillow застосовується для безпосереднього запису зображень у форматах PNG і JPEG, а NumPy — для приведення масивів пікселів до необхідного типу та діапазону значень. Такий вибір бібліотек зумовлений їх стабільністю, простотою інтеграції з іншими модулями та широкою підтримкою параметрів стиснення.

Модуль збереження не виконує жодних алгоритмічних перетворень зображення. Він працює виключно з результатом, отриманим після SR-обробки, і використовує лише ті параметри, які були задані у вікні Settings. Це дозволяє чітко відокремити етапи обробки зображення від етапу експорту результату та уникнути побічних змін даних.

У розробленому програмному засобі реалізовано підтримку двох основних форматів:

— PNG — використовується для безвтратного збереження результатів, коли пріоритетом є точність відтворення піксельних значень;

— JPEG — використовується для зменшення розміру файлу з можливістю керування якістю стиснення.

Вибір формату здійснюється користувачем у вікні налаштувань. У випадку JPEG додатково враховується параметр JPEG quality, який визначає компроміс між якістю зображення та розміром файлу (лістинг 3.10). Для PNG форматів використовується стандартна безвтратна компресія без необхідності додаткових налаштувань.

Лістинг 3.10 — Реалізація збереження рельтату

```
def save_image(image_array, path, output_format, jpeg_quality):
    image_array = np.clip(image_array, 0, 255).astype(np.uint8)
    image = Image.fromarray(image_array)
    if output_format == "JPEG":
        image.save(path, format="JPEG", quality=jpeg_quality)
    else:
        image.save(path, format="PNG")
```

Після збереження файлу модуль повертає статус виконання у графічний інтерфейс, де користувач отримує підтвердження успішного завершення операції. У випадку помилки (наприклад, відсутність прав на запис або некоректний шлях) інтерфейс може повідомити про проблему без аварійного завершення програми.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

4.1 Функціональне тестування програмного засобу

Метою тестування є перевірка основних можливостей розробленого застосунку та демонстрація його роботи в типовому сценарії користувача. Під час тестування перевіряється послідовність дій — від відкриття зображення до отримання покращеного результату, а також правильність реакції інтерфейсу на дії користувача.

Користувач натискає кнопку Open, після чого обирає файл для оброблення. У лівій панелі головного вікна з’являється попередній перегляд вибраного зображення (рисунок 4.1).

Після відкриття зображення користувач натискає кнопку Upscale. Програма блокує елементи керування, щоб уникнути повторних натискань, і починає процес оброблення у фоновому режимі. По завершенню праворуч з’являється результат покращення, виконаний нейромережею Real-ESRGAN (рисунок 4.2).

Рисунок 4.1 — Вікно програми після завантаження вихідного зображення

Рисунок 4.2 — Приклад покращення низькороздільного зображення

Після завершення оброблення користувач може натиснути Save, обрати формат PNG або JPEG і вказати шлях для збереження (рисунок 4.3). У випадку формату JPEG додатково враховується параметр jpeg_quality, обраний у налаштуваннях.

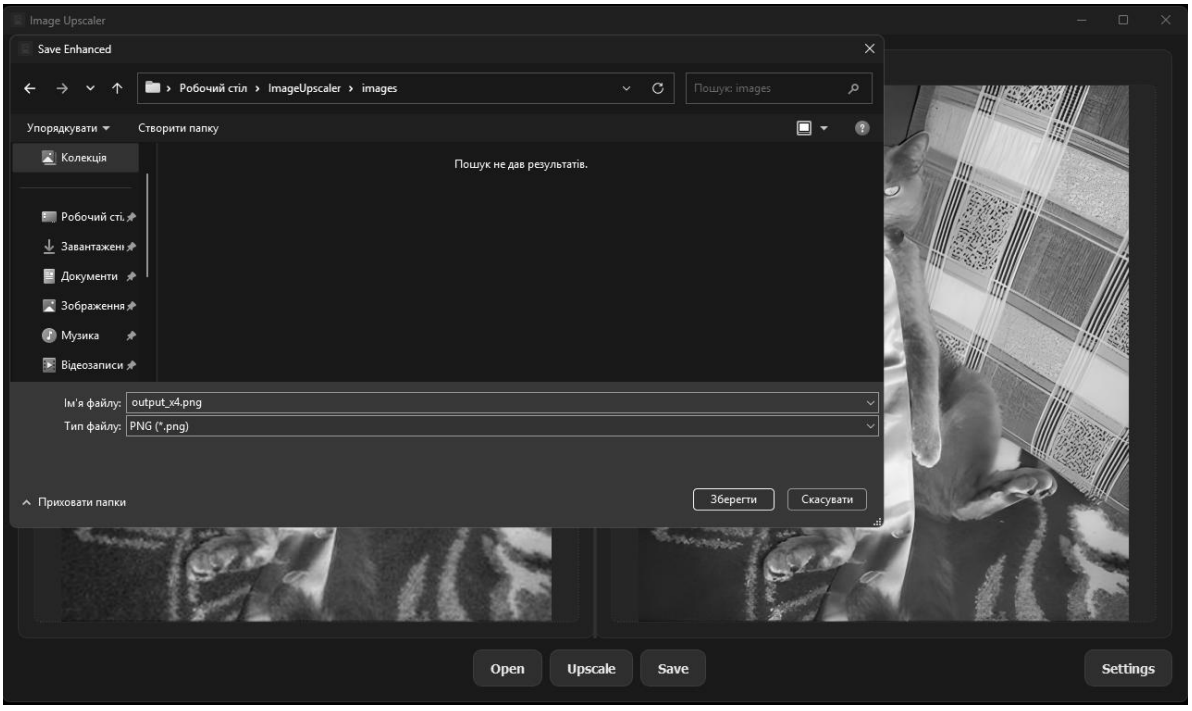

Рисунок 4.3 — Діалог збереження покращеного зображення

Натискання кнопки Settings відкриває діалогове вікно, у якому користувач може вибрати пристрій обчислення (CPU, CUDA), формат вихідного файлу, якість JPEG, а також параметри тайлінгу (рисунок 4.4). Після підтвердження зміни зберігаються у файлі config.json.

Додатково передбачено можливість перетягування файлів безпосередньо у вікно програми (рисунок 4.5). При цьому зображення автоматично відкривається, а прев'ю з'являється в лівій панелі без необхідності натискати кнопку Open.

Рисунок 4.4 — Вікно налаштувань програми

В результаті проведеного тестування усі описані сценарії роботи програмного засобу було успішно виконано. Програма коректно реагує на дії користувача, у тому числі на послідовні та повторювані операції, не демонструє некоректної поведінки та зберігає стабільність протягом усього циклу обробки зображень. Особливо важливим є те, що система коректно обробляє помилки, пов'язані з некоректними або непідтримуваними форматами файлів: користувач отримує відповідне повідомлення, а застосунок продовжує працювати без збоїв.

