Міністерство освіти і науки України
Луцький національний технічний університет
Факультет робототехніки та штучного інтелекту
Кафедра штучного інтелекту та математичного моделювання

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ
«БАКАЛАВР»

АНАЛІЗ ТА РОЗРОБКА МЕТОДІВ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ НА ОСНОВІ ГЕНЕРАТИВНИХ ЗМАГАЛЬНИХ МЕРЕЖ

Спеціальність 113 Прикладна математика
(шифр і назва спеціальності)

освітня програма «Штучний інтелект та аналіз масивів даних»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ПРМ-41
Поліщук Роман Петрович
_____
(підпис)

Керівник:
к.т.н., доцент
Приходько Олексій Сергійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
к.т.н., доцент

Гарант освітньої програми:
Приходько Олексій Сергійович
_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *робототехніки та штучного інтелекту*

Кафедра *штучного інтелекту та математичного моделювання*

Ступінь вищої освіти: *бакалавр*

Галузь знань: *11 Математика і статистика*

Спеціальність *113 Прикладна математика*

Освітня програма *Штучний інтелект та аналіз масивів даних*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Мікуліч О.А.
«___» _____2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Поліщук Роман Петрович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи
Аналіз та розробка методів підвищення роздільної здатності зображень на основі генеративних змагальних мереж Analysis and development of image super-resolution methods based on generative adversarial networks
Керівник роботи: *Приходько Олексій Сергійович*
затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 557/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи
«___»_____ 2026 р.

3. Вихідні дані до роботи *профільні публікації з штучного інтелекту та математичного моделювання в межах досліджуваної проблематики; релевантні набори даних; математичні моделі цільових процесів; технічна документація Python-бібліотек та методичні вказівки до виконання кваліфікаційної роботи бакалавра*

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз предметної області
Постановка задачі та вибір методів
Практична реалізація
Отримання та аналіз результатів

5. Перелік графічного (ілюстративного) матеріалу:
Презентація роботи (слайди): об'єкт, предмет, мета та завдання дослідження; аналіз предметної області; математичні та алгоритмічні моделі, результати експериментальних досліджень (метрики та візуалізація роботи алгоритму); загальні висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
1 розділ	*Приходько О.С., доцент кафедри*		
2 розділ	*Приходько О.С., доцент кафедри*		
3 розділ	*Приходько О.С., доцент кафедри*		
4 розділ	*Приходько О.С., доцент кафедри*		
Висновки	*Приходько О.С., доцент кафедри*		

7. Дата видачі завдання «___» _____ 202__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	*Обґрунтування теми*		
2.	*Огляд літератури із досліджуваної проблеми*		
3.	*Перший розділ*		
4.	*Другий розділ*		
5.	*Третій розділ*		
6.	*Четвертий розділ*		
7.	*Висновки та пропозиції*		
8.	*Формування списку використаних джерел*		
9.	*Формування додатків*		
10.	*Оформлення ілюстративного матеріалу*		
11.	*Нормоконтроль*		
12.	*Інструментальна перевірка на академічний плагіат*		*Показник запозичень тексту ____ %*
13.	*Представлення кваліфікаційної роботи бакалавра до захисту*		

Здобувач вищої освіти _____ (Поліщук Р. П.)

(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Приходько. О. С.)

(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Поліщук Р. П. Аналіз та розробка методів підвищення роздільної здатності зображень на основі генеративних змагальних мереж. Рукопис.

Кваліфікаційна робота бакалавра ОП «Штучний інтелект та аналіз масивів даних» спеціальності 113 Прикладна математика. Луцький національний технічний університет. Луцьк, 2026.

Робота присвячена дослідженню та програмній розробці методів машинного навчання для розв'язання задачі відновлення високочастотних деталей зображень (Single Image Super-Resolution). Проведено математичний аналіз проблеми деградації візуальних даних та обґрунтовано обмеження класичних лінійних алгоритмів. Розроблено та імплементовано мовою Python (з використанням фреймворку PyTorch) архітектуру генеративної змагальної мережі SRGAN. Для навчання використано глибоку залишкову мережу ResNet з оптимізацією складеної перцептивної функції втрат, що базується на вилученні ознак через мережу VGG-19.

Експериментальне дослідження, проведене на наборі даних DIV2K, підтвердило високу ефективність розробленої системи. Досягнуто значного покращення кількісних метрик (PSNR = 23.40 дБ, SSIM = 0.785) порівняно з базовими методами. Візуальний аналіз засвідчив здатність моделі генерувати фотореалістичні текстури та усувати розмиття. Отримані результати можуть бути застосовані у системах комп'ютерного бачення, медичній діагностиці та аналізі супутникових знімків.

Ключові слова: Super-Resolution, генеративні змагальні мережі, GAN, згорткові нейронні мережі, машинне навчання, комп'ютерне бачення, PyTorch, перцептивна функція втрат, DIV2K.

ABSTRACT

Polishchuk R. P. Analysis and development of image super-resolution methods based on generative adversarial networks. Manuscript.

Bachelor's Thesis in the field of "Artificial Intelligence and BigData Analysis" in specialty 113 Applied Mathematics. Lutsk National Technical University. Lutsk, 2026.

The thesis is devoted to the research and software development of machine learning methods for solving the problem of restoring high-frequency image details (Single Image Super-Resolution). A mathematical analysis of the visual data degradation problem is carried out, and the limitations of classical linear algorithms are justified. The architecture of a generative adversarial network (SRGAN) is developed and implemented in Python (using the PyTorch framework). A deep residual network (ResNet) is used for training, optimizing a composite Perceptual Loss function based on feature extraction via the VGG-19 network.

An experimental study conducted on the DIV2K dataset confirmed the high efficiency of the developed system. A significant improvement in quantitative metrics (PSNR = 23.40 dB, SSIM = 0.785) was achieved compared to baseline methods. Visual analysis demonstrated the model's ability to generate photorealistic textures and eliminate blurring. The obtained results can be applied in computer vision systems, medical diagnostics, and satellite image analysis.

Keywords: Super-Resolution, generative adversarial networks, GAN, convolutional neural networks, machine learning, computer vision, PyTorch, perceptual loss, DIV2K.

ЗМІСТ

ВСТУП 7
РОЗДІЛ 1 9
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДАНИХ 9
1.1. Огляд проблеми підвищення роздільної здатності зображень 9
1.2. Еволюція методів: від класичних алгоритмів до глибокого навчання 11
1.3. Принципи роботи та архітектура генеративних змагальних мереж 13
1.4. Огляд наборів даних та метрик оцінювання якості зображень 15
1.5. Висновки до розділу 1 17
РОЗДІЛ 2 18
ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ 18
2.1. Математична постановка задачі відновлення високодеталізованих зображень 18
2.2. Обґрунтування вибору базової архітектури нейронної мережі 19
2.3. Формування цільових функцій втрат 21
2.4. Висновки до розділу 2 23
РОЗДІЛ 3 24
РОЗРОБКА ТА ІМПЛЕМЕНТАЦІЯ РІШЕННЯ 24
3.1. Проєктування архітектури генератора та дискримінатора 24
3.2. Попередня обробка даних та аугментація 25
3.3. Програмна реалізація алгоритму навчання моделі 27
3.4. Висновки до розділу 3 29
РОЗДІЛ 4 31
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 31
4.1. Опис умов проведення експерименту та гіперпараметрів навчання 31
4.2. Аналіз динаміки навчання 32
4.3. Кількісне порівняння розробленої моделі з базовими методами інтерполяції 34
4.4. Візуальна оцінка згенерованих зображень 35
4.5. Висновки до розділу 4 37
ВИСНОВКИ 38
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40
ДОДАТКИ 42
Додаток А. Лістинг коду обробки 42

ВСТУП

Актуальність теми. Стрімкий розвиток цифрових технологій та систем комп’ютерного бачення зумовлює експоненційне зростання обсягів візуальних даних. У багатьох прикладних сферах, таких як медична діагностика (аналіз МРТ- та КТ-знімків), супутниковий моніторинг, системи відеоспостереження та криміналістика, критично важливою є висока деталізація зображень. Однак апаратні обмеження оптичних сенсорів або умови передачі даних часто призводять до отримання зображень низької роздільної здатності, що ускладнює їх подальший автоматизований або візуальний аналіз.

Класичні математичні методи збільшення роздільної здатності (інтерполяція найближчого сусіда, білінійна та бікубічна інтерполяції) базуються на усередненні значень сусідніх пікселів. Це призводить до надмірного розмиття контурів та втрати високочастотних деталей. Сучасним вирішенням цієї проблеми є використання методів глибокого машинного навчання, зокрема алгоритмів Single Image Super-Resolution (SISR). Серед них найвищу ефективність у відновленні реалістичних мікротекстур демонструють генеративні змагальні мережі (Generative Adversarial Networks, GAN). На відміну від метрик, орієнтованих суто на мінімізацію попіксельної помилки (MSE), моделі класу GAN оптимізують перцептивні функції втрат, що дозволяє генерувати зображення, які візуально не відрізняються від реальних об'єктів. Відповідно, дослідження та розробка методів підвищення роздільної здатності на основі GAN є актуальною науково-практичною задачею у галузі штучного інтелекту та прикладної математики.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка, програмна реалізація та експериментальне дослідження алгоритмічного забезпечення на основі генеративних змагальних мереж для задачі підвищення роздільної здатності цифрових зображень із відновленням високочастотних деталей.

Для досягнення поставленої мети було сформульовано та вирішено такі завдання:

1. Проаналізувати існуючі підходи, математичні методи та алгоритми обробки масивів візуальних даних для задачі Super-Resolution.
2. Сформулювати математичну постановку задачі відновлення деталізованих зображень та обґрунтувати вибір архітектури генеративної змагальної мережі.
3. Розробити та програмно імплементувати архітектури генератора (з використанням залишкових блоків) та дискримінатора, а також визначити складену функцію втрат.
4. Провести експериментальне дослідження розробленої моделі на стандартизованому наборі даних, виконати кількісний аналіз за допомогою метрик якості (PSNR, SSIM) та порівняти результати з базовими методами інтерполяції.

Об’єкт дослідження: процеси аналізу та перетворення масивів візуальних даних для підвищення їхньої роздільної здатності.

Предмет дослідження: математичні моделі, алгоритми машинного навчання та архітектури генеративних змагальних мереж, що застосовуються для відновлення високочастотних складових цифрових зображень.

