Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ГЕНЕРАТИВНО-ЗМАЗАЛЬНИХ МЕРЕЖ
IMAGE RESTORATION SOFTWARE USING GENERATIVE ADVERSARIAL NETWORKS

Виконав: студент 4 курсу, групи КНТ-127
Спеціальності 121 Інженерія програмного забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Діденко А.Є.
(прізвище та ініціали)

Керівник Олійник. А.О.
(прізвище та ініціали)

Рецензент Скрупський С.Ю.
(прізвище та ініціали)

2021

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____С.О. Субботін
"____" _____20___року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Діденка Андрія Євгеновича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для відновлення зображень з використанням генеративно-змагальних мереж. Image Restoration Software Using Generative Adversarial Networks

керівник проєкту (роботи) Олійник Андрій Олександрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 30 " березня 2021 року № 103
2. Строк подання студентом проєкту (роботи) 10 травня 2021 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури мереж і програми. 3. Реалізація програмного забезпечення та аналіз результатів. 4. Експлуатація програмного забезпечення. Висновки. Перелік джерел посилання. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Олійник А.О., доцент		
Нормоконтролер	Андрєєв М.О., асистент		

7. Дата видачі завдання “ 22 ” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4-5 тиждень	Розділ 2
4	Розробка програми	6-7 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	8 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	9 тиждень	Додатки
7	Захист роботи	10 тиждень	

Студент _____ Діденко А.Є.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Олійник А.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 120 с., 48 рис., 8 табл., 3 дод., 51 джерело.

ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ, ГЕНЕРАТИВНО-ЗМАГАЛЬНІ МЕРЕЖІ, ГЛИБИННЕ НАВЧАННЯ, ОБРОБКА ЗОБРАЖЕНЬ, ШТУЧНІ НЕЙРОННІ МЕРЕЖІ.

Об’єкт дослідження – процес відновлення зображень.

Предмет дослідження – методи відновлення зображень з використанням глибинного навчання.

Мета роботи – програмна реалізація методів відновлення зображень з використанням генеративно-змагальних мереж.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, мова програмування Python, фреймворки для глибокого навчання Tensorflow, Keras, хмарний сервіс Google Colab з використанням відеокарти Nvidia Tesla T4, персональний комп’ютер з використанням відеокарти RTX 2060 Super.

Результати. Створено програмний застосунок, який відновлює зображення за допомогою генеративно-змагальних мереж шляхом колоризації, видалення шуму та збільшення роздільної здатності відсутніх областей зображення.

Висновки. Розроблено програмний застосунок для відновлення зображень, що автоматизує та прискорює процес обробки зображення.

Галузь використання – відновлення історичних та/або пошкоджених зображень різними категоріями користувачів.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 120 pages, 48 figures, 8 tables, 3 appendixes, 51 sources.

ARTIFICIAL NEURAL NETWORKS, DEEP LEARNING, GENERATIVE ADVERSARIAL NETWORKS, IMAGE PROCESSING, IMAGE RESTORATION.

Object of study is the process of image restoration.

Subject of study is methods of image restoration using deep learning.

The purpose of the work is implementation of image restoration methods using generative adversarial networks.

Materials, methods and tools: structured and object-oriented programming, programming language Python, deep learning frameworks Tensorflow, Keras, cloud service Google Colab with Nvidia Tesla T4 graphic card, personal computer with RTX 2060 Super graphic card.

Results. Software, that restore images using generative adversarial networks by colorizing, denoising and resolution increasing images was created.

Conclusions. Software for image restoration was created. It automates and accelerates image processing.

The field of use is restoration of historical and/or damaged images by every user category.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ГЗМ – генеративно-змагальна мережа;

ЗНМ – згорткова нейронна мережа;

УГЗМ – умовна генеративно-змагальна мережа;

ШНМ – штучна нейронна мережа.

ВСТУП

Останнім часом в Інтернеті зростає популярність візуальної інформації. Через це підвищується попит на програми, які дозволяють якісно оброблювати зображення.

Звичайні редактори використовують класичні алгоритми для обробки зображень. Такі алгоритми дозволяють вирішувати прості задачі, наприклад, зміна колірного тону зображення чи вирівнювання яскравості. Однак, для вирішення складніших задач за допомогою таких редакторів користувач повинен володіти професійними навичками обробки зображень.

Із розвитком технологій глибинного навчання зростає популярність редакторів, які можуть вирішувати складні задачі і потребують від користувача мінімальних навичок у роботі із зображеннями. Такі редактори спроможні якісніше оброблювати зображення, оскільки штучні нейронні мережі більш гнучкі, ніж класичні алгоритми.

Актуальність роботи полягає в дослідженні предметної області і різноманітних архітектур штучних нейронних мереж для обробки зображень, а саме колоризації, шумозниження і збільшення роздільної здатності.

У випускній кваліфікаційній роботі створюється програма для відновлення зображень, розроблюються і навчаються генеративно-змагальні мережі, а також аналізуються результати і описуються шляхи покращення якості роботи програми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття

Колоризація зображення – процес отримання кольорового зображення із вхідного чорно-білого зображення із збереженням семантичних кольорів і тонів [1].

Зниження шуму – процес усування шумів з корисного сигналу з метою підвищення його суб’єктивної якості. Шумозниження при відтворенні зображень служить для покращення візуального сприйняття [2].

Збільшення роздільної здатності – процес збільшення розміру зображення. Методи, що використовуються для цього, намагаються зменшити втрати при збільшенні зображення [3].

Відновлення зображення – процес отримання чистого/оригінального зображення із пошкодженого зображення [4]. У даній роботі під поняттям «відновлення зображення» мається на увазі колоризація, шумозниження і збільшення роздільної здатності.

У залежності від способу дискретизації цифрове зображення може одного з двох основних типів: растрове або векторне. Растрове зображення є двомірним масивом даних (матрицею пікселів). Векторне зображення складається з векторів і кривих Безьє. Растрові зображення використовуються у всіх випадках, коли необхідно відтворити аналоговий оригінал: фотографію, малюнок чи складний елемент оформлення, який нераціонально переводити в векторну форму. Векторні зображення найчастіше використовуються у друкованих ЗМІ та в рекламній продукції завдяки можливості якісного поліграфічного відтворення чітких ліній, яскравих кольорів, рівних заливок і геометрично правильних контурів [5]. У даній роботі оброблюються саме растрові зображення.

Колірна модель являє собою засоби для концептуального і кількісного опису кольору. Колірні моделі використовуються для математичного опису певних колірних областей спектру. Кожному основному кольору

присвоюється певне значення цифрового коду, після чого всі інші кольори визначаються як комбінація основних кольорів [6]. Найпоширенішою колірною моделлю є модель RGB, у якій колір задається як поєднання трьох каналів: червоного (R), зеленого (G) і синього (B).

Глибинне навчання – це розділ машинного навчання, спрямований на побудову ієрархічних моделей шляхом використання високорівненевих масових абстракцій даних на основі глибинного графу с багатьма обробними шарами, які здійснюють лінійні або нелінійні перетворення. У центрі глибинного навчання є глибинні нейронні мережі і методи їхньої побудови [7].

Штучна нейронна мережа (ШНМ) – математична модель, а також її програмне або апаратне втілення, побудована на принципі організації та функціонування біологічних нейронних мереж – мереж нервових клітин живого організму [7].

Штучний нейрон – це примітивний обчислювальний пристрій (або його модель), що має кілька входів і один вихід, і є основним обчислювальним елементом ШНМ [7]. На рис. 1.1 зображено схему штучного нейрона.

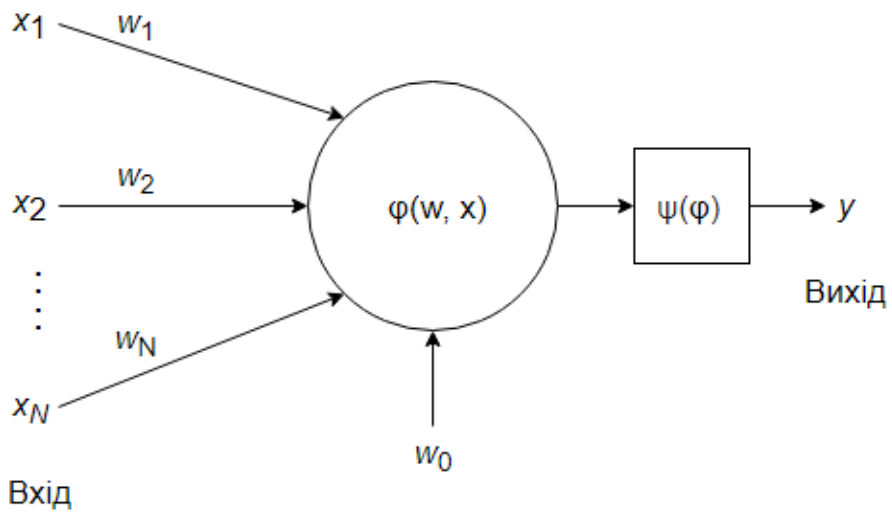


Рисунок 1.1 – Схема штучного нейрона

На вхід одношарового нейрона надходить вхідний вектор – набір вхідних сигналів $x=\{x_j\}$, $j = 1, 2, \dots, N$, де N– кількість входів. Кожний вхідний сигнал x_j зважується (масштабується певним чином) відносно зіставленої йому ваги зв'язку (вагового коефіцієнта) w_j, яка моделює перетворення сигналу у синапсі (міжнейронному контакті) [7].

Дискримінантна (вагова) функція нейрона φ поєднує зважені вхідні сигнали, отримуючи постсинаптичний потенціал та подає його значення до фукнції активації ψ, яка видає скалярне значення, що видається на виході нейрона [7].

Отже, математична модель функціонування штучного нейрона описується співвідношенням (1.1):

$$y = \psi(\varphi(w, x)), \tag{1.1}$$

де x – вектор вхідних аргументів (сигналів);

y – значення на виході нейрона;

ψ – функція активації;

φ – дискримінантна функція;

$w=\{w_j\}$ – вектор, що містить значення вагових w_j коефіцієнтів і значення зсуву (порогове значення) w_0 [7].

Глибинна нейронна мережа – це різновид ШНМ, що має багато шарів обробки даних, яка перетворює вхідні данні у вихідні, ієрархічно виділяючи та агрегуючи ознаки, підвищуючи рівень абстракції даних в напрямку від входів до виходів [7].

1.2 Огляд існуючих методів

У даній роботі досліджується завдання відновлення зображень, що включає в себе колоризацію, зниження шуму та збільшення роздільної

здатності. Оскільки завдання є комплексним, описані нижче існуючі методи можуть використовуватися для вирішення декількох частин завдання.

Існуючі методи можна умовно розділити на дві категорії: методи з використанням детермінованих алгоритмів і методи з використанням алгоритмів глибинного навчання. До методів з використанням детермінованих алгоритмів належать методи просторової фільтрації та вейвлет-перетворення (для зниження шуму), методи інтерполяції (для збільшення роздільної здатності зображення). До методів з використання алгоритмів глибинного навчання належать згорткові нейронні мережі і генеративно-змагальні мережі.

1.2.1 Методи просторової фільтрації для шумозниження

У загальному випадку, значення *v* цифрового зображення *f* у координаті *(x, y)* можна представити у наступному вигляді (1.2):

$$v = f(x, y), \tag{1.2}$$

де *f* – зображення;

x, y – цілі координати;

v – значення у координаті *(x, y)* [8].

Ділянка дійсної площини, охоплена координатами зображення називається просторовою областю [8].

Просторові операції виконуються безпосередньо над пікселями цифрового зображення. Просторові операції поділяються на однопіксельні операції, операції над групою сусідніх пікселів та геометрично просторові перетворення [8].

Просторові операції можна формалізувати наступним вираженням (1.3):

$$g(x, y) = T[f(x, y)], \tag{1.3}$$

де *f(x, y)* – вхідне зображення;

g(x, y) – вихідне зображення;

T – оператор, визначений на сусідстві точки *(x, y)* [8].

На рис. 1.2 схематично зображено рівняння (1.3).

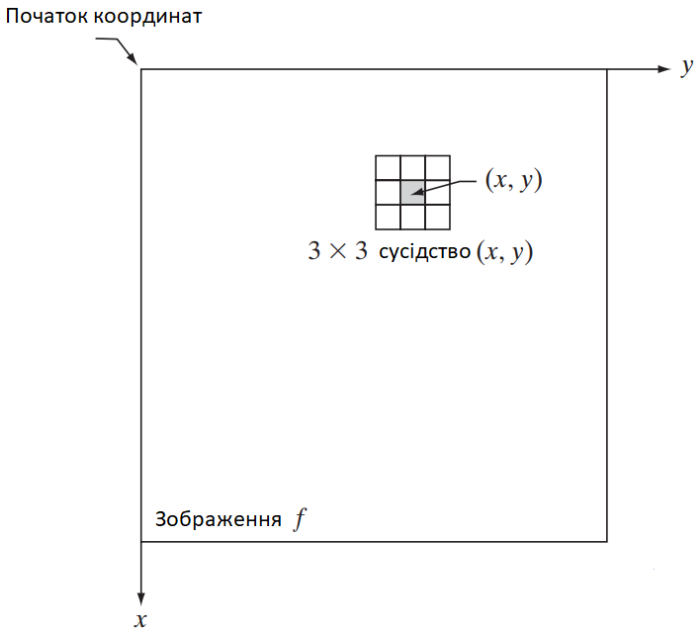


Рисунок 1.2 – Схематичне зображення просторової операції [8]

Процес просторової операції полягає в пересуванні центру області сусідства піксель за пікселем і застосуванні оператора *T* на пікселях у отриманій області. Коли центр області сусідства знаходиться на границі зображення, частина сусідства виходить за межі зображення. Існує два способи рішення цієї проблеми: ігнорувати область сусідства, що вийшла за межі зображення при обчислені оператора *T*, або розширити зображення шляхом додання границі з пікселів. Ширина доданої границі залежить від розміру області сусідства [8].