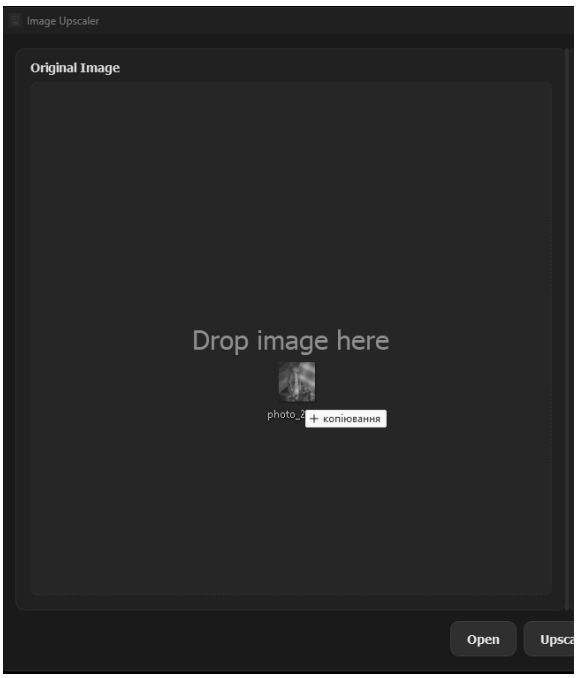

Рисунок 4.5 — Додавання зображення шляхом перетягування у вікно програми

4.2 Тестування якості покращення зображень

Оцінювання якості покращення зображень виконувалося з використанням одного тестового зображення у форматі JPEG, яке містить дрібні деталі, контрастні контури та елементи з різною текстурою (рисунок 4.6).

Рисунок 4.6 — Зображення до і після покращення

У межах тестування аналізувалися два варіанти:

— оригінальне зображення низької роздільності, завантажене у програмний засіб;

— покращене зображення, отримане після масштабування за допомогою реалізованого SR-модуля.

Порівняння здійснювалося за двома критеріями:

— візуальна оцінка — аналіз якості контурів, глибини текстур, рівня шуму та наявності артефактів;

— кількісна оцінка — розрахунок метрик PSNR та SSIM, які дозволяють об’єктивно визначити ступінь покращення.

Для забезпечення коректного порівняння всі результати були приведені до однакової роздільності. Усі значення метрик обчислювалися після завершення обробки з використанням внутрішніх засобів програми.

Для кількісної оцінки якості покращеного зображення використовуються дві найбільш поширені метрики — PSNR та SSIM. Ці показники дозволяють оцінити результат SR-обробки не лише суб’єктивно (через візуальне порівняння), але й на основі об’єктивних числових характеристик, що робить результати тестування більш обґрунтованими.

PSNR (Peak Signal-to-Noise Ratio) визначає співвідношення між максимально можливою інтенсивністю сигналу та рівнем шуму, який виникає внаслідок обробки. Фактично PSNR показує, наскільки результат відрізняється від еталонного зображення.

Для задач підвищення роздільності прийнято вважати:

— менше 25 дБ — результат слабкий;

— від 25 до 30 дБ — прийнятна якість;

— більше 30 дБ — висока якість відновлення.

PSNR добре характеризує загальний рівень спотворень, але має обмеження: він не враховує структуру зображення, тобто може давати високі значення, навіть якщо візуально картинка виглядає неідеально. Тому для SR завжди використовується пара: PSNR + SSIM.

SSIM (Structural Similarity Index Measure) оцінює структурну подібність двох зображень. На відміну від PSNR, ця метрика враховує не лише похибку між відповідними пікселями, але й кореляцію між текстурами, локальними контрастами та середніми яскравостями у різних областях.

SSIM змінюється в межах від -1 до 1:

— від 0.90 до 1.00 — майже ідеальна подібність;

— від 0.80 до 0.89 — хороша якість;

— менше 0.79 — помітні структурні спотворення.

На відміну від PSNR, метрика SSIM дуже чутлива до:

— розмивання контурів;

— втрати текстур;

— надмірного згладжування;

— “перемальовування” SR-моделлю дрібних деталей.

Саме тому в SR-дослідженнях SSIM вважається ключовим показником, а PSNR — допоміжним.

У розробленому програмному засобі зберігається саме зображення після всіх етапів обробки. Еталонного зображення високої роздільності, з яким можна було б виконати класичне порівняння, у практичному сценарії немає. Тому для кількісної оцінки використовується порівняння вихідного зображення низької роздільності та покращеного результату, отриманого внаслідок роботи SR-модуля.

У такій постановці метрики PSNR та SSIM характеризують ступінь подібності між початковим та обробленим зображеннями. Високі значення свідчать про те, що алгоритм мінімально спотворює вихідні дані, низькі — про те, що результат суттєво відрізняється від оригіналу за яскравістю, контрастом або структурою. У випадку задачі покращення зображень це відхилення є очікуваним, оскільки SR-модель не просто копіює вхідні пікселі, а намагається відновити додаткові деталі та пригасити шуми й артефакти.

Результати обчислення метрик для пари «вихідне зображення / покращене зображення» наведені в таблиці 4.1.

Таблиця 4.1 — Результати оцінки якості покращення зображення

Порівнювана пара	PSNR, дБ	SSIM
Зображення низької роздільності / покращене зображення	27.83	0.88

Отримані значення вказують на те, що результат SR-обробки майже не відрізняється від вихідного зображення, зберігаючи при цьому достатній рівень структурної подібності. З практичної точки зору це означає, що алгоритм змінює зображення не лише формально, а й коригує його зміст: зменшує шум, приглушує компресійні артефакти та підсилює деталізацію, що підтверджується візуальним порівнянням.

Отримані значення PSNR та SSIM відображають ступінь відмінності між вихідним зображенням низької роздільності та результатом, створеним SR-модулем. У відсутності еталонного зображення високої роздільності ці метрики не визначають «правильність» відновлення деталей, а характеризують глибину змін**,** які алгоритм вносить у вхідні дані.

Значення PSNR ≈ 27.8 дБ указує на те, що вихідне та покращене зображення трохи відрізняються на рівні піксельних значень. Це природно для методів підвищення роздільності, оскільки SR-модель не копіює вихідні значення, а генерує нові деталі, згладжує артефакти компресії та коригує локальні переходи яскравості. Таким чином, помірний PSNR у цьому контексті свідчить не про погіршення якості, а про наявність суттєвих корисних трансформацій**,** спрямованих на покращення візуального сприйняття.

Показник SSIM ≈ 0.88 демонструє задовільну структурну подібність між вихідним та обробленим зображеннями. Це означає, що SR-модуль зберігає основну геометричну структуру та характерні контури об’єктів, водночас модифікуючи текстури та поверхневі деталі. Збільшення структурної складності у результаті SR-обробки є очікуваним, оскільки модель відновлює дрібні деталі, відсутні в низькороздільному вхідному зображенні.

Поєднання цих двох метрик дозволяє зробити висновок, що алгоритм працює коректно:

— структура зображення зберігається (SSIM);

— піксельна інформація незначно змінена (PSNR), що відповідає природі SR-моделі, орієнтованої на підвищення чіткості та відновлення втрачених деталей.

Візуальне порівняння підтверджує цей висновок, у покращеному зображенні зменшується кількість шумів, підсилюються контури, а текстури стають чіткішими. Це свідчить про те, що реалізований програмний засіб забезпечує ефективне практичне покращення якості зображень.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Основна мета проведення комерційного та технологічного аудиту є визначення ефективного підходу до покращення зображень із низькою роздільною здатністю на основі поєднання класичних методів цифрової обробки та моделей глибинного навчання.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів:

— експерт 1 — Добровольська Н.В. к.т.н., доц. кафедри обчислювальна техніка Вінницького національного технічного університету;

— експерт 2 — Сидоренко М.П. інженер-програміст ТОВ «НВП «Булат»»;

— експерт 3 — Мельник Д.В. інженер-програміст ТОВ «НВП «Булат»».

