

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Розробка та дослідження ефективних моделей
глибокого навчання для підвищення роздільної
здатності зображень і відео послідовностей»**

Виконав студент групи ІІЗм-24-1
КОСТЮЧЕНКО Артем Валерійович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
САВІЦЬКИЙ Роман Святославович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОСТЮЧЕНКО Артем Валерійович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Розробка та дослідження ефективних моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 491/с**
Вихідні дані для роботи: **об'єктом дослідження є процес підвищення роздільної здатності зображень і відео послідовностей за допомогою моделей глибокого навчання, а предметом дослідження є створення моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	22.09.2025р.	22.09.2025р.
2	Локтікова Т.М.	13.10.2025р.	13.10.2025р.
3	Локтікова Т.М.	29.10.2025р.	29.10.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота освітнього ступеня «магістр» присвячена розробці та дослідженню ефективних моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей. Пояснювальна записка до кваліфікаційної магістерської роботи містить 69 сторінок друкованого тексту, 44 ілюстрацій та 4 таблиці. Список використаних джерел нараховує 40 найменувань та займає 4 сторінки. В роботі є 2 додатки на 13 сторінках.

Метою роботи є розробка та дослідження ефективних моделей глибокого навчання для підвищення роздільної здатності зображень зі зручним графічним інтерфейсом для тренування, тестування та застосування моделей.

У роботі описано процес створення моделей, програмного продукту для зручності використання моделей з можливістю тренування та розробку інтерфейсу. Досліджено моделі ESRGAN, R-ESRGAN, RT4KSR, а також створену модель RTSR для підвищення роздільної здатності зображень через нейронні мережі. Проведено тестування моделей на незалежних наборах даних.

КЛЮЧОВІ СЛОВА: РОЗДІЛЬНА ЗДАТНІСТЬ, ЗОБРАЖЕННЯ, ВІДЕО, ОБРОБКА, МАСШТАБУВАННЯ, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ.

ABSTRACT

The final qualification work for the educational degree of “Master” is devoted to the development and research of effective deep learning models for improving the resolution of images and video sequences. The explanatory note to the qualification master's thesis contains 69 pages of printed text, 44 illustrations, and 4 tables. The list of references contains 40 titles and occupies 4 pages. The thesis has 2 appendices on 13 pages.

The goal of this work is to develop and study effective deep learning models for improving image resolution with a user-friendly graphical interface for training, testing, and using models.

The paper describes the process of creating models, software products for convenient use of models with training capabilities, and interface development. The ESRGAN, R-ESRGAN, RT4KSR models, as well as the RTSR model created to improve image resolution through neural networks, were studied. The models were tested on independent datasets.

KEY WORDS: RESOLUTION, IMAGE, VIDEO, PROCESSING, SCALING, DEEP LEARNING, NEURAL NETWORKS.

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 АНАЛІЗ ПРОЦЕСУ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ	10
1.2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ	12
1.3 АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ	18
1.4 ПОСТАНОВКА ЗАДАЧІ	22
Висновки до першого розділу	24
РОЗДІЛ 2 РОЗРОБКА МОДЕЛЕЙ ДЛЯ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ	25
2.1 ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ МОДЕЛЕЙ	25
2.2 ПОБУДОВА МОДЕЛЕЙ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ	26
2.3 СТВОРЕННЯ НАБОРІВ ДАНИХ	35
2.4 ПРОЦЕС НАВЧАННЯ СТВОРЕНИХ МОДЕЛЕЙ	37
Висновки до другого розділу	39
РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА ТЕСТУВАННЯ СТВОРЕНИХ МОДЕЛЕЙ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ	40
3.1 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ.....	40
3.2 ОПИС ІНТЕРФЕЙСУ ПРОГРАМНОГО ПРОДУКТУ	43
3.3 ТЕСТУВАННЯ РОБОТИ СТВОРЕНИХ МОДЕЛЕЙ	56
Висновки до третього розділу	61
ВИСНОВКИ	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NN – нейронна мережа (англ. neural network)

CNN – згорткова нейронна мережа (англ. convolutional neural network)

GAN – генеративно-змагальна мережа (англ. generative adversarial network)

SR – надвисока роздільна здатність (англ. super resolution)

ISR – надвисока роздільна здатність зображень (англ. image super resolution)

VSR – надвисока роздільна здатність відео (англ. video super resolution)

HR – висока роздільна здатність (англ. high resolution)

LR – низька роздільна здатність (англ. low resolution)

BN – пакетна нормалізація (англ. batch normalization)

Adam – адаптаційний імпульс (англ. adaptive momentum)

ВСТУП

Актуальність теми зумовлено зростанням потреби у якісному цифровому контенті. Роздільна здатність впливає на чіткість деталей, можливість аналізу сцени та точність подальшої обробки. У багатьох сферах використовуються дані, які мають низьку якість через обмеження обладнання або умови зйомки, саме такі матеріали потребують відновлення, щоби їх можна було застосовувати в роботі та різних дослідженнях.

Глибоке навчання дозволяє відновити структуру зображення та покращити його якість. Моделі здатні відтворювати втрачені деталі та формувати якісніші версії вхідних даних, що підвищує можливість проведення точного аналізу у різних напрямках. Якість відео також має значення, оскільки відео послідовності часто містять рухомі об'єкти, які важко відновити традиційними методами.

Проблема зростає разом зі збільшенням обсягів даних і вимог до їхньої точності. У багатьох системах застосовуються матеріали, які не можуть бути повторно зняті, тому покращення роздільної здатності є одним із способів для підвищення їхньої якості. Відповідно через моделі глибокого навчання є можливість працювати з такими складними випадками, де стандартні алгоритми не забезпечують потрібної чіткості.

Через ці проблеми, дослідження методів підвищення роздільної здатності є важливим для розвитку сучасних цифрових технологій. Це має практичне значення, тому що підвищена якість даних покращує точність рішень, ефективність систем та надійність подальшої обробки.

Метою випускної кваліфікаційної роботи магістра є розробка та дослідження ефективних моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей з можливістю зручного використання через програмний продукт.

Поставлена мета обумовлює необхідність виконання наступних **завдань**:

1. Дослідити роботу нейронних моделей для підвищення роздільної здатності зображень та відео послідовностей.

2. Дослідити різні архітектури для побудови нейронних моделей для підвищення роздільної здатності зображень і відео послідовностей.

3. Розробити ефективні нейронні моделі на основі проведених досліджень.

4. Створити програмний продукт з використанням глибокого навчання, який дозволить тренувати різні моделі на різних архітектурах; тестувати якість, точність та швидкість обробки на різних наборах даних; обробляти зображення та відео послідовності з можливістю їхнього порівняння та перегляду; конвертувати створені моделі в формати FP16 або INT8 для пришвидшення обробки.

Об'єктом дослідження є процес підвищення роздільної здатності зображень і відео послідовностей за допомогою моделей глибокого навчання.

Предметом дослідження є створення моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей.

Наукова новизна роботи полягає у застосуванні архітектур та методів глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей, побудові архітектур моделей, здатних адаптуватися до різних типів вхідних даних і покращувати якість зображення без втрати змісту.

Практичне значення полягає у використанні розроблених моделей глибокого навчання для покращення якості зображень і відео на різних цифрових пристроях. Отримані моделі можуть бути використанні у різних програмних продуктах, які виконують відновлення та обробку візуальних даних.

Апробація. За темою випускної кваліфікаційної роботи магістра:

1) опубліковано тези доповіді Kostyuchenko A., Loktikova T., Lysogor Iu. Development of a Neural Model for Image and Video Resolution Enhancement. Тези доповідей XV Міжнародної науково-технічної конференції

«Інформаційно-комп'ютерні технології», 28-29 березня 2025 року. Житомир: «Житомирська політехніка», 2025. С. 41-42 [1];

2) подана та прийнята до опублікування стаття Kostyuchenko A., Kushnir N., Loktikova T. Development and research of effective deep learning models for improving the resolution of images and video sequences. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія Технічні науки. 2025. №6 [2].

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз процесу підвищення роздільної здатності зображень і відео послідовностей

Підвищення роздільної здатності – це процес збільшення кількості пікселів зображення, головною метою якого є збереження цілісності та покращення його якості. В більшості випадків збільшене зображення чи відео має більш кращу деталізацію та краще сприймається візуально порівняно з оригінальним. Підвищення роздільної здатності використовується у різних випадках, але переважно тоді, коли зображення отримується в поганій якості через камеру низької якості, застарілість, розмитість, артефакти чи освітленість, тим самим втрачаються деякі деталі. Використання цього процесу допомагає відновити втрачені деталі та покращити якість зображення чи відео.

Підвищення роздільної здатності відбувається наступним чином: вхідне зображення чи відео низької роздільної здатності оброблюється за допомогою певного методу чи алгоритму, в результаті чого отримується результат з вищою роздільною здатністю. Метод чи алгоритм виконує збільшення пікселів на основі сусідніх пікселів з збереженням природності та точності відтворення до оригіналу. Порівнюючи з простим масштабуванням, при підвищенні роздільної здатності відбувається передача нових деталей, які були приховані чи втрачені зі збереженням структури, контурів та текстур зображення чи відео [3, 4].

Існують різні методи підвищення роздільної здатності, розглянемо найбільш відомі. До появи нейронних мереж використовувалися традиційні алгоритми, які базуються на так званій інтерполяції. Інтерполяція є процесом, де відбувається знаходження проміжних значень величини за наявним дискретним набором відомих значень. В підвищенні роздільної здатності, інтерполяція виконує роль створення нових пікселів на основі значень сусідніх

пікселів. Найбільш відомими інтерполяціями є найближчий сусід, білінійна та бікубічна. Найближчий сусід бере значення найближчого пікселя без обчислень з невеликим помітним результатом на межах зображення, білінійна інтерполяція обчислює середнє значення на основі чотирьох сусідніх пікселів між якими утворюється плавніший перехід, бікубічна інтерполяція враховує шістнадцять сусідніх пікселів з складними обчисленнями, в результаті чого отримується більш м'яке та детальне зображення. Вони прості у використанні, але мають проблеми при збільшенні. При збільшенні зображення за допомогою інтерполяції, зображення матиме розмиття, а контури втратять якість і стануть менш чіткими. Традиційні методи не враховують закономірність як повинно виглядати зображення при збільшенні, тому й не здатні відновити дрібні деталі, які були в оригіналі [5, 6].

З розвитком технологій та машинного навчання, в підгалузі глибокого навчання було створено архітектури, які збільшують зображення, додатково аналізують деталі та відтворюють їх з високою точністю. Моделі, створені на таких архітектурах, працюють з парою зображень “низька роздільна здатність” – “висока роздільна здатність”, завдяки чому навчаються передбачати, як зображення повинне виглядати у високій роздільній здатності. Архітектури включають різні можливості, завдяки яким на основі пікселів реконструюються текстури, тіні, освітлення та інші особливості. Для відеопослідовностей враховується часовий проміжок між кадрами, оскільки можуть бути стрибки між зображеннями та зміни контрасту, в результаті чого результат буде менш привабливим [7, 8].

Перед підвищенням роздільної здатності зображення відбувається попередня обробка оригінального зображення. Зображення нормалізується, усуваються так звані шуми, які виникають з різних причин, наприклад, проблеми з камерою, пошкодження при передачі через мережу чи пристрій; якщо відео, то його стабілізація для кращого вихідного результату. Після обробки відбувається масштабування або реконструкція за допомогою певного методу чи алгоритму. Після збільшення на зображенні проводиться

тестування за показниками якості, такими як різкість, структурна подібність, відношення сигналу до шуму і т.п. Якщо це нейронна мережа, показники часто використовуються під час навчання, щоби модель поступово покращувала якість відновлення.

Для архітектур підвищення роздільної здатності відео послідовностей додатково використовуються і інші архітектури та методи, які враховують просторову інформацію, рух об'єктів між кадрами, що додає плавність переходів між кадрами. Під час обробки відео модель використовує методи оптичного потоку або рекурентні архітектури, які аналізують контекст попередніх та наступних кадрів. У результаті відео залишається у високій роздільній здатності без втрати плавності руху [9, 10].

Під час підвищення роздільної здатності потрібно зберігати баланс між якістю та швидкістю обробки, де обладнання має обмежені обчислювальні можливості. Для цього потрібно досліджувати різні архітектури, які здатні якісно виконувати обрахунки без значних обчислень з найменшою втратою точності. Правильно спроектувавши ефективну модель, її можна використовувати на мобільних пристроях, потокових сервісах та інших системах.

1.2 Аналіз існуючих методів підвищення роздільної здатності

У задачах підвищення роздільної здатності зображень та відео послідовностей використовуються різні архітектури для побудови моделей глибокого навчання. Кожна модель побудована з урахуванням певних вимог до якості, швидкості обробки та типу вхідних даних. Наразі існує велика кількість архітектур, здатних підвищити роздільну здатність зображення чи відео у декілька разів. Розглянемо найбільш відомі, які використовувалися чи використовуються донині.

Найвідомішою й найпершою нейронною мережею, у якій було використано згорткову архітектуру для підвищення роздільної здатності, була

SRCNN (Super Resolution Convolutional Neural Network) [11]. Задача мережі полягала в покращенні зображення, яке розглядалося як відображення з простору зображень низької роздільної здатності у простір зображень високої роздільної здатності. Вхідне зображення спочатку масштабується простим методом, після чого передається до мережі для уточнення масштабування.

Класичний варіант SRCNN архітектури складається з трьох послідовних згорткових шарів. Перший шар виділяє базові ознаки (features) та локальні структури з зображення, наступний шар відображає ці ознаки у більш придатне для відновлення представлення і в останньому шарі формується фінальне зображення з покращеними значеннями пікселів, завдяки чому зображення має відновлені деталі, покращені контури, границі та плавні переходи кольорів [12]. Вигляд архітектури SRCNN показано на рис. 1.1.

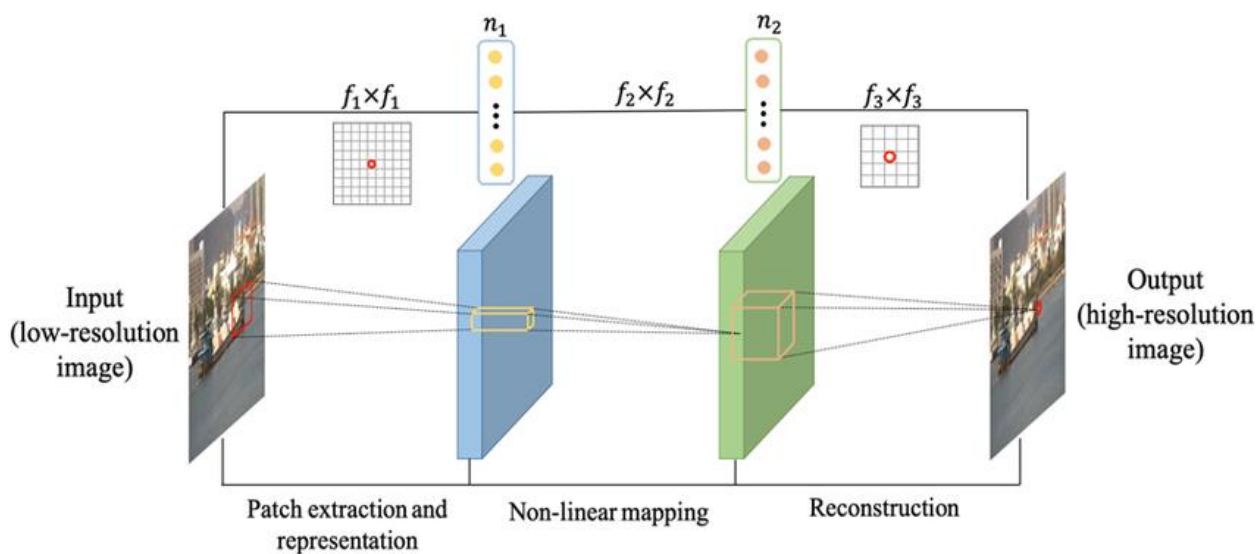


Рисунок 1.1 – Архітектура SRCNN

Однією з переваг архітектури є її простота побудови та зрозумілий принцип роботи. Мережа має кращі результати якості порівняно з традиційними методами інтерполяції, при цьому не має велику глибину та ширину, тому її навчання та використання не потребують великих обчислювальних ресурсів.

Невелика кількість шарів хоч і дозволяє покращити зображень, але повноцінно відновити складніші текстури та дрібніші деталі мережа не здатна. Якщо масштаб збільшення занадто великий, фінальне зображення може мати розмитість, деякі структури стають згладженими. Через ці недоліки продовжилося дослідження створення більш глибоких та гнучкіших архітектур.

З розвитком підходів до згорткових мереж, які використовуються для підвищення роздільної здатності, було розроблено модель EDSR (Enhanced Deep Super Resolution) [13]. Модель використовує залишкові блоки, які дозволяють будувати глибокі мережі без деградації якості навчання. В кожному блоці подається сигнал через послідовність шарів та пропускну зв'язку, завдяки чому полегшується оптимізація мережі. У залишкових блоках було прибрано пакетну нормалізацію (Batch Normalization, BN) за міні-пакетами, завдяки чому модель зберігає інформацію про рівень яскравості та контрасту під час відновлення зображень. В результаті модель краще відтворює структуру та переходить між пікселями, не спотворюючи зображення.

Архітектура EDSR використовує різну кількість залишкових блоків, через які проходять ознаки. Чим глибше модель, тим більше ознак буде витягнуто з зображення і навпаки. Завдяки більшій глибині, модель, з великою кількістю даних для тренування, навчається краще на прикладах і накопичує інформацію про різні типи текстур та контурів. В кінцевому блоці просторовий розмір зображення зростає, через що кінцеве зображення отримує більше деталей. На рис. 1.2 наведено архітектуру моделі EDSR.

Натренована модель демонструє високі показники за різними метриками під час тестування. Модель здатна відновлювати дрібні деталі, працює з багатьма різними масштабами збільшення зображень, підходить для різних практичних задач. Але є й недолік: чим глибша модель, тим більше обчислювальних ресурсів вона використовує. Її можна використовувати на слабких пристроях, але потрібно обмежувати глибину, тим самим втрачаючи якість результату.

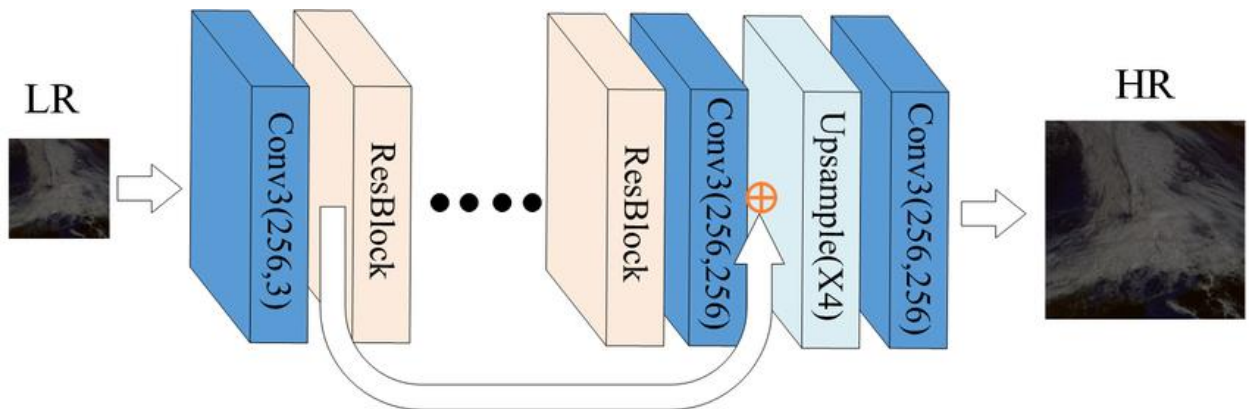


Рисунок 1.2 – Архітектура моделі EDSR

Наступним розвитком стали моделі, які беруть до уваги якості та візуальне сприйняття людським оком. Однією з таких моделей є ESRGAN (Enhanced Super Resolution Generative Adversarial Networks) [14]. На відміну від інших моделей, де використовуються згорткові шари, ESRGAN використовує генеративний підхід, який відновлює піксельні значення та формує правдоподібні текстури.

Модель складається з двох частин: генератора та дискримінатора. Генератор створює зображення високої роздільної здатності, а дискримінатор оцінює роботу генератора, чи зображення є схожим на справжнє. Між ними відбувається так зване змагання, де генератор та дискримінатор через деякий час стають сильнішими, в результаті чого генератор створює зображення, які складно відрізнити від справжніх. У архітектурі генератора використовуються покращені залишкові блоки, які складаються з декількох згорткових шарів з пропускними зв'язками без пакетної нормалізації за міні-пакетом, як і в EDSR. Модель використовує перцепційну функцію втрат, яка враховує різницю між пікселями та відмінність у просторі ознак, виділених іншою мережею. Завдяки використанню цієї втрати, модель навчається відтворювати зображення, що краще сприймається людиною. Архітектуру моделі зображено на рис. 1.3.