Просторова фільтрація є однією з просторових операцій. Термін «фільтр» запозичений з обробки частотної області, де «фільтрація» означає прийняття або викидання певних частотних компонентів [8].

Просторовий фільтр складається з області сусідства певної точки (зазвичай, невеликий квадрат) і заданої операції, що вираховується на пікселях зображення, охоплених областю сусідства. Результат операції – новий піксель з координатами, що дорівнюють координатам центру області сусідства, і значенням, що є результатом операції фільтрації. Якщо операція, що виконується над зображенням, лінійна, то фільтр називають лінійним просторовим фільтром, інакше – нелінійним [8].

У загальному випадку, лінійна просторова фільтрація на зображені розміру $M \times N$ з фільтром $m \times n$ можна формалізувати наступним виразами (1.4) – (1.6):

$$m = 2a + 1, \tag{1.4}$$

$$n = 2b + 1, \tag{1.5}$$

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t), \tag{1.6}$$

де f – вхідне зображення;

g – вихідне зображення;

w – фільтр;

a, b – цілі додатні числа;

x, y – координати пікселя [8].

Статистичні фільтри – це нелінійні просторові фільтри, результат яких базується на впорядкуванні пікселів, що містяться в області, охопленій фільтром. Центральний піксель області замінюється на значення, визначене результатом впорядкування [8].

Перевагами методів просторової фільтрації є простота та швидкість роботи. Головним недоліком цих методів є порівняно низька якість результату роботи.

1.2.1.1 Фільтр середнього арифметичного

Найпростіший метод фільтрації для шумозниження – це фільтр, що базується на середньому значенні, а саме на середньому арифметичному. Загальна формула фільтру має наступний вигляд (1.7):

$$\hat{f}(x,y) = \frac{1}{mn}\sum_{(s,t)} g(s,t), \tag{1.7}$$

де $\hat{f}$ - відновлене зображення;

g – пошкоджене зображення;

x, y – координати поточного фільтру;

s, t – координати у пікселів у області сусідства поточного пікселю;

m, n – ширина і висота фільтру [8].

1.2.1.2 Медіанний фільтр

Ще один метод фільтрації для шумозниження використовує медіанний фільтр, що відноситься до статистичних фільтрів. Медіанний фільтр замінює значення пікселю на медіану значень у області сусідства (1.8):

$$\hat{f}(x,y) = \underset{s,t}{median}\{g(s,t)\}, \tag{1.8}$$

де $\hat{f}$ - відновлене зображення;

g – пошкоджене зображення;

x, y – координати поточного фільтру;

s, t – координати у пікселів у області сусідства поточного пікселю [8].

1.2.1.3 Білетеральний фільтр

Білатеральний фільтр – це нелінійний фільтр для шумозниження, основна ідея якого полягає в заміні інтенсивності кожного пікселя середньозваженим значенням інтенсивності пікселів у сусідстві [9].

Білатеральний фільтр визначається наступним чином (1.9):

$$\hat{f}(x,y) = \frac{1}{W_p} \sum_{s,t} g(s,t) w_r(\|g(s,t) - g(x,y)\|) w_s(\|(s,t) - (x,y)\|), \qquad (1.9)$$

і W_p визначається як (1.10):

$$W_p = \sum_{s,t} w_r(\|g(s,t) - g(x,y)\|) w_s(\|(s,t) - (x,y)\|), \qquad (1.10)$$

де $\hat{f}$ - відновлене зображення;

g – пошкоджене зображення;

x, y – координати поточного фільтру;

s, t – координати у пікселів у області сусідства поточного пікселю;

w_r – діапазонний фільтр для згладжування різниці в інтенсивності;

w_s – просторовий фільтр для згладжування різниці у координатах [9].

1.2.1.4 Метод нелокальних середніх

На відміну від методів з використанням локальних середніх, що розраховують середнє значення сусідства певного пікселя для

шумозниження, метод нелокальних середніх розраховує середнє значення всіх пікселів, зважених за подібністю до обраного пікселя [10].

Формально, метод нелокальних середніх можна описати наступним чином (1.11):

$$u(p) = \frac{1}{C(p)} \sum_{q \in \Omega} v(q) f(p,q), \tag{1.11}$$

де Ω – зображення;

u(p) – значення пікселя у точці *p*;

v(q) – значення пікселя у точці *q*;

f(p,q) – функція зважування;

C(p) – коефіцієнт нормалізації [10].

Коефіцієнт нормалізації розраховується наступним чином (1.12):

$$C(p) = \sum_{q \in \Omega} f(p,q), \tag{1.12}$$

Мета функції зважування визначити, наскільки близько значення пікселів у точках *p* і *q*. Функція зважування має наступний вигляд (1.13):

$$f(p,q) = e - \frac{\left|B(q) - B(p)\right|^2}{h^2}, \tag{1.13}$$

де *B(q)* і *B(p)* – середнє значення пікселів у сусідстві пікселів *q* і *p* відповідно, що розраховується за наступною формулою (1.14):

$$B(p) = \frac{1}{|R(p)|} \sum_{i \in R(p)} v(i), \tag{1.14}$$

де *R(p)* – сусідство пікселя *p*;

|R(p)| – кількість пікселів у сусідстві *p* [10].

1.2.2 Методи інтерполяції для збільшення роздільної здатності

Інтерполяція – це метод знаходження значень у діапазоні дискретного набору відомих значень [11].

Методи інтерполяції застосовують для масштабування зображення. Під час збільшення розміру зображення утворюються проміжки у матриці пікселів, які необхідно ефективно заповнити. Збільшене зображення не повинно містити значних деформацій форми об’єктів і кольорів.

Перевагами методів інтерполяції для масштабування зображення є математична формальність та простота реалізації. До недоліків можна віднести порівняно низьку якість результатів та повільну роботу.

1.2.2.1 Метод найближчого сусіда

Це найпростіший і найшвидший метод, що використовується для збільшення роздільної здатності зображень. Під час цього методу значенню пікселя, що повинен бути інтерпольованим, задається значення найближчого відомого пікселя [12].

На рис. 1.3 зображено принцип роботи методу.

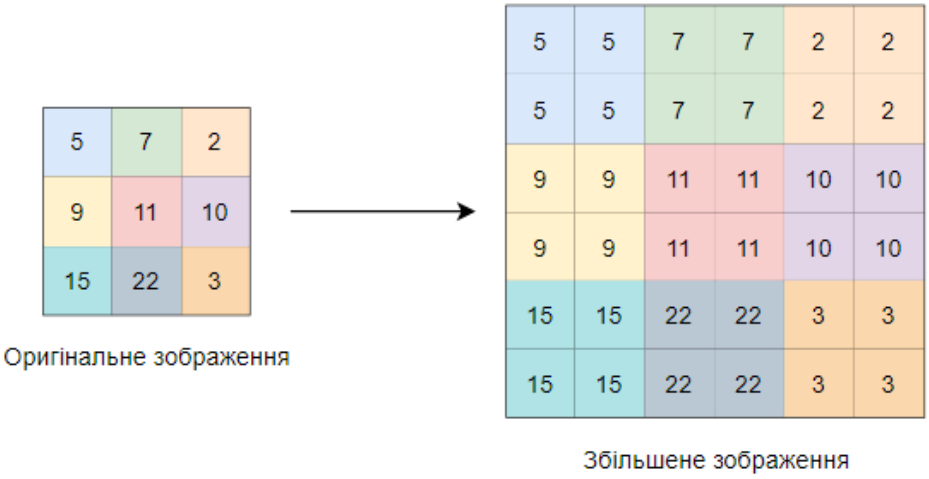


Рисунок 1.3 – Принцип роботи методу найближчого сусіда

При використанні цього методу сильно спотворюються рівні контури, що є головним недоліком цього методу [8].

1.2.2.2 Білінійна інтерполяція

На відміну від методу найближчого сусіда, білінійна інтерполяція розглядає сусідство 2×2 (4 сусідніх пікселі загалом) пікселю, значення якого повинно бути розраховане.

Формально білінійну інтерполяцію можна записати у наступному вигляді (1.15):

$$Q \approx a_0 + a_1 x + a_2 y + a_3 xy\,, \tag{1.15}$$

де Q – розраховане значення пікселю;

x, y – координати пікселю, значення якого повинне бути розраховане;

a_0, a_1, a_2, a_3 – коефіцієнти рівняння [13].

Коефіцієнти рівняння можна знайти із наступної системи рівнянь (1.16):

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 \\ 1 & x_1 & y_2 & x_1 y_2 \\ 1 & x_2 & y_1 & x_2 y_1 \\ 1 & x_2 & y_2 & x_2 y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} Q_{11} \\ Q_{12} \\ Q_{21} \\ Q_{22} \end{bmatrix}, \quad (1.16)$$

де x_i, y_j – координати сусіднього пікселю;

Q_{ij} – значення сусіднього пікселю [13].

Білінійна інтерполяція працює краще, ніж метод найближчого сусіда, оскільки використовує більше інформації про сусідні пікселі.

1.2.2.3 Бікубічна інтерполяція

Бікубічна інтерполяція використовує для розрахування значення пікселя сусідство 4×4 (16 пікселів).

Формально, бікубічну інтерполяцію можна записати наступним чином (1.17):

$$Q = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j , \quad (1.17)$$

де Q – розраховане значення пікселю;

x^i, y^j – координати пікселів у сусідстві;

a_{ij} – коефіцієнт рівняння [14].

Коефіцієнти рівняння можна знайти із наступної системи рівнянь (1.18):

$$Q = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \cdot \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}, \quad (1.18)$$

де *x*, *y* – координати пікселю, значення якого повинно бути розраховане [14].

1.2.3 Згорткові нейронні мережі

Згорткові нейронні мережі (ЗНМ) – різновид глибинних нейронних мереж, основним застосуванням яких є аналіз зображень.

Існує декілька видів шарів, що характерні для ЗНМ:

– згортковий шар;

– агрегуючий шар;

– шар розгортки;

– шар роз'єднання;

– шар нормалізації.

Згортка – математична операція двох функцій, що дозволяє отримати третю функцію. Операція згортки має наступний вигляд (1.19):

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-\tau) g(\tau) d\tau, \quad (1.19)$$

де *f(t)* і *g(t)* – дві задані функції [15].

Дискретна двовимірна операція згортки на цифровому зображенні має наступний вигляд (1.20):

$$w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t), \quad (1.20)$$

де w – фільтр,

f – вхідне зображення;

x, y – координати поточного фільтру;

s, t – координати у пікселів у області сусідства поточного пікселю;

a, b – цілі додатні числа, аналогічно до виразу 1.6 [8].

Під час операції згортки фільтр перевертається на 180°. Перевертання і зсув вхідного зображення замість фільтру зроблене для простоти нотації [8].

Хоча назва ЗНМ пішла від операції згортки, ЗНМ фактично використовують операцію взаємної кореляції (1.21):

$$w(x,y) \otimes f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t), \quad (1.21)$$

де w – фільтр,

f – вхідне зображення;

x, y – координати поточного фільтру;

s, t – координати у пікселів у області сусідства поточного пікселю;

a, b – цілі додатні числа, аналогічно до виразу 1.6 [8].

Єдина відмінність між операціями згортки та взаємної кореляції полягає в тому, що під час операції взаємної кореляції фільтр не перевертається. Надалі в тексті для позначення операції в ЗНМ замість словосполучення «взаємна кореляція» буде використовуватись «згортка» для підтримання загальної конвенції.

Згортковий шар – основний шар у ЗНМ. Кожен згортковий шар складається з набору фільтрів (ядер). Під час опрацювання даних у згортковому шарі кожен фільтр проходить через вхідні дані у напрямку зліва направо та зверху вниз, застосовуючи операцію згортки та утворюючи

двовимірну вихідну матрицю. Крок, на який переміщується фільтр, називається зсувом. Якщо вхідна інформація має декілька каналів, кожен фільтр набуває такої ж глибини, як і вхідна інформація. Операція згортки відбувається на кожному каналі, і результати додаються поелементно [16].

На рис. 1.4 зображено приклад роботи операції згортки.