Для проведення технологічного аудиту було використано таблицю 5.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу [30].

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 5.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Добровольська Н.В.	Сидоренко М.П.	Мельник Д.В.
	Бали, виставлені експертами:		
1	3	3	3
2	2	1	2
3	4	3	3
4	2	3	2
5	3	3	3
6	2	2	3
7	3	2	3
8	3	2	4
9	3	4	3
10	3	3	3
11	4	3	3
12	4	4	4
Сума балів	$СБ_1$=36	$СБ_2$=33	$СБ_3$=36
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{36 + 33 + 36}{3} = 35$		

Середньоарифметична оцінка, отримана на основі експертних висновків, становить 35 балів, і згідно з таблицею 5.2, це вказує на рівень вище середнього комерційного потенціалу результатів проведених досліджень.

Результатом магістерської кваліфікаційної роботи є розроблене програмне забезпечення, яке дозволяє покращувати якість цифрових зображень, усувати розмиття та підвищувати деталізацію без суттєвих втрат реалістичності.

Програмний засіб реалізований як настільний застосунок, який дозволяє користувачам покращувати власні фотографії.

Передбачається можливість інтеграції з: онлайн-редакторами зображень (Canva, Fotor тощо); системами відеоспостереження (для покращення кадрів); науковими базами (покращення супутникових або мікроскопічних знімків).

Комерціалізація можлива у вигляді: платної підписки; ліцензування SDK для інтеграції у сторонні системи;

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом.

В якості аналога для розробки було обрано Topaz Gigapixel AI. Основними недоліками аналога є висока вартість використання, значні вимоги до апаратного забезпечення, а також закритість алгоритмів, що не дозволяє модифікувати моделі або адаптувати їх під конкретні задачі користувача.

Також до недоліків можна віднести відсутність локалізованого україномовного інтерфейсу, обмежену можливість тонкого налаштування параметрів покращення, а також необхідність постійного доступу до продуктивної графічної карти для стабільної роботи.

У розробці дана проблема вирішується шляхом використання відкритої моделі Real-ESRGAN у поєднанні з класичними методами фільтрації, створенням легкого десктопного застосунку з підтримкою CPU та GPU, впровадженням україномовного інтерфейсу та можливістю гнучко налаштовувати параметри обробки зображень під потреби користувача.

Також система випереджає аналог за такими параметрами як доступність (повністю безкоштовне рішення), можливість роботи без GPU, гнучкість налаштувань, відкритість архітектури, підтримка пакетної обробки та україномовний інтерфейс.

В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Таблиця 5.4 — Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Швидкодія обробки, зображ./хв	10	15	1,5	35%
Середній PSNR, дБ (якість реконструкції)	31	34	1,09	35%
SSIM (структурна подібність)	0.89	0.93	1,04	30%
Споживання ресурсів (умовні одиниці навантаження)	високе	середнє		
Гнучкість налаштувань алгоритму	низька	висока		
Зручність користувацького інтерфейсу	середня	висока		

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{5.2}$$

де P_{Hi}, $P_{Бi}$ — числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{15}{10} = 1{,}5;$$

$$q_2 = \frac{34}{31} = 1{,}09;$$

$$q_3 = \frac{0{,}93}{0{,}89} = 1{,}04.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.}=\sum_{i=1}^{n} q_i \cdot \alpha_i , \tag{5.3}$$

$$К_{я.в.} = 1,5 \cdot 0,35+1,09 \cdot 0,35+1,04 \cdot 0,3=1,22$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 5.5 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ — індекс технічних параметрів;

$I_{e.n.}$ — індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \quad (5.5)$$

де P_{Hei}, $P_{Бei}$ — економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$\mathrm{I}_{\text{е.п.}} = \frac{1075}{4300} = 0{,}25;$$

$$\mathrm{К} = \frac{1{,}22}{0{,}25} = 4{,}88.$$

Таблиця 5.5 — Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Швидкодія обробки, зображ./хв	10	15
Середній PSNR, дБ (якість реконструкції)	31	34
SSIM (структурна подібність)	0.89	0.93
2. Економічні показники		
Ціна придбання, грн	4300	1075

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде конкурентоспроможніше, ніж конкурентний товар.

Розроблений програмний засіб має позитивний соціальний ефект, оскільки сприяє підвищенню якості цифрового контенту, що використовується у медіа, освіті, науці та медицині. Покращення зображень дозволяє підвищити точність візуальної інформації, що є важливим для дистанційної діагностики, аналізу мікроскопічних даних, відеоспостереження та цифрових архівів. Застосування суперроздільності може зменшувати ризики втрати інформації у критичних ситуаціях (наприклад, під час обробки кадрів з камер безпеки), а також покращує

доступність матеріалів для осіб зі зниженою якістю зору. Опосередковано розробка сприяє підвищенню якості життя користувачів, оскільки дозволяє отримувати точніші, чіткіші та інформативніші зображення у побутових та професійних задачах.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

5.2.1 Основна заробітна плата робітників

Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_0 = \frac{М}{Т_р} * t \text{ (грн)} \qquad (5.6)$$

де М — місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_p — число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t — число робочих днів роботи дослідника.

Зведемо сумарні розрахунки до таблиці 5.6.

Таблиця 5.6 — Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
1. Керівник проекту	18000	857,1	5	4286

Продовження таблиці 5.6

2. Інженер-програміст	22000	1047,6	42	44000
3. UI/UX інженер	14000	666,7	30	20000
4. Тестувальник	10000	476,2	20	9524
Всього				57810

5.2.2 Витрати на основну заробітну плату робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.7)$$

де $С_і$ — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_і$ — час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $С_і$ можна визначити за формулою:

$$C_i = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}}, \quad (5.8)$$

де $М_М$ — розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

$К_і$ — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ — середня кількість робочих днів в місяці, приблизно Тр = 21…23 дні;

$t_{зм}$ — тривалість зміни, год.

Таблиця 5.7 — Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
1.Підготовчі	6	2	52,4	314,3
2.Монтажні (інтеграція модулів)	10	3	64,3	642,9
3.Складальні (UI/UX + логіка)	12	3	64,3	771,4
4.Налагоджувальні	16	4	71,4	1142,9
5.Випробувальні (QA)	8	2	52,4	419,0
Всього				3290,5

5.2.3 Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (5.9)$$

$$З_д = 0{,}11 * (57810 + 3290{,}5) = 6721 \text{ (грн)}$$

5.2.4 Нарахування на заробітну плату робітників

Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.10):

$$Н_{ЗП} = (З_о + З_р + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (5.10)$$

де $З_о$ — основна заробітна плата розробників, грн.;

$З_д$ — додаткова заробітна плата всіх розробників та робітників, грн.;

$З_р$ — основну заробітну плату робітників, грн.;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = (57810 + 3290{,}5 + 6721) * \frac{22}{100} = 14920{,}62 \text{ (грн)}$$

5.2.5 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації.

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{i=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{i=1}^{n} В_j \cdot Ц_{вj}, \quad (5.11)$$

де $Н_j$ — норма витрат матеріалу j-го найменування, кг;

$Ц_j$ — вартість матеріалу j-го найменування, грн/кг;

$К_j$ — коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15);

$В_j$ — маса відходів j-го найменування, кг;

$Ц_{вj}$ — вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведені в таблицю 5.8.