Однією з переваг моделі є її привабливий результат. Відновлені зображення мають чіткі краї, виразні структури та текстури, які більше

нагадують реальні. Але є і мінуси такого відновлення. Генератор під час створення може додати деталі, яких не було в оригіналі, тому задачі, де важливо мати результат, строго схожий до оригінального, відпадають. Через це такі моделі часто використовуються для обробки контенту, орієнтованого на перегляд, реставрацію архівних зображень або підвищення якості графіки, наприклад в іграх.

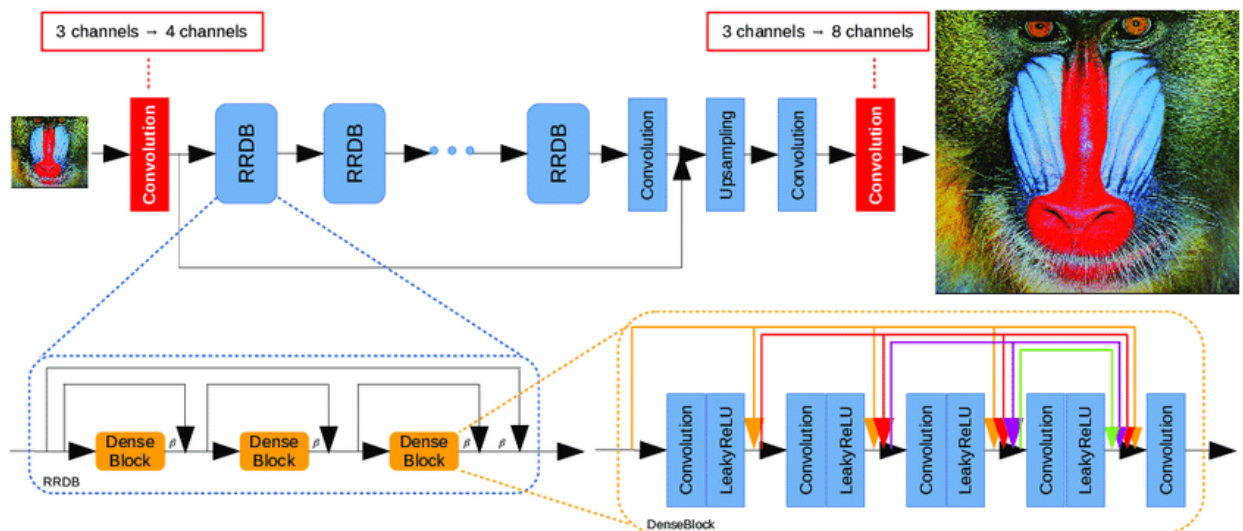


Рисунок 1.3 – Архітектура моделі ESRGAN

Модель Real-ESRGAN [15] є нащадком ESRGAN, яка була створена з урахуванням реальних умов, в яких отримуються зображення. Часто такі зображення містять шуми, розмиття різного походження, артефакти від кодеків, стиснення та інші спотворення. Для обробки таких зображень модель повинна бути стійкою до спотворень.

Під час навчання, модель використовує складнішу схему формування зображень, ніж ESRGAN. Зображення низької роздільної здатності штучно перетворюється у варіанти з низькою роздільною здатністю з додаванням шуму, розмиття, стиснення та інших спотворень. Мережа навчається відновлювати структуру зображення за наявності цих спотворень з масштабуванням. Архітектура моделі, на відміну від ESRGAN, адаптується

під грубіші дані. У ній використовуються ті ж залишкові блоки та генеративна схема, але з іншими налаштуваннями втрат та підбором параметрів.

Модель можна використовувати у реальних системах, де неможливо контролювати джерело даних. Джерелом можуть бути зображення з мобільних пристроїв, старих відеозаписів, матеріалів із мережі і т.п. Недоліки залишаються однакові, як в ESRGAN.

Модель RT4KSR (Real Time 4K Super Resolution) [16] є орієнтованою на підвищення роздільної здатності відео до 4K у режимі, наближеному до реального часу. Модель побудована так, щоби поєднати якість відновлення деталей та швидкість обробки із помірним навантаженням на обчислювальні ресурси.

Архітектура моделі використовує компактні згорткові блоки та ефективні операції збільшення роздільної здатності, які споживають найменшу кількість ресурсів. Зображення збільшується поступово з використанням піксельного перемножування або подібних операцій, тим самим виконуючи більшу частину обчислень у просторі нижчої роздільної здатності, в результаті чого швидкість обробки зростає. На рис. 1.4 наведено архітектуру моделі RT4KSR.

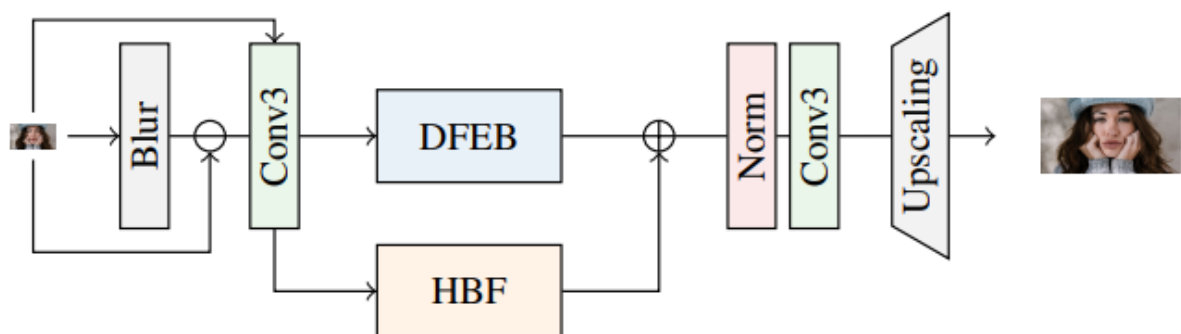


Рисунок 1.4 – Архітектура моделі RT4KSR

З розвитком глибокого навчання з'явилися трансформери, на основі яких було побудовано моделі SwinIR [17]. Модель використовує зв'язки між зображеннями з виділенням ознак. Зображення розділяється на невеликі

частини (вікна), в межах яких аналізуються відносини між пікселями, далі частини зміщуються для того, щоби модель бачила більшу область без різкого зростання обчислень.

В архітектурі моделі використовується багат шарова структура з послідовністю блоків уваги. У кожному блоці зображення представляється у вигляді набору ознак, де механізм уваги визначає, які елементи є найбільш важливими для відновлення структури, завдяки чому модель враховує локальні зв'язки в зображенні та зовнішні між іншими. Для підвищення роздільної здатності використовується спеціальні шари, які перетворюють зображення у вищу роздільну здатність з збереженням ознак, отриманих на попередніх шарах. Архітектуру моделі показано на рис. 1.5.

Модель дуже добре працює з складними зображеннями, де звичайні згорткові мережі не мають можливості відстежувати зв'язки між іншими зображеннями. Однак моделі, створені на основі трансформерів, повинні бути правильно налаштованими та добре натренованими на великих наборах даних [18].

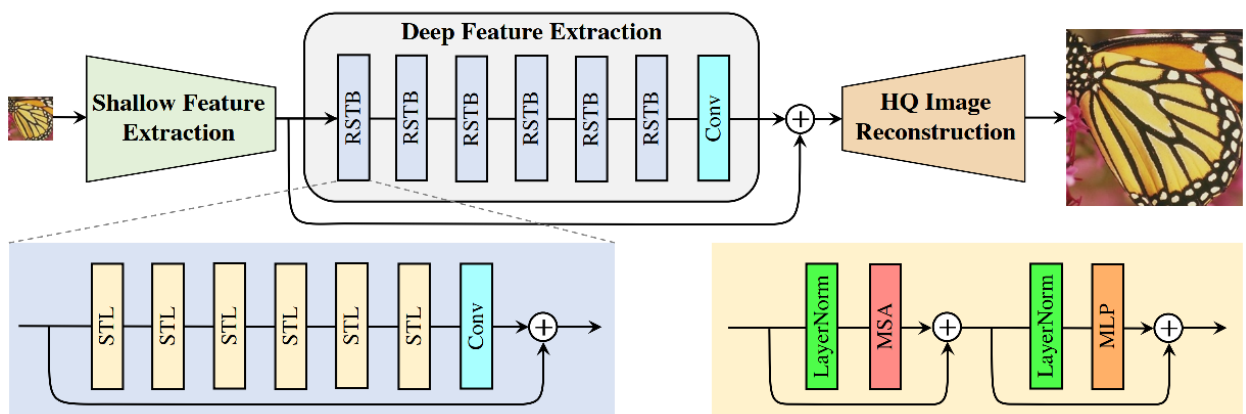


Рисунок 1.5 – Архітектура моделі SwinIR

1.3 Аналіз існуючих засобів підвищення роздільної здатності

Для підвищення роздільної здатності зображень і відео послідовностей використовується велика кількість програмних засобів. Для аналізу було

обрано найбільш оптимальні програмні засоби: ComfyUI, chainNNet та Upscayl.

ComfyUI [19] – програмний продукт, який дозволяє працювати з моделями глибокого навчання через вузловий інтерфейс. Процес обробки зображення або відео передається у вигляді графа, який складається з окремих вузлів. Кожен вузол має власну операцію, наприклад, завантаження моделі, обробки зображення, підвищення роздільної здатності, фільтрації, збереження результату і т.п. Інтерфейс ComfyUI показано на рис. 1.6.

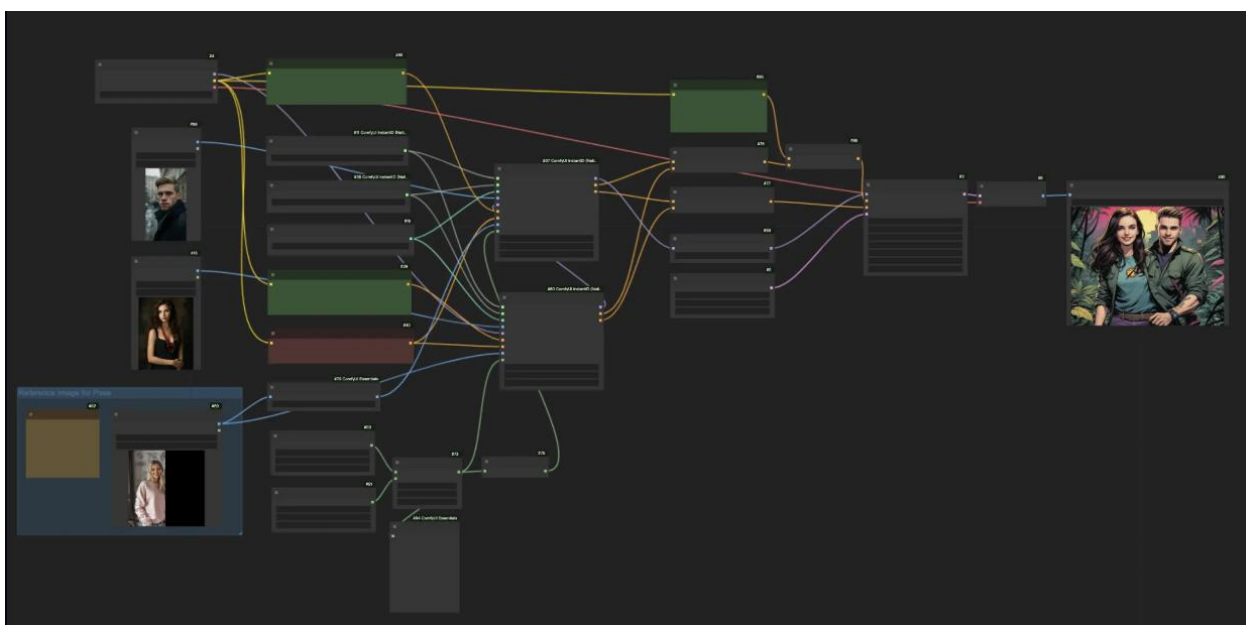


Рисунок 1.6 – Інтерфейс ComfyUI

ComfyUI підтримує використання різних типів моделей, наприклад дифузійних. Підвищити роздільну здатність зображення можна різними способами: просте масштабування на основі піксельних методів чи використання сторонніх моделей підвищення роздільної здатності, наприклад ESRGAN. Після додавання моделі, створюються окремі вузли або групи вузлів, які відповідають за певний етап обробки. Під час робочого процесу виконуються різні операції, які можна гнучко налаштовувати у вигляді ланцюжка обробки під конкретну задачу. Користувачу надається повний контроль зміни параметрів моделі.

ComfyUI має складний інтерфейс для початківця через складність налаштування, підходить досвідченим користувачам, які розуміють структуру налаштування моделей і їх використання. Потрібно самостійно підбирати вузли, підключати моделі, налаштовувати їхні параметри. Програмний продукт не має модуля для навчання моделей та керування наборами даних, оскільки моделі тренуються окремо від програми і тільки використовуються в ній.

chaiNNer [20] є вузловим графічним інтерфейсом, який використовується для підвищення роздільної здатності та обробки зображень через нейронні мережі. У програмному продукті зображення проходить через послідовність вузлів підготовки даних, обробки моделями та пост обробки. Вузли можуть включати зміну формату, нормалізацію, застосування підвищення роздільної здатності, корекція різкості або шуму. На рис. 1.7 наведено інтерфейс програмного продукту chaiNNer.

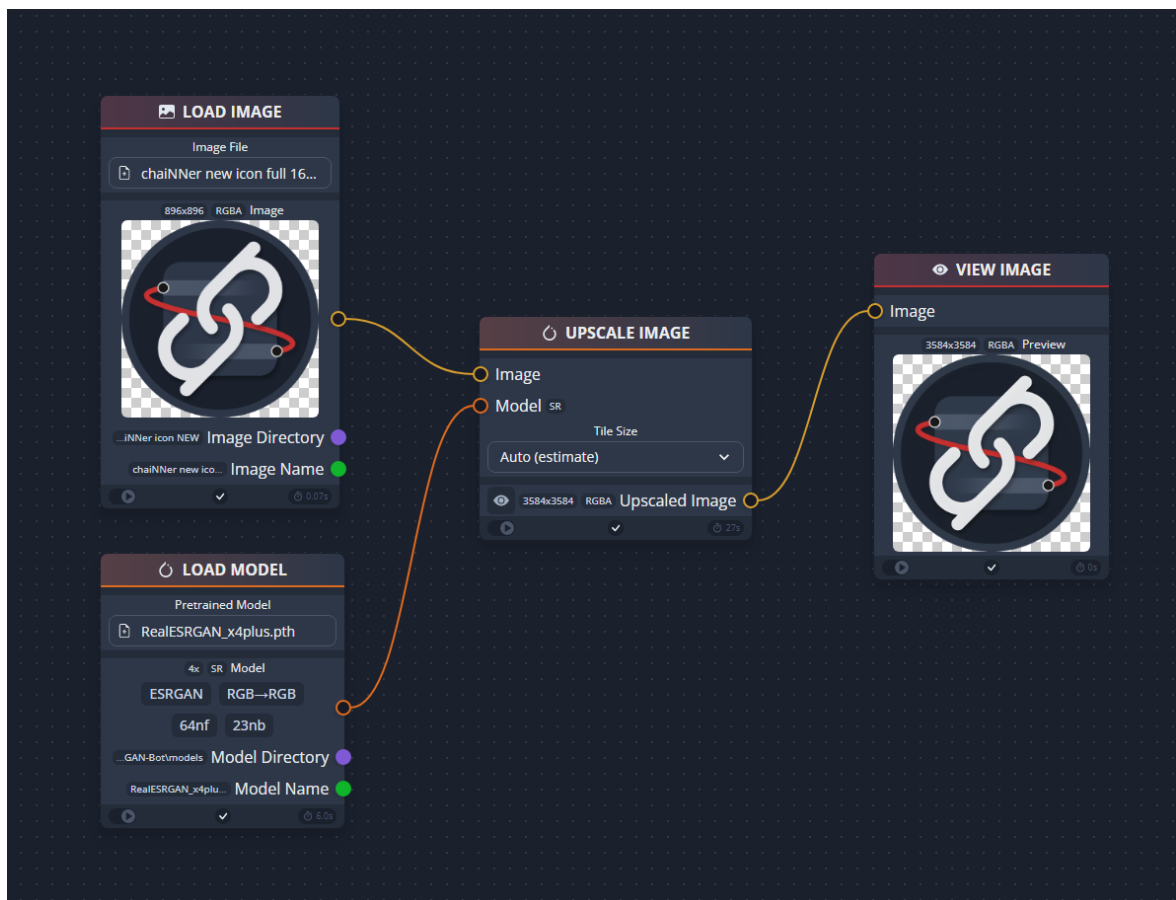


Рисунок 1.7 – Інтерфейс chaiNNer

Найбільшою перевагою chaiNNer є зручність створення конвейєра обробки зображень. Інтерфейс має зрозумілу послідовність дій між зв'язками вузлів, можливість перевикористання готових схем для повторного налаштування моделей під типові задачі.

Як і ComfyUI, chaiNNer не надає можливість тренувати власні моделі. Користувачу пропонується використання натренованих моделей. Інтерфейс може бути незрозумілим через велику кількість вузлів та параметрів, якщо відсутній досвід користування.

Upscayl [21] – програмний продукт, який використовується для підвищення роздільної здатності зображень. Upscayl використовує натреновані моделі, які побудовані на архітектурі Real-ESRGAN. Програмний продукт має простий та зручний інтерфейс. Користувач може обрати вхідне зображення, масштаб збільшення, обрати модель (за потреби), змінити декілька параметрів (формат зображення). Інтерфейс Upscayl показано на рис. 1.8.

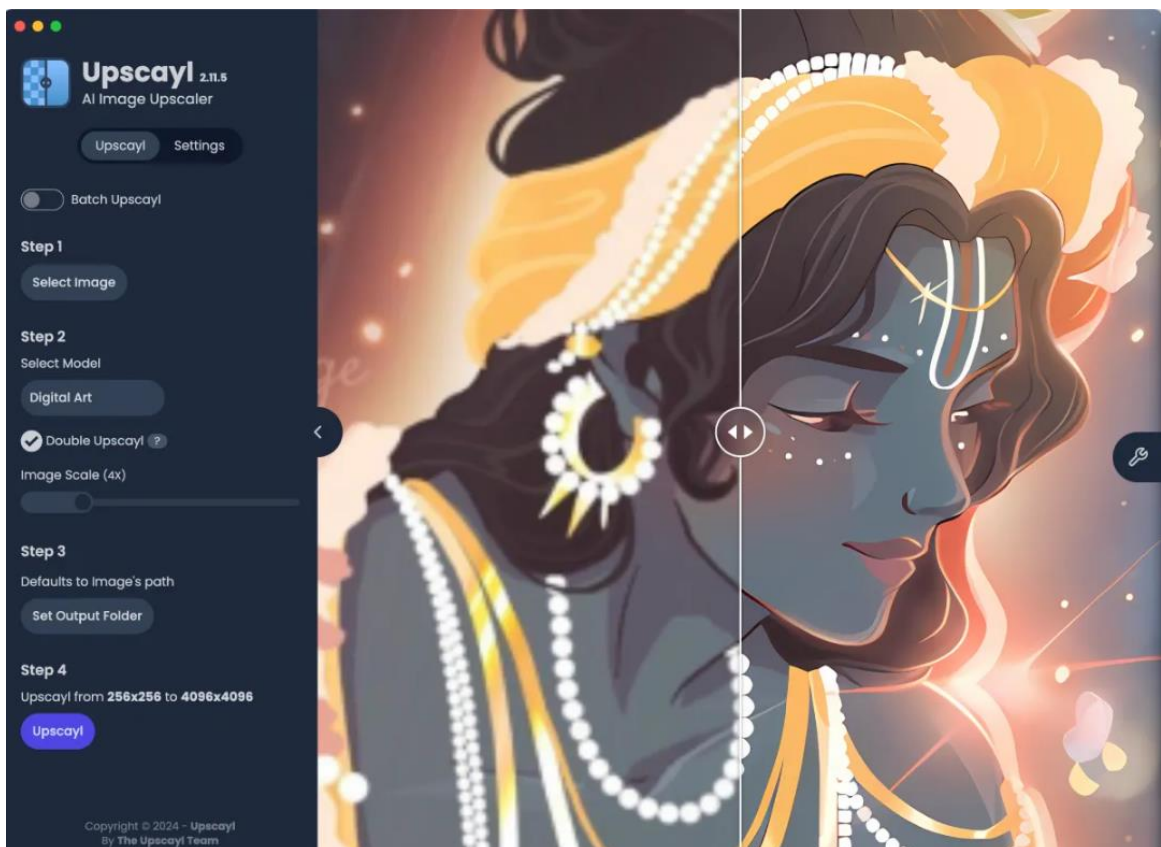


Рисунок 1.8 – Інтерфейс Upscayl

Для використання не потрібно мати якісь знання у підгалузі глибокого навчання чи обробки зображень, все, що потрібно зробити, – встановити програмний продукт, обрати зображення та запустити.