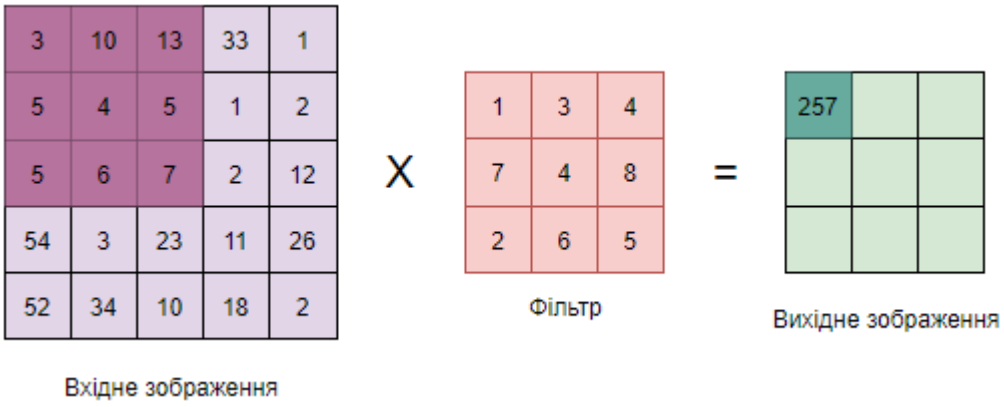


Рисунок 1.4 – Приклад роботи операції згортки

Основна мета роботи згорткового шару – виділити певні ознаки із зображення. Перші згорткові шари виділяють низькорівневі ознаки, такі як границі об’єктів. Чим більше згорткових шарів має ЗНМ, тим більше високорівневих ознак може виділити нейромережа [16].

Також у згортковому шарі часто розширюють вхідну матрицю, додаючи навколо неї рядки та стовпчики, заповнені нулями. Це робиться для збільшення розміру вихідної матриці та для підвищення впливу значень, що знаходяться по краях вхідної матриці [17].

Наступний вид шарів, який часто зустрічається в архітектурі ЗНМ – це агрегуючий шар. Його основна функція – зниження розмірності вхідних даних та виділення домінуючий ознак [16].

Процес роботи агрегуючого шару складається із переміщення вікна по вхідній матриці зліва направо та зверху вниз і вибору домінуючої ознаки у вікні за певним правилом. Розмір вікна та величина зсуву задаються заздалегідь [16].

Існує два види агрегуючий шару: максимізаційне агрегування та усереднювальне агрегування. Під час масимізаційного агрегування у вікні обирається найбільше значення, а під час усереднювального агрегування значення у вікні усереднюються [16].

На рис. 1.5 зображено процес роботи агрегуючого шару із розміром вікна 2×2 і з величиною зсуву 2.

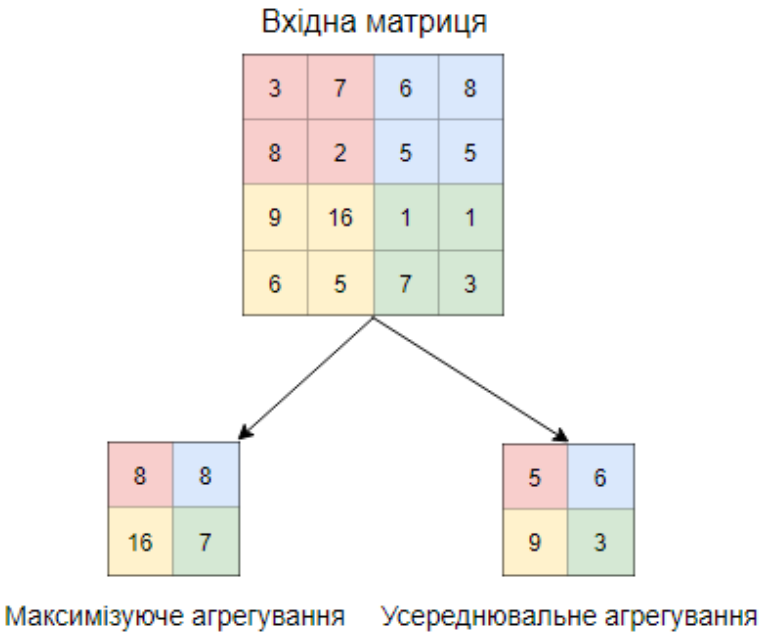


Рисунок 1.5 – Процес роботи агрегуючого шару

Ще один шар, що інколи використовується у ЗНМ – це шар розгортки. Цей шар виконує операцію, обернену до згортки, розширюючи при цьому розмір вхідної матриці. Фільтри у шарі розгортки навчаються аналогічно до фільтрів у згортковому шарі [18].

Інший шар, що використовується для розширення вхідних даних – шар роз’єднання. Цей шар працює обернено до агрегуючого шару, заповнюючи пусті комірки у вихідній матриці значеннями найближчої комірки вхідної матриці [18].

Принцип роботи шару роз’єднання зображено на рис. 1.6.

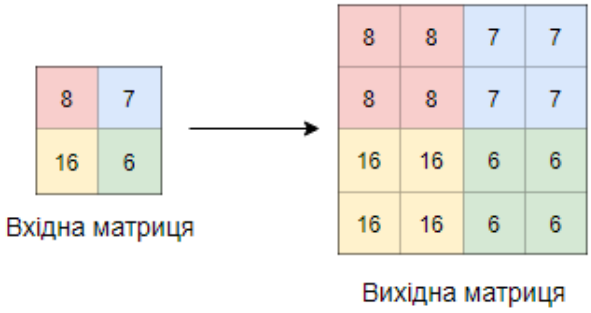


Рисунок 1.6 – Принцип роботи шару роз’єднання

Інколи у ЗНМ використовують нормалізуючий шар. Під час навчання ШНМ розподіл виходів прихованих шарів постійно змінюється. Це сповільнює процес навчання, оскільки кожен шар повинен адаптуватись до нового розподілу кожну ітерацію. За допомогою нормалізуючого шару вихід прихованих шарів масштабується до розподілу з нульовим математичним сподіванням та одиничною дисперсією шляхом навчання [19].

Перевагами ЗНМ є висока точність результатів, порівняно до класичних алгоритмів. Головними недоліками ЗНМ є потреба у високій обчислювальній потужності для навчання мережі і складність архітектури та реалізації.

1.2.3.1 ЗНМ для колоризації зображень

Метод «Colorful Image Colorization» [20] був запропонований у 2016 році.

Мережа оперує із зображеннями, які представлені у колірній моделі Lab. На вхід подається канал L (L – рівень освітлення), на виході отримуються спрогнозовані канали ab, що містять інформацію про співвідношення кольорів [20].

В основі ЗНМ, що була використана в досліджені, лежить архітектура VGG [21], але з деякими відмінностями: у мережі відсутні агрегуючі шари, кожен блок шарів згортки використовує нормалізуючий шар та функцію

активації ReLU. Крім того, ЗНМ повертає розподіл кольорів як результат, який потім поєднується із вхідним каналом L, для утворення кольорового зображення. Архітектуру ЗНМ зображено на рис. 1.7.

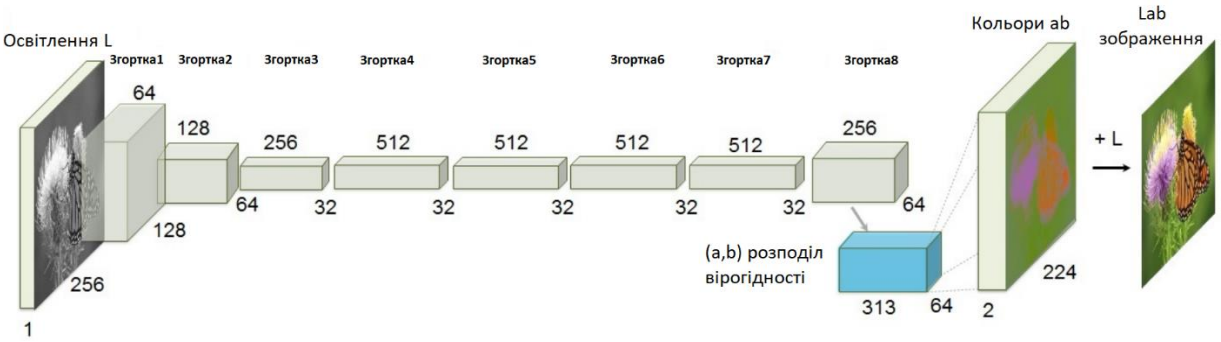


Рисунок 1.7 – Архітектура мережі методу Colorful Image Colorization [20]

У методі також запропоновано функцію втрат для колоризації. Задача колоризації є мультимодальною: багато об’єктів можуть мати декілька правдоподібних кольорів. Наприклад, яблуко, як правило, червоне, зелене або жовте, але навряд чи воно буде синім. Для вирішення цієї проблеми мережа видає розподіл можливих кольорів для кожного пікселя. Крім того, функція втрат перезважується під час навчання для того, щоб надати більше значення рідкісним кольорам [20].

Функція втрат має наступний вигляд (1.22):

$$L_{cl}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \log(\hat{Z}_{h,w,q}), \tag{1.22}$$

де Z – оригінальний розподіл кольорів;

$\hat{Z}$ – розподіл кольорів, отриманий мережею;

$v(Z_{h,w})$ – коефіцієнт перезважування;

h, w – координати пікселю на зображенні;

q – індекс каналу у зображенні [20].

Для тестування моделі було проведено тест Тьюринга, під час якого людям показували справжнє та колоризоване зображення. У результаті тестування 32% людей помилилися у виборі справжнього зображення [20].

Результати роботи методу зображено на рис. 1.8.

Рисунок 1.8 – Результати роботи методу Colorful Image Colorization [20]

1.2.3.2 ЗНМ для шумозниження

Один з популярних методів застосування ЗНМ для шумозниження – використання згорткового автокодувальника.

Автокодувальник – ШНМ, що навчається ефективно кодувати вхідну інформацію та відновлювати її із закодованого виду таким чином, щоб відновлена інформація була максимально схожа на первинну. Автокодувальники зменшують розмірність даних, навчаючись ігнорувати шум у даних [22].

Автокодувальник складається з 3 основних частин:

– кодувальника, за допомогою якого модель навчається зменшувати розмірність даних та кодувати інформацію;

– латентного представлення (коду, проміжного шару), у якому містяться стиснені дані;

– декодувальника, за допомогою якого модель навчається відновлювати закодовану інформацію [23].

Схематично архітектура автокодувальника зображена на рис. 1.9.

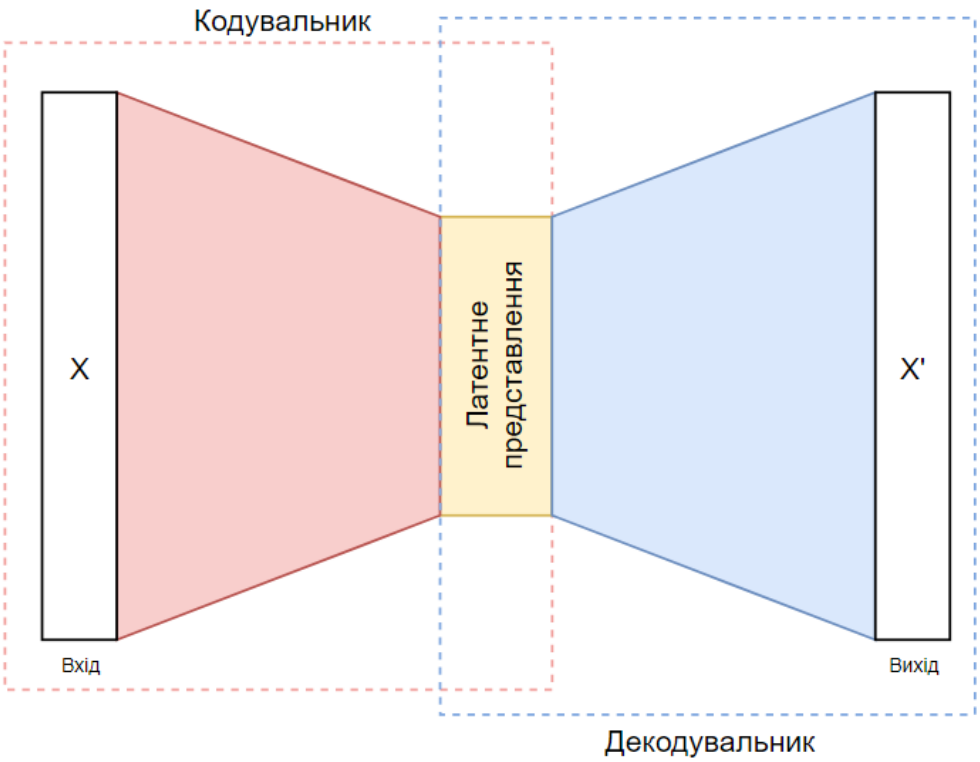


Рисунок 1.9 – Архітектура автокодувальника

Згортковий автокодувальник – автокодувальник, що складається із шарів, характерних для ЗНМ.

У дослідженні [24] згортковий автокодувальник використовується для шумозниження на рентгенографії органів грудної клітки. Для створення навчальної вибірки до зображень було додано випадковий шум.

Метод було порівняно з медіанним фільтром, методом нелокальних середніх і методом зменшення загальної варіації. Серед порівняних методів згортковий автокодувальник показав найкращий результат.

На рис. 1.10 зображено архітектуру мережі, що використовувалась у дослідженні.