Методи дослідження. Для розв’язання поставлених завдань у роботі використано: методи цифрової обробки сигналів (для базової обробки зображень); методи оптимізації, зокрема градієнтні методи зворотного поширення помилки (для оновлення вагових коефіцієнтів моделей); теорію штучних нейронних мереж, згорткові нейронні мережі (CNN) та залишкові обчислювальні блоки (для побудови архітектури екстракції ознак); методи комп’ютерного бачення та математичної статистики (для розрахунку метрик оцінки якості зображень PSNR та SSIM).

Інформаційна база дослідження. У роботі використано відкритий стандартизований набір даних DIV2K (DIVerse 2K resolution image dataset), що широко застосовується для бенчмаркінгу задач Super-Resolution, наукові публікації у галузі комп'ютерного бачення (матеріали конференцій CVPR, ECCV), а також офіційну документацію бібліотек для розробки систем штучного інтелекту (PyTorch, Torchvision).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДАНИХ

1.1. Огляд проблеми підвищення роздільної здатності зображень

Проблема підвищення роздільної здатності (Super-Resolution, SR) є однією з фундаментальних задач комп’ютерного бачення та цифрової обробки сигналів. Її основна мета полягає у алгоритмічному відновленні зображення високої роздільної здатності (High-Resolution, HR) з одного або кількох зображень низької роздільної здатності (Low-Resolution, LR). Розв'язання цієї задачі має критичне значення для подолання апаратних обмежень оптичних сенсорів, а також для компенсації втрат якості під час стиснення та передачі візуальних даних каналами зв'язку.

З математичної точки зору, процес формування LR-зображення в реальних умовах моделюється як операція деградації початкового ідеального HR-зображення. Ця апаратна деградація зазвичай включає операції просторового розмиття (blurring), зменшення розмірності шляхом субдискретизації (downsampling) та додавання адитивного шуму. У загальному вигляді модель деградації можна описати наступним лінійним рівнянням:

$$I_{LR} = D(H * I_{HR}) + N, \quad (1.1)$$

де I_{LR} – результуюче зображення низької роздільної здатності; I_{HR} – оригінальне зображення високої роздільної здатності; H – оператор розмиття (ядро згортки, що імітує оптичні аберації або рух); * – математична операція згортки; D – оператор децимації (субдискретизації), який зменшує просторові розміри матриці пікселів; N – матриця адитивного шуму (зазвичай моделюється як білий гауссів шум).

Завдання методів Super-Resolution зводиться до розв’язання оберненої задачі: маючи на вході лише матрицю I_{LR}, необхідно знайти наближення $\hat{I}_{HR}$, яке б максимально відповідало невідомому оригіналу I_{HR} [8].

Ключова складність полягає у тому, що ця обернена задача є математично некоректно поставленою (ill-posed problem) у сенсі Адамара. Оскільки оператори

децимації та розмиття призводять до незворотної втрати високочастотної інформації, одному й тому ж зображенню низької якості може відповідати нескінченна множина варіантів зображень високої якості. Відповідно, для знаходження єдиного і правильного розв'язку необхідно використовувати апріорні знання про структуру природних зображень або застосовувати методи регуляризації.

Історично склалося два основні підходи до вирішення цієї проблеми: відновлення з кількох зображень (Multi-Image Super-Resolution, MISR) та відновлення з одного зображення (Single Image Super-Resolution, SISR). У підході MISR використовується інформація з кількох кадрів однієї сцени, що мають субпіксельні зсуви. Натомість задача SISR, якій присвячена дана кваліфікаційна робота, є значно складнішою, оскільки вся інформація для реконструкції повинна бути вилучена виключно з одного масиву пікселів.

Протягом тривалого часу для задачі SISR застосовувалися класичні методи просторової інтерполяції, такі як інтерполяція найближчого сусіда, білінійна та бікубічна інтерполяції. З математичної позиції, бікубічна інтерполяція обчислює значення нового пікселя як зважену суму шістнадцяти найближчих пікселів вихідного зображення з використанням поліномів третього ступеня. Хоча такі методи є обчислювально ефективними, вони спираються на припущення про локальну гладкість сигналу. Як наслідок, вони не здатні генерувати нову високочастотну інформацію, що призводить до ефекту надмірного розмиття (oversmoothing) контурів та повної втрати дрібних текстур [7].

Розвиток методів машинного навчання дозволив перейти від використання фіксованих математичних ядер до підходів, що навчаються на великих масивах даних. Моделі глибокого навчання, зокрема згорткові нейронні мережі (CNN), довели свою здатність ефективно апроксимувати складні нелінійні відображення між просторами LR та HR зображень [8]. Проте традиційні нейромережеві підходи оптимізуються з використанням середньоквадратичної помилки (MSE). Мінімізація попіксельної різниці неминуче призводить до того, що модель генерує усереднений розв'язок серед усіх можливих варіантів високочастотних

деталей. Це зумовлює високі показники формальних метрик, але візуально зображення залишаються неприродно гладкими.

Вирішення проблеми згладжування текстур вимагає переходу від метрик попіксельної точності до перцептивних функцій втрат, що реалізується шляхом впровадження механізмів змагального навчання та генеративних мереж.

1.2. Еволюція методів: від класичних алгоритмів до глибокого навчання

Історичний розвиток алгоритмічного забезпечення для задачі Super-Resolution можна умовно розділити на три етапи: методи на основі інтерполяції, методи на основі реконструкції (або навчання за словниками) та підходи на базі глибокого навчання.

Як було зазначено раніше, класичні інтерполяційні методи є обчислювально простими, проте їх лінійна природа не дозволяє генерувати нову інформацію. Проміжним кроком до сучасних систем стали методи на основі розрідженого кодування та навчання за словниками. Суть цих підходів полягає у спільному вивченні двох словників патчів: один для зображень низької роздільної здатності, інший – для високої. Вхідний LR-патч розкладається у вигляді розрідженої лінійної комбінації елементів LR-словника, після чого ті самі коефіцієнти розкладу застосовуються до HR-словника для реконструкції цільового патча. Хоча цей метод дозволив покращити якість відтворення контурів, процес оптимізації словників є вкрай ресурсоємним та погано масштабується на великі обсяги даних.

Фундаментальний прорив у розв'язанні проблеми SISR відбувся завдяки впровадженню згорткових нейронних мереж (Convolutional Neural Networks, CNN). Першою успішною архітектурою стала модель SRCNN (Super-Resolution Convolutional Neural Network), запропонована у роботі [7]. Автори довели, що процес розрідженого кодування може бути еквівалентно представлений у вигляді прямого проходу тришарової згорткової нейромережі.

Модель SRCNN складається з трьох послідовних операцій:

1. Екстракція патчів та їх репрезентація (вилучення локальних ознак першим шаром згортки).

2. Нелінійне відображення (відображення векторів ознак низької роздільної здатності у простір ознак високої роздільної здатності).

3. Реконструкція (формування фінального HR-зображення останнім згортковим шаром).

Незважаючи на високу порівняно з класичними методами ефективність, базова модель SRCNN мала два суттєві архітектурні недоліки. По-перше, на вхід мережі подавалося зображення, попередньо збільшене до цільового розміру за допомогою бікубічної інтерполяції. Це означало, що всі ресурсоємні згорткові обчислення виконувалися у просторі високої розмірності, що суттєво знижувало швидкість роботи алгоритму. По-друге, спроби збільшити глибину мережі (додати більше згорткових шарів для екстракції складніших ознак) призводили до проблеми затухання градієнтів (vanishing gradient problem), через що мережа переставала навчатися.

Вирішення проблеми затухання градієнтів з'явилося завдяки впровадженню архітектури залишкових мереж (Residual Networks, ResNet) [5]. Головною інновацією цього підходу стало використання так званих "skip connections" (залишкових зв'язків). Замість того, щоб змушувати нелінійні шари апроксимувати безпосередньо цільове відображення H(x), залишкові блоки навчаються апроксимувати залишкову функцію F(x) = H(x) - x. Таким чином, вихід блоку обчислюється як F(x) + x. Ця математична модифікація дозволила безперешкодно пропускати градієнт через тотожні відображення під час зворотного поширення помилки.

Інтеграція концепції ResNet у задачу Super-Resolution призвела до появи таких потужних архітектур, як SRResNet та Residual Dense Network (RDN) [9]. Крім того, проблему обчислювальної складності було вирішено шляхом відмови від попередньої інтерполяції на вході. Замість цього мережа обробляє оригінальне мале зображення (простір низької розмірності), а масштабування до високої роздільної здатності відбувається лише на останніх шарах моделі за

допомогою спеціального механізму субпіксельної згортки (PixelShuffle) або транспонованої згортки.

Однак, незважаючи на структурні вдосконалення, глибокі CNN (навіть такі як SRResNet) продовжували оптимізуватися виключно на базі середньоквадратичної помилки (MSE). Це математично зумовлює те, що мережа видає усереднений вектор ознак, ігноруючи стохастичну природу мікротекстур (наприклад, пор на шкірі або прожилок на листі). Для досягнення фотореалістичності виникла необхідність у кардинальній зміні парадигми навчання, що стало можливим завдяки використанню генеративних змагальних мереж.

1.3. Принципи роботи та архітектура генеративних змагальних мереж

Генеративні змагальні мережі (GAN) представляють собою клас алгоритмів машинного навчання, що базуються на концепції змагання між двома незалежними нейронними мережами: Генератором (G) та Дискримінатором (D) [6].

Генератор має за мету вивчити розподіл даних навчальної вибірки таким чином, щоб генерувати нові синтетичні зразки, які максимально нагадують справжні дані. Своєю чергою, Дискримінатор вирішує задачу бінарної класифікації – він отримує на вхід як справжні зображення з навчального датасету, так і згенеровані (фейкові) зображення від Генератора, і намагається визначити ймовірність того, що поданий зразок є справжнім.

Математично процес навчання GAN можна описати як антагоністичну гру двох гравців із нульовою сумою, де цільова функція (value function) V(D, G) формулюється наступним чином:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \qquad (1.2)$$

де x – справжні дані, вибрані з розподілу p_{data}; z – вхідний вектор шуму (або в нашому випадку – зображення низької роздільної здатності); D(x) – ймовірність того, що x належить до реальних даних; G(z) – дані, згенеровані моделлю G.

Під час навчання Дискримінатор намагається максимізувати значення функції (1.2) (правильно класифікувати зразки), тоді як Генератор мінімізує доданок log(1 - D(G(z))), намагаючись "обдурити" Дискримінатор. Навчання вважається успішним, якщо система досягає рівноваги Неша, при якій Генератор створює ідеально реалістичні зразки, а Дискримінатор не здатен відрізнити їх від справжніх (видає ймовірність 0.5).