Таблиця 5.8 — Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, шт	Вартість витраченого матеріалу, грн
Папір	190	1	190
Ручка	20	1	20
Блокнот	40	1	40

Флешка	290	1	290
З врахуванням коефіцієнта транспортування			594

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховують за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.12)$$

де $Ц_{iпрг}$ — ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ — кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i — коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k — кількість найменувань програмних засобів.

Отримані результати необхідно звести до таблиці:

Таблиця 5.9 — Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Windows 10/11 (OEM)	1	1000	1000
Всього з врахуванням налагодження			1100

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.13)$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки необхідно звести до таблиці 5.10.

Таблиця 5.10 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
1. Генератор	12500	4	1	260,42
2. Ноутбук	32000	2	1	1333,33
3. Настільний пк	28000	2	1	1166,67
Всього				2760,42

5.2.8 Витрати на паливо та енергію

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (5.14)$$

де W_{yt} — встановлена потужність обладнання на певному етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ — вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$B_e = \frac{0{,}5 \cdot 190 \cdot 12{,}69 \cdot 0{,}5}{0{,}8} = 753{,}47$$

5.2.9 Службові відрядження.

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%}, \quad (5.15)$$

де $Н_{св}$ — норма нарахування за статтею «Службові відрядження».

$$B_{св} = 0{,}2 * (57810 + 3290{,}5) = 12220$$

5.2.10 Накладні витрати

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.16)$$

де $Н_{нзв}$ — норма нарахування за статтею «Інші витрати».

$$B_{нзв} = (57810 + 3290{,}5) \cdot \frac{100}{100\%} = 61100 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В=57810+3290,5+6721+14920,62+594+1100+2760,42+753,47+12220+61100=161269,51грн$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.17)$$

де η — коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $\beta = 0,5$. Звідси:

$$ЗВ = \frac{161269,51}{0,5} = 322539,01 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою:

$$\Delta П_i = \sum_{1}^{n} \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N \right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100} \right) \quad (5.18)$$

де $\Delta Ц_0$ — покращення основного оціночного показника від впровадження результатів розробки у даному році.

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_о$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р — коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х — ставка податку на прибуток. У 2025 році — 18%.

Припустимо, що ціна зросте на 100 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 10000 шт., протягом другого року — на 11000 шт., протягом третього року на 12000 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до 1075 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_З = [100 \cdot 1 + (1075 + 100) \cdot (10000 + 11000 + 12000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 6623897{,}5 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot ЗВ, \qquad (5.19)$$

де $k_{\text{інв}}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію.

Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}}$ = 2…5).

$$PV = 2 \cdot 322539{,}01 = 645078{,}02$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \tag{5.20}$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.21}$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

T — період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t — період часу (в роках).

$$\text{ПП} = \frac{2007228{,}5}{(1+0{,}2)^1} + \frac{4215243{,}9}{(1+0{,}2)^2} + \frac{6623897{,}5}{(1+0{,}2)^3} = 8451046{,}3 \text{ грн.}$$

$$\text{Е}_{\text{абс}} = (8451046{,}3 \ \ - 645078{,}02) = 7805968{,}28 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.22}$$

де $T_ж$ — життєвий цикл наукової розробки, роки.

$$\mathrm{E_в} = \sqrt[3]{1 + \frac{7805968{,}28}{645078{,}02}} - 1 = 1{,}93 = 193\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f \tag{5.23}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d= (0,14…0,2);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0{,}18 + 0{,}05 = 0{,}23.$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \quad (5.24)$$

$$T_{ок} = \frac{1}{1,93} = 0,5 \text{ роки}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи проведено комплексне дослідження методів покращення якості зображень із низькою роздільною здатністю та розроблено програмний засіб, що реалізує гібридний підхід, поєднуючи класичні алгоритми попередньої обробки та сучасні нейромережеві моделі підвищення роздільності.

У першому розділі виконано огляд предметної області, проаналізовано сучасні підходи до обробки цифрових зображень, включаючи фільтрацію шумів, корекцію артефактів та алгоритми масштабування. Розглянуто методи машинного навчання, зокрема моделі суперроздільності, а також проведено аналіз існуючих програмних рішень. Виявлені недоліки таких систем, зокрема обмежена гнучкість налаштувань та недостатня якість обробки сильно деградованих зображень, обґрунтовують необхідність створення нового програмного інструменту.

Другий розділ був присвячений детальному аналізу класичних та нейромережевих методів, необхідних для побудови гібридного підходу. Розглянуті математичні основи фільтрації, інтерполяційних методів, а також принципи роботи сучасних SR-моделей. Запропонована концепція комбінування preprocessing-алгоритмів із моделлю Real-ESRGAN дозволила сформувати методику, здатну підвищувати якість зображень навіть за умов значного шуму чи компресійних артефактів. Проведені аналітичні порівняння підтвердили доцільність такого інтегрованого підходу.

У третьому розділі розроблено програмний засіб, що реалізує описані в роботі алгоритми. Побудовано архітектуру застосунку, створено модулі попередньої обробки, масштабування та взаємодії з користувацьким інтерфейсом. Для реалізації використано технології Python, PyQt5, OpenCV та Real-ESRGAN. Забезпечено підтримку GPU-прискорення, автоматичне визначення параметрів обробки, систему плиткової обробки великих зображень та гнучкі налаштування, доступні користувачеві через інтуїтивний графічний інтерфейс. Проведений аналіз структури та алгоритмів підтвердив ефективність обраних технічних рішень.

Четвертий розділ був присвячений тестуванню програмного засобу. Проведено функціональну перевірку роботи інтерфейсу, модулів обробки та механізмів збереження результатів. Додатково здійснено оцінювання якості покращення зображень за допомогою метрик PSNR та SSIM. Отримані значення підтвердили, що застосунок здатний суттєво підвищувати деталізацію та покращувати візуальні характеристики зображень, знижуючи прояви шумів і артефактів. Зміна структурних показників свідчить про коректність роботи SR-модуля та відповідність алгоритмічних трансформацій очікуваній поведінці моделі.

Розроблений програмний засіб є комплексним рішенням для підвищення якості цифрових зображень, що поєднує гнучкість класичних підходів і можливості сучасних нейромережевих моделей. Практичні результати підтверджують ефективність запропонованого методу за умов низької початкової якості вхідних даних. Система може бути використана в задачах обробки фотографій, цифрової реставрації, технічного аналізу та в інших сферах, де важливо отримати деталізоване та візуально якісне зображення. Проведене дослідження демонструє актуальність теми та робить вагомий внесок у розвиток програмних рішень для інтелектуальної обробки зображень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Галімон Р.С., Добровольська Н.В.. Метод покращення якості зображень із низькою роздільною здатністю Конференція ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН-2026). Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2026/paper/view/25933

2. Gonzalez, R. C., & Woods, R. E., Digital Image Processing, 4th Edition, Pearson, 2018. 1104 p.

3. Jain, A. K., Fundamentals of Digital Image Processing, Prentice-Hall, 1989. 569 p.

4. Park, C., Park, M., & Kang, M., “Super-Resolution Image Reconstruction: A Technical Overview,” IEEE Signal Processing Magazine, vol. 20, no. 3, pp. 21– 36, 2003.

5. Dong, C., Loy, C. C., He, K., & Tang, X., “Image Super-Resolution Using Deep Convolutional Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 2, pp. 295–307, 2016.

6. Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., Qiao, Y., & Loy, C. C., “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” European Conference on Computer Vision (ECCV) Workshops, 2018.