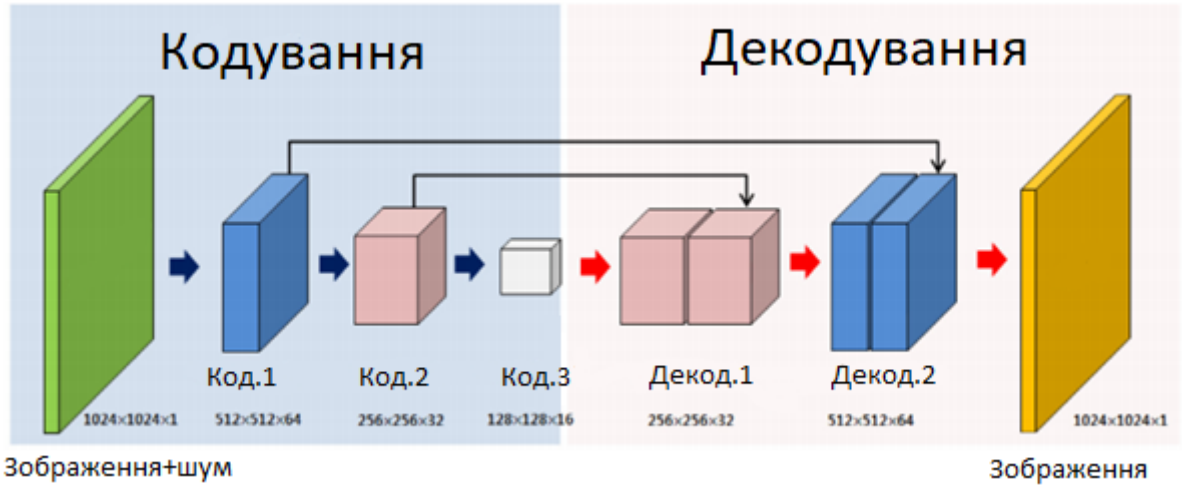


Рисунок 1.10 – Архітектура мережі, що використана у дослідженні [24]

1.2.4 Генеративно-змагальні мережі

Генеративно-змагальна мережа (ГЗМ) – це вид ШНМ, що складається з двох мереж: генератора і дискримінатора. Генератор G генерує екземпляри даних, а дискримінатор D намагається відрізнити справжні екземпляри від згенерованих.

Формально принцип роботи ГЗМ можна сформулювати наступним чином. Основною задачею генератора G є вивчити розподіл p_g з набору даних x. Для цього задається вхідний шум $p_z(z)$, що відображається у простір даних як $G(z;\theta_g)$, де z – вхідний шум, θ_g – параметри генератора. Також задається дискримінатор $D(x;\theta_d)$. $D(x)$ – це вірогідність того, що x належить до справжніх даних. Дискримінатор тренується таким чином, щоб максимізувати вірогідність правильної класифікації справжніх та згенерованих даних (1.23) [25]:

$$L(D) = \log(D(x)) + \log(1 - D(G(z))) \tag{1.23}$$

Генератор, у свою чергу, тренується таким чином, щоб мінімізувати наступну функцію (1.24):

$$L(G) = \log(1 - D(G(z))) \tag{1.24}$$

Таким чином, генератор і дискримінатор змагаються один з одним у гру з нульовою сумою [25].

Схематично ГЗМ зображено на рис. 1.11.

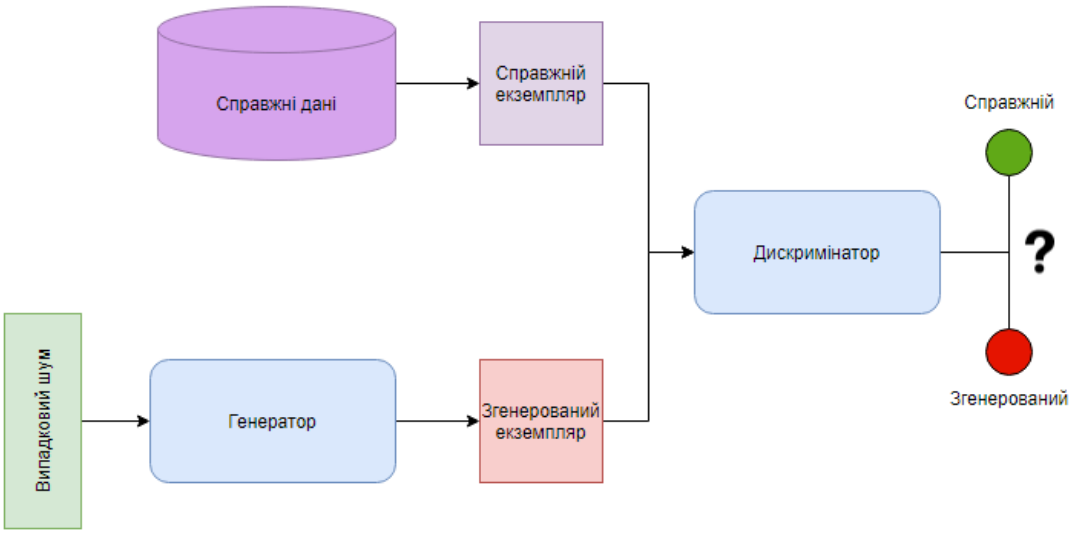


Рисунок 1.11 – Схема ГЗМ

Різновидом ГЗМ є умовні ГЗМ (УГЗМ). УГЗМ відрізняється від ГЗМ тим, що у вхідні дані до генератора і дискримінатора додаються деякі дані *y*, щоб обумовити генератор і дискримінатор. Ця додаткова інформація *y* може бути будь-якими даними, наприклад, класом даних *x*. Така техніка допомагає ГЗМ швидше та точніше навчатися [26].

Схематично УГЗМ зображено на рис. 1.12.

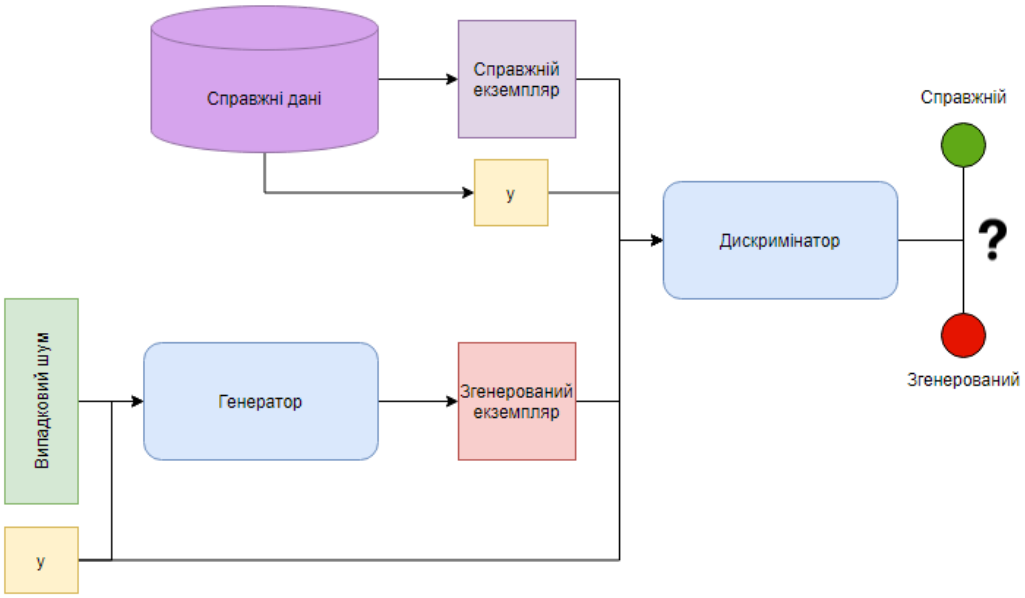


Рисунок 1.12 – Схема УГЗМ

Основною перевагою ГЗМ є якісне генерування даних, що схожі на реальні дані. Головним недоліком ГЗМ є велика складність навчання моделей генератора і дискримінатора для досягнення рівноваги між ними.

1.2.4.1 ГЗМ для колоризації зображень

ГЗМ може бути використана для колоризації зображень. Оскільки колоризація зображень є задачею трансляції зображень, при якій необхідно побудувати відповідності між вхідними на вихідними зображеннями, найчастіше використовують УГЗМ. На вхід подається чорно-біле зображення, на виході отримується кольорове зображення.

У дослідженні [27] для генератора використано архітектуру U-Net.

U-Net – це різновид згорткового автокодувальника з так званими пропускними зв’язками [28]. Під час проходження інформації через автокодувальник деяка інформація втрачається, і декодувальнику важко відновлювати дані. Для забезпечення стабільності та зниження втрат інформації до архітектури генератора було додано пропускні зв’язки між

шарами *i* та *n* – *i*, де *n* – загальна кількість шарів. Кожен пропускний зв’язок конкатенує вихід шару *i* із входом шару *n* – *i* [27].

На рис. 1.13 зображено порівняння архітектури автокодувальника і мережі U-Net.

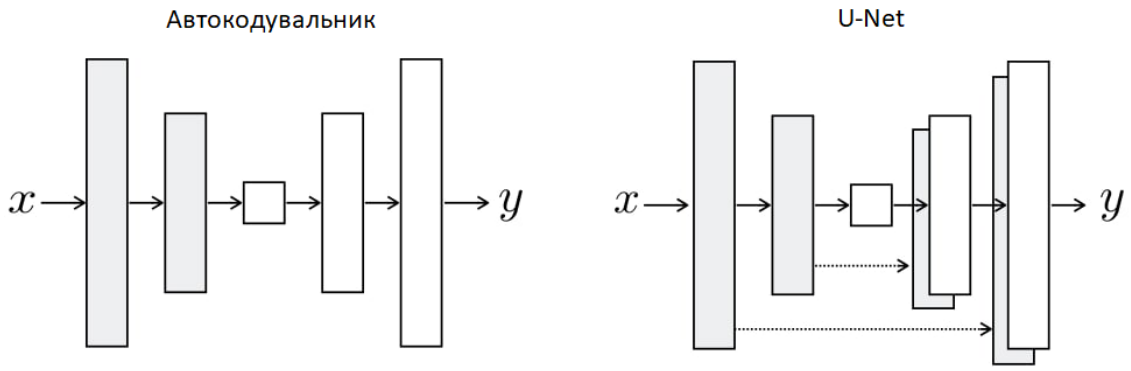


Рисунок 1.13 – Порівняння архітектури автокодувальника і U-Net [27]

На рис. 1.14 зображено архітектуру генератора, використану у дослідженні.

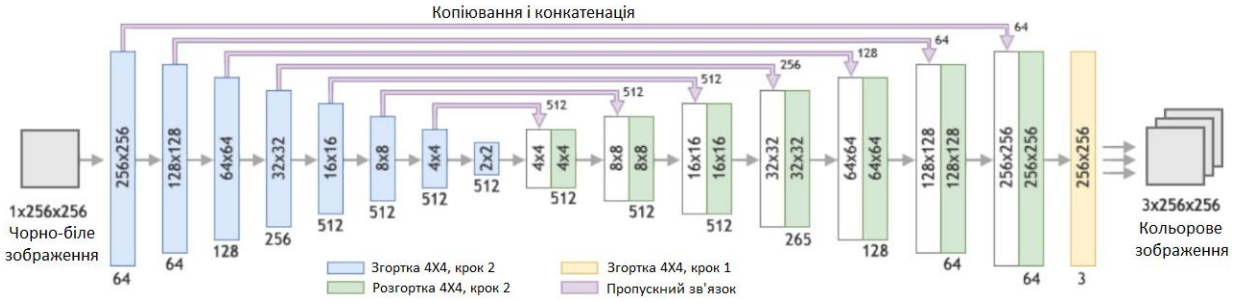


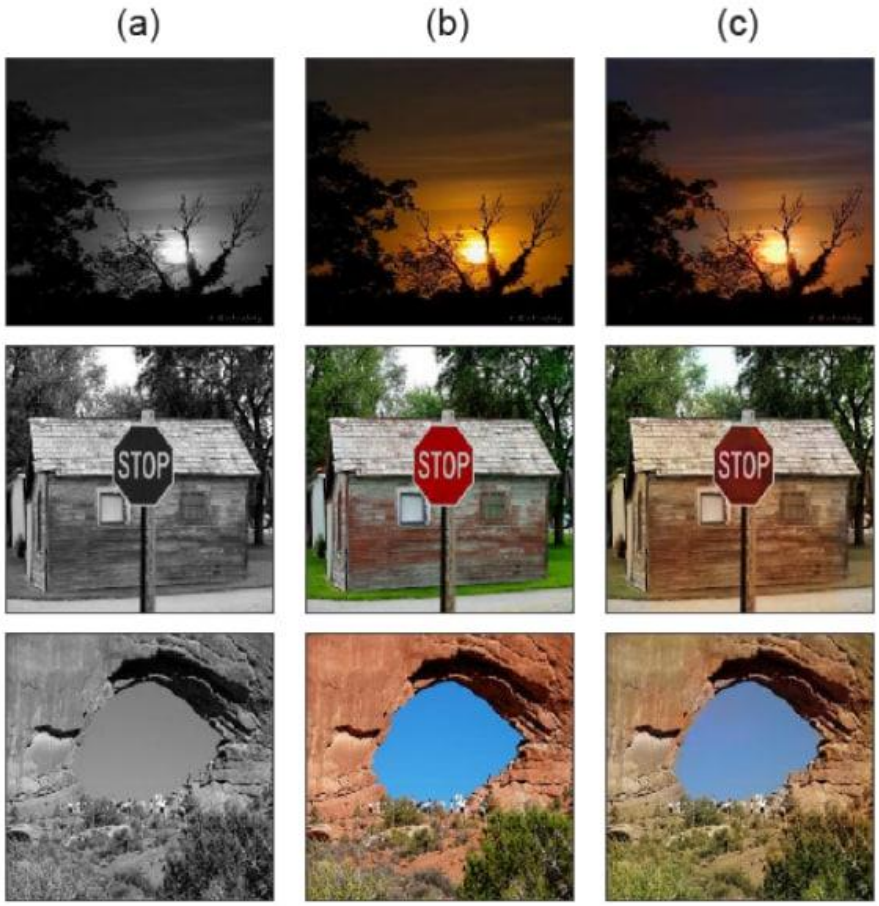
Рисунок 1.14 – Архітектура генератора, що використана у дослідженні [27]

Також у дослідженні було використано колірну модель Lab, замість звичайної RGB. Такий вибір зумовлений тим, що канал L містить інформацію про освітлення зображення, що може слугувати входом ГЗМ як чорно-біле зображення. Крім того, канали ab містять повну інформацію про кольори у зображенні. Таким чином, це запобігає різким змінам кольору і

яскравості через невеликі хвилювання значень інтенсивності, які виникають у колірній моделі RGB [27].