Як і інші програмні продукти, Upscayl не має можливостей для тренування та тестування моделей за різними метриками. Засіб орієнтований на користувача, якому потрібний швидкий та простий спосіб покращити зображення.

У табл. 1.1 подана порівняльна характеристика розглянутих засобів підвищення роздільної здатності за основними параметрами.

Таблиця 1.1

Порівняння існуючих засобів підвищення роздільної здатності

	ComfyUI	chaiNNer	Upscayl
Навчання моделей	Не підтримується	Не підтримується	Не підтримується
Тестування моделей	Створення власних схем	Так	Тільки порівняння
Зручність використання	Низька	Середня	Висока

1.4 Постановка задачі

Під час аналізу методів підвищення роздільної здатності було обрано декілька архітектур, на яких будуть побудовані моделі. В якості дослідження було обрано архітектури ESRGAN, Real-ESRGAN та RT4KSR. Архітектури використовують різні підходи збільшення зображення, тому їхнє порівняння надасть можливість оцінити їхні якості відновлення у різних умовах. Моделі повинні підтримувати гнучке налаштування параметрів та мета параметрів, обробку зображень. Додатково, буде створено власну модель на основі існуючих архітектур, яка повинна працювати з різними типами та наборами даних, з підтримкою роботи на різних пристроях.

Аналіз існуючих засобів підвищення роздільної здатності зображень і відео послідовностей показав, що деякі аспекти потребують покращення у

майбутньому рішенні. Найбільшим недоліком розглянутих засобів є відсутність єдиного середовища, де відбуватиметься навчання, тестування та практичне використання різних моделей для підвищення роздільної здатності. Їхня основна мета – здійснювати просте підвищення роздільної здатності або будувати складні конвейєри, які для простого користувача є незрозумілими. Не менш вагомим недоліком є складність частин інтерфейсу, які виглядають не зрозумілими, що у свою чергу сповільнює опанування роботи з програмним продуктом. Додатково, деякі з них працюють виключно з зображеннями без підтримки обробки відео.

Головними задачами дослідження є створення та порівняння роботи моделей глибокого навчання для підвищення роздільної здатності зображень і відео послідовностей, розробка програмного продукту як частина для їхнього використання. Кожна модель повинна зберігати баланс між якістю відновлення та швидкістю обробки на доступному обладнанні. Під час порівняння роботи моделей потрібно врахувати різні метрики оцінювання якості відновлених зображень, за допомогою яких можна покращити роботу моделей через проведення різних експериментів.

Програмний додаток повинен мати зручний інтерфейс, підтримувати створення, тренування та тестування різних моделей без складних налаштувань, повинен обробляти зображення та відео з можливістю порівняння результатів під час тестування та без нього. Тестування повинне підтримувати: різні параметри запуску, наприклад вибір декількох моделей, наборів даних та метрик якості; відображення результатів обраних метрик з візуальним порівнянням оброблених моделями зображень.

Задачі будуть розділені на декілька етапів. Спочатку буде створено різні набори даних для тренування моделей з розподілом на тренувальну, валідаційну та тестову вибірки, після чого проектування архітектур моделей ESRGAN, Real-ESRGAN, RT4KSR та власної на основі існуючих. Наступним етапом відбуватиметься тренування та тестування роботи моделей з подальшими змінами будови моделей, параметрів та мета параметрів з метою

знаходження найкращого рішення на основі результатів метрик якості. Останнім етапом стане створення програмного продукту, який повинен підтримувати обрані моделі, надаватиме зручний інтерфейс для використання, обробки зображень та відео з можливістю перегляду, порівняння та оцінки отриманих результатів.

Висновки до першого розділу

В даному розділі було описано процес підвищення роздільної здатності зображень і відео послідовностей. Розглянуто методи та засоби підвищення роздільної здатності, проаналізовано їхні переваги та недоліки, на основі чого було сформовано задачі розробки та дослідження.

Було визначено етапи виконання відповідних задач. На основі існуючих засобів проаналізовано елементи, які потрібно реалізувати у власному програмному продукті. Основою стане набір досліджуваних моделей глибокого навчання, які підвищуватимуть роздільну здатність зображень і відео послідовностей.

РОЗДІЛ 2 РОЗРОБКА МОДЕЛЕЙ ДЛЯ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ

2.1 Вибір інструментів розробки моделей

Для побудови моделей підвищення роздільної здатності зображень і відео послідовностей доцільно використати мову програмування Python, оскільки вона поширена для застосування у підгалузі глибокого навчання. Python є інтерпретованою мовою з простою та зрозумілою структурою коду. Мова підтримує модульну організацію програм, об'єктно-орієнтований підхід та роботу з функціями вищого порядку, що дозволяє будувати гнучку структуру проекту: від окремих модулів обробки даних до повного циклу тренування, валідації та тестування моделей глибокого навчання. Найбільшою перевагою Python є наявність розвинутого набору бібліотек для наукових обчислень, машинного та глибокого навчання, роботи із зображеннями та відео, які активно підтримуються спільнотою розробників. Присутні вбудовані засоби інтеграції з кодом на C/C++ та CUDA для виконання складних обчислювальних операцій на графічних процесорах без необхідності прямої реалізації низькорівневого коду.

Для реалізації процесів тренування, тестування та обробки зображень розробленими моделями використовуватимуться такі бібліотеки, що описані нижче.

PyTorch є популярною бібліотекою, яка використовується для побудови різних архітектур моделей, організації процесу їхнього навчання та виконання на різних обчислювальних пристроях.

Для оцінки якості результатів обробки зображень моделями – torchmetrics, яка включає в себе метрики оцінки якості зображень.

Бібліотека pandas є підходящим варіантом для обробки, підготовки наборів даних, а саме зображень, для збереження результатів тестування у вигляді структурованих даних.

Для збереження результатів навчання моделей під час тренування та валідації доцільно використати бібліотеку TensorBoard, яка дозволяє зберігати скалярні дані та зображення, що можуть бути корисними для аналізу роботи моделей.

Для роботи з відео послідовностями найбільше підходить модуль ffmpeg. За допомогою цього модуля можна легко зчитувати кадри з відео та обробляти їх в моделях. Додатково, можна використати базові операції бібліотеки OpenCV та бібліотеку Pillow для роботи з різними форматами зображень.

Опрацьовувати різні масиви та тензори можна через бібліотеку NumPy з використанням різних математичних операцій.

2.2 Побудова моделей підвищення роздільної здатності зображень і відео послідовностей

Будова моделі ESRGAN базується на генеративному підході, де зображення по своїй суті генерується у вищій роздільній здатності на основі зображення низької роздільної здатності. Як вже зазначалося, модель складається з двох частин: генератора та дискримінатора. Генератор створює зображення високої роздільної здатності, дискримінатор оцінює його якість роботи через порівняння створеного зображення з оригінальним високо роздільним зображенням [22].

Генератор складається з послідовності згорткових шарів та спеціальних блоків. Основна частина генератора складається з RRDB (Residual-in-Residual Dense Block) блоків, які містять в собі три залишкові блоки (Residual Block). Кожен залишковий блок містить декілька згорткових шарів, з'єднаних поміж собою пропускними зв'язками, окрім останнього згорткового шару [23]. Вигляд RRDB блока наведено на рис. 2.1.

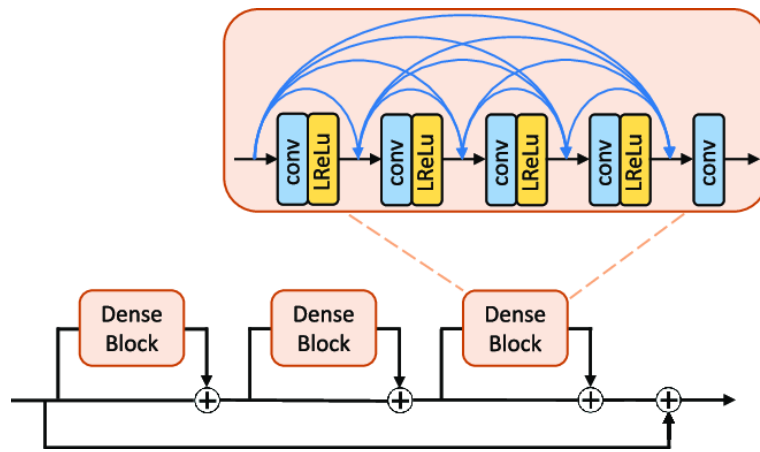


Рисунок 2.1 – Структура блока RRDB

Як видно з рисунку, в блоці RRDB вихід кожного шару додається до його вхідного значення для того, щоби зменшити втрату градієнтів та підвищити стабільність при високій глибині під час навчання.

Після кожного згорткового шару залишкового блока (окрім останнього) використовується функція активації LeakyReLU (рис. 2.2) для уникнення зависання нейронів та кращого проходження інформації поміж шарами.

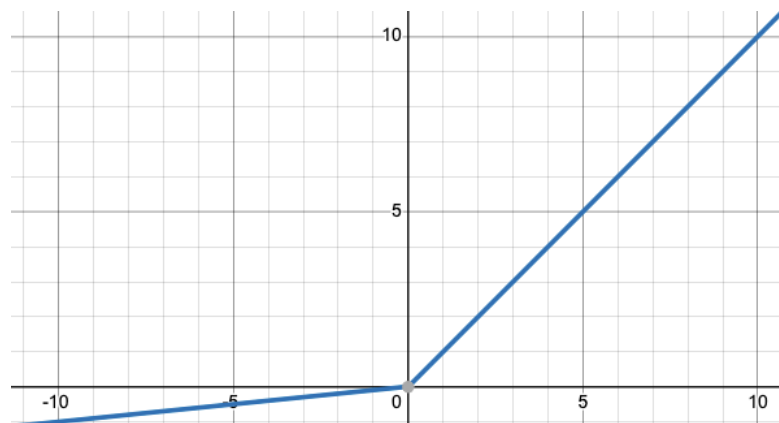


Рисунок 2.2 – Функція активації LeakyReLU

Після проходження зображення через RRDB блоки ознаки зображення проходять через додатковий згортковий шар. Перед підвищенням роздільної здатності зображення ознаки конкатенуються (перед та після проходження RRDB блоків). Для підвищення роздільної здатності використовуються операції PixelShuffle або подібні, які перетворюють канали на додаткові

пікселі між іншими пікселями. Після підвищення ознаки зображення проходять через фінальний згортковий шар, який формує кінцеве зображення. Повний вигляд побудованої моделі наведено на рис 2.3.

Для того, щоби генератор ставав потужнішим, використовують дискримінатор. В якості дискримінатора зазвичай застосовують релятивістичний дискримінатор (RaD) або подібні до нього [24]. Така мережа містить певну послідовність згорткових шарів, які виконують роль зменшення розмірності зображення зі збільшенням кількості його каналів, завдяки чому дискримінатор навчається аналізувати текстури та дрібні деталі і вказувати на помилки, які допускаються генератором. Останній шар мережі забезпечує отримання ймовірності того, що зображення є реальним. Якщо генератор ввів в оману дискримінатора, то наступним кроком генератор оновлює власні параметри, щоби наділі формувати такі зображення, які дискримінатор не зможе відрізнити від справжніх.

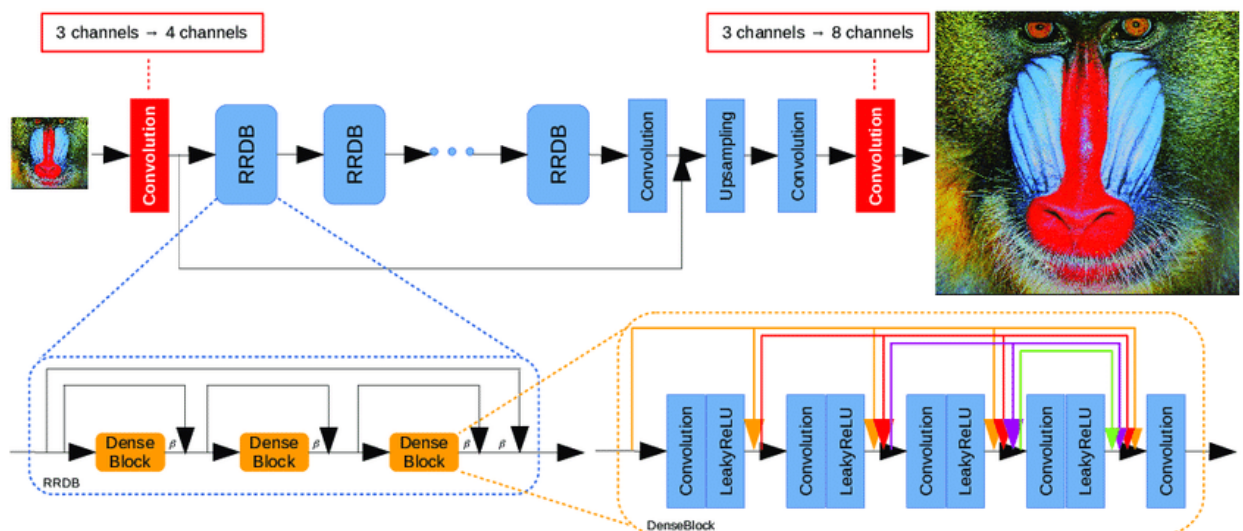


Рисунок 2.3 – Архітектура моделі ESRGAN

Під час навчання ESRGAN використовує різні функції втрат, оскільки тренуються дві моделі: генератор та дискримінатор. Основною функцією втрати є перцептивна. Вона порівнює ознаки в проміжних шарах моделі VGG, щоби оцінити структуру зображень, а не тільки його колір. Змагальну втрату

використовує генератор для того, щоби створювати текстури, які виглядатимуть природними. Піксельну втрату (L1 або L2) використовує дискримінатор для оцінки різниці між вихідним зображенням генератора та зображенням високої роздільної здатності [25].

R-ESRGAN має однакову будову, що й ESRGAN, принцип роботи залишається незмінним. У навчанні використовуються перцептивна, змагальна та піксельна втрати.

Модель R-ESRGAN, на відміну від ESRGAN, адаптована до реальних умов та спотворень. У класичній ESRGAN під час тренування використовуються пари зображень, де низьке роздільне зображення створюється через просте зменшення масштабу, наприклад, через бікубічну інтерполяцію, відповідно створене зображення не матиме ніяких спотворень. R-ESRGAN додає ці спотворення через двоступеневу модель деградації.

На першій ступені деградації до створеного низького роздільного зображення, додаються згладжування, шуми та інші спотворення, на другій ступені деградації – зниження чіткості, зміна освітлення і т.п, у результаті чого зображення виглядає близьким до зображень, які були створені в реальних умовах.

Додавання деградацій до зображень надає можливість моделі працювати з різними зображеннями з різних джерел і не залежати від того, як це зображення було створене.

В моделі R-ESRGAN дискримінатор був замінений на мережу U-NET з спектральною нормалізацією [26]. До нього було додано додаткові операції нормалізації для того, щоби зробити навчання стабільнішим при роботі з шумом та зменшити перевагу дискримінатора над генератором.

Побудована модель RT4KSR складається з послідовності згорткових шарів та блоків, які працюють з ознаками низької роздільної здатності та їхніми високочастотними ознаками. Зображення проходить через блок виділення високочастотних ознак, згортковий шар, дві гілки DFEB та HBF,

блок нормалізації із кінцевою згорткою та блоком підвищення роздільної здатності зображення. На рис. 2.4 зображено будову моделі RT4KSR.

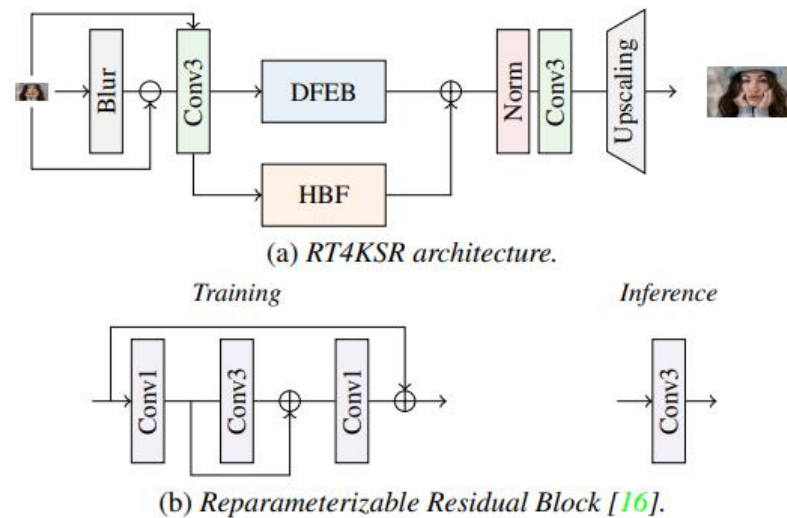


Рисунок 2.4 – Архітектура моделі RT4KSR

На вхід моделі надходить низько роздільне зображення. В Blur блоці формується згладжена версія зображення. Блок використовує усереднення з подальшим збільшенням до початкового розміру або, як варіант, згортка з гаусовим ядром, який створює розмиття. Обидва варіанти дають зображення з приглушеними деталями. Після цього обчислюється високочастотна карта як різниця між вихідним зображенням та згладженою версією:

$$H(x) = x - G(x), \quad (2.1)$$

де x – вихідне зображення;

$G(x)$ – результат згладжування.

Паралельно вихідне зображення проходить через згортковий шар з ядром розміром 3×3 . Шар виконує перетворення зображення у початкову карту ознак фіксованої кількості каналів. Після чого ознаки розподіляються на дві гілки: гілку глибоких ознак DFEB (Deep Feature Extraction Branch) та гілку високих частот HBF (High Frequency Branch) [27]. Високочастотна карта, яку було отримано, надходить в HBF. Обидві гілки працюють з ознаками, які

пройшли через спільну згортку, але мають різні завдання. DFEB опрацьовує стиснуті ознаки, а HBF уточнює саме високочастотні ознаки.

DFEB використовує стиснення просторових розмірів ознак через операцію PixelUnshuffle. Кожна карта ознак має певний розмір $H \times W \times C$. Після операції PixelUnshuffle з коефіцієнтом s , вона перетворюється у тензор $H/s \times W/s \times (C \times s^2)$, тим самим просторовий розмір карти зменшується, а кількість каналів збільшується. Потім отримані ознаки проходять через згортку розміром 3×3 , яка стискає їх до потрібної кількості каналів з коригуванням розподілу інформації після перестановлення пікселів. Через це зменшується кількість операцій, які потрібно виконати над зображенням, але водночас зберігається інформація, важлива для відновлення.

Всередині DFEB та HBF для обробки ознак використовуються модифіковані блоки NAFNet [28]. Кожний блок складається зі згортки, що виконується поканально, за якою слідує точкова згортка розміром 1×1 для зміни кількості каналів. Після першої точкової згортки розмір каналу збільшується з C до $2C$. До результату застосовується функція активації GeLU для згладженої нелінійності:

$$GeLU(x) \approx \frac{1}{2}x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.0447x^3))), \quad (2.2)$$

де x – вихідні дані.

Після активації використовується спрощена канална увага, яка масштабує канали за коефіцієнтами, обчисленими з глобального усереднення. Потім застосовується нормалізація по шарах для стабілізації розподілу ознак. Наприкінці блоку ще одна точкова згортка переводить кількість каналів назад до C . Вихід блока додається до його входу через залишковий зв'язок.

У кінці виходи DFEB та HBF комбінуються. Спочатку до ознак гілки DFEB додаються покращені ознаки з HBF. Сума виконується по елементам для того, щоби доповнити глибокі ознаки деталями.

Отримана карта ознак надходить до блока нормалізації (Norm), де знову застосовується шарова нормалізація для уніфікації масштабу, після чого згортка розміром 3×3 об'єднує всі канали для передачі їх на підвищення роздільної здатності. Блок підвищення роздільної здатності (Upscaling) виконує збільшення просторової роздільної здатності. Блок використовує згортку для формування необхідної кількості каналів та операцію PixelShuffle для перетворення каналів у додаткові пікселі. Результат на виході блока являє собою зображення з підвищеною роздільною здатністю.