7. Real-ESRGAN GitHub repository [Електронний ресурс] — Режим доступу: https://github.com/xinntao/Real-ESRGAN (дата звернення: 14.10.2024).

8. Ho, J., Jain, A., & Abbeel, P., “Denoising Diffusion Probabilistic Models,” Advances in Neural Information Processing Systems (NeurIPS), 2020.

9. Adobe Photoshop Documentation [Електронний ресурс] — Режим доступу: https://helpx.adobe.com/photoshop/user-guide.html (дата звернення: 14.10.2024).

10. Topaz Labs Gigapixel AI Documentation [Електронний ресурс] — Режим доступу: https://www.topazlabs.com/gigapixel-ai (дата звернення: 14.10.2024).

11. Waifu2x GitHub repository [Електронний ресурс] — Режим доступу: https://github.com/nagadomi/waifu2x (дата звернення: 14.10.2024).

12. OpenCV Documentation. [Електронний ресурс] — Режим доступу: https://docs.opencv.org/4.x/ (дата звернення: 16.10.2024).

13. Bovik, A. C., The Essential Guide to Image Processing, 2nd Edition, Academic Press, 2009.

14. Sonka M., Hlavac V., Boyle R. Image Processing, Analysis and Machine Vision. Cengage Learning, 2014. 920 p.

15. Jain A. K. Fundamentals of Digital Image Processing. Prentice-Hall, 1989. 569p.

16. Szeliski R. Computer Vision: Algorithms and Applications. Springer, 2022. — 950 p.

17. Keys R. Cubic convolution interpolation for digital image processing. IEEE Transactions on Acoustics, 1981.

18. Burger W., Burge M. Digital Image Processing: An Algorithmic Introduction Using Java. Springer, 2016. — 811 p.

19. Turkowski K. Filters for Common Resampling Tasks. Apple Computer Technical Report, 1990.

20. Tomasi C., Manduchi R. Bilateral Filtering for Gray and Color Images. IEEE ICCV, 1998.

21. Mitchell D., Netravali A. Reconstruction filters in computer graphics. ACM SIGGRAPH, 1988.

22. Buades A., Coll B., Morel J. A non-local algorithm for image denoising. CVPR, 2005.

23. Dong C. et al. Learning a Deep Convolutional Network for Image Super-Resolution (SRCNN). ECCV, 2014.

24. Ledig C. et al. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (SRGAN). CVPR, 2017.

25. Zhang K. et al. Beyond a Gaussian Denoiser: Residual Learning for Image Restoration (DnCNN). IEEE TIP, 2017.

26. PyTorch Official Documentation. [Електронний ресурс]. — Режим доступу: https://pytorch.org/docs (дата звернення: 20.10.2024).

27. Nvidia. Tensor Cores in DL Applications. Technical Whitepaper, 2022.

28. NumPy documentation. [Електронний ресурс]. — Режим доступу: https://numpy.org/doc/stable/ (дата звернення: 20.10.2024).

29. PyQt5 Reference Guide. [Електронний ресурс]. — Режим доступу: https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата звернення: 20.10.2024).

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. — Вінниця : ВНТУ, 2021. 42 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О. Д.
« 03 » 10 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
«Метод і програмний засіб покращення якості зображень із низькою роздільною здатністю»

Науковий керівник: к.пед.н., доц. каф. ОТ
_____ Добровольська Н. В.

Студент групи 1КІ-24м
_____ Галімон Р. С.

ВНТУ 2025

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність дослідження полягає в тому, що у багатьох практичних сферах часто використовуються зображення низької роздільної здатності, які містять шуми, артефакти та втрату дрібних деталей. Підвищення їхньої якості за допомогою сучасних методів є важливим для забезпечення точнішого аналізу, кращої візуальної інтерпретації та відновлення інформативності даних.

1.2 Наказ про затвердження теми МКР №313 від 24.09.2025 р.

2 Мета МКР і призначення розробки

2.1 Мета роботи — визначення ефективного підходу до покращення зображень із низькою роздільною здатністю та розробка програмного засобу для підвищення якості зображень шляхом застосування попередньої обробки та нейромережевої надроздільності.

2.2 Призначення розробки — програмний засіб призначений для покращення якості зображень із низькою роздільною здатністю шляхом автоматизованого застосування процедур попередньої обробки та нейромережевої надроздільності.

3 Вихідні дані для виконання МКР

3.1 Вхідні зображення низької роздільної здатності.

3.2 Попередньо навчені ваги моделі Real-ESRGAN.

3.3 Програмні інструменти: Python, PyTorch, OpenCV, PyQt5.

3.4 Необхідність забезпечення можливості збереження результатів та керування параметрами обробки через графічний інтерфейс.

4 Вимоги до виконання МКР

4.1 Проведення аналізу існуючих методів і підходів до підвищення якості зображень.

4.2 Дослідження методів підвищення якості зображень.

4.3 Роббробка програмного засобу.

4.4 Тестування розробленого програмного засобу.

4.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

5 Етапи МКР та очікувані результати

Етапи роботи та очікуванні результати приведено в таблиці А.1.

Таблиця А.1 — Етапи МКР

№	Назва етапів дипломної роботи	Термін виконання		Очікувані результати
		початок	закінчення	
1	Аналіз предметної області	25.09.2025	17.10.2025	Розділ 1
2	Дослідження методів	18.10.2025	25.10.2025	Розділ 2
3	Розробка програмного засобу	26.10.2025	02.11.2025	Розділ 3
4	Тестування програмного засобу	03.11.2025	10.11.2025	Розділ 4
5	Розрахунок економічної частини	11.11.2025	17.11.2025	Розділ 5
6	Апробація та впровадження результатів дослідження	18.11.2025	19.11.2025	Тези доповідей
7	Опублікування результатів	19.11.2025	20.11.2025	Стаття
8	Оформлення пояснювальної записки, графічного матеріалу і презентації	21.11.2025	01.12.2025	ПЗ, графіч. матеріал і презентація
9	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	02.12.2025	04.12.2025	Оформлені документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Едина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Метод і програмний засіб покращення якості зображень із низькою роздільною здатністю

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки, ФІТКІ, 1КІ-24м
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 13,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Азаров О. Д., д.т.н., зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Мартинюк Т. Б., д.т.н., проф. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Захарченко С. М., к.т.н., проф. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Галімон Р. С.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Блок-схема алгоритму роботи програми