Для тестування у якості функції втрат було використано середню абсолютну помилку, що у результаті дало значення 7.5 [27].

На рис. 1.15 зображені деякі результати роботи ГЗМ.

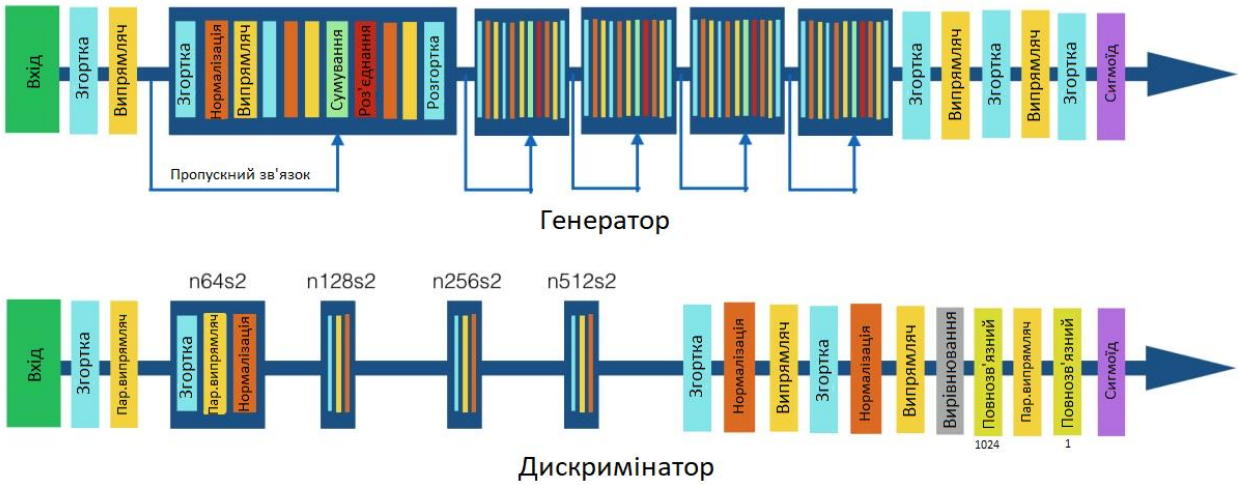


(a) – Чорно-біле зображення, (b) – справжнє зображення, (c) – колоризоване зображення

Рисунок 1.15 – Результати роботи ГЗМ

1.2.4.2 ГЗМ для шумозниження

ГЗМ також може бути використана для шумозниження. У дослідженні [29] використовуються наступні архітектури для генератора і дискримінатора (рис. 1.16).



Умовні познаки:
n – кількість фільтрів;
s – крок згортки.

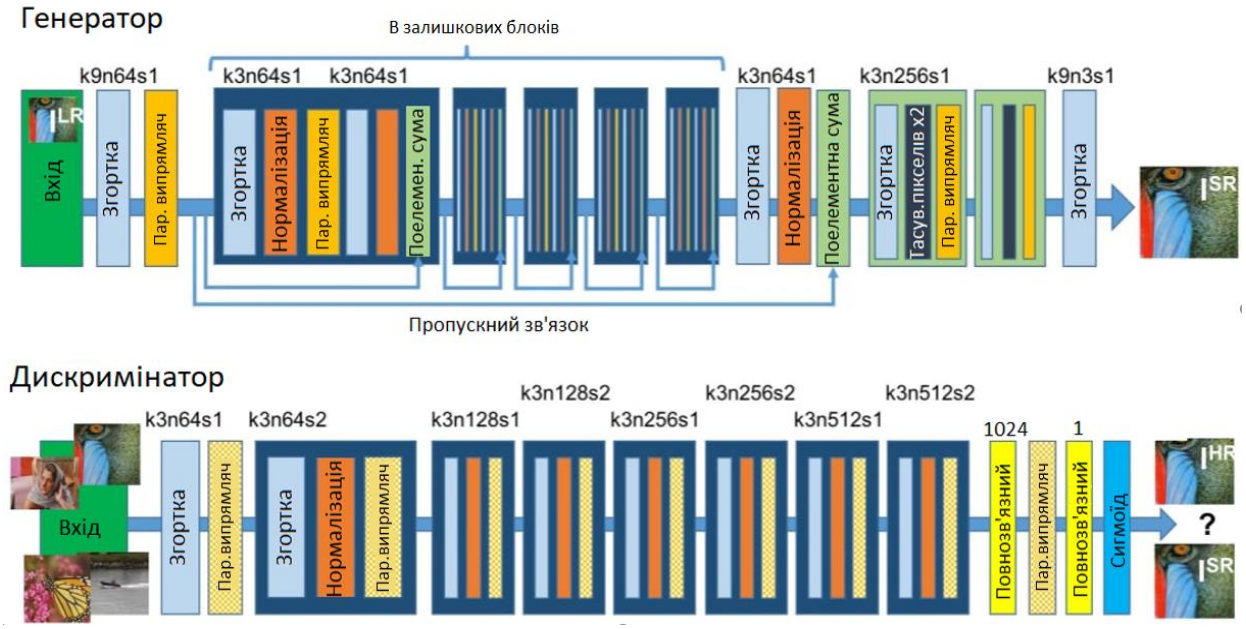
Рисунок 1.16 – Архітектури генератора і дискримінатора [29]

У дослідженні ця архітектура також використовується для збільшення роздільної здатності та для зменшення розмиття зображення.

Для шумозниження у вхідні зображення було додано випадковий шум. Для тестування було PSNR (пікове відношення сигналу до шуму), використання ГЗМ показало кращий результат, ніж медіанний фільтр на метод нелокальних середніх [29].

1.2.4.3 ГЗМ для збільшення роздільної здатності

У дослідженні [30] було використано ГЗМ для збільшення роздільної здатності зображень. На рис. 1.17 зображено архітектуру генератора і дискримінатора.



Умовні познаки:
k – розмір ядра фільтру;
n – кількість фільтрів;
s – крок згортки.

Рисунок 1.17 – Архітектура генератора і дискримінатора [30]

Відмінністю цього дослідження є використання особливої функції втрат генератора. Вона має наступний вигляд (1.25):

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR}, \tag{1.25}$$

де l_X^{SR} – функція втрат змісту;

l_{Gen}^{SR} – змагальна функція втрат [30].

Змагальна функція втрат підштовхує генератор до природнього різноманіття зображень за допомогою дискримінатора, що навчений розрізняти оригінальні та згенеровані зображення і визначається наступним чином (1.26):

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta D}(G_{\theta G}(I^{LR})), \tag{1.26}$$

де $G_{\theta G}(I^{LR})$ – згенероване зображення;

N – кількість згенерованих зображень [30].

Функція втрат змісту використовується для підтримки подібності між пікселями у справжньому та згенерованому зображеннях.

Функція втрат змісту базується на виходах активаційних шарів у навченій моделі VGG і визначається наступним чином (1.27):

$$l_{VGG/i.j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta G}(I^{LR}))_{x,y})^2, \tag{1.27}$$

де $G_{\theta G}(I^{LR})$ – згенероване зображення;

I^{HR} – оригінальне зображення;

$\phi_{i,j}$ – j-та мапа ознак перед i-м агрегуючим шаром у VGG;

$W_{i,j}, H_{i,j}$ – ширина та висота відповідної мапи ознак [30].

1.3 Порівняння методів відновлення зображень

Проаналізувавши методи відновлення зображень можна зробити їхню порівняльну характеристику (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика існуючих методів відновлення зображень

Критерій	Класичний алгоритм	Згорткова нейронна мережа	Генеративно-змагальна мережа
Складність реалізації	Низька	Помірна	Висока
Точність роботи	Низька	Помірна	Висока
Швидкість роботи	Висока	Помірна	Помірна
Здатність до узагальнення	Низька	Помірна	Висока
Залежність від обчислювальної потужності	Ні	Так	Так

Із порівняння у табл. 1.1 можна побачити, що точні методи працюють повільніше, ніж методи з низькою точністю. Серед точних методів – ЗНМ та ГЗМ. Завдяки тому, що ГЗМ здатна краще узагальнювати інформацію шляхом навчання, точність її роботи і якість отриманих результатів будуть вище, ніж у ЗНМ. Оскільки якість результатів важливіше, ніж швидкість роботи, доцільно використовувати ГЗМ для вирішення задачі відновлення зображень.

1.4 Постановка задачі до роботи

У даній роботі розглядається задача відновлення зображень, що включає в себе колоризацію, шумозниження та збільшення роздільної здатності зображення. Основним етапом є створення моделей для вирішення поставлених підзадач.

Нехай задано вихідне зображення I. Також задані зображення, що є пошкодженими варіантами вихідного зображення I, а саме: I_c, – чорно-біле вихідне зображення, I_n – зашумлене вихідне зображення, I_r – зменшене вихідне зображення. Необхідно створити наступні моделі: m_c – модель колоризації, m_n – модель шумозниження, m_r – модель збільшення роздільної здатності.

Колоризоване зображення можна формально записати наступним чином (1.28):

$$\hat{I}_c = m_c(I_c) \tag{1.28}$$

Знешумлене зображення виражається формулою (1.29):

$$\hat{I}_n = m_n(I_n) \tag{1.29}$$

Аналогічно, відновлене зменшене зображення виражається формулою (1.30):

$$\hat{I}_r = m_r(I_r) \tag{1.30}$$

У якості функції втрат для оцінки моделі доцільно використати індекс структурної подібності, що виражається наступною формулою (1.31):

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{1.31}$$

де x, y – області зображення розміром $N \times N$;

μ_x – середнє x;

μ_y – середнє y;

σ_x – дисперсія *x*;

σ_y – дисперсія *y*;

c_1, c_2 – змінні [31].

Змінні c_1, c_2 визначаються наступною формулою (1.32):

$$c_i = (k_i L)^2, \tag{1.32}$$

де *i* – індекс змінної (1 чи 2);

L – діапазон значень пікселів зображення;

k_1 = 0,01 та k_2 = 0,03 – константи [31].

При наявності декількох моделей для вирішення певної підзадачі необхідно обрати найкращу. Формально це можна записати наступним чином (1.33):

$$M = \{m_1, m_2, ..., m_N\}, \tag{1.33}$$

де *M* – множина моделей відновлення певної підзадачі;

N – кількість моделей.

Вибір найкращої моделі описується наступним чином (1.34):

$$m^* = \arg\min_m L(m) \tag{1.34}$$

Метою роботи є програмна реалізація методів відновлення зображення з використанням генеративно-змагальних мереж.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– підготувати дані для навчання і тестування моделей;

– створити та навчити моделі для вирішення кожної з підзадач;

– створити програмне забезпечення для використовування моделей.

1.5 Висновки за розділом 1

У даному розділі було досліджено предметну область та проаналізовано існуючі методи і приклади вирішення задачі відновлення зображень, виділено їхні основні переваги та недоліки. На основі аналізу було порівняно існуючі методи, найкращим серед який виявився метод із застосуванням генеративно-змагальних мереж. Було формально сформульовано завдання до роботи та необхідні кроки для вирішення поставленого завдання.

2 РОЗРОБКА АРХІТЕКТУРИ МЕРЕЖ І ПРОГРАМИ

Перед початком розроблення системи і архітектури мереж необхідно визначитися із набором програмних засобів. Обрання найефективнішого набору програмних засобів забезпечить швидкість і якість реалізації системи, а також полегшить підтримку та оновлення системи.

До програмних засобів відносяться мова програмування, фреймворк глибинного навчання, середовище розробки.

2.1 Вибір мови програмування

Для вибору мови програмування необхідно розглянути можливі варіанти, їхні переваги та недоліки і провести порівняльну характеристику відносно поставленої задачі. Нижче розглянуто такі популярні мови програмування, як C++, Java та Python.

C++ – це високорівнева статично типізована мова програмування, що підтримує об’єктно-орієнтовану, процедурну та узагальнену парадигми програмування [32].

До переваг C++ можна віднести:

– швидкість виконання програм;

– підтримка різних технологій і парадигм програмування, наприклад, ООП, директивне програмування, метапрограмування;

– можливість працювати з пам’яттю ефективно на низькому рівні [32].

Недоліками C++ є:

– відсутність збирача сміття і автоматичного контролю за життям об’єктів, вказівників і посилань;

– складність у вивчені;

– порівняно низька популярність у рішенні задач аналізу даних [32].

Java – об’єктно-орієнтована мова програмування високого рівня, що виконується на віртуальній машині Java [33].

Переваги Java:

– відносна простота у вивченні;

– платформна незалежність;

– наявність збирача сміття;

– багатопоточність [33].

Недоліки Java:

– високе споживання пам’яті;

– відсутність підтримки низькорівневого програмування;

– відсутність контролю за збирачем сміття;

– невелика кількість бібліотек для аналізу даних [33].