Окремою складовою моделі є репараметризований залишковий блок [29], який використовується всередині блока DFEB замість звичайних згорток. Під час навчання блок містить декілька паралельних згорткових гілок: точкові згортки, згортку з ядром розміром 3×3 та прямий залишковий зв'язок.

Після завершення навчання ядра та зміщення всіх гілок піддаються згортці розміром 3×3 під час обробки зображення для зменшення кількості операцій без зміни вихідної функції.

Для дослідження, як вже зазначалося, було спроектовано просту архітектуру моделі підвищення роздільної здатності. Архітектура включає в себе згорткові шари, залишкові блоки та блок підвищення роздільної здатності через операцію PixelShuffle. Метою дослідження було розроблення такої архітектури, яка зменшує складні обчислення та зберігає можливість відтворювати складні деталі зображення. На рис. 2.5 наведено архітектуру створеної моделі.

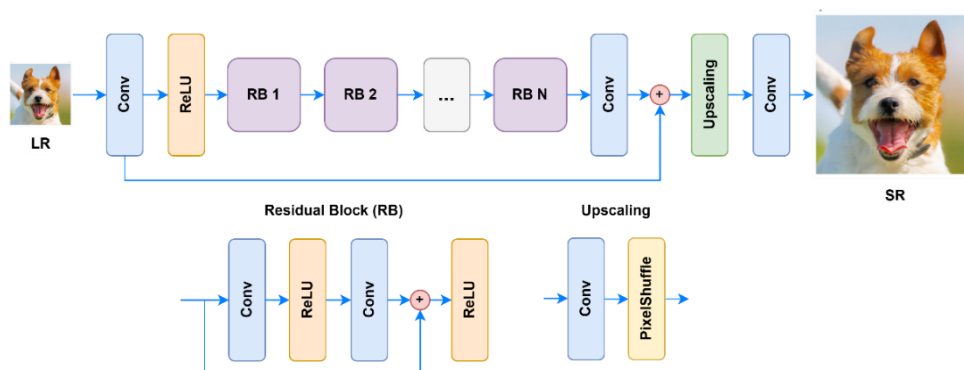


Рисунок 2.5 – Архітектура створеної моделі

Більшу частину моделі становлять спрощені залишкові блоки. Блок складається з двох послідовних згорткових шарів з ядром розміром 3x3. Після першої згортки ознак застосовується функція активації ReLU для обнулення від'ємних значень, щоби залишити тільки додатні. Результат першої згортки проходить через другу згортку, після чого до виходу додаються вхідні ознаки та пропускаються через функцію активації ReLU. Спрощений блок описується так:

$$y = \text{ReLU}(\text{Conv}_2(\text{Relu}(\text{Conv}_1(x)) + x), \quad (2.3)$$

де x – вихідні дані.

Для побудови моделі достатньо всього трьох параметрів: масштаб збільшення (scale), кількість каналів (nc) та кількість залишкових блоків (nb). Через масштаб зображення буде збільшено по кожній осі в n разів. Кількість каналів встановлює розмірність простору ознак, з якими працює модель. Кількість блоків є глибиною моделі, через які створюються складні залежності між ознаками.

На вході моделі використовується згортковий шар, який виконує перетворення три каналного зображення у певну кількість каналів ознак. Шар виділяє початкові структури, контури та базові текстури. До виходу шару додається функція активації ReLU для нелінійності та подальшої обробки ознак. Після чого ознаки проходять через певну послідовність залишкових блоків для уточнення ознак зображення. Ознаки, які пройшли через останній залишковий блок, надходять у згортковий шар з розміром ядра 3x3 для об'єднання отриманих ознак та їхньої підготовки до довгострокового пропускового зв'язку. Далі вихід згорткового шару додається до виходу першого згорткового шару для утворення довгострокового зв'язку, що дозволяє моделі вивчати різницю між покращеними та початковими ознаками без повного відновлення зображення з нуля.

Після поєднання ознак виконується збільшення масштабу. Згортка з розміром ядра 3×3 перетворює початкову кількість каналів на кількість каналів, помножених на масштаб збільшення в квадраті. Далі виконується переставлення каналів у просторові координати зі збільшенням ширини та висоти зображення відповідно до масштабу збільшення через операцію PixelShuffle.

До виходу блока збільшення застосовується останній згортковий шар з розміром ядра 3×3 , який перетворює карту ознак у три каналне зображення.

Ініціалізація ваг моделі представлена через ініціалізацію Каймінга [30] для всіх згорткових шарів. В ній задаються початкові ваги з врахування дисперсії сигналу, яка може зникати та зростати при проходженні через шари. При такій ініціалізації пришвидшується збіжність під час навчання та зменшується кількість запусків, які можуть бути невдалими при неправильному виборі мета параметрів моделі.

Шари пакетної нормалізації були наявні в моделі, але під час дослідження вони були прибрані, оскільки викликали артефакти (биті пікселі) та зміну яскравості та контрасту, якщо порівнювати з оригінальним зображенням. Дослідження показало, що прибирання шарів пакетної нормалізації зменшує обчислювальні ресурси, що є корисним при використанні моделі на пристроях з обмеженими обчислювальними ресурсами.

Під час дослідження використовувався механізм каналної уваги для підвищення важливих каналів ознак та послаблення другорядних ознак за рахунок множення карт ознак на коефіцієнти ваг, обчислених через їхні середні значення. Результати з каналною увагою помітно не відрізнялися від тих, які були отримані без неї, тому її було прибрано з метою зменшення часу обробки зображення моделлю [31].

2.3 Створення наборів даних

Для тренування та порівняння роботи моделей було створено 3 набори даних, які складаються з різних типів зображень: звичайні зображення з обличчями та без, анімаційні зображення та рентгенівські знімки. Розмір кожного датасету складається приблизно з 20000 прикладів високої роздільної здатності. Рішення щодо вибору такого розміру було прийнято через обмеження, пов'язані з доступними обчислювальними ресурсами та часом тренування моделей.

Набір даних зі звичайних зображень створений на основі наборів даних DIV2K, Flickr2K, OST+ та IMDB-WIKI [32]. Набір DIV2K містить 1000 зображень з роздільною здатністю до 2K, з яких 800 призначені для тренування та по 100 для валідації та тестування; Flickr2K складається з 2650 зображень високої роздільної здатності з широким різновидом сцен; набір даних OST, який входить до OST+, містить зображення вуличних та природних типів з різними деталями. Для доповнення набору даних було використано зображення обличч різних знаменитостей у різних вікових групах з набору даних IMDB-WIKI, який містить близько 500000 зображень. З усіх наборів даних обиралися унікальні зображення з наявністю дрібних деталей та достатньою роздільною здатністю для тренування, валідації та тестування. На рис. 2.6 зображено вигляд створеного набору даних.

Для тренування моделей на анімаційному наборі даних було використано набір даних Anita Dataset [33] (рис. 2.7). Він складається з 16000 зображень 2D-анімації у високій роздільній здатності з різними типами малювання – ескізи та кольорові варіанти. Під час тренування переважно використовувалися кольорові варіанти, хоча ескізи також є корисними для анімаційних зображень, оскільки можуть використовуватися після тренування моделей.

Для рентгенівських зображень грудної клітини було використано набір даних ChestX-ray8 [34]. Набір даних налічує приблизно 109000 знімків 33000 пацієнтів із текстовими мітками поширених захворювань. З набору даних було

взято частину знімків з достатньою якістю та відсутністю технічних дефектів. Додатково, зображення обрізалися до однакового розміру для зручного використання під час тренування моделей. На рис. 2.8 зображено набір даних рентгенівських знімків ChestX-ray8.

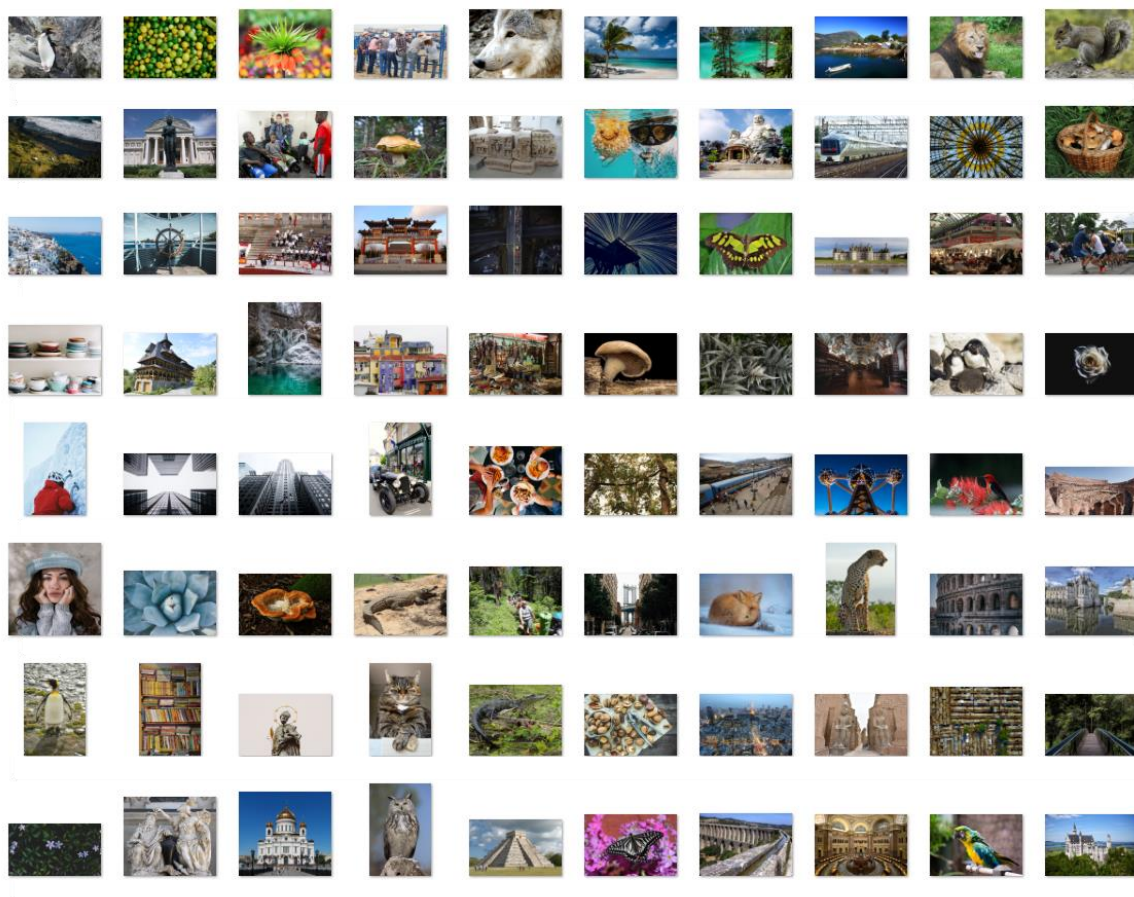


Рисунок 2.6 – Створений набір даних



Рисунок 2.7 – Набір даних Anita Dataset

На обраних наборах даних під час тренування використовувалися лише високо роздільні зображення. Низько роздільні зображення створювалися штучно через зменшення масштабу високо роздільних зображень за допомогою бікубічної інтерполяції.

Для завантаження зображень та передачі їх для використання в моделях було створено класи наборів даних для звичайного та деградованого тренування моделей. Код створених класів наведено в Додатку А.

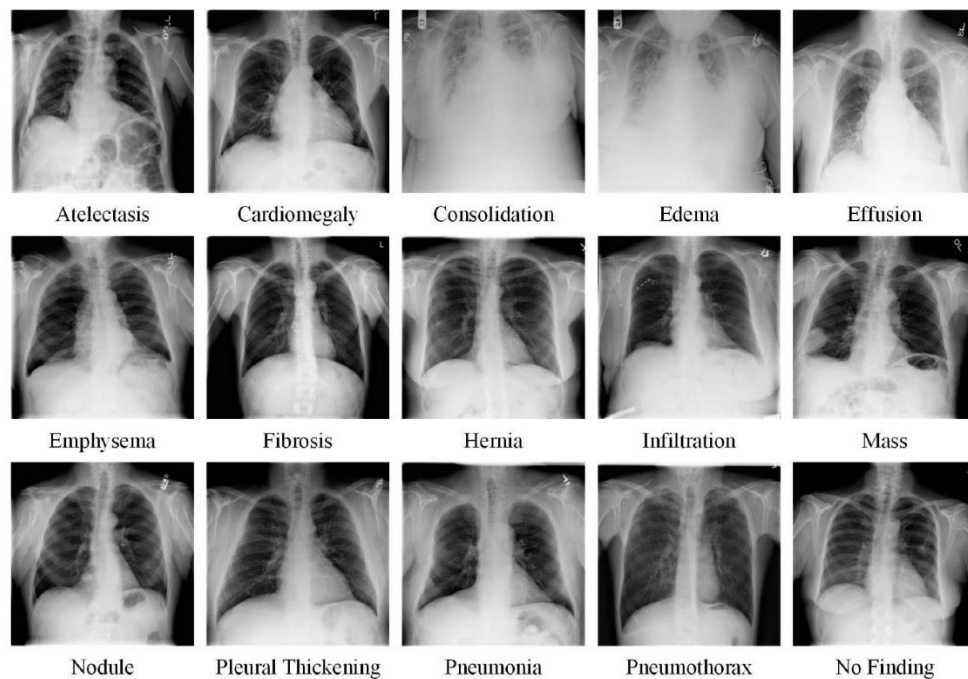


Рисунок 2.8 – Набір даних ChestX-ray8

2.4 Процес навчання створених моделей

Навчання кожної моделі проводиться на різних класах наборів даних. Для тренування використовуються вирізані з високо роздільних зображень фрагменти (патчі) фіксованого розміру для більшої різноманітності охоплення різних частин зображення. Для всіх моделей використовується навчання зі змішаною точністю (AMP) для зменшення використання відео пам'яті графічного процесора. Оптимізація параметрів (ваг) реалізується через різні

алгоритми градієнтного спуску з міні партіями. Розбиття набору даних наступне: 90% для тренування, 5% для валідації та 5% для тестування.

В моделі ESRGAN навчалися дві моделі: генератор та дискримінатор. Масштаб збільшення зображення становить 2. Для тренування використовуються патчі розміром 128x128 пікселів в низько роздільному варіанті. Розмір міні партії навчання становить 8 для того, щоби не навантажувати обчислювальний пристрій. Навчання виконується протягом 50 епох. Обидві моделі використовують оптимізатор типу Adam із початковою швидкістю навчання 0,0001. Для генератора використовується функція втрати на основі середньоквадратичної помилки та перцепційної складової, яка обчислюється на ознаках попередньо навченої моделі VGG, дискримінатор – логістична функція втрат.

Модель Real-ESRGAN тренується на класі набору даних, який застосовує двоступеневу деградацію для низько роздільних зображень. Масштаб збільшення зображення становить 2, розмір патчів – 256x256 пікселів. Розмір міні партії становить 4, оскільки використовуються складніші блоки, які потребують більш складних обчислень. Тривалість навчання моделі становить 100 епох. В ролі оптимізатора використовується AdamW з початковою швидкістю навчання 0,0002. Швидкість навчання зменшується поступово відповідно до зміни епох. Для піксельної втрати використовується функція Шарбаньє, перцепційна втрата обчислюється на основі ознак моделі VGG, а змагальна втрата через функцію втрати hinge-GAN. Щоби запобігти перенавчанню моделі, використовується рання зупинка з контролем приросту метрики якості.

Тренування RT4KSR має масштаб збільшення кратне 2. Під час тренування використовуються патчі розміром 256x256 пікселів. Розмір партії становить 32, оскільки модель не використовує складні обчислювальні операції. Для оптимізації параметрів (ваг) моделі використовується AdamW з швидкістю навчання 0,0003. Функція втрат у вигляді середньої абсолютної помилки (L1) для обчислення різниці між відновленим та високо роздільним

зображенням. Швидкість навчання змінюється за багатоступеневим планувальником, що й у Real-ESRGAN.

Створена модель навчається на масштабі збільшення зображення, який дорівнює 2. Модель використовує 8 залишкових блоків та простір ознак розміром 64 канали. Для навчання використовуються патчі розміром 256x256 пікселів, розмір міні партій становить 8. Повна тривалість навчання моделі дорівнює 200 епохам. В якості оптимізатора застосовується AdamW з початковою швидкістю навчання 0,0001, функція втрат – L1. Швидкість навчання змінюється через багатоступеневий планувальник з поетапним зменшенням по епохам.

Висновки до другого розділу

У другому розділі було обрано інструменти розробки моделей на мові програмування Python. Розглянуто будову, основні компоненти та процес роботи моделей ESRGAN, R-ESRGAN, RT4KSR. На основі досліджених моделей було створено власну модель зі спрощеною архітектурою.

Було створено та обрано набори даних для тренування, валідації та тестування моделей на звичайних та анімаційних зображеннях, а також на рентгенівських знімках. Описано процес навчання створених моделей з визначеними параметрами, які найкраще підходять для таких архітектур.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ТА ТЕСТУВАННЯ СТВОРЕНИХ МОДЕЛЕЙ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ І ВІДЕО ПОСЛІДОВНОСТЕЙ

3.1 Проектування програмного продукту

Для проектування програмного продукту потрібно визначити, як саме він буде використовуватися користувачем, і який функціонал повинен бути наявним. Для цього було створено діаграму варіантів використання, яка зображена на рис. 3.1.



Рисунок 3.1 – Діаграма варіантів використання

Користувач має такі функціональні можливості, які наведені нижче.

Створювати, редагувати та видаляти моделі з можливістю перегляду та їхнього пошуку.

Імпортувати та експортувати моделі, які є натренованими, для використання на інших пристроях.

Створювати, редагувати, очищати та видаляти тренування з можливістю перегляду та їхнього пошуку.

Переглядати конкретне тренування з можливістю перегляду параметрів навчання та наборів даних, графіків тренування та валідації, а також запуску тренування.

Тестувати натреновані моделі на декількох наборах даних та обраних метриках якості.

Обробляти локальні зображення та відео послідовності з можливістю порівняння обробленого зображення з його низько роздільним оригіналом, перегляду обробленого відео.

Створювати, редагувати, фільтрувати та видаляти набори даних з можливістю перегляду та їхнього пошуку.

Функціональні вимоги передбачають: можливість збереження інформації, а саме система повинна зберігати інформацію і надавати можливість користувачу керувати нею; програма повинна функціонувати без збоїв, навіть якщо вони виникнуть.

Нефункціональні вимоги є такими: час, який необхідний для навчання звичайних користувачів не повинен перевищувати 30 хвилин, для навчання просунутих користувачів – 10-15 хвилин; час відгуку програмного продукту для типових задач – не більше 10 секунд, для складних завдань – не більше 25 секунд; інтерфейс має бути зрозумілим з точки зору вигляду та зручності користування з відповідними підказками; максимальна норма помилок та дефектів в роботі системи – одна помилка на десять тисяч рядків коду; система повинна працювати на різних операційних системах без залежності від конкретного обчислювального пристрою; при оновленні системи весь попередній функціонал повинен працювати коректно, навіть якщо збільшується кількість нових архітектур та моделей.

При аналізі функціональних можливостей програмного продукту було виділено сутності, які дозволили його створити. На рис. 3.2 наведено діаграму класів у вигляді сервісів, які використовуються для взаємодії між інтерфейсом та програмною реалізацією програмного продукту.