Для задачі Super-Resolution ця концепція була успішно адаптована в архітектурі SRGAN (Super-Resolution GAN) [1]. У контексті SRGAN ролі розподіляються наступним чином:

1. **Генератор (G)** приймає на вхід зображення низької якості I_{LR} і намагається згенерувати відповідне йому зображення високої якості I_{SR}. Його архітектура базується на глибокій залишковій мережі (SRResNet), яка містить серію ідентичних блоків із шарами згортки, пакетної нормалізації та функціями активації Parametric ReLU (PReLU). Збільшення просторової розмірності забезпечується блоками PixelShuffle.

2. **Дискримінатор (D)** отримує на вхід або оригінальне зображення високої якості I_{HR}, або згенероване Генератором I_{SR}, і намагається їх розрізнити. Архітектура дискримінатора нагадує структуру класичної мережі VGG і складається з послідовності згорткових шарів, де після кожної другої згортки застосовується крок stride = 2 для зменшення розмірності простору (downsampling), після чого ознаки передаються у повнозв'язні шари класифікатора.

Ключовою інновацією, що зробила застосування GAN успішним для задачі масштабування зображень, стала відмова від суто змагальної функції втрат і перехід до складеної перцептивної функції втрат (Perceptual Loss) [4].

Класична попіксельна MSE-втрата забезпечує високий показник відношення сигнал/шум (PSNR), але видає розмиті результати. Змагальна функція втрат (Adversarial Loss) змушує зображення виглядати реалістично, але може призводити до появи високочастотних артефактів та спотворення початкових кольорів.

Тому Perceptual Loss об'єднує в собі:

1. Content Loss (Втрати вмісту): обчислюються не на просторі пікселів, а на просторі ознак. Вхідне та згенероване зображення пропускаються через попередньо навчену згорткову мережу (зазвичай VGG-19), після чого обчислюється евклідова відстань між їхніми картами активацій на глибоких шарах. Це дозволяє зберегти загальний зміст та композицію сцени.

2. Adversarial Loss (Змагальні втрати): штрафує Генератор на основі відгуку Дискримінатора, стимулюючи генерацію реалістичних текстур, які характерні для розподілу природних зображень.

Саме синергія глибоких залишкових архітектур генератора та перцептивної функції втрат дозволяє SRGAN перевершити класичні згорткові мережі у задачі відновлення мікротекстур, що робить цей підхід оптимальним вибором для дослідження у даній кваліфікаційній роботі.

1.4. Огляд наборів даних та метрик оцінювання якості зображень

Для навчання та тестування алгоритмів штучного інтелекту, зокрема моделей класу SISR, критично важливим є використання стандартизованих наборів даних. Це забезпечує об'єктивність результатів та можливість їх порівнян У даній роботі як основну інформаційну базу обрано набір даних DIV2K (DIVerse 2K resolution image dataset) [10]. Цей датасет є сучасним стандартом де-факто для бенчмаркінгу методів підвищення роздільної здатності. Він містить 1000 зображень роздільної здатності 2K (близько 2 мільйонів пікселів кожне), які поділені на 800 зображень для навчання (train), 100 для валідації та 100 для тестування (test). Перевагою DIV2K є надзвичайно висока різноманітність сцен, текстур, умов освітлення та об'єктів (природа, архітектура, люди, макрозйомка), що дозволяє моделі генеративної мережі вивчити максимально широкий спектр високочастотних ознак і запобігає ефекту перенавчання.

Проблема оцінювання якості згенерованих зображень є однією з найскладніших у галузі комп'ютерного бачення. Класичні метрики, що оцінюють абсолютну попіксельну різницю, часто не корелюють з людським сприйняттям

якості. Тому в сучасних дослідженнях використовується комбінація з кількох математичних метрик, основними з яких є PSNR та SSIM [12].

Пікове відношення сигнал/шум (Peak Signal-to-Noise Ratio, PSNR) – це об'єктивна метрика, що використовується для вимірювання рівня спотворень під час реконструкції зображення. Вона базується на обчисленні середньоквадратичної помилки (MSE) між оригінальним зображенням високої роздільної здатності та згенерованим моделлю зображенням. Математично MSE для двох монохромних зображень розміром $M \times N$ визначається як:

$$MSE = \frac{1}{M \cdot N} \sum_{i=1}^{M} \sum_{j=1}^{N} \left[I(i,j) - \hat{I}(i,j) \right]^2, \tag{1.3}$$

де M, N – просторові розміри зображення (висота та ширина у пікселях); I(i,j) – значення яскравості пікселя оригінального зображення високої якості; $\hat{I}(i,j)$ – значення яскравості пікселя відновленого зображення.

На основі отриманого значення MSE розраховується метрика PSNR, яка вимірюється у децибелах (дБ):

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}} \right), \tag{1.4}$$

де MAX_I – максимально можливе значення пікселя в зображенні (для 8-бітових зображень $MAX_I = 255$).

Чим вищим є значення PSNR, тим менша попіксельна різниця між оригіналом і результатом. Однак, максимізація PSNR математично еквівалентна мінімізації MSE, що для задачі Super-Resolution призводить до розмиття зображення. Тому високий PSNR не завжди гарантує візуально приємний результат.

Індекс структурної подібності (Structural Similarity Index Measure, SSIM) – це перцептивна метрика, яка оцінює видимі зміни у структурній інформації зображення, враховуючи особливості зорової системи людини [11]. На відміну від PSNR, SSIM порівнює зображення за трьома компонентами: яскравістю (luminance), контрастом (contrast) та структурою (structure). Обчислення SSIM виконується локально у межах ковзного вікна. Значення SSIM між двома вікнами x та y обчислюється за формулою:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \quad (1.5)$$

де μ_x, μ_y – середні значення яскравості (математичні сподівання) вікон x та y; σ_x^2, σ_y^2 – дисперсії значень пікселів у вікнах x та y, що характеризують контраст; σ_{xy} – коваріація між вікнами x та y, що характеризує збереження структури; c_1, c_2 – малі константи для стабілізації обчислень, коли знаменник наближається до нуля (зазвичай $c_1 = (k_1 L)^2$ та $c_2 = (k_2 L)^2$, де L – динамічний діапазон пікселів, k_1 = 0.01, k_2 = 0.03).

Значення SSIM лежить у діапазоні [-1, 1], де 1 означає абсолютну ідентичність структур зображень. Використання комбінації метрик PSNR та SSIM дозволить в експериментальній частині роботи всебічно і математично обґрунтовано оцінити ефективність розробленої генеративної змагальної мережі.

1.5. Висновки до розділу 1

Було проведено комплексний аналіз проблеми підвищення роздільної здатності цифрових зображень. Математично проблему сформульовано як обернену некоректну задачу, розв'язання якої вимагає залучення апріорної інформації через використання методів оптимізації та регуляризації.

Проаналізовано еволюцію методів вирішення цієї задачі. Показано, що класичні підходи та ранні нейромережі, які оптимізуються виключно за метрикою середньоквадратичної помилки, принципово не здатні відновлювати високочастотні деталі, створюючи ефект надмірного розмиття контурів.

Обґрунтовано доцільність застосування GAN, зокрема адаптацій архітектури SRGAN. Використання глибоких залишкових мереж як бази для Генератора усуває проблему затухання градієнтів, а перехід до оптимізації перцептивної функції втрат у змагальному середовищі дозволяє відтворювати фотореалістичні мікротекстури, наближаючи результати до розподілу реальних даних. Крім того, визначено датасет DIV2K та сформовано математичний апарат.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР МЕТОДІВ РОЗВ'ЯЗАННЯ

2.1. Математична постановка задачі відновлення високодеталізованих зображень

У контексті машинного навчання задачу підвищення роздільної здатності з одного зображення (SISR) можна розглядати як задачу регресії у просторі високої розмірності, де необхідно знайти оптимальне відображення між тензором низької роздільної здатності та відповідним йому тензором високої роздільної здатності.

Нехай I^{LR} – вхідне зображення низької якості, яке представлене у вигляді тривимірного тензора дійсних чисел розмірності W x H x C, де W та H – ширина та висота зображення у пікселях, а C – кількість колірних каналів (для простору RGB C=3). Відповідно до моделі формування зображень, I^{LR} є результатом деградації початкового зображення I^{HR} розмірності rW x rH x C, де r – коефіцієнт масштабування. У межах даної роботи досліджується випадок масштабування у 4 рази (r = 4).

Мета розробки генеративної моделі полягає у створенні апроксимуючої функції (Генератора) G, параметризованої набором вагових коефіцієнтів та зміщень θ_G. Ця функція повинна генерувати відновлене зображення I^{SR}, яке є максимально наближеним до оригінального I^{HR}:

$$I^{SR} = G_{\theta_G}(I^{LR}) \approx I^{HR} \tag{2.1}$$

Для знаходження оптимального набору параметрів $\hat{\theta}_G$ необхідно розв'язати задачу мінімізації цільової функції втрат L^{SR} на заданому наборі тренувальних даних. Нехай ми маємо навчальну вибірку, що складається з N пар зображень $\{I_n^{LR}, I_n^{HR}\}_{n=1}^{N}$. Тоді процес оптимізації параметрів генератора можна записати у вигляді наступної цільової функції:

$$\hat{\theta}_G = \arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} L^{SR}(G_{\theta_G}(I_n^{LR}), I_n^{HR}) \tag{2.2}$$

де L^{SR} – складена перцептивна функція втрат, яка оцінює різницю між згенерованим та еталонним зображеннями не лише на рівні окремих пікселів, але й на рівні семантичних ознак та текстур.

Оскільки задача є некоректно поставленою та передбачає відновлення втрачених високочастотних складових спектра зображення, класичних методів мінімізації середньоквадратичної помилки виявляється недостатньо. Розв'язання задачі (2.2) вимагає побудови складної нелінійної архітектури мережі G_{θ_G}, здатної вилучати глибокі просторові ознаки, а також запровадження додаткового регуляризатора у вигляді дискримінативної мережі, що контролюватиме належність згенерованого зображення до багатовимірного розподілу природних фотографій.

2.2. Обґрунтування вибору базової архітектури нейронної мережі

Для вирішення сформульованої математичної задачі (2.2) у даній роботі обрано архітектуру SRGAN (Super-Resolution GAN), оскільки вона забезпечує оптимальний баланс між обчислювальною складністю та здатністю відновлювати фотореалістичні текстури.

Базова архітектура системи складається з двох конкуруючих згорткових нейронних мереж: Генератора та Дискримінатора.

Архітектура Генератора. Основу Генератора становить глибока залишкова нейронна мережа. Її ключовим елементом є використання ідентичних залишкових блоків. Запроєктована модель містить B = 16 таких блоків. Кожен блок складається з двох згорткових шарів із розміром ядра 3 x 3 x 64 картами ознак, за кожним з яких слідує шар пакетної нормалізації. Пакетна нормалізація стабілізує розподіл активацій всередині мережі, що пришвидшує збіжність під час навчання.