Python – високорівнева інтерпретована мова програмування, з динамічною типізацією і підтримкою об’єктно-орієнтованого програмування [34].

Переваги Python:

– простота у вивченні;

– читабельність;

– велика кількість різних бібліотек, особливо для аналізу даних;

– автоматичне виділення пам’яті [34].

Недоліки Python:

– повільність;

– відсутність контролю за пам’яттю;

– динамічна типізація [34].

Таким чином, кожна з мов підходить для вирішення певної задачі і повністю реалізує всі необхідне інструменти. Оскільки поставлена задача належить до аналізу даних і машинного навчання, доцільно звернути увагу на мову, що має для цього найкращі інструменти.

На основі аналізу мов програмування можна зробити їхню порівняльну характеристику (табл. 2.1).

Таблиця 2.1 – Порівняльна характеристика мов програмування

Критерій	C++	Java	Python
Кросплатформенність	Так	Так	Так
Типізація	Статична	Статична	Динамічна
Швидкість виконання	+++	++	+
Простота і читабельність	+	++	+++
Наявність сторонніх бібліотек	+	++	+++
Підтримка аналізу даних	+	+	+++

Отже, для вирішення поставленої задачі доцільно вибрати Python, оскільки ця мова програмування проста у використанні, має велику кількість сторонніх бібліотек і бібліотек для аналізу даних та машинного навчання. Незважаючи на те, що це досить повільна мова програмування, багато фреймворків машинного навчання є обгорткою над бібліотеками, що написані мовами C, C++. Це означає, що код буде виконуватись значно швидше, залишаючись при цьому простим і читабельним.

2.2 Вибір фреймворку глибинного навчання

Оскільки основна частина роботи – створення ШНМ, необхідно вибрати найефективніший фреймворк для глибинного навчання.

Tensorflow – це відкрита бібліотека для машинного та глибинного навчання, розроблена у компанії Google [35].

Переваги Tensorflow:

– можливість створити модель будь-якої складності;

– швидкість навчання мереж;

– велика спільнота розробників та користувачів;

– сумісність з різними мовами програмування;

– детальна документація;

– візуалізація;

– підтримка операцій на декількох GPU;

– постійна підтримка та оновлення [35].

Недоліки:

– складність у написанні коду для створення моделей;

– часте оновлення, що ускладнює підтримку старого коду;

– низька зрозумілість повідомлень про помилки [35].

Keras – це бібліотека глибинного навчання, що спроєктована поверх Tensorflow [36].

Переваги Keras:

– простота у написанні коду для створення моделі;

– велика спільнота розробників та користувачів;

– детальна документація;

– наявність попередньо навчених моделей [36].

Недоліки Keras:

– обмеження у гнучкості;

– швидкість нижча, порівняно з Tensorflow [36].

PyTorch – відкрита бібліотека глибинного навчання, що є версією бібліотеки Torch. Розробляється у Facebook [37].

Переваги PyTorch:

– простота у вивченні;

– гнучкість у створенні моделей;

– простота у відлагодженні [37].

Недоліки PyTorch:

– не оптимізована для використання моделей у виробництві, оскільки розроблюється як інструмент для досліджень;

– порівняно менша спільнота розробників і користувачів;

– відсутність власних інструментів для візуалізації процесу навчання [37].

Theano – відкрита бібліотека для маніпулювання і обчислення математичних виразів, особливо у вигляді матриць. Також застосовується для машинного і глибинного навчання [38].

Переваги Theano:

– оптимізовано під матричні операції;

– паралелізм [38].

Недоліки Theano:

– можливе використання лише одного GPU;

– складність у написанні коду створення моделей;

– низька зрозумілість повідомлень про помилки;

– порівняно менша спільнота розробників і користувачів;

– невелика кількість оновлень [38].

Caffe – відкрита бібліотека для глибинного навчання, що була створена для розробки ЗНМ [39].

Переваги Caffe:

– оптимізовано для створення ЗНМ і обробки зображень;

– зрозумілий інтерфейс для Python;

– простота у створені моделей [39].

Недоліки Caffe:

– відсутність оновлень;

– не оптимізовано для рекурентних нейронних мереж;

– складність у використанні GPU [39].

ONNX – відкрита бібліотека для глибинного навчання [40].

Переваги ONNX:

– підтримка різних мов програмування;

– підтримка інших фреймворків глибинного навчання [40].

Недоліки ONNX:

– недостатньо документації;

– невелика кількість прикладів;

– незрілість програмних засобів [40].

На основі аналізу фреймворків глибинного навчання можна зробити їхню порівняльну характеристику (табл. 2.2).

Таблиця 2.2 – Порівняльна характеристика фреймворків глибинного навчання

Критерій	Tensorflow	Keras	PyTorch	Theano	Caffe	ONNX
Швидкодія	++++	+++	+++	++	++	++
Інтуїтивна зрозумілість	++	++++	+++	+	++	++
Документація	++++	+++	+++	++	+	+
Спільнота	++++	+++	++	+	+	+
Підтримка GPU	++++	+++	+++	+	+	+
Гнучкість	+++	++	++	+	+	+
Відлагодження	++	+	+++	+	+	+
Оновлення	+++	+++	+++	+	+	++

Таким чином, для створення ШНМ можна обрати фреймворк Tensorflow. Оскільки Keras є обгорткою над Tensorflow і спрощує створення моделей, доцільно використовувати Keras разом з Tensorflow.

2.3 Вибір датасету

Для навчання ШНМ необхідно мати датасет (набір даних). До основних критеріїв вибору датасету можна віднести розмір, репрезентативність і якість даних. Було розглянуто декілька датасетів для виконання завдання, а саме CelebA, Places365, ImageNet.

Репрезентативність – це властивість, при якій вибірка даних містить усі типові випадки, подані у генеральній сукупності, та їхні частоти у вибірці близькі до частот у генеральній сукупності [41].

CelebA – датасет, що містить зображення облич знаменитостей. Крім того, кожне фото має 45 атрибутів, серед яких координати певних точок на обличчі та бінарні ознаки (стать, форма обличчя, наявність бороди, тощо). Датасет містить близько 200000 зображень [42]. Основний недолік – відсутність репрезентативності. Модель, навчена на цьому датасеті, зможе видавати якісні результати лише на зображеннях облич людей, тому використовувати цей набір даних доцільно лише для вирішення вузьких задач, що пов’язані, наприклад, із розпізнаванням облич.

На рис. 2.1 зображено приклад даних датасету CelebA.

Рисунок 2.1 – Приклад даних датасету CelebA [42]

Places365 – датасет, що містить зображення різноманітних місць. Загалом, у датасеті містяться більш ніж 10 мільйонів зображень, що підпадають близько до 400 класів (пляж, ліс, магазин, вулиця, тощо) [43]. Оскільки більшість класів пов’язана з природою, модель колоризації буде

схильна обирати відтінки синього (колір неба) і зеленого (колір трави, дерев), що впливає на репрезентативність.

На рис 2.2 зображено приклад даних датасету Places365.

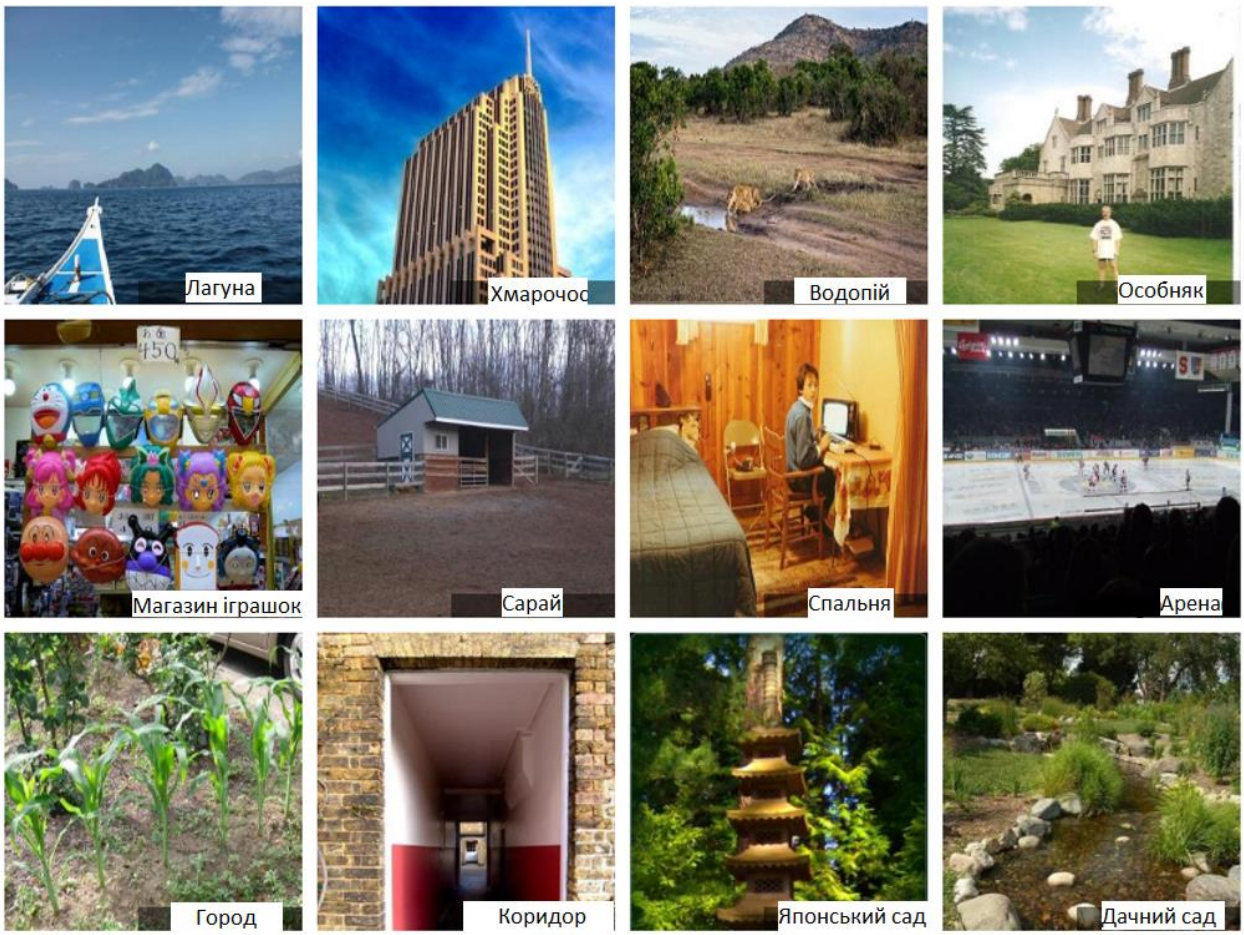


Рисунок 2.2 – Приклад даних датасету Places365 [43]

ImageNet – база даних зображень, що налічує більш ніж 14 мільйонів зображень у більш ніж 20 тисяч категорій. Для завантаження пропонується вибірка із 1.2 мільйона зображень у 1000 категорій [44]. Основний недолік датасету полягає в різноманітті розмірів зображень.

На рис. 2.3 зображено приклад даних датасету ImageNet.

Рисунок 2.3 – Приклад даних датасету ImageNet [44]

У табл. 2.3 наведено порівняльну характеристику датасетів.

Таблиця 2.3 – Порівняльна характеристика датасетів

Критерій	CelebA	Places365	ImageNet
Тематика зображень	Обличчя знаменитостей	Різноманітні місця	Різноманітні об’єкти
Кількість зображень	Більше 200 тисяч	Більше 10 мільйонів	Більше 14 мільйонів
Кількість класів	40 категорій	Більше 400 класів	Більше 20 тисяч класів
Умови використання	Дослідження	Дослідження	Дослідження
Репрезентативність	+	++	+++
Якість даних	+++	++	++

Таким чином, для виконання завдання було обрано датасет ImageNet, а саме вибірку із нього, що містить 50 тисяч зображень (45 тисяч – навчальна вибірка і 5 тисяч – тестова вибірка).

2.4 Обробка даних

Перед початком створення архітектури моделі необхідно зібрати дані і підготувати їх для навчання моделі.

Усі зображення із обраної вибірки датасету ImageNet були масштабовані до розміру 256 на 256 пікселів. Крім того, значення пікселів було масштабовано до інтервалу [-1;1].

Для задачі колоризації була обрана колірна модель Lab. Канал L (освітлення) використовується у якості вхідного чорно-білого зображення. Мережа бере на вхід канал L і повертає згенеровані канали ab. Потім, згенеровані канали ab порівнюються зі справжніми каналами ab. Колірна модель Lab була обрана, оскільки канали ab повністю кодують інформацію про колір зображення і є більш стійкими до невеликих хвилювань у кольорі і освітленні, ніж канали у колірній моделі RGB [20].

Для задачі шумозниження до вхідного зображення було додано випадковий шум із нормальним розподілом з нульовим математичним сподіванням і дисперсією, що дорівнює 10. Модель бере на вхід зображення із шумом і повертає знешумлене зображення. Потім, знешумлене зображення порівнюється із оригінальним зображенням.

Для задачі підвищення роздільної здатності вхідне зображення було зменшено у два рази (128 на 128 пікселя). Модель бере на вхід зменшене зображення і повертає збільшене зображення. Потім, збільшене зображення порівнюється із оригінальним зображенням.