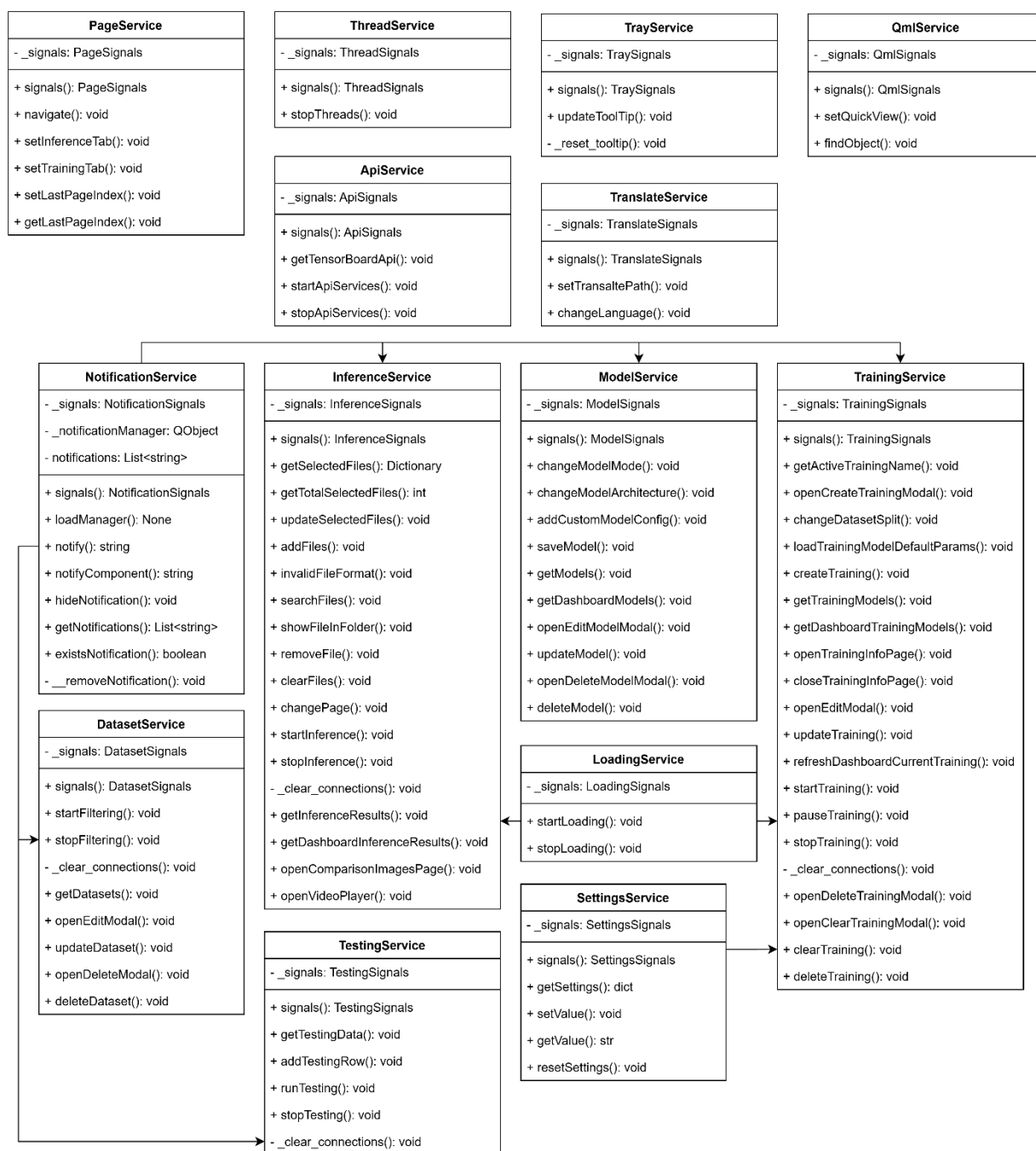


Рисунок 3.2 – Діаграма класів

Для відображення взаємодії між компонентами системи було створено діаграму розгортання (рис. 3.3). На діаграмі зображено взаємодію користувача з програмним продуктом. Середовище декларативної мови програмування QML бібліотеки QtQuick для побудови інтерфейсів взаємодіє через міст з бібліотекою PySide6 в середовищі виконання мови програмування Python, яка використовується для передачі та отримання даних з обох сторін [35]. Для

створення, тренування, обробки та тестування моделей використовується бібліотека PyTorch, для збереження результатів обробки конкретною моделлю – база даних SQLite 3.

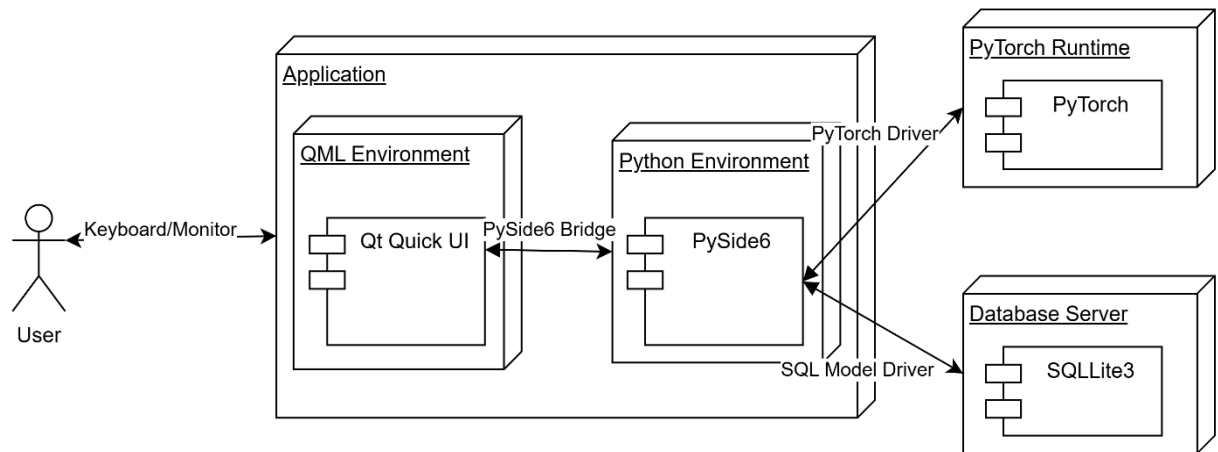


Рисунок 3.3 – Діаграма розгортання програмного продукту

3.2 Опис інтерфейсу програмного продукту

При відкритті програмного продукту користувач бачить стартове вікно, на якому відображається інформація про завантаження модулів, які використовуються ним. На рис. 3.4 зображено прогрес завантаження модулів.

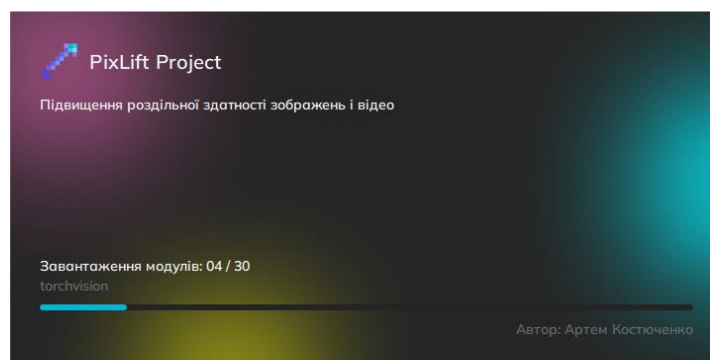


Рисунок 3.4 – Стартове вікно програми

Після завантаження модулів та запуску TensorBoard сервера буде відображено головне вікно програми зі сторінкою головної панелі (рис. 3.5).

На головній панелі відображається інформація про кількість створених моделей, активних тренувань та сесій обробки. Користувач може побачити створені ним останні три моделі та тренування. Для зручності користування було додано можливість запускати останнє тренування, яке було запущене в попередній сесії програми. Додатково, відображено останні результати обробки шістьох зображень або відео, оброблених моделями.

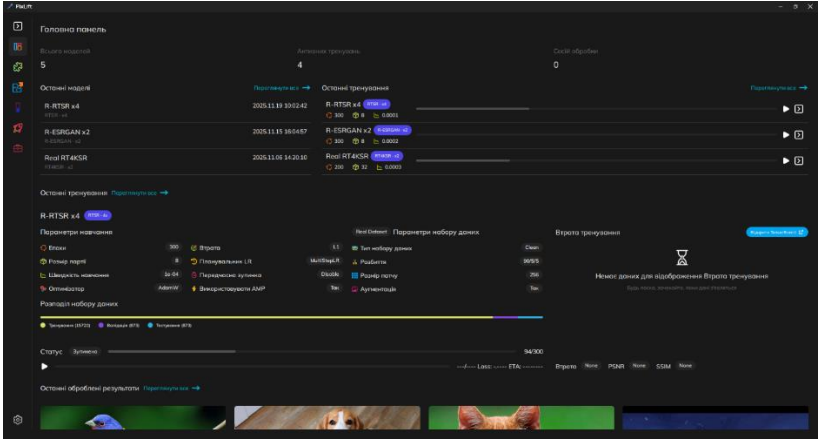


Рисунок 3.5 – Головна панель

На сторінці моделей (рис. 3.6) користувач може переглянути інформацію про створені моделі: назва, архітектура, масштаб, дата створення. Є зручний пошук для знаходження різних моделей з пагінацією. На цій сторінці можна створити власну модель, яка буде використовуватися для навчання та майбутньої обробки зображень та відео послідовностей.

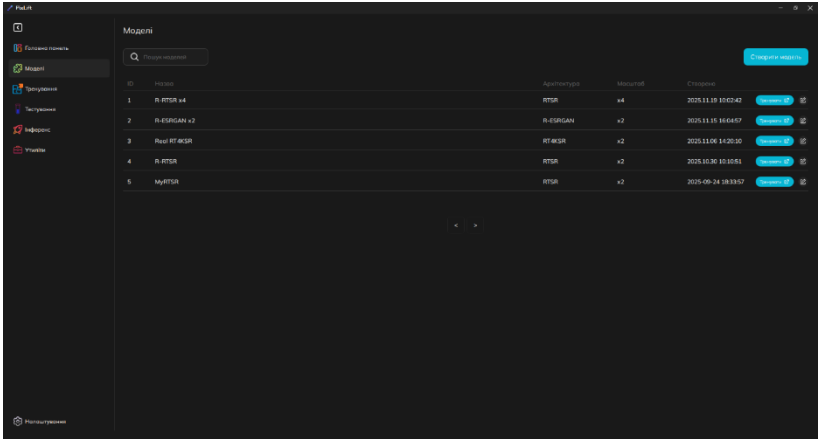


Рисунок 3.6 – Сторінка моделей

При створенні моделі відкривається модальне вікно з формою, яку потрібно заповнити (рис. 3.7). В форму потрібно ввести назву моделі, обрати архітектуру та масштаб збільшення, а також режим, за яким буде будуватися модель. Є чотири режими: продуктивність, баланс, якість та користувацький. Режим продуктивності дозволяє створити модель, яка буде працювати швидко. За режиму якості модель видаватиме більш кращі результати, тому і швидкість обробки буде менша. Режим балансу є серединою між режимами продуктивності і якості.

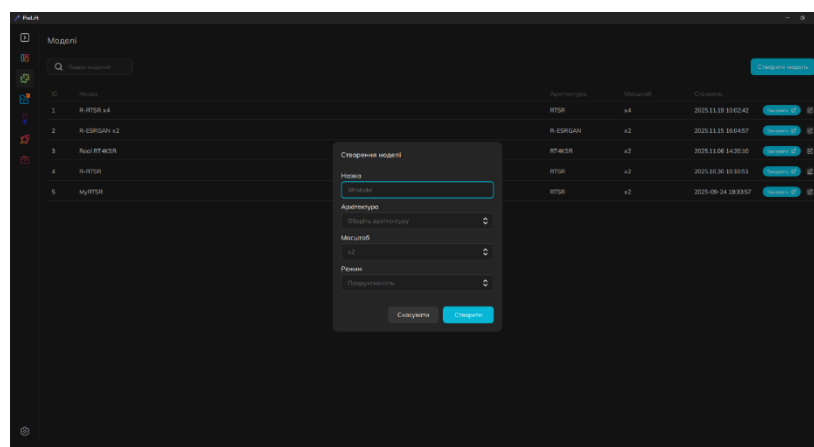


Рисунок 3.7 – Модальне вікно створення моделі

Користувацький режим дозволяє налаштувати параметри моделі самостійно, якщо це потрібно. Значення параметрів моделі автоматично підставляються до режиму балансу. Кожна модель має власні параметри, які потрібно налаштувати. На рис. 3.8 наведено приклад налаштування власних параметрів моделі.

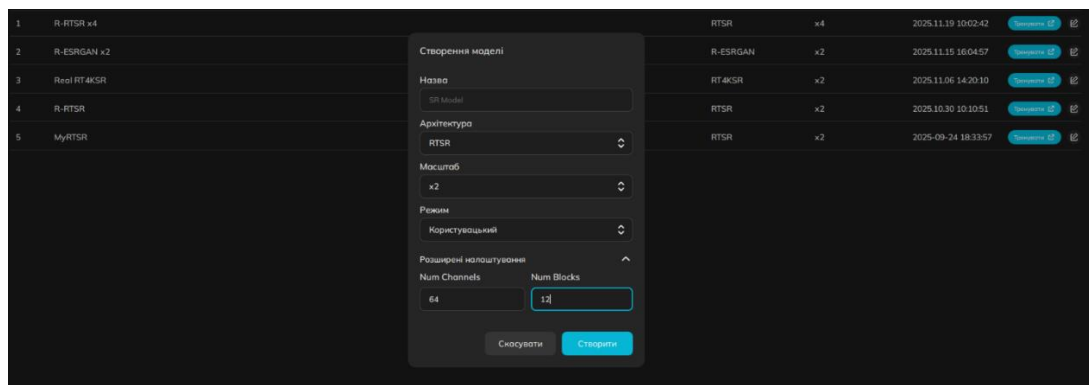


Рисунок 3.8 – Розширені налаштування при користувацькому режимі

Кожне наступне модальне вікно має валідацію форми з відображенням помилок. Валідація є гнучкою, тому може використовуватися в різних компонентах. Форма буде оброблена програмним продуктом відповідно до правил валідацій. На рис. 3.9 показано приклад роботи валідації форми.

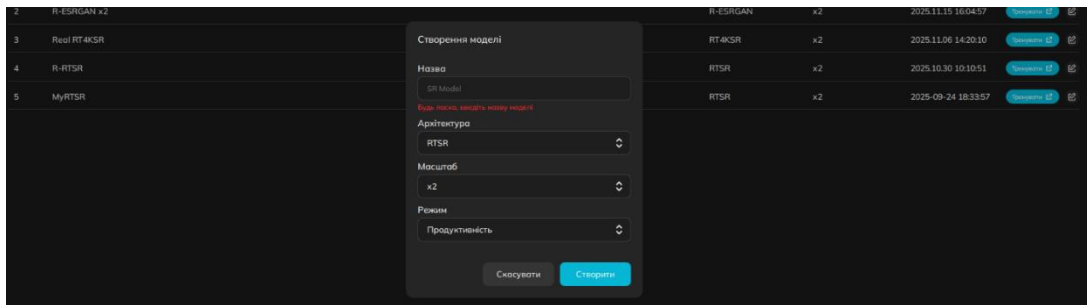


Рисунок 3.9 – Приклад валідації форми

Після створення моделі, вона з'явиться в списку моделей. Додатково буде відображено повідомлення про її успішне створення. Якщо виникне помилка, то форма створення залишиться відкритою, а користувачу виведеться інформативне повідомлення. На рис. 3.10 відображено сторінку зі створеною моделлю.

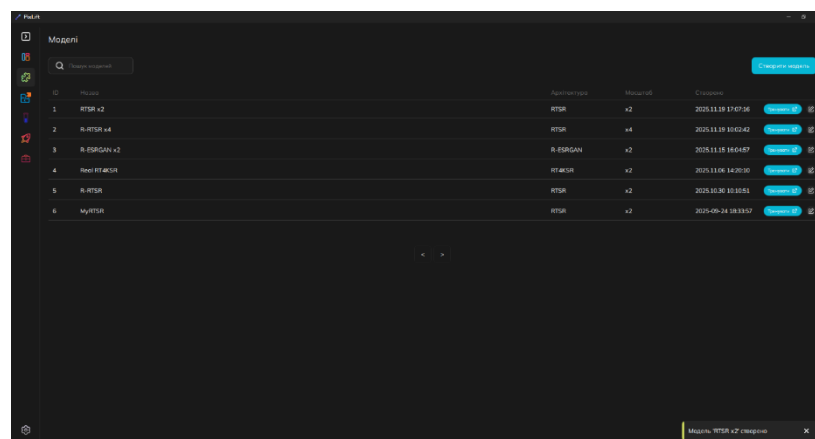


Рисунок 3.10 – Успішно створена модель

Створену модель можна почати навчати. Навчання усіх моделей відбувається на сторінці тренування. На цій сторінці відображаються усі моделі, які були обрані для тренування. Як і на сторінці моделей, присутній

пошук та пагінація тренувань, є можливість запускати навчання моделей. Кожне тренування має інформацію про назву моделі, архітектуру, масштаб та коротку інформацію з параметрами тренування. На рис. 3.11 зображено сторінку тренувань.

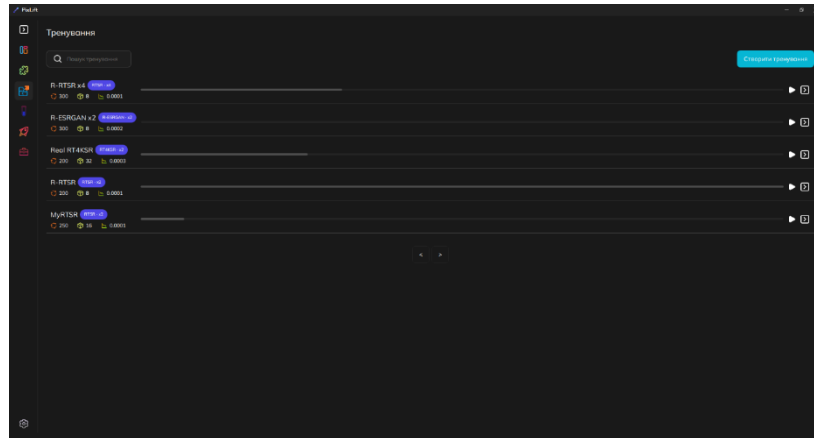


Рисунок 3.11 – Сторінка тренувань

Для того, щоби створити тренування, потрібно натиснути кнопку тренування створеної моделі на сторінці моделей або тренувань. Після чого з'явиться модальне вікно з формою, яку потрібно заповнити. На рис. 3.12 зображено модальне вікно створення тренування.

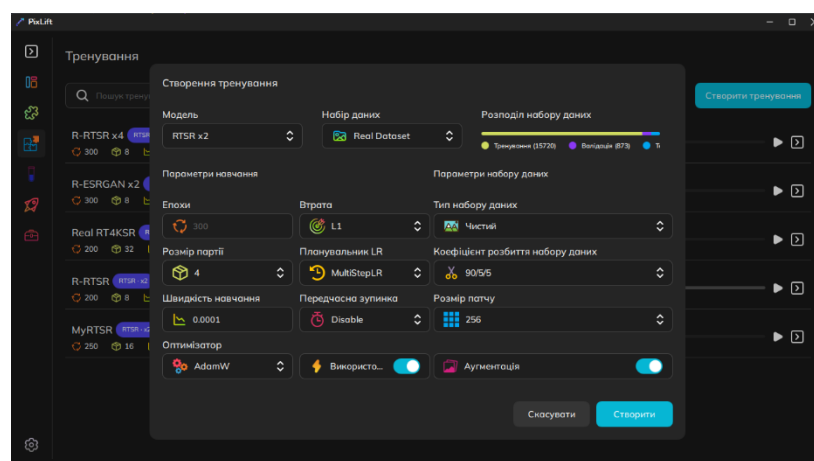


Рисунок 3.12 – Модальне вікно створення тренування

При створенні моделі обов'язково потрібно обрати модель для навчання та набір даних, на яких модель буде навчатися. Всі інші параметри є необов'язковими для заповнення, оскільки вони за замовчуванням підставляються відповідно обраної архітектури моделі та її режиму. Додатково відображається інформація про розбиття набору даних через прогрес розподілу.

Після створення тренування можна розпочинати навчання моделі. На сторінці конкретного тренування, яку можна відкрити через натискання на кнопку у вигляді стрілки, відображається інформація у лівій колонці блока, яка була заповнена під час створення тренування. На рис. 3.13 наведено вигляд сторінки конкретного тренування.

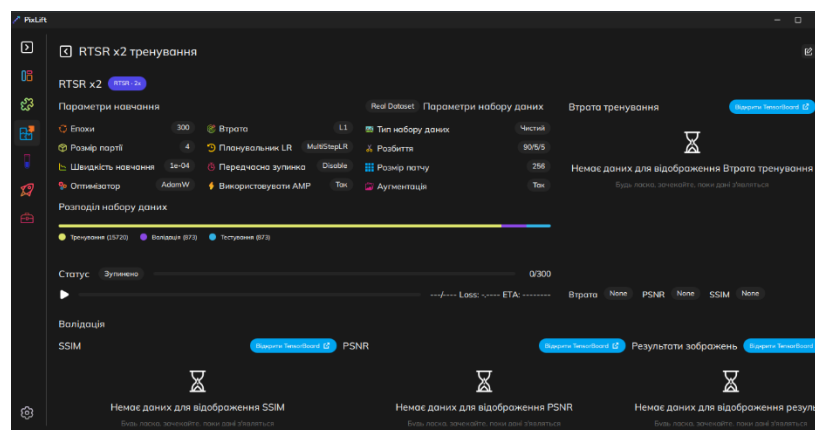


Рисунок 3.13 – Сторінка конкретного тренування

У правій колонці блока знаходиться графік втрати тренування. Зараз він порожній, оскільки модель ще ніколи не навчалася. Нижче під графіком відображається інформація про поточну втрату тренування та дві метрики якості, які також є порожніми і будуть розглянуті пізніше. Тренування має власний статус, який змінюється під час навчання. Біля статусу відображається прогрес закінчених епох тренування, під ним і ще один прогрес, який відображає дані поточного статусу та інформацію щодо запущеного тренування справа. Додатково, на відміну від сторінки головної

панелі, є відображення валідації під час тренування: два графіка метрик якості та результати зображень.