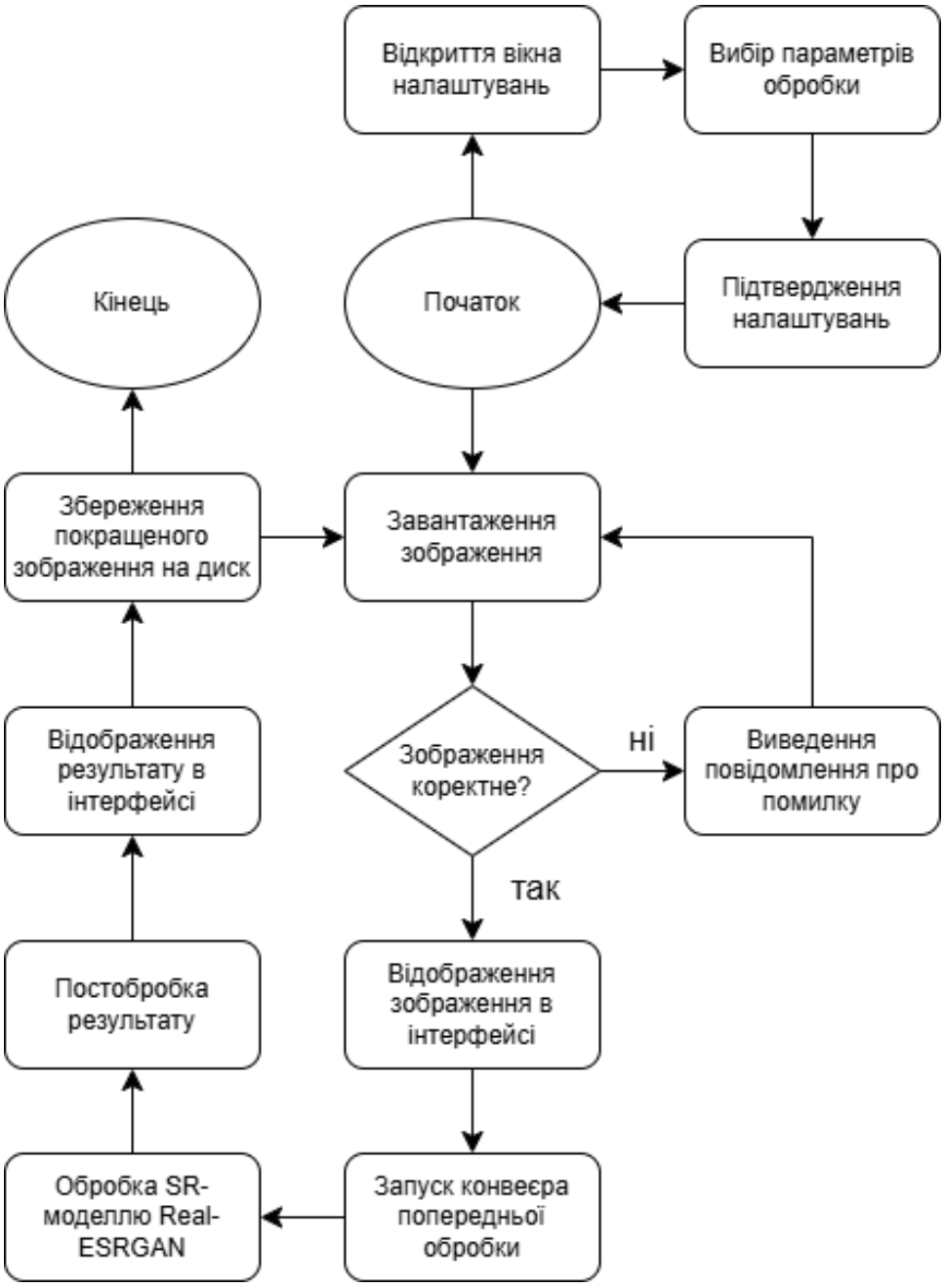

Рисунок В.1 — Блок-схема алгоритму роботи програми

ДОДАТОК Г

Лістинг налаштування моделі

```
from pathlib import Path
import time
import torch
from realesrgan import RealESRGANer
from basicsr.archs.rrdbnet_arch import RRDBNet

class ImageEnhancer:
    def __init__(self, model_path: Path):
        self.model_path = str(model_path)
        self._engine = None
        self._params = None

    def configure(self, *, device: str, half: bool, tile: int, tile_pad: int, pre_pad: int):
        if device == "cuda":
            dev = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        elif device == "cpu":
            dev = torch.device("cpu")
        else:  # "auto"
            dev = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        use_half = bool(half and dev.type == "cuda")

        rrdb = RRDBNet(
            num_in_ch=3, num_out_ch=3,
            num_feat=64, num_block=23, num_grow_ch=32,
            scale=4
        )

        new_params = dict(
            device=dev,
            half=use_half,
            scale=4,
            model_path=self.model_path,
            tile=int(tile),
            tile_pad=int(tile_pad),
            pre_pad=int(pre_pad),
        )

        if self._engine is not None and new_params == self._params:
            return
```

```
    self._engine = RealESRGANer(
        scale=new_params["scale"],
        model_path=new_params["model_path"],
        model=rrdb,
        tile=new_params["tile"],
        tile_pad=new_params["tile_pad"],
        pre_pad=new_params["pre_pad"],
        half=new_params["half"],
        device=new_params["device"],
    )
    self._params = new_params

def enhance_bgr(self, bgr):
    """
    Приймає BGR (OpenCV), повертає (BGR, seconds).
    """
    if self._engine is None:
        raise RuntimeError("ImageEnhancer is not configured. Call configure(...) first.")
    t0 = time.time()
    with torch.inference_mode():
        out, _ = self._engine.enhance(bgr, outscale=self._params["scale"])  # x4
    return out, time.time() - t0
```