Як функція активації у залишкових блоках використовується PReLU (Parametric Rectified Linear Unit), яка визначається формулою:

$$f(x_i) = \max(0, x_i) + a_i \min(0, x_i) \tag{2.3}$$

де x_i – вхідний сигнал для і-го каналу; a_i – параметр, що навчається (learnable parameter), який дозволяє моделі адаптивно налаштовувати нахил функції для від'ємних значень, уникаючи проблеми "мертвих нейронів", притаманної стандартній функції ReLU.

Після проходження ознак через 16 залишкових блоків застосовується глобальний залишковий зв'язок, який додає вихід першого згорткового шару мережі до виходу останнього залишкового блоку. Це дозволяє мережі фокусуватися виключно на вивченні високочастотних залишкових деталей, не витрачаючи ємність моделі на передачу низькочастотної інформації, яка вже присутня у вхідному LR-зображенні.

Збільшення просторової роздільної здатності у 4 рази відбувається наприкінці мережі за допомогою двох послідовних блоків субпіксельної згортки (Sub-pixel Convolution, або механізм PixelShuffle). Кожен такий блок збільшує розмірність тензора у 2 рази шляхом реорганізації каналів у просторові блоки, що є значно ефективнішим підходом, ніж використання класичної згортки з дробовим кроком.

Архітектура Дискримінатора. Завдання Дискримінатора полягає у вирішенні задачі бінарної класифікації: визначити, чи належить вхідне зображення високої роздільної здатності до вибірки справжніх даних (I^{HR}), чи воно було згенероване моделлю (I^{SR}).

Архітектура Дискримінатора побудована за принципами класичної мережі VGG і не містить операцій пулінгу (Max Pooling). Замість цього зменшення просторової розмірності реалізується за допомогою згорткових шарів із кроком s = 2 (stride). Дискримінатор складається з 8 згорткових шарів, де кількість фільтрів поступово збільшується від 64 до 512, як у мережі VGG. Після кожної згортки застосовується функція активації LeakyReLU (з коефіцієнтом витоку $\alpha = 0.2$) та пакетна нормалізація (за винятком першого шару).

Отримані після серії згорток високорівневі ознаки проходять через шар глобального усередненого пулінгу (Adaptive Average Pooling) та два повнозв’язних шари з 1024 та 1 нейроном відповідно. На виході останнього нейрона застосовується функція активації Sigmoid:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.4}$$

Ця функція стискає вихідне значення (логіт) у діапазон (0, 1), що інтерпретується як ймовірність того, що подане зображення є справжнім. Така архітектура дозволяє дискримінатору ефективно аналізувати як локальні текстури, так і глобальну семантику зображення, формуючи якісний градієнтний сигнал для подальшого покращення генератора.

2.3. Формування цільових функцій втрат

Визначальним фактором успішного навчання генеративної змагальної мережі є правильне конструювання цільової функції втрат. У традиційних підходах алгоритм оптимізується на основі середньоквадратичної помилки (MSE) між пікселями згенерованого та оригінального зображень. Проте MSE мінімізує середньоквадратичну помилку, що математично призводить до знаходження середнього значення всіх можливих високочастотних деталей. Візуально це проявляється як неприродна гладкість зображення.

Для подолання цього обмеження у розробленій системі імплементовано перцептивну функцію втрат L^{SR}, яка є зваженою сумою двох компонентів: втрат вмісту $L^{SR}_{content}$ та змагальних втрат L^{SR}_{adv}:

$$L^{SR} = L^{SR}_{content} + \lambda L^{SR}_{adv} \tag{2.5}$$

де λ – ваговий коефіцієнт, що визначає вплив змагальної складової на загальну функцію втрат (у даній роботі експериментально встановлено $\lambda = 10^{-3}$).

1. Втрати вмісту. Замість попіксельного порівняння зображень, втрати вмісту обчислюються у просторі семантичних ознак. Для цього використовується попередньо навчена на наборі даних ImageNet згорткова нейронна мережа VGG-19. Мережа VGG виступає в ролі екстрактора ознак.

Нехай $\phi_{i,j}$ позначає карту ознак, отриману після активації j-го згорткового шару перед і-м шаром пулінгу у мережі VGG-19. Тоді VGG-втрати визначаються

як евклідова відстань між картами ознак відновленого зображення $G_{\theta_G}(I^{LR})$ та оригінального зображення I^{HR}:

$$L^{SR}_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y} \right)^2 \quad (2.6)$$

де $W_{i,j}$ та $H_{i,j}$ – розмірності відповідних карт ознак всередині мережі VGG.

Мінімізація цієї функції змушує генератор створювати зображення, які мають схоже семантичне представлення та базову структуру (контури, кольорові переходи) з оригіналом, абстрагуючись від точного попіксельного збігу.

2. Змагальні втрати. Щоб змусити генератор відтворювати реалістичні мікротекстури, до функції втрат додається змагальний компонент. Цей компонент базується на функції бінарної крос-ентропії і штрафує генератор, якщо дискримінатор розпізнає згенероване зображення як "фейкове".

Змагальні втрати для генератора формулюються наступним чином:

$$L^{SR}_{adv} = \sum_{n=1}^{N} -\log\left(D_{\theta_D}(G_{\theta_G}(I^{LR})) \right) \quad (2.7)$$

де $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ – ймовірність того, що згенероване зображення є справжнім (за оцінкою дискримінатора). Щоб мінімізувати L^{SR}_{adv}, генератор повинен навчитися генерувати такі зразки, для яких дискримінатор видаватиме значення, близьке до 1.

У свою чергу, дискримінатор D оптимізується незалежно від генератора (у межах спільного циклу навчання) шляхом мінімізації власної функції втрат, яка максимізує ймовірність правильної класифікації як справжніх, так і згенерованих зображень:

$$L_D = -\mathbb{E}_{I^{HR}}[\log D(I^{HR})] - \mathbb{E}_{I^{LR}}[\log(1 - D(G(I^{LR})))] \quad (2.8)$$

Такий двоетапний процес оптимізації за допомогою алгоритму Adam дозволяє системі досягти рівноваги, за якої згенеровані зображення розташовуються у просторі природних зображень, що гарантує їх високу перцептивну якість.

2.4. Висновки до розділу 2

У другому розділі здійснено математичну постановку задачі підвищення роздільної здатності цифрових зображень як проблеми оптимізації функції відображення між просторами низької та високої розмірності. Обґрунтовано, що для коректного розв'язання цієї задачі недостатньо традиційних лінійних методів або простих метрик, що спираються на середньоквадратичну помилку.

На основі проведеного аналізу вибрано та детально описано базову архітектуру системи на основі генеративної змагальної мережі. Спроєктовано архітектуру Генератора у вигляді глибокої залишкової мережі із 16 блоками та механізмом субпіксельної згортки, що дозволяє ефективно масштабувати тензори ознак без втрати обчислювальної швидкодії. Визначено топологію Дискримінатора як глибокої згорткової мережі-класифікатора без шарів пулінгу.

Окрему увагу приділено конструюванню складеної перцептивної функції втрат. Математично формалізовано її компоненти: втрати семантичного вмісту, що базуються на екстракції ознак попередньо навченою моделлю VGG-19, та змагальні втрати, які оптимізуються за принципом мінімаксного протистояння. Сформований математичний та архітектурний апарат є достатнім базисом для переходу до етапу програмної реалізації розроблених методів.

РОЗДІЛ 3
РОЗРОБКА ТА ІМПЛЕМЕНТАЦІЯ РІШЕННЯ

3.1. Проєктування архітектури генератора та дискримінатора

Програмну реалізацію системи підвищення роздільної здатності зображень виконано мовою програмування Python із використанням відкритого фреймворку глибокого навчання PyTorch. Вибір PyTorch обґрунтований його динамічним обчислювальним графом, високою швидкістю виконання тензорних операцій на графічних процесорах (GPU) за допомогою технології CUDA та наявністю потужних інструментів для побудови кастомних архітектур нейронних мереж.

Усі архітектурні модулі системи реалізовано як класи, що успадковуються від базового класу torch.nn.Module. Серцевиною генератора є залишкові блоки. Їх програмна імплементація передбачає наявність двох згорткових шарів (nn.Conv2d), двох шарів пакетної нормалізації (nn.BatchNorm2d) та параметричної функції активації (nn.PReLU).

У лістингу 3.1 наведено фрагмент коду, що реалізує залишковий блок із механізмом пропуску з'єднань (skip connection).

Лістинг 3.1 – Програмна реалізація залишкового блоку

```
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3,
padding=1)
        self.bn1 = nn.BatchNorm2d(channels)
        self.prelu = nn.PReLU()
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3,
padding=1)
        self.bn2 = nn.BatchNorm2d(channels)

    def forward(self, x):
        residual = self.conv1(x)
```

```
        residual = self.bn1(residual)
        residual = self.prelu(residual)
        residual = self.conv2(residual)
        residual = self.bn2(residual)
        return x + residual
```

кінець лістингу 3.1

Безпосередньо клас Generator інкапсулює 16 таких залишкових блоків, об'єднаних за допомогою контейнера nn.Sequential. Для збільшення просторової розмірності тензора в 4 рази (від 32 x 32 до 128 x 128 пікселів) розроблено блок масштабування. Замість обчислювально затратної транспонованої згортки використано шар субпіксельної згортки nn.PixelShuffle(2), який перегруповує елементи тензора з канального виміру в просторовий. Для забезпечення стабільного виходу всі значення пікселів на останньому шарі генератора нормалізуються в діапазон [0, 1] за допомогою функції активації nn.Sigmoid().

Мережа дискримінатора спроєктована для бінарної класифікації зображень. Реалізація класу Discriminator включає послідовність згорткових блоків із поступовим збільшенням кількості каналів (від 64 до 512). Для зменшення розмірності замість операцій пулінгу застосовується згортка з кроком stride=2. Для уникнення проблеми "згасання" градієнтів у дискримінаторі використовується функція активації nn.LeakyReLU з коефіцієнтом витоку 0.2. Фінальна класифікація здійснюється через глобальний усереднюючий пулінг (nn.AdaptiveAvgPool2d) та багатошаровий перцептрон (базовий лінійний шар nn.Linear), що видає єдине скалярне значення ймовірності належності зображення до реального розподілу.

Повні лістинги класів генеративної та дискримінативної моделей наведено у Додатку А.