2.5 Архітектура компонентів програми

2.5.1 Вибір архітектури генератора

Для задачі колоризації генератор є мережею U-Net із попередньо навченою основою ResNet50.

ResNet50 – один з видів залишкових ЗНМ. Залишкові ЗНМ були створені через те, що звичайні глибокі ЗНМ починають втрачати точність при зростанні кількості згорткових шарів. Завдяки використанню пропускних зв’язків, під час яких інформація з виходів нижніх шарів безпосередньо додається до входів верхніх шарів, підвищується точність ЗНМ і зменшується втрата корисної інформації [45]. Блоки, що використовують пропускні зв’язки, називаються залишковими. Якщо на шляху пропускного зв’язку знаходиться згортковий шар, то такий блок називається згортковим.

Для підвищення точності і зменшення часу навчання була взята попередньо навчена основа генератора ResNet50. Такий підхід називається передавальним навчанням, під час якого знання, набуті ШНМ у вирішенні однієї задачі, застосовуються до вирішення схожої задачі. Оскільки взята мережа ResNet50 була навчена для класифікації зображень, можна припустити, що мережа має певні знання про кольори, форми і текстури об’єктів.

На рис. 2.4 зображено архітектуру генератора для вирішення задач колоризації.

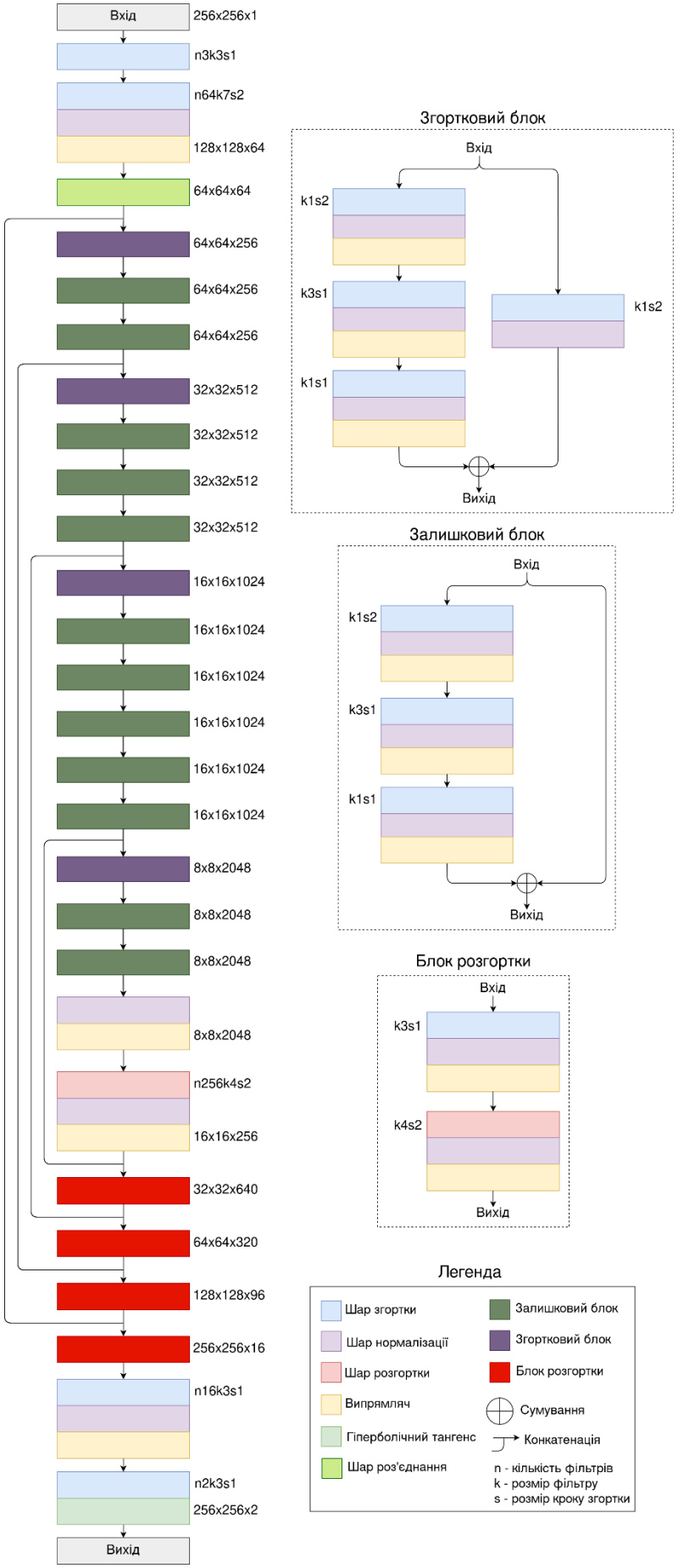


Рисунок 2.4 – Генератор для колоризації

Генератор використовує два види функцій активації, а саме ReLU (випрямляч) для прихованих шарів і tanh (гіперболічний тангенс) для вихідного шару. Нижче наведено формулу (2.1) випрямляча:

$$f(x) = \max(0, x), \tag{2.1}$$

де *f(x)* – функція ReLU (rectified linear unit);

x – аргумент функції.

Гіперболічний тангенс має наступну формулу (2.2):

$$\tanh(x) = \frac{e^{x} = e^{-x}}{e^{x} + e^{-x}}, \tag{2.2}$$

де tanh(x) – гіперболічний тангенс;

e – експонента;

x – аргумент функції.

Для вирішення задачі шумозниження було використано генератор із дослідження [27] (рис. 2.5).

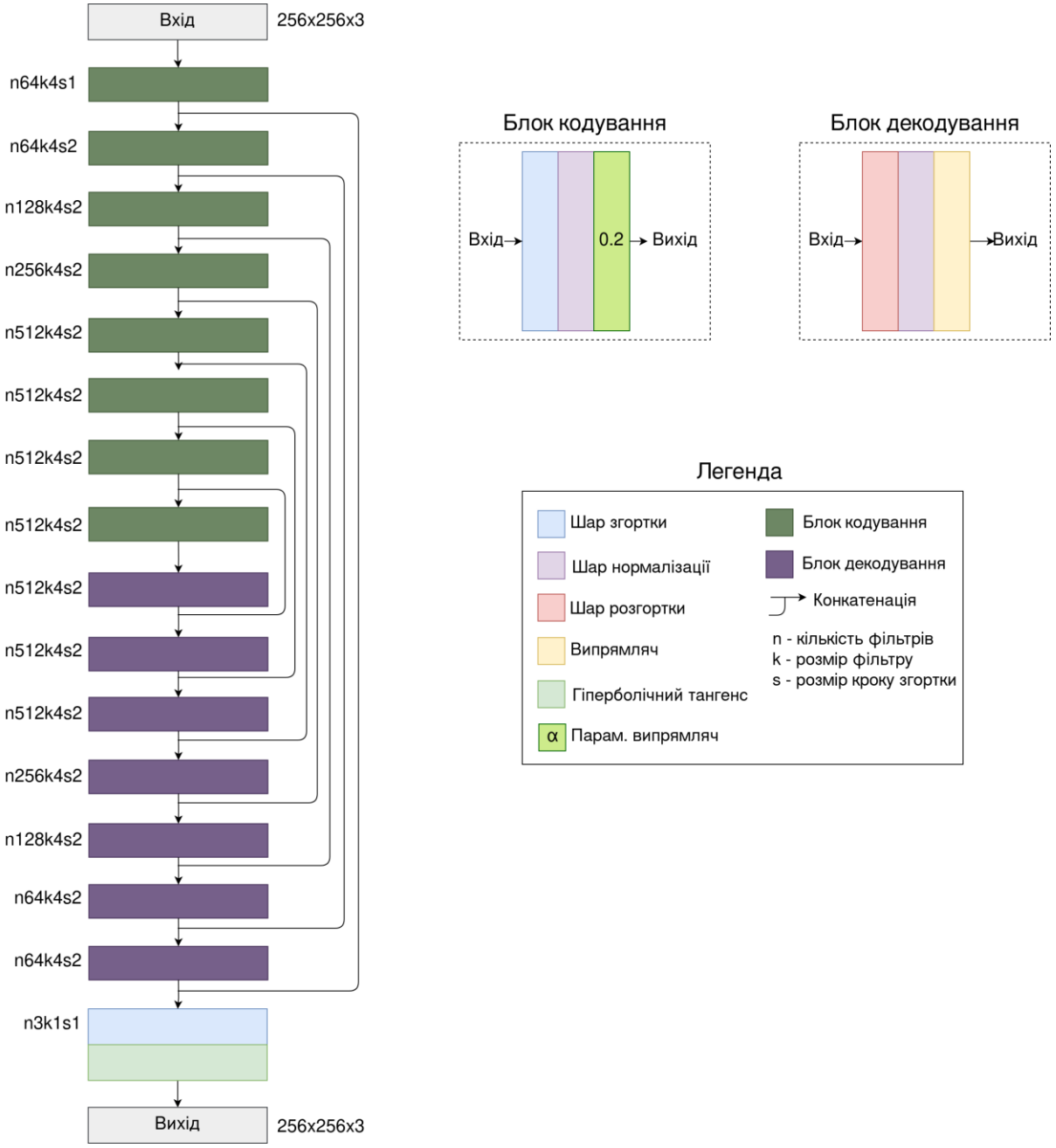


Рисунок 2.5 – Генератор для задачі шумозниження

Для вирішення задачі збільшення роздільної здатності було використано видозмінений генератор із дослідження [30] (рис. 2.6).

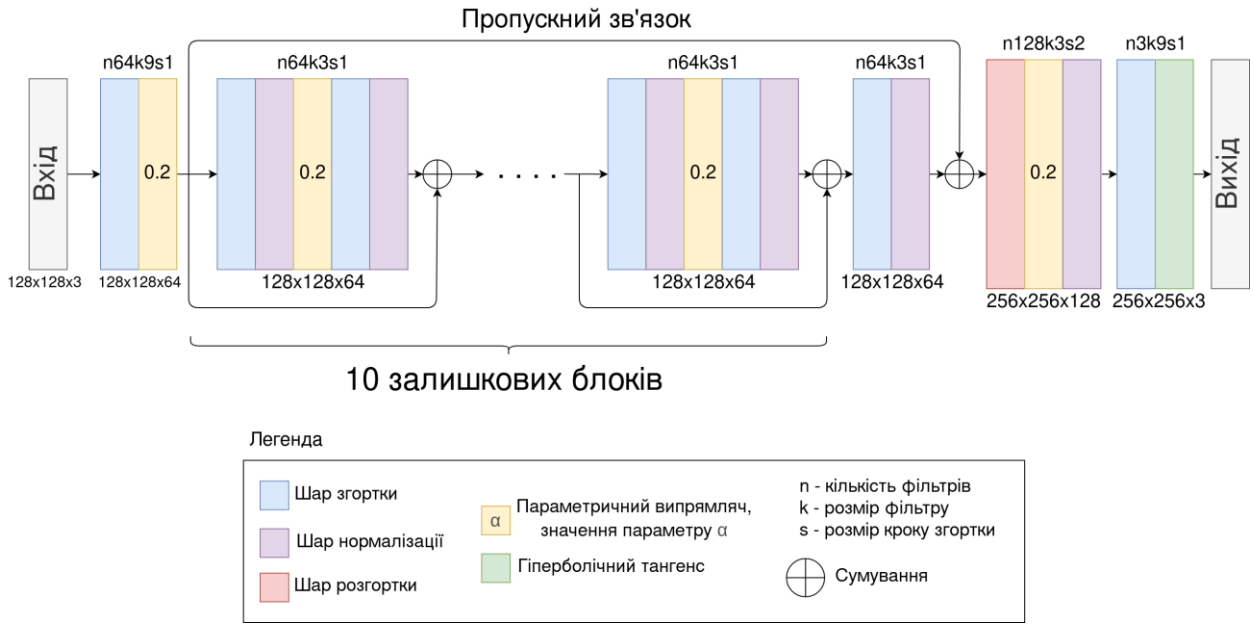


Рисунок 2.6 – Генератор для задачі збільшення роздільної здатності

Даний генератор замість звичайного випрямляча використовує параметричний випрямляч у якості функції активації прихованих шарів. Параметричний випрямляч має наступний вигляд (2.3):

$$f(x)=\begin{cases}0, x \le 0\\ ax, x > 0\end{cases}, \tag{2.3}$$

де *f(x)* – параметричний випрямляч;

a – параметр випрямляча;

x – аргумент функції.

2.5.2 Вибір архітектури дискримінатора

Для кожної з трьох ГЗМ для дискримінатора була обрана архітектура PatchGAN [46].

При використанні звичайного згорткового класифікатора у якості дискримінатора класифікація здійснюється на всьому зображенні. Це може призвести до зниження якості генератора, якому буде легше обманювати

дискримінатор. Для вирішення цієї проблеми застосовується дискримінатор PatchGAN, що аналізує не все зображення цілком, а кожен сегмент зображення розміром $N \times N$. Після аналізу кожного сегменту результати усереднюються. Такий підхід дозволяє дискримінатору краще «роздивитися» зображення, що у свою чергу змушує генератор видавати більш якісні дані [46].

На рис. 2.7 зображено архітектуру дискримінатора.