При старті навчання моделі статус тренування та кольори прогресів зміняться. Статус тренування зміниться на статус валідації роботи моделі при проходженні всіх ітерацій епохи. На рис. 3.14 показано прогрес навчання моделі.

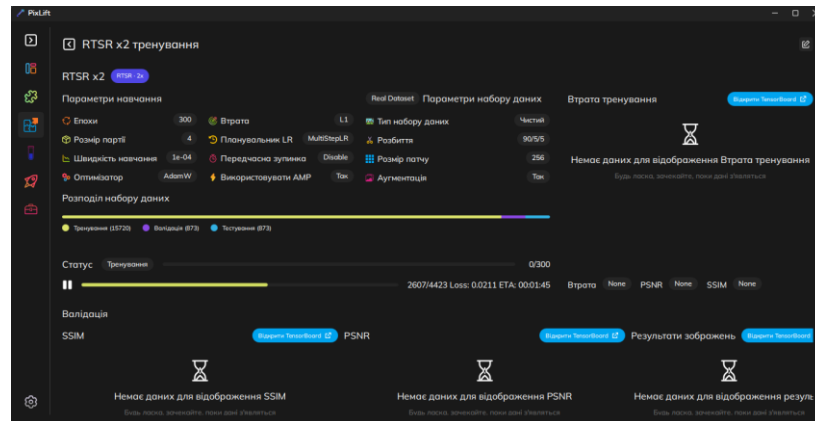


Рисунок 3.14 – Прогрес навчання моделі

Після проходження першої епохи тренування графік втрати тренування оновиться. Графік демонструє, наскільки модель вміє відновлювати текстури та контури зображень. Значення цього графіку повинні зменшуватися, оскільки низькі значення вказують на кращу роботу моделі. На рис. 3.15 зображено графік втрати тренування.

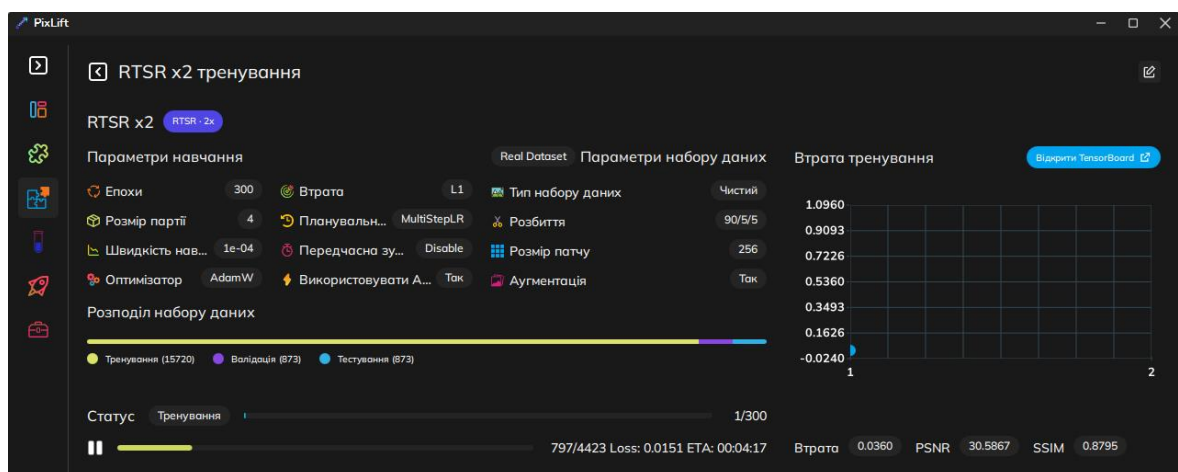


Рисунок 3.15 – Оновлений графік втрати тренування

Наступним кроком відбувається валідація моделі, після якої буде відображено графіки метрик якості та результати зображень (рис. 3.16). Зображення є мозаїкою, яка складається з 9 зображень: по 3 приклади зображень низької роздільної здатності, підвищеної роздільної здатності та високої роздільної здатності.

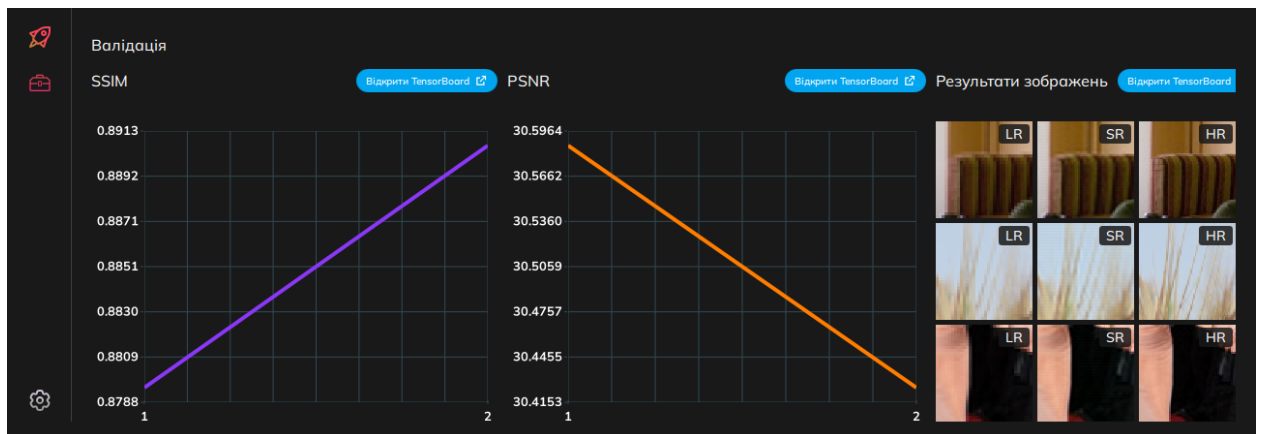


Рисунок 3.16 – Графіки метрик якості та результати зображень

Натреновані моделі можна тестувати. На сторінці тестування користувач може обрати: моделі, які він хоче протестувати; набори незалежних даних для тестування моделей; різні метрики. На рис. 3.17 показано порожню сторінку тестування.

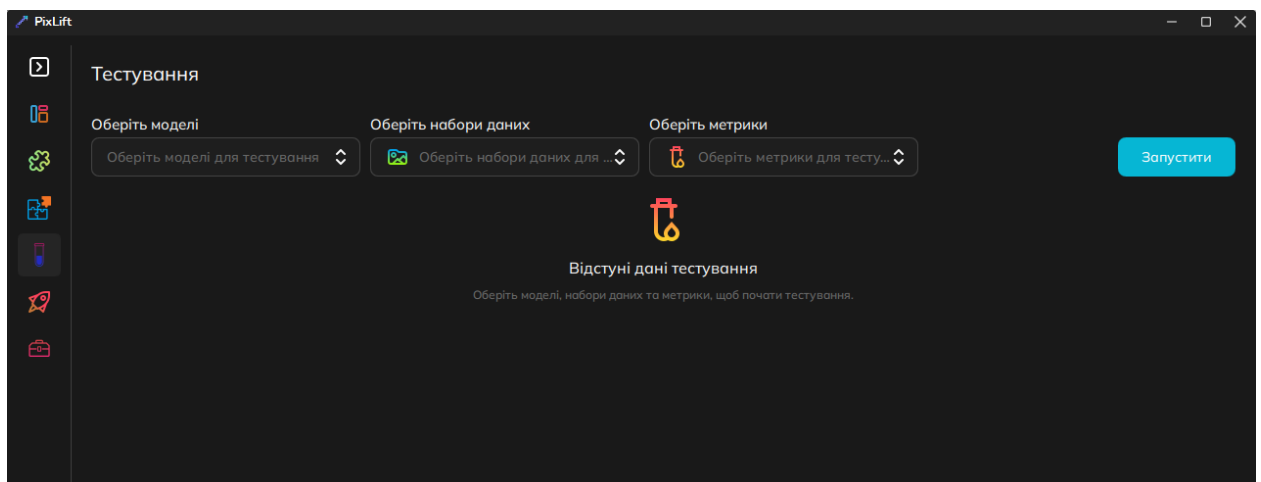


Рисунок 3.17 – Сторінка тестування

Після запуску тестування, з'являються розгорнуті списки з обраними наборами даних для тестування, де відображається інформація про результати метрик у вигляді таблиці для кожної моделі та порівняння зображень з різними моделями у вигляді мозаїки. Результати тестування зберігаються у вигляді CSV даних та зображень, які знаходяться в папці результатів. Результати зображень можна приближувати для порівняння пікселів кожного результату. На рис. 3.18 зображено результати тестування моделей на наборі даних Set14.

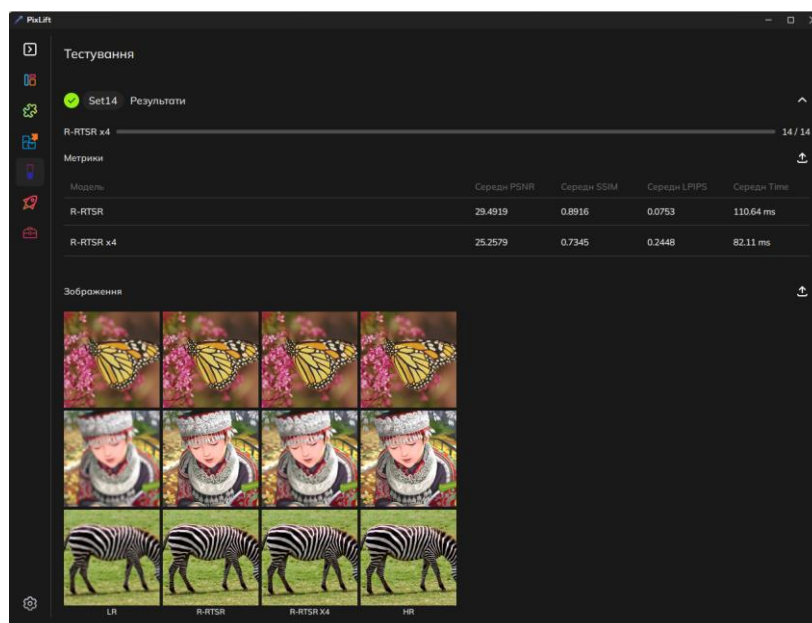


Рисунок 3.18 – Результати тестування моделей

Програмний продукт підтримує обробку зображень та відео послідовностей. На сторінці обробки користувач обирає модель та тип ресурсу: зображення або відео, додатково місце збереження оброблених результатів. Для вибору файлів обробки достатньо їх перетягнути в зону вибору файлів або обрати через файловий діалог. Нижче відображається розгорнутий список з обраними файлами. Список підтримує пошук, видалення та відкриття розташування файлів. На рис. 3.19 наведено обрані зображення для обробки конкретною моделлю.

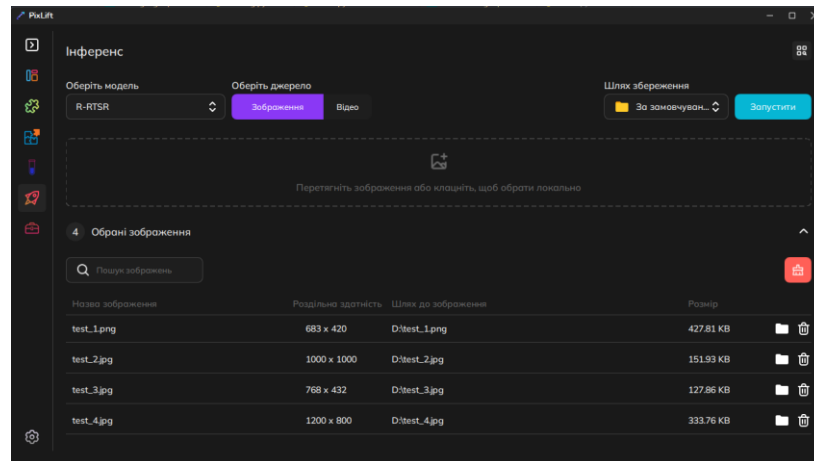


Рисунок 3.19 – Вибір зображень для обробки

При натисканні кнопки обробки інтерфейс вибору вище описаних параметрів блокується та починається обробка обраних файлів. Під час обробки відображається прогрес обробки конкретного ресурсу та скелетони майбутніх оброблених ресурсів. Після обробки ресурсу зображення будуть підставлені замість скелетонів. На рис. 3.20 наведено приклад декількох оброблених зображень.

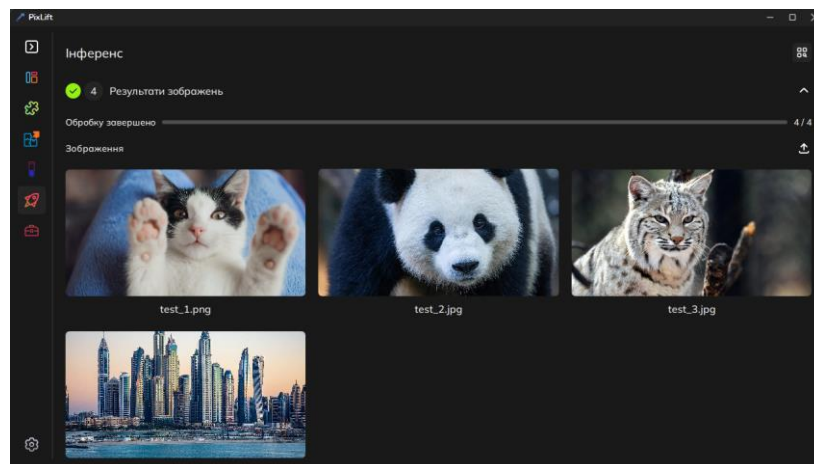


Рисунок 3.20 – Оброблені зображення

Для перегляду всіх оброблених результатів різними моделями потрібно перейти на сторінку результатів обробки, на яку можна потрапити натисканням кнопки, яка розташована в правому верхньому куті сторінки

обробки ресурсів. Сторінка результатів обробки включає пошук та сортування по даті створення. На рис. 3.21 зображено сторінку результатів обробки.

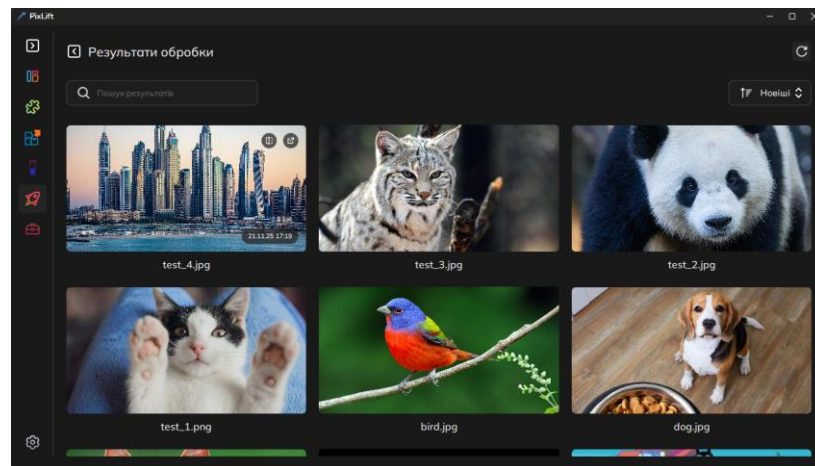


Рисунок 3.21 – Сторінка результатів обробки

Оброблені зображення можна порівняти з оригіналами. Для цього достатньо натиснути кнопку порівняння при наведенні на оброблене зображення, після чого відкриється сторінка порівняння зображень. На рис. 3.22 наведено сторінку порівняння зображень.

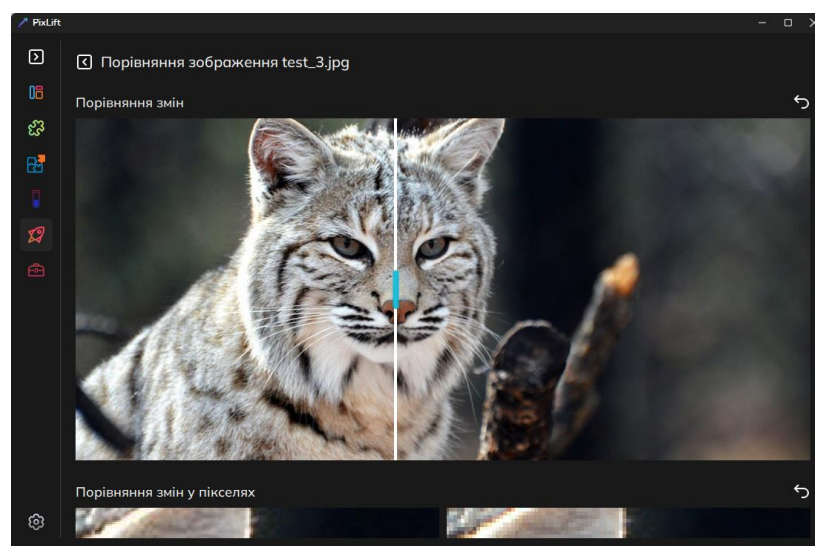
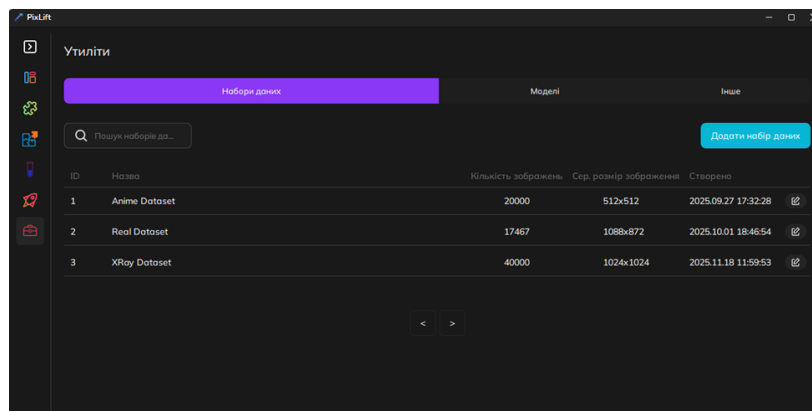


Рисунок 3.22 – Сторінка порівняння зображень

На сторінці є два способи порівняння: порівняння змін та порівняння пікселів. При порівняння змін використовується вертикальний повзунок, який

можна перетягувати і порівнювати зміни з лівим обробленим зображенням та правим оригінальним. Порівняння пікселів дозволяє переглянути різницю пікселів між обробленим та оригінальним зображеннями. Для збільшення та зменшення розміру зображень можна використовувати колесо, ліву та праву кнопку миші відповідно.

Для навчання моделей використовуються набори даних, які потрібно створити. На сторінці утиліт наявна вкладка “Набори даних”, на якій можна переглянути створені набори даних користувачем з пошуком та пагінацією. На рис. 3.23 зображено сторінку наборів даних.



Утиліти				
Набори даних				
Пошук наборів даних...				
Додати набір даних				
ID	Назва	Кількість зображень	Середній розмір зображення	Створено
1	Anime Dataset	20000	512x512	2025.09.27 17:32:28
2	Real Dataset	17467	1088x872	2025.10.01 18:46:54
3	XRay Dataset	40000	1024x1024	2025.11.18 11:59:53

Рисунок 3.23 – Сторінка наборів даних

Для додавання нового набору даних потрібно відкрити модальне вікно та заповнити форму: назва набору даних та шлях до набору даних. Додаткові фільтри призначені для фільтрування зображень набору даних під певний розмір чи формат. При натисканні кнопки додавання почнеться процес фільтрування зображень, після якого набір даних з’явиться на сторінці набору даних. Під час фільтрування відображається інформація про загальну кількість знайдених зображень відповідно до обраних форматів зображень, пропущених зображень через невідповідність обраного розміру та всього відфільтрованих зображень. На рис. 3.24 зображено процес фільтрування набору даних рентгенівських знімків.