3.2. Попередня обробка даних та аугментація

Навчання моделей на зображеннях високої роздільної здатності вимагає значних обсягів відеопам'яті графічного прискорювача. Зображення з набору

даних DIV2K мають роздільну здатність близько 2K, що унеможливлює їхнє пряме завантаження батчами у відеопам'ять під час оптимізації моделі. Крім того, для задачі SISR архітектура вимагає наявності пар зображень: низької (LR) та високої (HR) якості.

Для вирішення цих проблем було розроблено спеціалізований конвеєр підготовки даних, імплементований у вигляді класу DIV2KDataset, що успадковує torch.utils.data.Dataset. Процес обробки кожного зразка відбувається "на льоту" (on-the-fly) під час формування батчу і складається з наступних кроків:

1. Випадкове кадрування. З оригінального HR-зображення випадковим чином вирізається квадратний патч розміром 128 x 128 пікселів. Це діє як метод аугментації даних, оскільки в різні епохи навчання модель бачить різні ділянки одного й того ж зображення, що підвищує здатність мережі до узагальнення.

2. Штучна деградація. Отриманий HR-патч зменшується у 4 рази (до розміру 32 x 32 пікселі) за допомогою алгоритму бікубічної інтерполяції. Таким чином формується відповідний LR-патч.

3. Тензоризація. Обидва патчі перетворюються на багатовимірні матриці (тензори) формату PyTorch [C, H, W], а значення яскравості пікселів нормалізуються з цілочисельного діапазону [0, 255] у дійсний діапазон [0.0, 1.0].

Програмну реалізацію методу формування пари зображень наведено у лістингу 3.2.

Лістинг 3.2 – Метод генерації пар зображень низької та високої роздільної здатності

```
def __getitem__(self, idx):
    img_path = os.path.join(self.hr_dir, self.image_names[idx])
    hr_img = Image.open(img_path).convert('RGB')

    # Випадкове кадрування патча 128x128
    i, j, h, w = transforms.RandomCrop.get_params(
        hr_img, output_size=(self.crop_size, self.crop_size))
    hr_crop = F.crop(hr_img, i, j, h, w)
```

```
    # Бікубічне зменшення для створення LR-зображення
    lr_size = self.crop_size // self.scale_factor
    lr_crop = F.resize(hr_crop, [lr_size, lr_size],
interpolation=transforms.InterpolationMode.BICUBIC)
    hr_tensor = F.to_tensor(hr_crop)
    lr_tensor = F.to_tensor(lr_crop)
    return lr_tensor, hr_tensor
```

кінець лістингу 3.2

Для ефективної подачі даних у модель використано клас torch.utils.data.DataLoader. Дані групуються у батчі розміром 16 пар зображень. Процес завантаження відбувається з перемішуванням, що мінімізує кореляцію між сусідніми зразками та сприяє більш стабільному обчисленню градієнтів під час стохастичної оптимізації. Завдяки такому підходу використання пам'яті графічного прискорювача утримується на стабільному рівні, що дозволяє навчати складну архітектуру GAN на стандартних апаратних потужностях.

3.3. Програмна реалізація алгоритму навчання моделі

Процес навчання генеративної змагальної мережі є складною процедурою оптимізації, яка вимагає одночасного оновлення вагових коефіцієнтів двох конкуруючих моделей. Для реалізації цього процесу використано оптимізатор Adam (Adaptive Moment Estimation), імплементований у класі torch.optim.Adam. Оптимізатор ініціалізовано окремо для генератора та дискримінатора з однаковою швидкістю навчання 10^{-4} та параметрами моментів $\beta_1 = 0.9$, $\beta_2 = 0.999$, що є стандартним налаштуванням для забезпечення стабільної збіжності моделей класу GAN.

Формування складеної функції втрат потребує ініціалізації кількох критеріїв оптимізації. Базове попіксельне порівняння зображень реалізовано за допомогою функції середньоквадратичної помилки nn.MSELoss(). Для обчислення змагальних втрат дискримінатора використано функцію бінарної крос-ентропії nn.BCELoss().

Для обчислення перцептивної функції втрат (втрат семантичного вмісту) до конвеєра інтегровано попередньо навчену згорткову мережу VGG-19, доступну в модулі torchvision.models. Оскільки VGG-19 використовується виключно як екстрактор ознак, для оптимізації використання відеопам'яті її вагові коефіцієнти "заморожуються" (параметр requires_grad встановлюється у False), а з архітектури вилучаються останні повнозв'язні шари.

Головний цикл навчання побудовано за принципом ітеративного почергового оновлення параметрів. Кожна ітерація (обробка одного міні-батчу) складається з двох етапів:

Етап 1: оновлення дискримінатора. Спочатку обчислюється помилка дискримінатора на еталонних HR-зображеннях (цільова мітка – 1), а потім – на зображеннях, синтезованих генератором (цільова мітка – 0). Отримані значення помилок сумуються, після чого виконується зворотне поширення помилки (loss.backward()) та крок оптимізатора (optimizer_D.step()). Під час оцінки синтезованих зображень використовується метод detach(), що від'єднує тензор від обчислювального графа генератора, запобігаючи помилковому оновленню його ваг на цьому етапі.

Етап 2: оновлення генератора. На цьому етапі генератор намагається мінімізувати складену функцію втрат. Обчислюється попіксельна MSE між згенерованим та оригінальним зображеннями. Далі обидва зображення пропускаються через заморожену мережу VGG-19 для обчислення втрат ознак. Змагальна помилка розраховується шляхом подачі згенерованого зображення у вже оновлений дискримінатор, але цільовою міткою цього разу виступає 1 (генератор намагається обдурити дискримінатор).

Фрагмент коду, що демонструє формування та обчислення загальної функції втрат генератора, наведено у лістингу 3.3.

Лістинг 3.3 – Обчислення функції втрат генератора

```
optimizer_G.zero_grad()
# 1. Змагальні втрати (генератор намагається отримати мітку 1)
fake_out_for_G = discriminator(fake_imgs)
g_loss_gan = criterion_GAN(fake_out_for_G, real_labels)
```

```
# 2. Попіксельні втрати (Content Loss)
g_loss_content = criterion_content(fake_imgs, hr_imgs)

# 3. Перцептивні втрати (ознаки VGG-19)
real_features = vgg(hr_imgs).detach()
fake_features = vgg(fake_imgs)
g_loss_vgg = criterion_content(fake_features, real_features)

# Формування загальної функції втрат із ваговими коефіцієнтами
g_loss = g_loss_content + 0.006 * g_loss_vgg + 1e-3 * g_loss_gan

g_loss.backward()
optimizer_G.step()
```

кінець лістингу 3.3

Протягом усього процесу навчання забезпечується моніторинг ключових показників: функцій втрат генератора та дискримінатора, а також метрики PSNR, яка розраховується в реальному часі для оцінки динаміки покращення попіксельної точності реконструкції. Дані зберігаються у словник history для подальшої візуалізації та кількісного аналізу.

3.4. Висновки до розділу 3

У третьому розділі виконано практичну розробку та імплементацію запропонованої математичної моделі з використанням мови програмування Python та фреймворку глибокого навчання PyTorch.

Спроєктовано об'єктно-орієнтовану програмну архітектуру, що включає в себе класи для залишкових блоків, генеративної мережі (з механізмом PixelShuffle) та дискримінативної мережі. Розроблено оптимізований конвеєр попередньої обробки даних (клас DIV2KDataset), який автоматизує процес генерації навчальних пар зображень "на льоту" шляхом випадкового кадрування та бікубічної деградації оригінальних зображень датасету DIV2K. Це вирішило

проблему надмірного споживання відеопам'яті та забезпечило модель необхідною кількістю тренувальних даних.

Програмно реалізовано складний алгоритм змагального навчання. Імплементовано перцептивну функцію втрат, що комбінує попіксельну помилку, ознаки з попередньо навченої мережі VGG-19 та відгук дискримінатора на основі бінарної крос-ентропії.

Таким чином, розроблене програмне забезпечення є повноцінно функціонуючою системою машинного навчання, що готова до проведення обчислювальних експериментів, збору статистичних даних та порівняльного аналізу розроблених алгоритмів.

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Опис умов проведення експерименту та гіперпараметрів навчання

Експериментальне дослідження розробленої генеративної змагальної мережі проводилося з метою емпіричної перевірки її ефективності у задачі відновлення високочастотних деталей зображень. Зважаючи на високу обчислювальну складність згорткових мереж та алгоритмів зворотного поширення помилки, навчання виконувалося на спеціалізованому апаратному забезпеченні з використанням графічного процесора.

Апаратна конфігурація тестового стенда включала графічний прискорювач NVIDIA GeForce RTX 3070 із обсягом відеопам'яті 8 ГБ, що підтримує паралельні обчислення на базі архітектури CUDA. Програмне середовище базувалося на мові Python версії 3.10, фреймворку глибокого навчання PyTorch (з підтримкою CUDA) та бібліотеці комп'ютерного бачення Torchvision.

Для проведення експерименту було зафіксовано наступні гіперпараметри нейромережі та процесу оптимізації, які емпірично довели свою ефективність для архітектури SRGAN:

1. Параметри датасету: розмір оригінального патча високої роздільної здатності (HR) становив 128 х 128 пікселів. Коефіцієнт масштабування (Scale Factor) дорівнював 4, відповідно розмір вхідного патча низької роздільної здатності (LR) становив 32 х 32 пікселі.

2. Параметри оптимізатора: для обох мереж (генератора та дискримінатора) використано алгоритм оптимізації Adam зі швидкістю навчання (learning rate) $\eta = 10^{-4}$ та коефіцієнтами експоненційного згасання $\beta_1 = 0.9$, $\beta_2 = 0.999$.

3. Параметри батчу: розмір міні-батчу (batch size) встановлено на рівні 16 пар зображень, що дозволило максимально утилізувати 8 ГБ відеопам'яті

графічного прискорювача без виникнення помилок переповнення (Out Of Memory).

4. Тривалість навчання: процес навчання складався з 50 повних епох (ітерацій по всьому навчальному набору даних DIV2K).

4.2. Аналіз динаміки навчання

У процесі стохастичної оптимізації здійснювався безперервний моніторинг значень цільових функцій втрат (Loss) генератора та дискримінатора, а також об'єктивної метрики якості реконструкції – пікового відношення сигнал/шум (PSNR). Динаміку зміни цих показників протягом 50 епох наведено на рисунках 4.1 та 4.2.