ДОДАТОК Д

Лістинг головного вікна програми

```
from PyQt5 import QtWidgets, QtCore, QtGui
from pathlib import Path
from .image_processing import read_bgr, save_image
from .model_interface import ImageEnhancer
from .ui_settings import SettingsDialog, SettingsData
from .config import load_settings, save_settings

class PreviewPanel(QtWidgets.QFrame):
    fileDropped = QtCore.pyqtSignal(str)

    def __init__(self, title: str, parent=None, allow_drop: bool = False):
        super().__init__(parent)
        self.setWindowIcon(QtGui.QIcon("logo.png"))
        self.setObjectName("PreviewPanel")
        self.setFrameShape(QtWidgets.QFrame.StyledPanel)
        self.setFrameShadow(QtWidgets.QFrame.Raised)
        self.title = QtWidgets.QLabel(title)
        self.title.setObjectName("TitleLabel")
        self.view  = QtWidgets.QLabel(" ")
        self.view.setObjectName("PreviewArea")
        self.view.setAlignment(QtCore.Qt.AlignCenter)
        self.view.setMinimumSize(320, 180)
        self.view.setSizePolicy(QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
        if allow_drop:
            self.view.setText("Drop image here")
            self.view.setStyleSheet("color:#8a9099; font-size:30px;")
        self._allow_drop = bool(allow_drop)
        self.setAcceptDrops(True)
        lay = QtWidgets.QVBoxLayout(self)
        lay.setContentsMargins(16, 12, 16, 16)
        lay.setSpacing(8)
        lay.addWidget(self.title)
        lay.addWidget(self.view, 1)

    def dragEnterEvent(self, e: QtGui.QDragEnterEvent) -> None:
        if not self._allow_drop:
            e.ignore()
            return
        md = e.mimeData()
        ok = False
```

```
        if md.hasUrls():
            urls = [u for u in md.urls() if u.isLocalFile()]
            if urls:
                suffix = urls[0].toLocalFile().lower()
                ok = suffix.endswith((".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff", ".webp"))
        elif md.hasText():
            text = md.text().strip("").strip()
            ok = text.lower().endswith((".png", ".jpg", ".jpeg", ".bmp", ".tif", ".tiff",
".webp"))
        if ok:
            e.acceptProposedAction()
        else:
            e.ignore()

    def dragMoveEvent(self, e: QtGui.QDragMoveEvent) -> None:
        if self._allow_drop:
            e.setDropAction(QtCore.Qt.CopyAction)
            e.accept()
        else:
            e.ignore()

    def dragLeaveEvent(self, e: QtGui.QDragLeaveEvent) -> None:
        e.accept()

    def dropEvent(self, e: QtGui.QDropEvent) -> None:
        if not self._allow_drop:
            e.ignore()
            return
        md = e.mimeData()
        path = None
        if md.hasUrls():
            for u in md.urls():
                if u.isLocalFile():
                    path = u.toLocalFile()
                    break
        elif md.hasText():
            path = md.text().strip("").strip()
        if path:
            self.fileDropped.emit(path)
            e.acceptProposedAction()
        else:
            e.ignore()

    def set_scaled_pixmap(self, pm: QtGui.QPixmap):
        if pm is None or pm.isNull():
```

```
            self.view.setText("No image")
            self.view.setPixmap(QtGui.QPixmap())
            return
        avail = self.area.contentsRect().size()
        dpr = pm.devicePixelRatio() if pm.devicePixelRatio() > 0 else 1.0
        base = pm.size() / dpr
        target = base.scaled(avail, QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation)
        self.area.setPixmap(pm.scaled(target, QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation))

class Worker(QtCore.QThread):
    finished = QtCore.pyqtSignal(object, float, str)
    failed   = QtCore.pyqtSignal(str)

    def __init__(self, enhancer: ImageEnhancer, in_path: Path, out_path: Path, jpeg_q:
int, parent=None):
        super().__init__(parent)
        self.enhancer = enhancer
        self.in_path  = in_path
        self.out_path = out_path
        self.jpeg_q   = jpeg_q

    def run(self):
        try:
            bgr = read_bgr(self.in_path)
            out_bgr, secs = self.enhancer.enhance_bgr(bgr)
            pm = self._to_pixmap(out_bgr)
            self.finished.emit(pm, secs, "(not saved yet)")
        except Exception as e:
            self.failed.emit(f"{type(e).__name__}: {e}")

    @staticmethod
    def _to_pixmap(bgr):
        import cv2
        rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb.shape
        bytes_per_line = ch * w
        qimg = QtGui.QImage(rgb.data, w, h, bytes_per_line,
QtGui.QImage.Format_RGB888).copy()
        return QtGui.QPixmap.fromImage(qimg)

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self, enhancer, images_dir, default_outname, jpeg_quality):
        super().__init__()
```

```
        self.setWindowTitle("Image Upscaler")
        self.resize(1280, 720)
        self._pm_left_original: QtGui.QPixmap = None
        self._pm_right_original: QtGui.QPixmap = None
        self.enhancer = enhancer
        self.images_dir = images_dir
        self.default_outname = default_outname
        self.settings: SettingsData = load_settings()
        self._apply_settings_to_engine()
        self.current_in: Path = None
        self.current_out: Path = None
        self.result_pixmap: QtGui.QPixmap = None
        self._build_ui()

    def _build_ui(self):
        central = QtWidgets.QWidget()
        self.setCentralWidget(central)
        self.left  = PreviewPanel("Original Image", self, allow_drop=True)
        self.right = PreviewPanel("Enhanced Image", self, allow_drop=False)
        self.left.fileDropped.connect(self._on_left_drop)
        self.splitter = QtWidgets.QSplitter(QtCore.Qt.Horizontal)
        self.splitter.addWidget(self.left)
        self.splitter.addWidget(self.right)
        self.splitter.setSizes([1, 1])
        self.splitter.splitterMoved.connect(lambda *_: self._refit_previews())
        self.btnOpen     = QtWidgets.QPushButton("Open")
        self.btnUpscale  = QtWidgets.QPushButton("Upscale")
        self.btnSave     = QtWidgets.QPushButton("Save")
        self.btnSettings = QtWidgets.QPushButton("Settings")
        for b in (self.btnOpen, self.btnUpscale, self.btnSave, self.btnSettings):
            b.setMinimumHeight(40)
            b.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
        centerGroup = QtWidgets.QWidget()
        centerLay   = QtWidgets.QHBoxLayout(centerGroup)
        centerLay.setContentsMargins(0,0,0,0)
        centerLay.setSpacing(8)
        centerLay.addWidget(self.btnOpen)
        centerLay.addWidget(self.btnUpscale)
        centerLay.addWidget(self.btnSave)
        centerGroup.setSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        bottomGrid = QtWidgets.QGridLayout()
        bottomGrid.setContentsMargins(0, 0, 0, 0)
        bottomGrid.setVerticalSpacing(0)
        bottomGrid.setHorizontalSpacing(12)
```

```
        bottomGrid.setColumnStretch(0, 1)
        bottomGrid.setColumnStretch(1, 0)
        bottomGrid.setColumnStretch(2, 1)
        leftSpacer = QtWidgets.QSpacerItem(0, 0, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        bottomGrid.addItem(leftSpacer, 0, 0)
        bottomGrid.addWidget(centerGroup, 0, 1, alignment=QtCore.Qt.AlignHCenter |
QtCore.Qt.AlignVCenter)
        bottomGrid.addWidget(self.btnSettings, 0, 2, alignment=QtCore.Qt.AlignRight |
QtCore.Qt.AlignVCenter)
        root = QtWidgets.QVBoxLayout(central)
        root.setContentsMargins(16, 16, 16, 16)
        root.setSpacing(12)
        root.addWidget(self.splitter, 1)
        root.addLayout(bottomGrid)
        self.btnOpen.clicked.connect(self.on_open)
        self.btnUpscale.clicked.connect(self.on_upscale)
        self.btnSave.clicked.connect(self.on_save)
        self.btnSettings.clicked.connect(self.on_settings)
        self.current_in: Path = None
        self.current_out: Path = None
        self.result_pixmap: QtGui.QPixmap = None

    def on_settings(self):
        dlg = SettingsDialog(self.settings, self)
        if dlg.exec_() == QtWidgets.QDialog.Accepted:
            self.settings = dlg.result_data()
            save_settings(self.settings)
            self._apply_settings_to_engine()
            QtWidgets.QMessageBox.information(self, "Settings", "Settings have been
applied.")

        import torch
        print("Device:", torch.cuda.is_available(), torch.cuda.get_device_name(0) if
torch.cuda.is_available() else "CPU")

    def _apply_settings_to_engine(self):
        self.enhancer.configure(
            device=self.settings.device,
            half=self.settings.half,
            tile=self.settings.tile,
            tile_pad=self.settings.tile_pad,
            pre_pad=self.settings.pre_pad,
        )
```

```
    def on_open(self):
        p, _ = QtWidgets.QFileDialog.getOpenFileName(
            self, "Open Image", str(self.images_dir),
            "Images (*.png *.jpg *.jpeg *.bmp *.tif *.tiff *.webp)"
        )
        if not p:
            return
        self.current_in = Path(p)
        pix = QtGui.QPixmap(str(self.current_in))
        if pix.isNull():
            QtWidgets.QMessageBox.warning(self, "Image Upscaler", "Cannot open
image.")
            return

        self._pm_left_original = pix
        self._refit_previews()
        self.right.view.clear()
        self._pm_right_original = None

    def on_upscale(self):
        if not self.current_in:
            QtWidgets.QMessageBox.information(self, "Image Upscaler", "Open an image
first.")
            return
        self.current_out = self.current_in.parent / self.default_outname

        self._set_controls(False)

        self.worker = Worker(
            enhancer=self.enhancer,
            in_path=self.current_in,
            out_path=self.current_out,
            jpeg_q=self.settings.jpeg_quality,
            parent=self
        )
        self.worker.finished.connect(self._on_done)
        self.worker.failed.connect(self._on_failed)
        self.worker.start()

    def _on_done(self, pixmap: QtGui.QPixmap, secs: float, out_name: str):
        self.result_pixmap = pixmap
        self._pm_right_original = pixmap
        self._refit_previews()
```

```
        QtWidgets.QMessageBox.information(self, "Image Upscaler", f"Upscaled in
{secs:.2f}s.\nUse Save to export.")
        self._set_controls(True)

    def _on_failed(self, msg: str):
        QtWidgets.QMessageBox.critical(self, "Image Upscaler", msg)
        self._set_controls(True)

    def on_save(self):
        if not self.result_pixmap:
            QtWidgets.QMessageBox.information(self, "Image Upscaler", "There is no
enhanced image to save.")
            return

        default_path = (self.current_in.parent if self.current_in else self.images_dir) /
f"output_x{4}.{ 'jpg' if self.settings.output_format=='JPEG' else 'png'}"
        filt = "PNG (*.png);;JPEG (*.jpg *.jpeg)"
        p, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Save Enhanced",
str(default_path), filt)
        if not p:
            return

        if self.settings.output_format == "JPEG" or p.lower().endswith((".jpg", ".jpeg")):
            self.result_pixmap.save(p, "JPEG", quality=self.settings.jpeg_quality)
        else:
            self.result_pixmap.save(p, "PNG")

    def _scaled_for_label(self, label: QtWidgets.QLabel, pm: QtGui.QPixmap) ->
QtGui.QPixmap:
        avail = label.contentsRect().size()
        if pm is None or pm.isNull() or not avail.isValid():
            return pm
        dpr = pm.devicePixelRatio() if pm.devicePixelRatio() > 0 else 1.0
        base_size = QtCore.QSizeF(pm.size()) / dpr
        scaled_size = base_size.scaled(
            QtCore.QSizeF(avail.width(), avail.height()),
            QtCore.Qt.KeepAspectRatio
        )
        target_size = QtCore.QSize(int(scaled_size.width()), int(scaled_size.height()))
        return pm.scaled(target_size, QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation)

    def _refit_previews(self):
        if self._pm_left_original is not None:
```

```
            self.left.view.setPixmap(self._scaled_for_label(self.left.view,
self._pm_left_original))
        if self._pm_right_original is not None:
            self.right.view.setPixmap(self._scaled_for_label(self.right.view,
self._pm_right_original))

    def _on_left_drop(self, path: str):
        p = Path(path)
        if not p.exists():
            QtWidgets.QMessageBox.warning(self, "Image Upscaler", "File not found.")
            return
        pix = QtGui.QPixmap(str(p))
        if pix.isNull():
            QtWidgets.QMessageBox.warning(self, "Image Upscaler", "Unsupported or
corrupted image.")
            return

        self.current_in = p
        self._pm_left_original = pix
        self._pm_right_original = None
        self.right.view.clear()
        self._refit_previews()

    @staticmethod
    def _set_scaled(label: QtWidgets.QLabel, pix: QtGui.QPixmap):
        avail = label.contentsRect().size()
        if pix is None or pix.isNull() or not avail.isValid():
            label.setPixmap(QtGui.QPixmap())
            return
        dpr = pix.devicePixelRatio() if pix.devicePixelRatio() > 0 else 1.0
        base = pix.size() / dpr
        target = base.scaled(avail, QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation)
        label.setPixmap(pix.scaled(target, QtCore.Qt.KeepAspectRatio,
QtCore.Qt.SmoothTransformation))

    def resizeEvent(self, e: QtGui.QResizeEvent) -> None:
        self._refit_previews()
        super().resizeEvent(e)

    def _set_controls(self, enabled: bool):
        self.btnOpen.setEnabled(enabled)
        self.btnUpscale.setEnabled(enabled)
        self.btnSave.setEnabled(enabled)
```