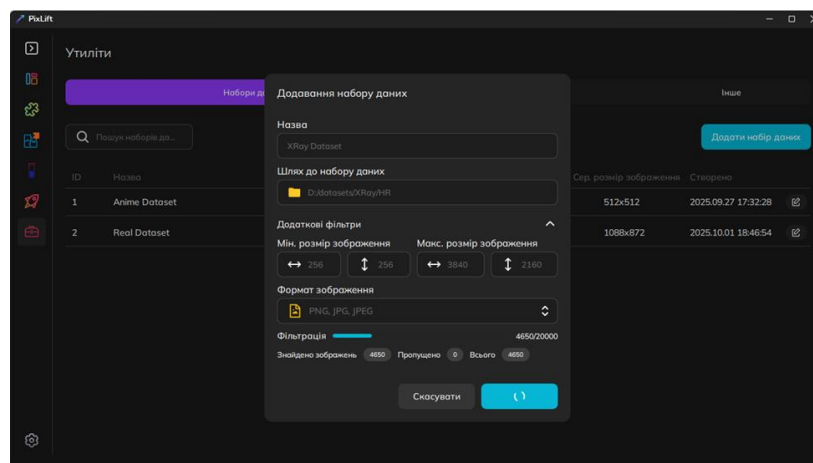


Рисунок 3.24 – Процес фільтрування набору даних рентгенівських знімків

На вкладці моделей сторінки утиліти є різні можливості взаємодії з моделями. Моделі можна експортувати та імпортувати, додатково додано експорт метрик, графіків та зображень натренованих моделей. У кожному блоці функціональності наявна валідація. На рис. 3.25 наведено вкладку утиліт моделей.

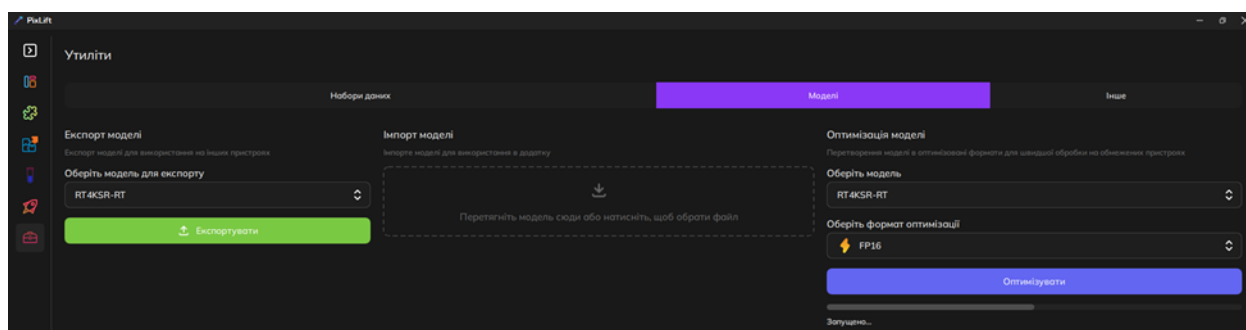


Рисунок 3.25 – Утиліти моделей

Вкладка “Інше” складається з різних функцій, які корисні для зображень та відео послідовностей. Зображення можна конвертувати в інший формат зображення, наприклад, для оптимізації розмірів набору даних. Відео послідовності підтримують також різні формати конвертацій для зручності використання на обмежених пристроях, які не підтримують деякі відеокодеки. На рис. 3.26 зображено вкладку інших утиліт.

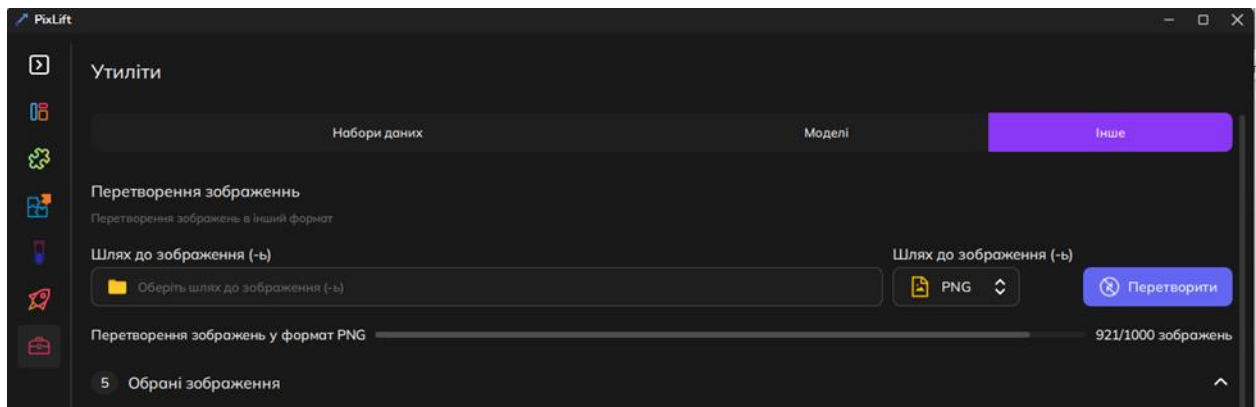


Рисунок 3.26 – Інші утиліти

Програмний продукт підтримує дві мови: українську та англійську. В майбутньому є можливість розширити пакет мов. На сторінці налаштування можна змінити мову, обрати обчислювальний пристрій, шлях оброблених результатів та тимчасової папки та згортання програми в сонний режим. На рис. 3.27 продемонстровано зміну мови на сторінці налаштувань.

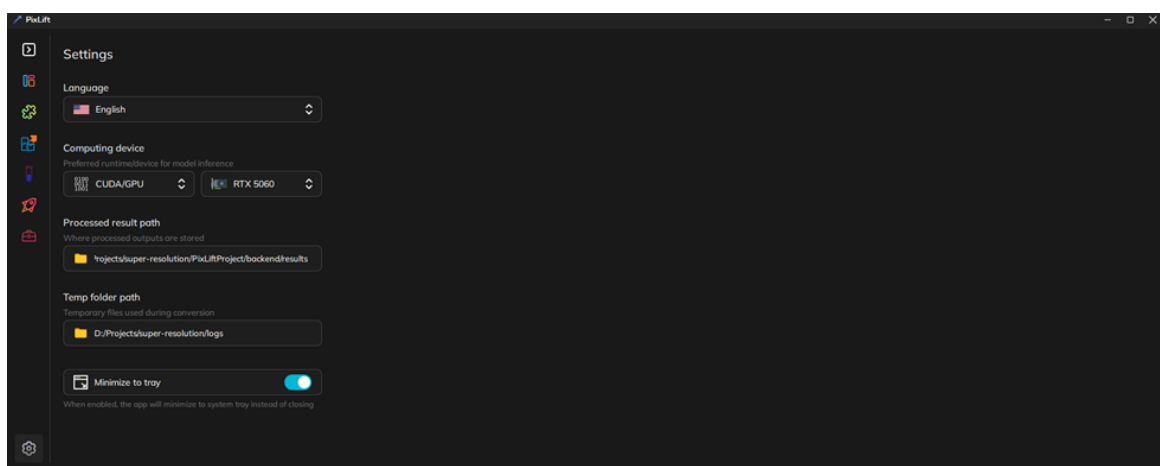


Рисунок 3.27 – Змінена мова програмного продукту

3.3 Тестування роботи створених моделей

Для тестування роботи моделей потрібно використати незалежні набори даних, які не використовувалися під час навчання. Для тестування були обрані такі тестові набори даних підвищення роздільної здатності: Set14, BSDS100 та General100 [36].

Set12 складається з 12 зображень природних сцен. Зображення підібрано випадковим чином з різним змістом та контрастом, тому вони й використовуються для оцінки якості реконструкції дрібних деталей та текстур.

BSDS100 є частиною великого набору даних Berkeley Segmentation Dataset. Ця версія має 100 зображень з різними текстурами та дрібними деталями елементів. Результати тестування набору даних показують, наскільки модель відновлює складні текстури та неоднорідні області.

General100 містить 100 зображень з рівномірним розподілом якості та складності сцен. Зазвичай, він використовується для повторної перевірки моделей для оцінки контрастності, якщо попередній набір даних має певні відхилення.

Для оцінки якості підвищення роздільної здатності, яка відображає наскільки модель якісно відновлює текстури та складні елементи зображення, було обрано три основні, найбільш використовувані, метрики.

Першою метрикою є PSNR (Peak-Signal-to-Noise Ratio) [37]. PSNR оцінює точність відновлення зображень після збільшення роздільної здатності. Якщо дослівно перекласти – це різниця, або відношення, між відновленим зображенням та оригіналом. Основою метрики є середньоквадратична помилка MSE, яка показує відхилення кожного пікселя від оригінального значення:

$$MSE = \frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W (I_{ij} - K_{ij})^2. \quad (3.1)$$

Після обчислення середньоквадратичної помилки визначається PSNR:

$$PSNR = 10 \log_{10} \left(\frac{MAX^2}{MSE} \right), \quad (3.2)$$

де MAX – діапазон кольорів (255).

Якщо результат обчислень є великим значенням, то відновлене зображення містить менше помилок, і навпаки, якщо невелике значення, то

зображення достатньо відрізняється від оригіналу. Метрика не враховує сприйняття зображення людиною, тому її доповнюють іншими метриками.

Метрика SSIM (Structural Similarity Index Measure) [38] оцінює структурну подібність між двома зображеннями. Метрика краще підходить для оцінки сприйняття людиною відновленого зображення, ніж PSNR. Обчислення виконується у невеликих вікнах, зміни яких впливають на значення індексу. Метрика обраховується шляхом порівнянь середніх значень, дисперсій та коваріацій між двома зображеннями. Формула обрахунку метрики є такою:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \quad (3.3)$$

де μ_x, μ_y – середні значення;

σ_x^2, σ_y^2 – дисперсії;

σ_{xy} – коваріації;

C_1, C_2 – стабілізуючі константи.

Значення SSIM, яке є ближчим до одиниці, демонструє, що відновлене зображення є подібним по структурі до оригінального.

Останньою метрикою якості для тестування було обрано LPIPS [39]. Вона оцінює якість порівнювальних зображень на основі ознак, отриманих із згорткових нейронних мереж. Обчислення відбуваються разом з навченою згортковою нейронною мережею, через яку проходить зображення. На базі отриманих ознак визначається відстань між ними. Кожний шар нейронної мережі має власну вагу для визначення важливості внеску кожної отриманої ознаки. Метрика LPIPS визначається так:

$$LPIPS(x, y) = \sum_l w_l \|f_l(x) - f_l(y)\|_2^2, \quad (3.4)$$

де f_l – ознаки на шарі l ;

w_l – навчені ваги на шарі l .

Невелике значення LPIPS вказує на високу подібність зображень у просторі ознак, через це можна дізнатися, наскільки модель зберігає природний вигляд деталей.

Додатково, для тестування було додано метрику швидкості для порівняння швидкодії обробки.

Для порівняння роботи моделей було обрано R-ESRGAN, RT4KSR та створену модель. Тестування проводилося при коефіцієнті масштабу збільшення два. В табл. 3.1-3.3 представлені результати метрик на обраних наборах даних.

Таблиця 3.1

Порівняння результатів обчислених метрик на наборі даних Set14

	PSNR	SSIM	LPIPS	Час обробки
R-ESRGAN (x2)	24.14	0.6144	0.4126	107.13 ms
RT4KSR (x2)	25.82	0.7964	0.2431	86.37 ms
Власна (x2)	29.49	0.8916	0.0753	90.30 ms

Таблиця 3.2

Порівняння результатів обчислених метрик на наборі даних BSDS100

	PSNR	SSIM	LPIPS	Час обробки
R-ESRGAN (x2)	24.78	0.7012	0.3412	80.30 ms
RT4KSR (x2)	25.43	0.7616	0.2977	57.94 ms
Власна (x2)	29.01	0.8908	0.1148	60.79 ms

Таблиця 3.3

Порівняння результатів обчислених метрик на наборі даних General100

	PSNR	SSIM	LPIPS	Час обробки
R-ESRGAN (x2)	25.43	0.7813	0.2134	93.16 ms
RT4KSR (x2)	26.14	0.8092	0.1813	70.04 ms
Власна (x2)	32.84	0.8534	0.0342	74.52 ms

Виходячи з результатів тестування, створена модель є кращою, ніж R-ESRGAN та RT4KSR. Результати метрик доводять точність відновлення дрібних деталей та структур зображень моделлю. Значення LPIPS є найменшим значенням на трьох наборах даних, тому і зображення краще сприймаються людським оком. Створена модель має низький рівень помилок при реконструкції зображень, доказом цього є значення метрики PSNR. Час також не перевищує норму серед моделей.

R-ESRGAN продемонструвала достатньо низький результат під час тестування. Результати є зрозумілими, оскільки модель використовує двоступеневу модель деградації, через це і метрики якості є нижчими за значеннями, якщо порівнювати їх з результатами інших моделей.

Модель RT4KSR має відповідні результати для такої архітектури. Швидкість обробки моделі є найвищою серед всіх протестованих моделей. Точність відновлення може бути кращою, якщо збільшити кількість блоків та каналів моделі, але бажано цього не робити, оскільки модель зазвичай використовується в реальному часі, а швидкість для такого режиму є важливою [40].

Одним із варіантів покращення результатів під час тестування є збільшення розмірів наборів даних. Велика кількість зображень дозволить моделям навчитися ще більшим варіантам відновлення текстур та різних деталей.

На рис. 3.28 зображено приклад відновлення різних зображень моделями набору даних Set14 для візуального порівняння. В першій колонці відображаються низько роздільні зображення, остання колонка – високо роздільні зображення, між ними – результати оброблення моделями. Кожна колонка має відповідний підпис для зручності.

Візуально можна побачити, що кожна модель здатна відновлювати різні деталі.

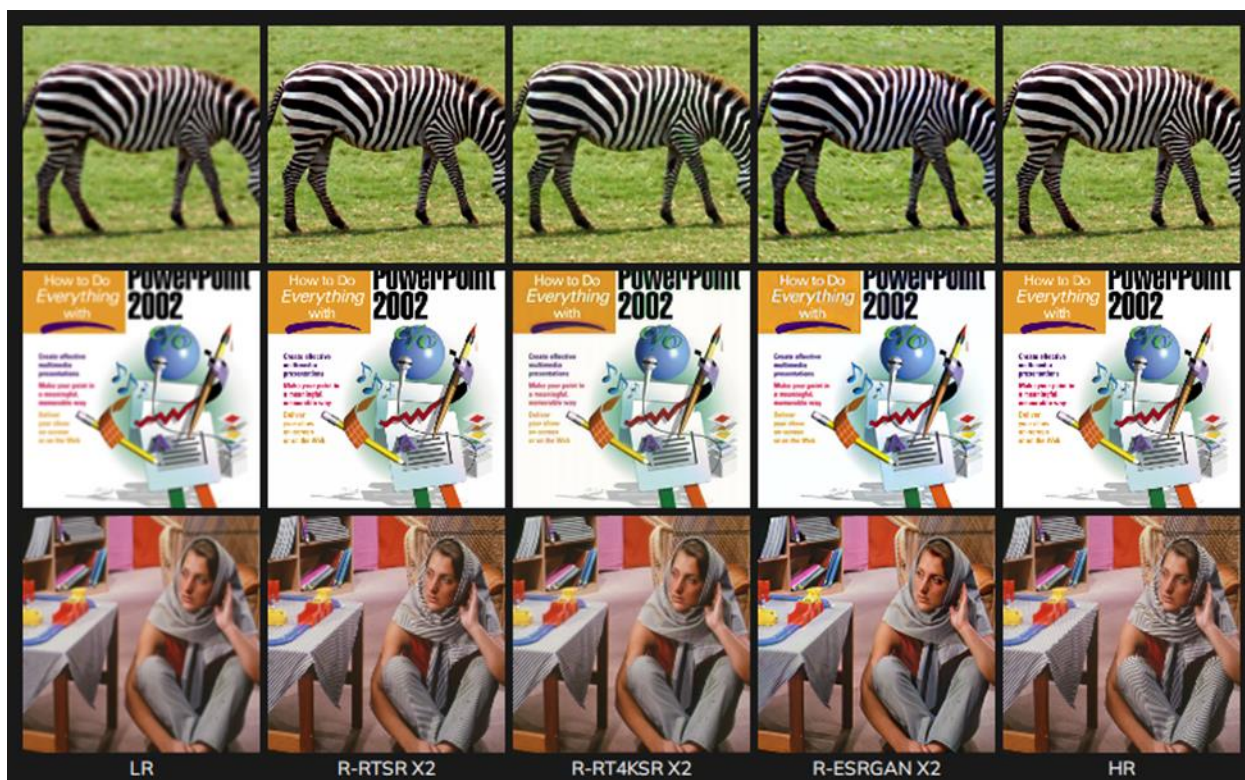


Рисунок 3.28 – Порівняння результатів оброблення моделями

Підсумовуючи результати тестування, можна відзначити, що кожна модель може бути застосованою для різних випадків відновлення унікальних зображень: R-ESRGAN найкраще використовувати для реальних зображень, які можуть мати спотворення; RT4KSR – для задач реального часу та різних відео послідовностей, де важливе значення має швидкодія, а не якість; створена модель найкраще підходить для задач, де потрібна точність відновлення текстур.

Висновки до третього розділу

У третьому розділі було описано проектування програмного продукту та наведено опис взаємодії користувача у вигляді діаграми. Розглянуто інтерфейс розробленого програмного продукту. Описано процес створення, навчання та тестування моделей, розглянуто обробку зображень та відео послідовностей з можливістю їхнього перегляду.

Було проведено тестування моделей R-ESRGAN, RT4KSR та власної створеної моделі на наборах даних Set14, BSDS100 та General100. На основі результатів тестування порівняно значення метрик якості PSNR, SSIM та LPIPS кожної моделі, визначено галузі, в яких моделі найкраще підходять для обробки зображень та відео послідовностей.

ВИСНОВКИ

Під час виконання випускної кваліфікаційної роботи магістра було проведено дослідження процесу підвищення роздільної здатності зображень і відео послідовностей, методів та засобів їхньої реалізації на сучасних обчислювальних пристроях.

Початковим етапом стало дослідження процесу підвищення роздільної здатності. Після чого було оглянуто та проаналізовано сучасні методи та засоби, які використовуються для вирішення таких задач. Визначені переваги та недоліки стали основою для формування завдань дослідження. Першим завданням стала розробка моделей підвищення роздільною здатності, другим завданням – розробка зручного програмного продукту для створення, навчання та тестування розроблених моделей.

Основними розроблювальними моделями були ESRGAN, R-ESRGAN та RT4KSR. Проаналізовано структуру, функції активації, методи відновлення текстур та особливості тренування кожної моделі. На основі дослідження відзначених моделей було створено власну модель зі спрощеною архітектурою. На наступному етапі було підготовлено набори даних для тренування, валідації та тестування розроблених моделей. Кожен набір даних було обмежено з міркувань наявних обчислювальних ресурсів, але цього вистачило для тренування та тестування роботи моделей. Для кожної моделі було описано процес навчання з найкращими параметрами відповідно до архітектур.

При розробці програмного продукту було спроектовано його структуру, визначено функціональні та нефункціональні вимоги. Програмний продукт складається з двох частин, які були реалізовані засобами QML та мовою програмування Python з використанням бібліотек PySide6 та PyTorch. Описано інтерфейс програмного продукту з конкретизацією дій створення, тренування, тестування моделей та обробки зображень і відео послідовностей.

На основі натренованих моделей було проведено тестування на незалежних наборах даних. Оцінювання моделей проводилися за допомогою різних метрик якості. Результати тестування продемонстрували, що створена модель показує найвищі значення метрик порівняно з іншими моделями. Дослідження показало, що модель R-ESRGAN краще підходить для реальних фотографій, а RT4KSR – для систем реального часу.

Створені моделі здатні якісно підвищувати роздільну здатність зображень і відео послідовностей за різних умов навчання та застосування. Завдяки розробленому програмному продукту створення, навчання та тестування моделей стала більш легкою та простішою задачею. Інтерфейс є зрозумілим та швидким для засвоєння будь-яким користувачем.

Виконане дослідження є основною для продовження розробок різних моделей у галузі підвищення роздільної здатності. Цей напрям досі залишається відкритим для вдосконалення через збільшення розмірів наборів даних, використання сучасних моделей, різні оптимізації для відео послідовностей. Створені модель та програмний продукт можуть бути використані як платформа для таких досліджень та допомогти розширити можливості відновлення інформації на зображеннях та відео послідовностях шляхом підвищення роздільної здатності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kostyuchenko A., Loktikova T., Lysogor Iu. Development of a Neural Model for Image and Video Resolution Enhancement. Тези доповідей XV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», 28-29 березня 2025 року. Житомир: «Житомирська політехніка», 2025. С. 41-42. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2025/04/povnyj-tekst.pdf>.
2. Kostyuchenko A., Kushnir N., Loktikova T. Development and research of effective deep learning models for improving the resolution of images and video sequences. Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія Технічні науки. 2025. №6. (Подано та прийнято до опублікування).
3. Image super-resolution: A comprehensive review, recent trends, challenges and applications / D. C. Lepcha et al. Information Fusion. 2023. Vol. 91. P. 230–260.
4. Can Maral B. Single Image Super-Resolution Methods: A Survey. arXiv. 2022. URL: <https://arxiv.org/abs/2202.11763>.
5. Mukherjee S. A Complete Guide to Image Super-Resolution in Deep Learning and AI. DigitalOcean. URL: <https://www.digitalocean.com/community/tutorials/image-super-resolution>.
6. S Malini M., Patil M. Interpolation Techniques in Image Resampling. International Journal of Engineering & Technology. 2018. Vol. 7, no. 3.34. P. 567. URL: <https://doi.org/10.14419/ijet.v7i3.34.19383>.
7. Unlocking Super-Resolution in Computer Vision. Number Analytics - AI Statistical Software. URL: <https://www.numberanalytics.com/blog/ultimate-guide-superresolution-computer-vision>.
8. Deep Learning Based Single Image Super-resolution: A Survey / V. K. Ha et al. International Journal of Automation and Computing. 2019. Vol. 16, no. 4. P. 413–426. URL: <https://doi.org/10.1007/s11633-019-1183-x>.