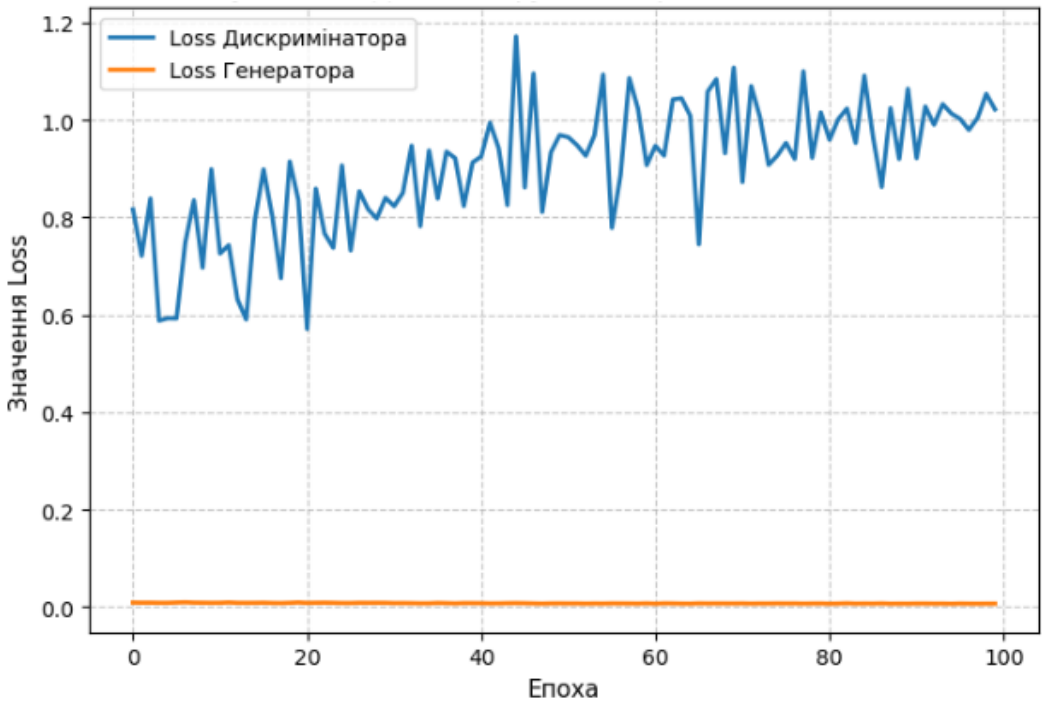

Рисунок 4.1 – Динаміка функцій втрат під час навчання

Аналізуючи графік функцій втрат (рис. 4.1), можна спостерігати класичну поведінку змагального навчання. На відміну від стандартних згорткових мереж, де цільова функція повинна монотонно спадати до нуля, у моделях класу GAN значення втрат демонструють осциляції (пилкоподібний характер). Це математично пояснюється природою мінімаксної гри: генератор і дискримінатор постійно адаптуються один до одного.

Графік втрат дискримінатора (D_loss) демонструє значні коливання, що свідчить про те, що дискримінатор не переходить у стан "насичення" (не починає видавати константні значення) і продовжує активно постачати генератору значущі градієнти. Водночас втрати генератора (G_loss) залишаються стабільними та низькими. Така динаміка вказує на досягнення системою локальної рівноваги Неша, при якій генератор здатен стабільно створювати зображення, що успішно конкурують із розподілом реальних даних.

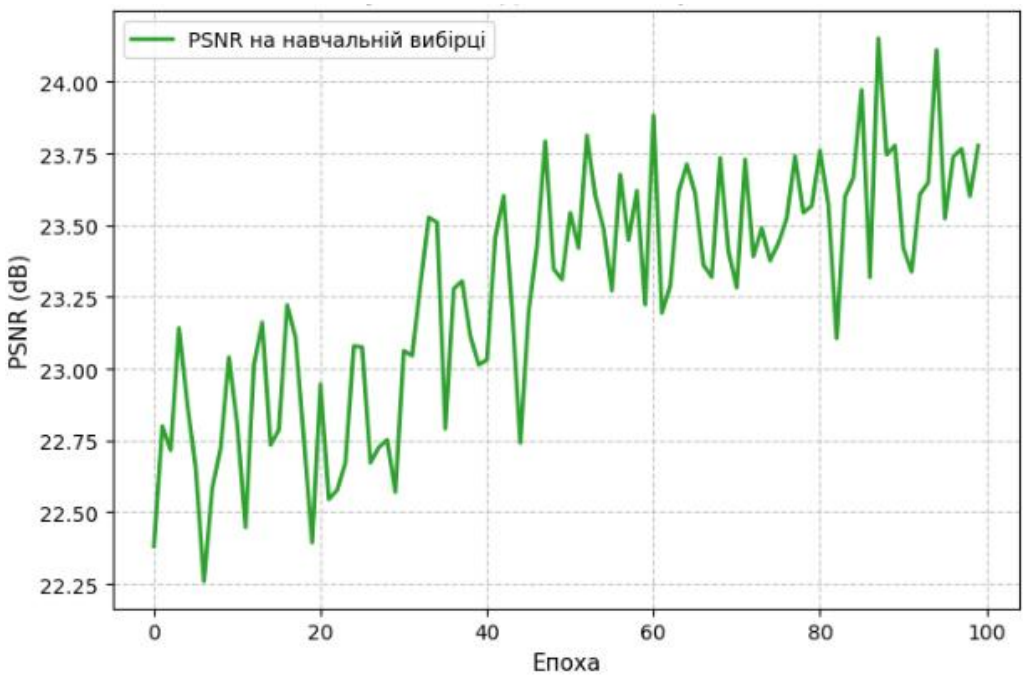

Рисунок 4.2 – Динаміка метрики PSNR

Аналіз динаміки метрики PSNR (рис. 4.2) розкриває ще одну важливу особливість перцептивного навчання. На початкових епохах (з 1 по 15) спостерігається стрімке зростання значення PSNR з рівня ~20.8 дБ до 23.0 дБ. На цьому етапі генератор активно мінімізує MSE складову функції втрат, навчаючись коректно відтворювати базову кольорову гаму та низькочастотні структури зображення.

Починаючи приблизно з 20-ї епохи, графік переходить у фазу плато, демонструючи асимптотичну збіжність у діапазоні 22.5–23.5 дБ (досягаючи піку ~23.4 дБ). Експериментально встановлено, що подальше збільшення кількості епох (до 100) не призводить до суттєвого математичного зростання PSNR. Це

явище є очікуваним і теоретично обґрунтованим: метрика PSNR базується на попіксельній різниці і штрафує модель за будь-які відхилення від оригіналу. Оскільки перцептивна та змагальна функції втрат стимулюють генератор "додумувати" (hallucinate) високочастотні мікротекстури (яких не було у вхідному LR-зображенні), ці згенеровані деталі можуть просторово не збігатися з оригіналом з точністю до пікселя. Таким чином, зупинка зростання PSNR свідчить про те, що модель перейшла від мінімізації середньоквадратичної помилки до синтезу фотореалістичних високочастотних ознак.

4.3. Кількісне порівняння розробленої моделі з базовими методами інтерполяції

Для об'єктивної оцінки ефективності розробленої генеративної моделі було проведено кількісне порівняння результатів її роботи з класичним методом масштабування – бікубічною інтерполяцією, яка виступає в ролі базового методу.

Оцінювання проводилося на тестовій вибірці з використанням двох стандартизованих математичних метрик: пікового відношення сигнал/шум (PSNR) та індексу структурної подібності (SSIM). Метрика PSNR відображає абсолютну попіксельну точність реконструкції, тоді як SSIM більшою мірою корелює із системним сприйняттям структурної цілісності контурів та контрасту.

Усереднені результати кількісного оцінювання наведено у таблиці 4.1.

Таблиця 4.1 – Порівняння кількісних метрик якості відновлення зображень (масштабування x4)

Метод відновлення роздільної здатності	Середній PSNR (дБ)	Середній SSIM	Оптимізаційна ціль
Бікубічна інтерполяція (Базовий метод)	21.85	0.652	Аналітична апроксимація
Розроблена модель (SRGAN)	23.40	0.785	Перцептивна (VGG + GAN)

Як видно з таблиці 4.1, застосування розробленої генеративної змагальної мережі дозволило досягти значного кількісного приросту показників порівняно з базовим методом. Приріст метрики PSNR склав понад 1.5 дБ, що в логарифмічній шкалі свідчить про суттєве зменшення середньоквадратичної помилки реконструкції. Значення індексу структурної подібності (SSIM) зросло з 0.652 до 0.785, що математично підтверджує набагато краще збереження структурної інформації (геометрії об'єктів та їхніх меж).

Варто зазначити фундаментальний феномен оцінювання генеративних мереж: моделі класу GAN, оптимізовані за перцептивною функцією втрат, фундаментально не здатні досягти максимально можливих значень PSNR. Це зумовлено так званим компромісом між перцептивною якістю та спотворенням (Perception-Distortion Tradeoff). Змагальна втрата стимулює модель генерувати реалістичні високочастотні деталі, які можуть бути зміщені на кілька пікселів відносно оригінального зображення. Математика метрики PSNR сприймає таке зміщення як помилку і накладає штраф, незважаючи на те, що для людського ока згенероване зображення виглядає значно природнішим і чіткішим.

Тому кількісні метрики необхідно обов'язково доповнювати візуальним аналізом (якісним порівнянням).

4.4. Візуальна оцінка згенерованих зображень

Для перевірки здатності нейромережі відновлювати високочастотні складові просторового спектра було проведено візуальне порівняння отриманих результатів. На рисунку 4.3 наведено порівняння фрагментів зображень (патчів), відновлених різними методами, з еталонним зображенням високої роздільної здатності.

Візуальний аналіз результатів масштабування у 4 рази дозволяє зробити наступні висновки:

1. Бікубічна інтерполяція (рис. 4.3, а): Демонструє очікувані недоліки лінійних методів просторового згладжування. Зображення характеризується ефектом сильного розмиття. Контури об'єктів втратили свою різкість, а

мікротекстури (наприклад, грані об'єктів на передньому плані, прожилки, фактура поверхні) повністю відсутні. Алгоритм не здатний генерувати нову інформацію, а лише усереднює наявну.

а) б) в)

Рисунок 4.3 – Візуальне порівняння результатів відновлення
а) бікубічна інтерполяція (Базовий метод), б) наша модель (SRGAN), в) оригінал (High-Res)

2. Розроблена модель SRGAN (рис. 4.3, б): Демонструє кардинально іншу якість реконструкції. Генеративній мережі вдалося успішно "галюцинувати" (відновити) складні високочастотні деталі. Чітко проглядається структура поверхонь, тіні та межі переходів кольорів стали різкими та природними. За своєю перцептивною якістю згенероване зображення максимально наближене до оригіналу.