ДОДАТОК Е

Лістинг вікна налаштувань

```
from PyQt5 import QtWidgets, QtGui, QtCore
from dataclasses import dataclass
@dataclass
class SettingsData:
    device: str
    half: bool
    outscale: int
    tile: int
    tile_pad: int
    pre_pad: int
    output_format: str
    jpeg_quality: int

class SettingsDialog(QtWidgets.QDialog):
    def __init__(self, s: SettingsData, parent=None):
        super().__init__(parent)
        from app.config import ICON_PATH
        self.setWindowIcon(QtGui.QIcon(str(ICON_PATH)))
        self.setWindowTitle("Settings")
        self.setModal(True)
        self._scale = 1.5
        self._build_ui(s)
        self._apply_dark_theme()
        try:
            from .win_titlebar import set_dark_titlebar_for_widget
            set_dark_titlebar_for_widget(self)
        except Exception:
            pass

    def _build_ui(self, s: SettingsData):
        self.cmbDevice = QtWidgets.QComboBox()
        self.cmbDevice.addItems(["auto", "cpu", "cuda"])
        self.cmbDevice.setCurrentText(s.device)
        self.chkHalf = QtWidgets.QCheckBox("Half precision (GPU only)")
        self.chkHalf.setChecked(s.half)
        self.spnTile = QtWidgets.QSpinBox();    self.spnTile.setRange(64, 1024);
self.spnTile.setValue(s.tile); self.spnTile.setSingleStep(32)
        self.spnTilePad = QtWidgets.QSpinBox(); self.spnTilePad.setRange(0, 64);
self.spnTilePad.setValue(s.tile_pad)
        self.spnPrePad = QtWidgets.QSpinBox();  self.spnPrePad.setRange(0, 64);
self.spnPrePad.setValue(s.pre_pad)
```

```
        self.cmbFormat = QtWidgets.QComboBox()
        self.cmbFormat.addItems(["PNG", "JPEG"])
        self.cmbFormat.setCurrentText(s.output_format)
        self.spnQ = QtWidgets.QSpinBox(); self.spnQ.setRange(1, 100);
self.spnQ.setValue(s.jpeg_quality)
        form = QtWidgets.QFormLayout()
        form.setVerticalSpacing(int(12 * self._scale))
        form.setLabelAlignment(QtCore.Qt.AlignLeft | QtCore.Qt.AlignVCenter)
        form.addRow("Device:", self.cmbDevice)
        form.addRow("", self.chkHalf)
        form.addRow("Tile:", self.spnTile)
        form.addRow("Tile pad:", self.spnTilePad)
        form.addRow("Pre pad:", self.spnPrePad)
        form.addRow("Output format:", self.cmbFormat)
        form.addRow("JPEG quality:", self.spnQ)
        btns = QtWidgets.QDialogButtonBox(QtWidgets.QDialogButtonBox.Ok |
QtWidgets.QDialogButtonBox.Cancel)
        btns.accepted.connect(self.accept); btns.rejected.connect(self.reject)
        root = QtWidgets.QVBoxLayout(self)
        root.addLayout(form)
        root.addWidget(btns)
    def result_data(self) -> SettingsData:
        return SettingsData(
            device=self.cmbDevice.currentText(),
            half=self.chkHalf.isChecked(),
            outscale=4,
            tile=int(self.spnTile.value()),
            tile_pad=int(self.spnTilePad.value()),
            pre_pad=int(self.spnPrePad.value()),
            output_format=self.cmbFormat.currentText(),
            jpeg_quality=int(self.spnQ.value()),
        )
```