9. Arbitrary-Scale Video Super-Resolution with Structural and Textural Priors / W. Shang et al. arXiv. 2024. URL: <https://arxiv.org/abs/2407.09919>.
10. Suitability of Single Image Super-Resolution Models for Video Super-Resolution. International Journal of Engineering and Advanced Technology. 2020. Vol. 9, no. 5. P. 368–371. URL: <https://doi.org/10.35940/ijeat.e9575.069520>.
11. Wang J., Song R. Improved Super-Resolution Convolution Neural Network for Large Images. arXiv. 2019. URL: <https://arxiv.org/abs/1907.12928>.
12. Research on Image Super-Resolution Reconstruction Mechanism based on Convolutional Neural Network / H. Yan et al. arXiv. 2024. URL: <https://arxiv.org/abs/2407.13211>.
13. Enhanced Deep Residual Networks for Single Image Super-Resolution / B. Lim et al. arXiv. 2017. URL: <https://arxiv.org/abs/1707.02921>.
14. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks / X. Wang et al. arXiv. 2018. URL: <https://arxiv.org/abs/1809.00219>.
15. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data / X. Wang et al. arXiv. 2021. URL: <https://arxiv.org/abs/2107.10833>.
16. Towards Real-Time 4K Image Super-Resolution / E. Zamfir et al. CVPRW. 2023. URL: https://openaccess.thecvf.com/content/CVPR2023W/NTIRE/html/Zamfir_Towards_Real-Time_4K_Image_Super-Resolution_CVPRW_2023_paper.html.
17. SwinIR: Image Restoration Using Swin Transformer / J. Liang et al. arXiv. 2021. URL: <https://arxiv.org/abs/2108.10257>.
18. Recovering Realistic Texture in Image Super-resolution by Deep Spatial Feature Transform / X. Wang et al. arXiv. 2018. URL: <https://arxiv.org/abs/1804.02815>.
19. ComfyUI | Generate video, images, 3D, audio with AI. URL: <https://www.comfy.org/>.
20. chaiNNer. URL: <https://chainner.app/>.
21. Upscayl - Free and Open Source AI Image Upscaler. URL: <https://upscayl.github.io/>.

22. Thakur A. Generative Adversarial Networks. International Journal for Research in Applied Science and Engineering Technology. 2021. Vol. 9, no. 8. P. 2307–2325. URL: <https://doi.org/10.22214/ijraset.2021.37723>.
23. Li H. Image Super-Resolution Algorithm Based on RRDB Model. IEEE Access. 2021. Vol. 9. P. 156260–156273. URL: <https://doi.org/10.1109/access.2021.3118444>.
24. Jolicoeur-Martineau A. The relativistic discriminator: a key element missing from standard GAN. arXiv. 2018. URL: <https://arxiv.org/abs/1807.00734>.
25. Choi Y., Park H. Improving ESRGAN with an additional image quality loss. Multimedia Tools and Applications. 2022. URL: <https://doi.org/10.1007/s11042-022-13452-4>.
26. Park S., Shin Y.-G. U-Net-Based Generative Adversarial Network. Journal of the Institute of Electronics and Information Engineers. 2021. Vol. 58, no. 5. P. 61–67. URL: <https://doi.org/10.5573/ieie.2021.58.5.61>.
27. Bodapati J. D., Veeranjanyulu N. Feature Extraction and Classification Using Deep Convolutional Neural Networks. Journal of Cyber Security and Mobility. 2018. URL: <https://doi.org/10.13052/2245-1439.825>.
28. Simple Baselines for Image Restoration / L. Chen et al. arXiv. 2022. URL: <https://arxiv.org/abs/2204.04676>.
29. Wang X., Dong C., Shan Y. RepSR: Training Efficient VGG-style Super-Resolution Networks with Structural Re-Parameterization and Batch Normalization. arXiv. 2022. URL: <https://arxiv.org/abs/2205.05671>.
30. Kaiming Initialization in Deep Learning. GeeksforGeeks. 2025. URL: <https://www.geeksforgeeks.org/deep-learning/kaiming-initialization-in-deep-learning/>.
31. Image Super-Resolution Using Very Deep Residual Channel Attention Networks / Y. Zhang et al. ECCV. 2018. URL: https://openaccess.thecvf.com/content_ECCV_2018/html/Yulun_Zhang_Image_Super-Resolution_Using_ECCV_2018_paper.html.

32. DF2K + OST. Kaggle. URL: <https://www.kaggle.com/datasets/thaihoa1476050/df2k-ost>.
33. Anita Dataset. URL: https://zhenglinpan.github.io/AnitaDataset_homepage/.
34. Chest X-ray8. Redivis. URL: <https://redivis.com/datasets/612s-dx4rxexky>.
35. Fitzpatrick M. Create GUI Applications with Python and Qt5: The Hands-On Guide to Making Apps with Python. Independently Published, 2020. 812 p.
36. SuperBench: A Super-Resolution Benchmark Dataset for Scientific Machine Learning / P. Ren et al. arXiv. 2025. URL: <https://arxiv.org/abs/2306.14070>.
37. On the Computation of PSNR for a Set of Images or Video / O. Keleş et al. arXiv. 2021. URL: <https://arxiv.org/abs/2104.14868>.
38. Nilsson J., Akenine-Möller T. Understanding SSIM. arXiv. 2020. URL: <https://arxiv.org/abs/2006.13846>.
39. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric / R. Zhang et al. arXiv. 2018. URL: <https://arxiv.org/abs/1801.03924>.
40. Efficient Test-Time Adaptation for Super-Resolution with Second-Order Degradation and Reconstruction / Z. Deng et al. arXiv. 2023. URL: <https://arxiv.org/abs/2310.19011>.

ДОДАТКИ

```

import torch
from torch.utils.data import Dataset
from torchvision import transforms
import torchvision.transforms.functional as F
from PIL import Image
from pathlib import Path
from ..downsamples import BicubicDownsample
from ..logger.message import message_logger
from typing import Union, Optional, List, Dict, Any
import random

class RandomRotation:
    def __init__(self, angles: List[int] = [90, 180, 270]) -> None:
        self.angles = angles

    def __call__(self, x: torch.Tensor) -> torch.Tensor:
        angle = random.choice(self.angles)
        return F.rotate(x, angle)

class SRDataset(Dataset):
    """
    Base class for Super Resolution Datasets.

    This class supports two modes:
    - filesystem mode (pass data_dir) – old behavior (uses split folders
    train/val/test/HR)
    - list mode (pass hr_image_list) – accepts a list of image-path dicts
    or strings and uses them directly

    Args:
        data_dir (Path | str): Path to the dataset directory (optional if
        hr_image_list is provided).
        hr_image_list (list): List of items describing images. Allowed items:
            - dict with key "image_path"
            - str / Path with full image path
        scale (int): Scale factor for super resolution.
        split (str): Dataset split (train/val/test).

```

```

        patch_size (int): Size of the image patches (if 0, no crop for
val/test).
        max_i_size (int): kept for backward compatibility (not used unless
filter_imgs=True).
        filter_imgs (bool): if True, run original size-based filtering checks
(kkept for compatibility).
    """

    def __init__(
        self,
        data_dir: Union[Path, str, None] = None,
        hr_image_list: Optional[List[Union[Path, str, Dict[str, Any]]]] =
None,
        scale: int = 2,
        split: str = "train",
        patch_size: Optional[int] = 128,
        max_i_size: int = 2100,
        filter_imgs: bool = False,
    ) -> None:
        super().__init__()
        self.scale = scale
        self.split = split
        self.patch_size = patch_size

        self.degrade = BicubicDownsample(scale)
        self.hr_paths = []

        if hr_image_list is not None:
            missing = 0
            for item in hr_image_list:
                if isinstance(item, dict):
                    p = Path(item.get("image_path") or item.get("path") or
""")
                    else:
                        p = Path(item)
                    if p.exists():
                        self.hr_paths.append(p)
                    else:
                        missing += 1
            if missing > 0:

```

```

        message_logger.warning(
            f"[{split.upper()}] {missing} images from list not found
on disk and were skipped")
    else:
        if data_dir is None:
            raise ValueError(
                "Either data_dir or hr_image_list must be provided")

    hr_i_paths = self.get_path_images_by_split(Path(data_dir), split)

    if filter_imgs:
        skipped = 0
        min_lr_size = (patch_size or 0) // scale if patch_size else 0
        for hr_path in hr_i_paths:
            if not hr_path.exists():
                skipped += 1
                continue
            try:
                with Image.open(hr_path) as img:
                    w, h = img.size
            except Exception:
                skipped += 1
                continue
            if split == "train":
                if w < (patch_size or 0) or h < (patch_size or 0):
                    skipped += 1
                    continue
                if w <= min_lr_size or h <= min_lr_size:
                    skipped += 1
                    continue
                if w > max_i_size or h > max_i_size:
                    skipped += 1
                    continue
                if w == 0 or h == 0:
                    skipped += 1
                    continue
            else:
                # val/test checks
                if patch_size and (w < patch_size or h < patch_size):
                    skipped += 1

```

```

        continue
    if w == 0 or h == 0:
        skipped += 1
        continue
    self.hr_paths.append(hr_path)
if skipped > 0:
    message_logger.warning(
        f"[{split.upper()}] {len(self.hr_paths)} valid images
found ({skipped} skipped)")
else:
    found = 0
    missing = 0
    for p in hr_i_paths:
        if p.exists():
            self.hr_paths.append(p)
            found += 1
        else:
            missing += 1
    if missing > 0:
        message_logger.warning(
            f"[{split.upper()}] {missing} images in filesystem
split not found and were skipped")
    if found == 0:
        message_logger.warning(
            f"[{split.upper()}] No existing images found in
{Path(data_dir)/split/'HR'}")

    assert len(
        self.hr_paths) > 0, f"No valid images found for {split} split
(check image paths)"

    # Create various augmentation sets for training, validation and
testing

    if split == "train":
        self.transforms = transforms.Compose([
            transforms.RandomCrop(self.patch_size),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            RandomRotation(),

```

```

    ])
elif split in ("val", "test"):
    val_transforms = []

    if self.patch_size is not None and self.patch_size > 0:
        val_transforms.append(transforms.CenterCrop(self.patch_size))

    self.transforms = transforms.Compose(val_transforms)
else:
    raise ValueError(
        f"Unknown split: {split} (Available: train, val, test)")

@staticmethod
def get_path_images_by_split(data_dir: Union[Path, str], split: str =
"train") -> list[Path]:
    """
    Get image paths for a specific split (train/val/test).

    Args:
        data_dir (Path): Path to the dataset directory.
        split (str): Dataset split (train/val/test).

    Returns:
        list[Path]: List of image paths.
    """
    split_dir = Path(data_dir) / split / "HR"

    if not split_dir.exists():
        raise FileNotFoundError(
            f"Split directory {split_dir} does not exist.")

    exts = ("*.png", "*.jpg", "*.jpeg")
    paths = []

    for ext in exts:
        paths.extend(split_dir.glob(f"**/{ext}"))

    return sorted(paths)

def __len__(self) -> int:

```

```

        return len(self.hr_paths)

def __getitem__(self, index: int) -> dict:
    # Load HR image
    hr_image = Image.open(self.hr_paths[index]).convert("RGB")

    # Apply augmentation to HR
    hr_image_patch = self.transforms(hr_image)

    # Convert HR Patch to tensor
    hr_tensor = F.to_tensor(hr_image_patch)

    # Generate LR tensor use HR tensor
    lr_tensor, hr_tensor_cropped = self.degrade(hr_tensor)

    return {"lr": lr_tensor, "hr": hr_tensor_cropped}

class SRDataModule(torch.nn.Module):
    def __init__(
        self,
        data_dir: Union[Path, str, None] = None,
        dataset_images: Optional[List[Dict[str, Any]]] = None,
        data_split: Union[List[float], List[int], tuple] = (90, 5, 5),
        scale: int = 2,
        patch_size: int = 128,
        batch_size: int = 16,
        num_workers: int = 4,
        shuffle_split: bool = True,
        split_seed: Optional[int] = None,
    ) -> None:
        """
        Base Data module for Super Resolution datasets.

        Args:
            data_dir (Union[Path, str, None]): Path to the dataset directory.
            dataset_images (Optional[List[Dict[str, Any]]]): List of image
descriptors (dicts).
            data_split (Union[List[float], List[int], tuple]): Data split
ratios or counts for train/val/test.

```

```

        scale (int): Super resolution scale factor.
        patch_size (int): Size of the image patches.
        batch_size (int): Batch size for training.
        num_workers (int): Number of workers for data loading.
        shuffle_split (bool): Whether to shuffle the dataset before
splitting.
        split_seed (Optional[int]): Random seed for shuffling the
dataset.
    """
    super().__init__()
    self.data_dir = Path(data_dir) if data_dir is not None else None
    self.dataset_images = dataset_images
    self.data_split = tuple(data_split)
    self.scale = scale
    self.patch_size = patch_size
    self.batch_size = batch_size
    self.num_workers = num_workers
    self.shuffle_split = shuffle_split
    self.split_seed = split_seed

    self.train_ds: Union[SRDataset, None] = None
    self.val_ds: Union[SRDataset, None] = None
    self.test_ds: Union[SRDataset, None] = None

    @staticmethod
    def _normalize_split(split_tuple):
        s = list(split_tuple)
        total = sum(s)

        if total == 0:
            raise ValueError("data_split sums to zero")

        if total > 1 and total <= 300:
            s = [float(x) / 100.0 for x in s]
        else:
            s = [float(x) / total for x in s]

        if len(s) != 3:
            raise ValueError(
                "data_split must have length 3 (train, val, test)"
            )

```

```

        return tuple(s)

    @staticmethod
    def _split_list(items: List[Any], ratios: tuple, shuffle: bool = True,
seed: Optional[int] = None):
        N = len(items)

        if N == 0:
            return [], [], []

        idxs = list(range(N))
        rng = random.Random(seed)

        if shuffle:
            rng.shuffle(idxs)

        train_count = int(round(N * ratios[0]))
        val_count = int(round(N * ratios[1]))

        if train_count + val_count > N:
            val_count = max(0, N - train_count)

        train_idx = idxs[:train_count]
        val_idx = idxs[train_count:train_count + val_count]
        test_idx = idxs[train_count + val_count:]
        train = [items[i] for i in train_idx]
        val = [items[i] for i in val_idx]
        test = [items[i] for i in test_idx]

        return train, val, test

    def setup(self, stage: str = None) -> None:
        ratios = self._normalize_split(self.data_split)

        if self.dataset_images is not None:
            all_items = list(self.dataset_images)
            train_list, val_list, test_list = self._split_list(
                all_items, ratios, shuffle=self.shuffle_split,
seed=self.split_seed)

```

```

        if stage == "fit" or stage is None:
            self.train_ds = SRDataset(hr_image_list=train_list,
split="train",
                                     scale=self.scale,
patch_size=self.patch_size, filter_imgs=False)
            self.val_ds = SRDataset(
                hr_image_list=val_list, split="val", scale=self.scale,
patch_size=0, filter_imgs=False)

        if stage == "eval" or stage is None:
            self.test_ds = SRDataset(
                hr_image_list=test_list, split="test", scale=self.scale,
patch_size=0, filter_imgs=False)
        else:
            if stage == "fit" or stage is None:
                self.train_ds = SRDataset(
                    data_dir=self.data_dir, split="train", scale=self.scale,
patch_size=self.patch_size)
                self.val_ds = SRDataset(
                    data_dir=self.data_dir, split="val", scale=self.scale,
patch_size=0)

            if stage == "eval" or stage is None:
                self.test_ds = SRDataset(
                    data_dir=self.data_dir, split="test", scale=self.scale,
patch_size=0)

    def train_dataloader(self) -> torch.utils.data.DataLoader:
        if self.train_ds is None:
            raise ValueError("Training dataset is not initialized")

        return torch.utils.data.DataLoader(
            self.train_ds, batch_size=self.batch_size,
num_workers=self.num_workers, shuffle=True, pin_memory=True
        )

    def val_dataloader(self) -> torch.utils.data.DataLoader:
        if self.val_ds is None:
            raise ValueError("Validation dataset is not initialized")

```

```

        return torch.utils.data.DataLoader(
            self.val_ds, batch_size=1, num_workers=self.num_workers,
            shuffle=False, pin_memory=True
        )

    def test_dataloader(self) -> torch.utils.data.DataLoader:
        if self.test_ds is None:
            raise ValueError("Test dataset is not initialized")

        return torch.utils.data.DataLoader(
            self.test_ds, batch_size=1, num_workers=self.num_workers,
            shuffle=False, pin_memory=True
        )

```

Додаток Б

```

import torch
import torch.nn as nn
import torch.nn.functional as F

class ResidualBlock(nn.Module):
    """
    A simplified residual block.
    Structure: Conv -> ReLU -> Conv -> + -> ReLU
    """

    def __init__(self, channels: int) -> None:
        """
        Initializes the residual block.
        Args:
            channels (int): The number of input and output channels.
        """
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass through the block.

```

```

    Args:
        x (torch.Tensor): The input tensor.
    Returns:
        torch.Tensor: The output tensor.
    """
    residual = x
    out = F.relu(self.conv1(x))
    out = self.conv2(out)
    # Add the input back to the output (the residual connection)
    out += residual
    return F.relu(out)

class RealTimeSRNet(nn.Module):
    """
    An optimized architecture for real-time super-resolution.
    Features:
    - Lightweight residual blocks for feature extraction.
    - Long skip connection to preserve image details.
    - PixelShuffle for efficient upsampling.
    - No BatchNorm layers to avoid validation instability and artifacts.
    """

    def __init__(self, scale: int = 4, nc: int = 64, nb: int = 8) -> None:
        """
        Initializes the model.
        Args:
            scale (int): The upscaling factor (e.g., 2, 3, or 4). Defaults to
4.
            nc (int): The number of channels in the convolutional layers.
Defaults to 64.
            nb (int): The number of residual blocks. Defaults to 8.
        """
        super(RealTimeSRNet, self).__init__()
        self.scale = scale
        self.nc = nc

        # First layer: extracts initial features from the input image
        self.conv_first = nn.Conv2d(3, nc, kernel_size=3, padding=1)

```

```

# Main body of the network: a sequence of residual blocks
self.residual_blocks = nn.Sequential(
    *[ResidualBlock(nc) for _ in range(nb)]
)

# A convolution layer after the residual blocks, before the long skip
connection
self.conv_mid = nn.Conv2d(nc, nc, kernel_size=3, padding=1)

# Upsampling block using PixelShuffle.
self.upscale = nn.Sequential(
    nn.Conv2d(nc, nc * (scale ** 2), kernel_size=3, padding=1),
    nn.PixelShuffle(scale)
)

# Final layer to produce the 3-channel (RGB) output image
self.conv_last = nn.Conv2d(nc, 3, kernel_size=3, padding=1)

# Initialize weights for better training convergence
self.__init_weights()

def __init_weights(self) -> None:
    """
    Initializes model weights using Kaiming Normal initialization.
    """
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            nn.init.kaiming_normal_(
                m.weight, mode='fan_out', nonlinearity='relu')
            if m.bias is not None:
                nn.init.constant_(m.bias, 0)

def forward(self, x: torch.Tensor) -> torch.Tensor:
    """
    Forward pass through the model.
    Args:
        x (torch.Tensor): The low-resolution input tensor.
    Returns:
        torch.Tensor: The super-resolved output tensor.
    """

```

```
# Initial feature extraction
x_first = F.relu(self.conv_first(x))

# Pass through residual blocks
res_out = self.residual_blocks(x_first)
res_out = self.conv_mid(res_out)

# Long skip connection: add initial features to the processed
features
out = res_out + x_first

# Upsample and produce the final image
out = self.upscale(out)
out = self.conv_last(out)

return out
```