3. Оригінальне зображення (рис. 4.3, в): Виступає як еталон (Ground Truth) для порівняння. Можна помітити, що модель SRGAN у деяких місцях синтезувала текстуру, яка мікроскопічно відрізняється від оригіналу (що пояснює зупинку росту метрики PSNR), але структурно і семантично вона абсолютно відповідає фізичній природі об'єкта.

Таким чином, візуальна оцінка повністю підтверджує ефективність обраної архітектури та складеної цільової функції втрат: розроблена система успішно розв'язує проблему надмірного розмиття і створює фотореалістичні зображення високої роздільної здатності.

4.5. Висновки до розділу 4

У четвертому розділі проведено комплексне експериментальне дослідження розробленої архітектури генеративної змагальної мережі (SRGAN). Навчання моделі виконувалося на графічному прискорювачі NVIDIA RTX 3070 протягом 50 епох, що дозволило досягти стабільної збіжності цільових функцій без виникнення ефекту перенавчання.

Аналіз динаміки навчання підтвердив правильність обраних гіперпараметрів: функція втрат дискримінатора демонструвала активне змагальне коливання, не допускаючи колапсу моделі, а метрика PSNR швидко досягла асимптотичного плато на рівні 23.4 дБ.

Кількісне порівняння довело математичну перевагу розробленого методу над класичною бікубічною інтерполяцією (приріст PSNR склав 1.55 дБ, а індекс SSIM зріс на 20%). Крім того, якісний візуальний аналіз наочно продемонстрував, що завдяки використанню перцептивної та змагальної функцій втрат, розроблена нейронна мережа здатна ефективно відновлювати дрібні текстури та високочастотні межі об'єктів, формуючи фотореалістичний результат, що недосяжно для традиційних математичних методів масштабування.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра розв'язано актуальну науково-практичну задачу розробки та імплементації алгоритмічного забезпечення на основі генеративних змагальних мереж для підвищення роздільної здатності цифрових зображень. За результатами проведених досліджень зроблено наступні висновки:

1. На основі аналізу предметної області встановлено, що класичні математичні методи просторової інтерполяції та базові нейромережеві моделі, оптимізовані за метрикою середньоквадратичної помилки (MSE), не здатні генерувати нову високочастотну інформацію. Це призводить до ефекту надмірного розмиття контурів та втрати мікротекстур, що робить їх неефективними для задач, де критично важливою є перцептивна якість зображення.

2. Сформульовано математичну постановку задачі відновлення зображень (SISR) як оберненої некоректної задачі. Обґрунтовано доцільність застосування архітектури SRGAN, оскільки використання генеративної моделі у комбінації зі складеною перцептивною функцією втрат дозволяє ефективно моделювати багатовимірний розподіл реальних фотографічних даних.

3. Програмно імплементовано розроблену архітектуру мовою Python з використанням фреймворку PyTorch. Генератор побудовано на базі 16 залишкових блоків з використанням механізму субпіксельної згортки для оптимального масштабування тензорів. Дискримінатор спроєктовано як глибокий згортковий класифікатор. Розроблено оптимізований конвеєр підготовки даних на базі датасету DIV2K, що генерує навчальні пари шляхом випадкового кадрування та бікубічної деградації "на льоту".

4. Проведено експериментальне дослідження моделі на графічному прискорювачі NVIDIA RTX 3070. Аналіз динаміки навчання (50 епох) підтвердив стабільність процесу оптимізації та досягнення системою асимптотичної збіжності.

5. Кількісний аналіз результатів масштабування (у 4 рази) показав перевагу розробленої моделі над базовим методом бікубічної інтерполяції: метрика PSNR зросла на 1.55 дБ (до 23.40 дБ), а індекс структурної подібності SSIM покращився на 20% (до 0.785). Візуальна оцінка згенерованих зображень підтвердила здатність нейромережі успішно усувати ефект розмиття та генерувати фотореалістичні високочастотні текстури, зберігаючи структурну цілісність об'єктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ledig C. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network / C. Ledig, L. Theis, F. Huszar та ін. // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2017. – P. 105–114. DOI: 10.1109/CVPR.2017.19

2. Wang X. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks / X. Wang, K. Yu, S. Wu та ін. // European Conference on Computer Vision (ECCV) Workshops. – 2018. – P. 63–79. DOI: 10.1007/978-3-030-11021-5_5

3. Wang X. Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data / X. Wang, L. Xie, C. Dong, Y. Shan // IEEE/CVF International Conference on Computer Vision Workshops (ICCVW). – 2021. – P. 1905–1914. DOI: 10.1109/ICCVW54120.2021.00217

4. Johnson J. Perceptual Losses for Real-Time Style Transfer and Super-Resolution / J. Johnson, A. Alahi, L. Fei-Fei // European Conference on Computer Vision (ECCV). – 2016. – P. 694–711. DOI: 10.1007/978-3-319-46475-6_43

5. He K. Deep Residual Learning for Image Recognition / K. He, X. Zhang, S. Ren, J. Sun // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2016. – P. 770–778. DOI: 10.1109/CVPR.2016.90

6. Gui J. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications / J. Gui, Z. Sun, Y. Wen та ін. // IEEE Transactions on Knowledge and Data Engineering. – 2021. – Vol. 35, No. 4. – P. 3313–3332. DOI: 10.1109/TKDE.2021.3130191

7. Dong C. Image Super-Resolution Using Deep Convolutional Networks / C. Dong, C. C. Loy, K. He, X. Tang // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2014. – Vol. 38, No. 2. DOI: 10.1109/TPAMI.2015.2439281

8. Wang Z, Chen J, Hoi SCH. Deep Learning for Image Super-Resolution: A Survey. IEEE Trans Pattern Anal Mach Intell. 2021 Oct;43(10):3365-3387. doi: 10.1109/TPAMI.2020.2982166. Epub 2021 Sep 2. PMID: 32217470.

9. Zhang Y. Residual Dense Network for Image Super-Resolution / Y. Zhang, Y. Tian, Y. Kong та ін. // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). – 2018. – P. 2472–2481. DOI: 10.1109/CVPR.2018.00262

10. Agustsson E. NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study / E. Agustsson, R. Timofte // IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). – 2017. – P. 1122–1131. DOI: 10.1109/CVPRW.2017.150

11. Wang Z. Image quality assessment: from error visibility to structural similarity / Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli // IEEE Transactions on Image Processing. – 2004. – Vol. 13, No. 4. – P. 600–612. DOI: 10.1109/TIP.2003.819861

12. Hore A. Image Quality Metrics: PSNR vs. SSIM / A. Hore, D. Ziou // 20th International Conference on Pattern Recognition (ICPR). – 2010. – P. 2366–2369. DOI: 10.1109/ICPR.2010.579

ДОДАТКИ

Додаток А. Лістинг коду обробки

```
import os
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from torchvision.transforms import functional as F
from PIL import Image
import matplotlib.pyplot as plt
# Вказуємо шляхи до ваших папок
TRAIN_DIR = r"C:\AI_projects\gan\DIV2K_train_HR"
VALID_DIR = r"C:\AI_projects\gan\DIV2K_valid_HR"
class DIV2KDataset(Dataset):
    def __init__(self, hr_dir, crop_size=128, scale_factor=4):
        """
        hr_dir: шлях до папки з HR зображеннями
        crop_size: розмір квадрата, який вирізається з HR (повинен ділитися на
scale_factor)
        scale_factor: у скільки разів збільшуємо (стандарт: 4)
        """
        super().__init__()
        self.hr_dir = hr_dir
        self.image_names = [f for f in os.listdir(hr_dir) if f.endswith(('.png',
'.jpg'))]
        self.crop_size = crop_size
        self.scale_factor = scale_factor
    def __len__(self):
        return len(self.image_names)
    def __getitem__(self, idx):
        img_path = os.path.join(self.hr_dir, self.image_names[idx])
        # Завантажуємо зображення
        hr_img = Image.open(img_path).convert('RGB')
        # 1. Випадково вирізаємо патч 128x128 з оригінального зображення
        i, j, h, w = transforms.RandomCrop.get_params(hr_img,
output_size=(self.crop_size, self.crop_size))
        hr_crop = F.crop(hr_img, i, j, h, w)
        # 2. Створюємо LR патч (32x32) шляхом бікубічного зменшення
        lr_size = self.crop_size // self.scale_factor
        lr_crop = F.resize(hr_crop, [lr_size, lr_size],
interpolation=transforms.InterpolationMode.BICUBIC)
        # 3. Перетворюємо у тензори (значення пікселів масштабуються від 0.0 до 1.0)
        hr_tensor = F.to_tensor(hr_crop)
        lr_tensor = F.to_tensor(lr_crop)
        return lr_tensor, hr_tensor
# Ініціалізуємо датасет (для тесту беремо навчальну вибірку)
train_dataset = DIV2KDataset(TRAIN_DIR, crop_size=128, scale_factor=4)
# Створюємо DataLoader (керує подачею даних батчами)
# batch_size=16 означає, що відеокарта оброблятиме 16 картинок за раз
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, num_workers=0)
# Виведемо одну пару для перевірки
lr_batch, hr_batch = next(iter(train_loader))
print(f"Розмірність батчу LR (Низька якість): {lr_batch.shape}") # має бути [16, 3, 32,
32]
print(f"Розмірність батчу HR (Висока якість): {hr_batch.shape}") # має бути [16, 3, 128,
128]
# Візуалізація
```

```
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
# Беремо перше зображення з батчу та перетворюємо тензор [C, H, W] у формат для малювання
[H, W, C]
axes[0].imshow(lr_batch[0].permute(1, 2, 0).numpy())
axes[0].set_title(f"LR патч {lr_batch[0].shape[1]}x{lr_batch[0].shape[2]}")
axes[0].axis("off")
axes[1].imshow(hr_batch[0].permute(1, 2, 0).numpy())
axes[1].set_title(f"HR патч {hr_batch[0].shape[1]}x{hr_batch[0].shape[2]}")
axes[1].axis("off")
plt.tight_layout()
plt.show()
#-------------------------------------------
# Код моделі
import torch.nn as nn
# ----------------- БЛОК ГЕНЕРАТОРА -----------------
class ResidualBlock(nn.Module):
    def __init__(self, channels):
        super().__init__()
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(channels)
        self.prelu = nn.PReLU()
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(channels)
    def forward(self, x):
        # Skip connection: додаємо вхідний тензор до результату згорток
        residual = self.conv1(x)
        residual = self.bn1(residual)
        residual = self.prelu(residual)
        residual = self.conv2(residual)
        residual = self.bn2(residual)
        return x + residual
class Generator(nn.Module):
    def __init__(self, scale_factor=4, num_res_blocks=16):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=9, padding=4),
            nn.PReLU()
        )
        # 16 залишкових блоків
        self.res_blocks = nn.Sequential(*[ResidualBlock(64) for _ in
range(num_res_blocks)])
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64)
        )
        # Блоки збільшення роздільної здатності (Upsampling)
        upsample_blocks = []
        for _ in range(scale_factor // 2): # Для збільшення в 4 рази потрібно 2 блоки
(2x2=4)
            upsample_blocks.append(nn.Conv2d(64, 256, kernel_size=3, padding=1))
            upsample_blocks.append(nn.PixelShuffle(2))
            upsample_blocks.append(nn.PReLU())
        self.upsample = nn.Sequential(*upsample_blocks)
        self.conv3 = nn.Conv2d(64, 3, kernel_size=9, padding=4)
        self.sigmoid = nn.Sigmoid() # Гарантує, що пікселі будуть в діапазоні [0, 1]
    def forward(self, x):
        out1 = self.conv1(x)
        out = self.res_blocks(out1)
        out2 = self.conv2(out)
```

```
        out = out1 + out2 # Глобальний skip connection
        out = self.upsample(out)
        out = self.conv3(out)
        return self.sigmoid(out)
# ----------------- БЛОК ДИСКРИМІНАТОРА -----------------
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        def discriminator_block(in_filters, out_filters, stride=1, normalize=True):
            layers = [nn.Conv2d(in_filters, out_filters, kernel_size=3, stride=stride,
padding=1)]
            if normalize:
                layers.append(nn.BatchNorm2d(out_filters))
            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers
        self.features = nn.Sequential(
            *discriminator_block(3, 64, normalize=False),
            *discriminator_block(64, 64, stride=2),
            *discriminator_block(64, 128),
            *discriminator_block(128, 128, stride=2),
            *discriminator_block(128, 256),
            *discriminator_block(256, 256, stride=2),
            *discriminator_block(256, 512),
            *discriminator_block(512, 512, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 1),
            nn.Sigmoid() # Видає ймовірність від 0 (Fake) до 1 (Real)
        )
    def forward(self, x):
        out = self.features(x)
        out = self.classifier(out)
        return out
# ----------------- ТЕСТУВАННЯ АРХІТЕКТУРИ -----------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Ініціалізуємо моделі та переносимо на відеокарту
generator = Generator().to(device)
discriminator = Discriminator().to(device)
print(f"Кількість параметрів Генератора: {sum(p.numel() for p in
generator.parameters()):,}")
print(f"Кількість параметрів Дискримінатора: {sum(p.numel() for p in
discriminator.parameters()):,}")
# Переносимо наш тестовий батч з попереднього кроку на GPU
lr_batch_gpu = lr_batch.to(device)
hr_batch_gpu = hr_batch.to(device)
# Тестовий прогін (Forward pass)
with torch.no_grad(): # Вимикаємо градієнти, бо ми поки не навчаємо, а лише тестуємо
    fake_hr_batch = generator(lr_batch_gpu)
    real_pred = discriminator(hr_batch_gpu)
    fake_pred = discriminator(fake_hr_batch)
print(f"\nРозмірність на вході в Генератор: {lr_batch_gpu.shape}")
print(f"Розмірність на виході з Генератора: {fake_hr_batch.shape} (Має збігатися з HR)")
print(f"Вихід Дискримінатора (на справжніх): {real_pred.shape} (Має бути [16, 1])")
#-------------------------------------------
# Тренування моделі
```

```
import torch.optim as optim
from torchvision.models import vgg19, VGG19_Weights
import math
# --- 1. Налаштування функцій втрат та оптимізаторів ---
vgg = vgg19(weights=VGG19_Weights.DEFAULT).features[:36].eval().to(device)
for param in vgg.parameters():
    param.requires_grad = False
criterion_GAN = nn.BCELoss()
criterion_content = nn.MSELoss()
optimizer_G = optim.Adam(generator.parameters(), lr=1e-4, betas=(0.9, 0.999))
optimizer_D = optim.Adam(discriminator.parameters(), lr=1e-4, betas=(0.9, 0.999))
# --- 2. Функція для розрахунку PSNR ---
def calculate_psnr(img1, img2):
    mse = torch.mean((img1 - img2) ** 2)
    if mse == 0:
        return 100.0
    return 20 * torch.log10(1.0 / torch.sqrt(mse)).item()
# --- 3. Ініціалізація списків для графіків ---
history = {
    'd_loss': [],
    'g_loss': [],
    'psnr': []
}
# --- 4. ГОЛОВНИЙ ЦИКЛ НАВЧАННЯ ---
EPOCHS = 50 # Для тесту ставимо 5
total_batches = len(train_loader)
print("🚀 Починаємо навчання...")
for epoch in range(EPOCHS):
    epoch_d_loss = 0
    epoch_g_loss = 0
    epoch_psnr = 0
    print(f"\n--- Епоха [{epoch+1}/{EPOCHS}] почалася ---")
    for idx, (lr_imgs, hr_imgs) in enumerate(train_loader):
        lr_imgs = lr_imgs.to(device)
        hr_imgs = hr_imgs.to(device)
        real_labels = torch.ones((hr_imgs.size(0), 1)).to(device)
        fake_labels = torch.zeros((hr_imgs.size(0), 1)).to(device)
        # ========================================
        #  Крок 1: Навчання Дискримінатора
        # ========================================
        optimizer_D.zero_grad()
        real_out = discriminator(hr_imgs)
        d_loss_real = criterion_GAN(real_out, real_labels)
        fake_imgs = generator(lr_imgs)
        fake_out = discriminator(fake_imgs.detach())
        d_loss_fake = criterion_GAN(fake_out, fake_labels)
        d_loss = d_loss_real + d_loss_fake
        d_loss.backward()
        optimizer_D.step()
        # ========================================
        #  Крок 2: Навчання Генератора
        # ========================================
        optimizer_G.zero_grad()
        fake_out_for_G = discriminator(fake_imgs)
        g_loss_gan = criterion_GAN(fake_out_for_G, real_labels)
        g_loss_content = criterion_content(fake_imgs, hr_imgs)
        real_features = vgg(hr_imgs).detach()
        fake_features = vgg(fake_imgs)
```

```
        g_loss_vgg = criterion_content(fake_features, real_features)
        g_loss = g_loss_content + 0.006 * g_loss_vgg + 1e-3 * g_loss_gan
        g_loss.backward()
        optimizer_G.step()
        # --- Збір статистики ---
        epoch_d_loss += d_loss.item()
        epoch_g_loss += g_loss.item()
        current_psnr = calculate_psnr(fake_imgs.detach(), hr_imgs)
        epoch_psnr += current_psnr
        # Виводимо статус кожні 10 батчів
        if (idx + 1) % 10 == 0 or (idx + 1) == total_batches:
            print(f"Крок [{idx+1}/{total_batches}] | D_loss: {d_loss.item():.4f} |
G_loss: {g_loss.item():.4f} | PSNR: {current_psnr:.2f} dB")
    # Усереднення результатів за епоху
    avg_d_loss = epoch_d_loss / total_batches
    avg_g_loss = epoch_g_loss / total_batches
    avg_psnr = epoch_psnr / total_batches

    history['d_loss'].append(avg_d_loss)
    history['g_loss'].append(avg_g_loss)
    history['psnr'].append(avg_psnr)
    print(f" Епоха [{epoch+1}/{EPOCHS}] завершена! Середній PSNR: {avg_psnr:.2f} dB")
print("\n Тестове навчання успішно завершено!")
#-------------------------------------------
# вивід результатів
#-------------------------------------------
import matplotlib.pyplot as plt
import torchvision.transforms.functional as TF
# --- 1. ПОБУДОВА ГРАФІКІВ НАВЧАННЯ ---
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
# Графік функцій втрат (Loss)
ax1.plot(history['d_loss'], label='Loss Дискримінатора', color='#1f77b4', linewidth=2)
ax1.plot(history['g_loss'], label='Loss Генератора', color='#ff7f0e', linewidth=2)
ax1.set_title("Рисунок 4.1 – Динаміка функцій втрат під час навчання", fontsize=12)
ax1.set_xlabel("Епоха", fontsize=11)
ax1.set_ylabel("Значення Loss", fontsize=11)
ax1.legend()
ax1.grid(True, linestyle='--', alpha=0.7)
# Графік метрики PSNR
ax2.plot(history['psnr'], label='PSNR на навчальній вибірці', color='#2ca02c',
linewidth=2)
ax2.set_title("Рисунок 4.2 – Динаміка метрики PSNR", fontsize=12)
ax2.set_xlabel("Епоха", fontsize=11)
ax2.set_ylabel("PSNR (dB)", fontsize=11)
ax2.legend()
ax2.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
# --- 2. ВІЗУАЛЬНЕ ПОРІВНЯННЯ ЗОБРАЖЕНЬ ---
generator.eval() # Переводимо генератор у режим оцінювання (без навчання)
# Беремо один тестовий батч
lr_imgs, hr_imgs = next(iter(train_loader))
lr_imgs_gpu = lr_imgs.to(device)
# Генеруємо зображення нашою моделлю
with torch.no_grad():
    sr_imgs = generator(lr_imgs_gpu).cpu()
# Вибираємо перше зображення з батчу
idx = 0
lr_img_show = lr_imgs[idx]
```

```
sr_img_show = sr_imgs[idx]
hr_img_show = hr_imgs[idx]
# Збільшуємо LR (32x32) базовим бікубічним методом до 128x128 для чесного порівняння
lr_bicubic = TF.resize(lr_img_show, [128, 128],
interpolation=TF.InterpolationMode.BICUBIC)
# Функція для безпечної конвертації тензора в картинку
def tensor_to_img(tensor):
    tensor = torch.clamp(tensor, 0, 1) # Відсікаємо аномалії
    return tensor.permute(1, 2, 0).numpy()
fig2, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(tensor_to_img(lr_bicubic))
axes[0].set_title("а) Бікубічна інтерполяція (Базовий метод)", fontsize=11)
axes[0].axis("off")
axes[1].imshow(tensor_to_img(sr_img_show))
axes[1].set_title("б) Наша модель (SRGAN)", fontsize=11)
axes[1].axis("off")
axes[2].imshow(tensor_to_img(hr_img_show))
axes[2].set_title("в) Оригінал (High-Res)", fontsize=11)
axes[2].axis("off")
plt.suptitle("Рисунок 4.3 – Візуальне порівняння результатів відновлення", fontsize=14)
plt.tight_layout()
plt.show()
# fig2.savefig("visual_comparison.png", dpi=300, bbox_inches='tight')
```