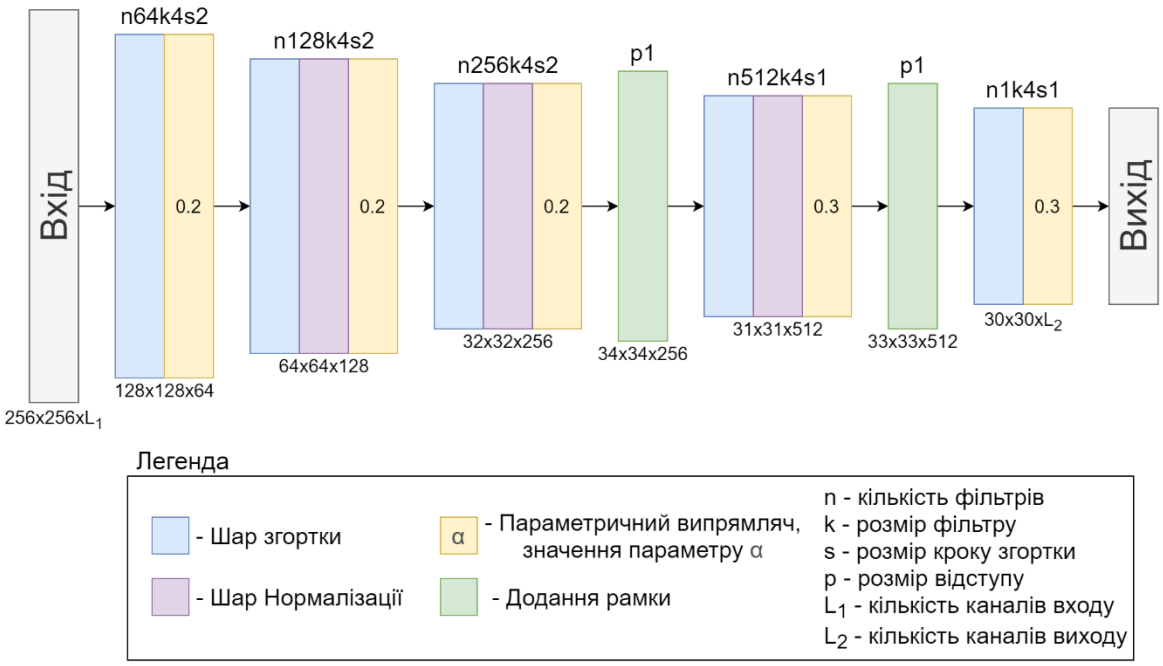


Рисунок 2.7 – Архітектура дискримінатора

2.5.3 Архітектури системи, що проєктується

Архітектура розроблюваної системи складається з наступних компонентів:

– три ГЗМ (колоризація, шумозниження і збільшення роздільної здатності), що оброблюють вхідні пошкоджені зображення і генерують відновлені зображення;

– ядро системи, у якому відбувається конвертація вхідних даних від користувача і вихідних даних від ГЗМ у необхідні формати, а також

попередня обробка вхідних даних (конвертація у чорно-білу колірну модель, додання шуму, зменшення роздільної здатності);

– інтерфейс користувача, за допомогою якого здійснюється зв’язок між ядром системи і користувачем.

2.6 Висновки за розділом 2

У даному розділі було проаналізовано мови програмування, фреймворки глибинного навчання та існуючі датасети. На основі порівняльних характеристик було обрано мову програмування Python, фреймворк глибинного навчання Tensorflow із обгорткою Keras і вибірку із датасету ImageNet.

Було описано методи обробки даних для вирішення поставлених задач.

Крім того, було обґрунтовано вибір архітектур генератора і дискримінатора.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вибір гіперпараметрів для навчання мереж

Перед початком навчання ШНМ необхідно визначитися з набором необхідних гіперпараметрів.

Гіперпараметри – це параметри, значення яких задаються для контролю процесу навчання [47].

До гіперпараметрів навчання відносять наступне:

– кількість епох;

– розмір пакету даних;

– крок навчання;

– алгоритм оптимізації;

– функція втрат.

Епоха – проходження всього датасету через ШНМ один раз для навчання її ваг [48]. Для всіх мереж, що розроблюються у роботі, кількість епох становить 40.

Пакет даних – кількість екземплярів, що одночасно проходять через ШНМ для навчання її ваг [48]. Виходячи з обчислювальної потужності комп’ютера, на якому відбувалось навчання, розмір пакету даних для всіх мереж, що розроблюються у роботі, становить 16 зображень. Отже, враховуючи розмір навчальної вибірки, кількість пакетів даних (ітерацій) за весь час навчання становить 112,5 тисяч.

3.1.1 Функції втрат генератора і дискримінатора

Оцінкою роботи ШНМ є її помилка – міра різниці між справжнім та спрогнозованим виходом. Функція втрат – функція, що використовується для розрахування помилки ШМН [49]. Вибір функції втрат є дуже важливим етапом, оскільки від цього залежить якість навчання ШНМ.

Для генератора функція втрат має наступний вигляд (3.1):

$$L_G = \alpha \log(1 - D(G(x))) + \beta L_c(y, G(x)), \quad (3.1)$$

де D – дискримінатор;

G – генератор;

x – вхідне зображення;

y – вихідне зображення;

$G(x)$ – згенероване зображення;

L_c – функція втрат змісту;

α, β – параметри ваг.

У випадку генератора для колоризації вхідне зображення x – це чорно-біле зображення (канал L у кольоровій моделі Lab), вихідне зображення y – канали ab у кольоровій моделі Lab, згенероване зображення $G(x)$ – згенеровані канали ab.

У випадку генератора для шумозниження вхідне зображення x – це зашумлене зображення, вихідне зображення y – справжнє зображення, згенероване зображення $G(x)$ – знешумлене зображення.

У випадку генератора для збільшення роздільної здатності вхідне зображення x – зменшене зображення, вихідне зображення y – справжнє зображення, згенероване зображення $G(x)$ – збільшене зображення.

Для задач колоризації і шумозниження параметри α і β дорівнюють 1 і 100 відповідно. Для задачі збільшення роздільної здатності параметри α і β дорівнюють 0,001 і 1 відповідно.

Для задач колоризації і шумозниження функція втрат змісту – це середня абсолютна помилка, що має наступний вигляд (3.2):

$$MAE(I,\hat{I}) = \frac{1}{HW}\sum_{i=1}^{W}\sum_{j=1}^{H}\left|I(i,j) - \hat{I}(i,j)\right|, \quad (3.2)$$

де *H, W* – висота і ширина зображення;

I – вихідне зображення;

$\hat{I}$ – відновлене зображення;

m – модель відновлення зображення;

i, j – координати пікселя зображення.

Для задачі збільшення роздільної здатності функція втрат змісту – це функція втрат VGG, що описана у формулі (1.27).

Для дискримінатора функція втрат має наступний вигляд (3.3):

$$L_D = \log(D(y)) + \log(1 - D(G(x))), \quad (3.3)$$

де *D* – дискримінатор;

G – генератор;

x – вхідне зображення;

y – вихідне зображення;

G(x) – згенероване зображення.

3.1.2 Вибір алгоритму оптимізації

Алгоритм оптимізації – метод оптимізації параметрів ШНМ для зменшення функції втрат. Від вибору алгоритму оптимізації залежать швидкість і якість навчання ШНМ.

Градієнтний спуск (GD) – найпростіший метод оптимізації. Формально, цей метод можна записати наступним чином (3.4):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta), \tag{3.4}$$

де θ – параметри (ваги) ШНМ;

η – крок навчання;

$J(\theta)$ – функція втрат;

$\nabla(J(\theta))$ – градієнт функції втрат [50].

Градієнтний спуск мінімізує функцію втрат $J(\theta)$ шляхом зміни параметрів ШНМ у напрямку, протилежному до градієнта функції втрат $\nabla(J(\theta))$. Крок навчання η визначає швидкість наближення до мінімуму [50].

Переваги:

– простий у реалізації;

– інтуїтивно зрозумілий.

Недоліки:

– може застрягнути у локальному мінімумі;

– потребує розрахунку градієнтів для всього датасету, що займає багато часу та пам’яті [51].

Для подолання останнього недоліку використовується модифікація градієнтного спуску, що називається стохастичний градієнтний спуск (SGD). Замість розрахунку градієнту на всьому датасеті за один раз, розрахунок градієнтів і оновлення параметрів ШНМ відбувається на кожному екземплярі датасет [50]. Формально це можна записати наступним чином (3.5):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}), \tag{3.5}$$

де $x^{(i)}$ – ознаки i-го екземпляру;

$y^{(i)}$ – вихід i-го екземпляру [50].

Переваги:

– швидкість;

– потребує менше пам’яті.

Недоліки:

– висока варіативність оновлення параметрів ШНМ, що зменшує стабільність навчання;

– може «перестрибнути» глобальний мінімум [51].

Пакетний градієнтний спуск (minibatch GD) поєднує знаходиться між звичайним і стохастичним градієнтними спусками. Розрахунок градієнтів відбувається на невеликому пакеті даних [50]. Формально це можна записати наступним чином (3.6):

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}), \tag{3.6}$$

де n – розмір пакету даних;

$x^{(i:i+n)}$, $y^{(i:i+n)}$ – ознаки і виходи екземплярів у пакеті даних [50].

Переваги:

– швидкість;

– менше варіації у оновленні параметрах ШНМ;

– невеликий об’єм пам’яті.

Недоліки:

– може застрягнути у локальному мінімумі;

– складність вибору оптимального кроку навчання [51].

Методам градієнтного спуску складно покинути область біля локального мінімуму. Метод Momentum (імпульс) допомагає прискорити градієнтний спуск у відповідному напрямі та зменшити коливання [50]. Математично, цей метод імпульс описується наступним чином (3.7):

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta), \tag{3.7}$$

де v – імпульс;

t – момент часу;

γ – коефіцієнт імпульсу [50].

Оновлення параметрів ШНМ відбувається наступним чином (3.8):

$$\theta = \theta - v_t, \tag{3.8}$$

Переваги:

– зменшує коливання і варіативність зміни параметрів;

– сходиться швидше, ніж градієнтний спуск.

Недоліки:

– складність вибору значення коефіцієнту γ [51].

Один з недоліків вищеописаних методів полягає в тому, що вони використовують один незмінний крок навчання для всіх параметрів. Алгоритм Adagrad вирішує цю проблему наступним чином: крок навчання адаптується під параметри, роблячи більші зміни для менш інформативних і затребуваних параметрів і менші зміни для більш інформативних і затребуваних параметрів. Цей підхід також підходить до розріджених даних [50].

Математично це можна представити наступним чином (3.9):

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}), \tag{3.9}$$

де $g_{t,i}$ – це градієнт функції втрат відносно параметра θ_i у момент часу t [50].

Оновлення параметрів відбувається наступним чином (3.10):

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} \cdot g_{t,i}, \tag{3.10}$$

де G_t – діагональна матриця, де кожен елемент i,i – це сума квадратів градієнтів відносно параметра θ_i до моменту часу t[11];

ε – згладжувальний коефіцієнт для уникання ділення на нуль [50].

Переваги:

– адаптивний крок навчання для кожного параметра;

– відсутність потреби задавати значення кроку навчання вручну;

– висока ефективність з розрідженими даними.

Недоліки:

– великі обчислювальні витрати;

– крок навчання постійно зменшується, що сповільнює навчання [51].

Основний недолік алгоритму Adagrad в тому, що крок навчання постійно зменшується, що приводить до сповільнення навчання. Для вирішення цієї проблеми у методі RMSprop (і у дуже схожому методі AdaDelta) використано ідею рухомого вікна розміру *w*, у якому накопичуються останні градієнти. Замість того, щоб зберігати останні *w* квадрати градієнтів, сума градієнтів рекурсивно визначається як рухоме середнє *w* квадратів градієнтів [50].

Математично це можна представити наступним чином (3.11):

$$E[g^2] = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2, \tag{3.11}$$

де $E[g^2]_t$ – рухоме середнє у момент часу t;

γ – імпульс [50].

Замінивши діагональну матрицю Gt у формулі (3.10) отримаємо (3.12):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t \tag{3.12}$$

Знаменник – це корінь із середнього квадратів градієнтів (3.13):

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon} \tag{3.13}$$

Переваги:

– відсутність затухання кроку навчання.

Недоліки:

– значні обчислювальні витрати [51].

Алгоритм Adam поєднує у собі ідею імпульсу і рухомого середнього квадратів [50].

Рухоме середнє накопичується наступним чином (3.14):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \tag{3.14}$$

де β_1 – швидкість затухання [50].

Нецентрована дисперсія розраховується наступним чином (3.15):

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{3.15}$$

де β_2 – швидкість затухання [50].

Оскільки m_t і v_t ініціалізуються нулями, було помічено, що вони довго накопичуються, особливо при малих значення швидкостей затухання [50].

Таким чином, скореговане середнє визначається наступним чином (3.16):

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3.16}$$

Скорегована дисперсія визначається наступним чином (3.17):

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.17}$$

Параметри ШНМ у алгоритмі Adam оновлюються наступним чином (3.18):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t \tag{3.18}$$

Переваги:

– швидке сходження;

– відсутність затухаючого кроку навчання.

Недоліки:

– значні обчислювальні витрати [51].

На основі аналізу алгоритмів оптимізації можна зробити їхню порівняльну характеристику (табл. 3.1).

Таблиця 3.1 – Порівняльна характеристика алгоритмів оптимізації

Критерій	GD	SGD	Minibatch GD	Momentum	Adagrad	RMSprop	Adam
Швидкість	+	+++	++	+++	+++	+++	++++
Точність сходження	++	+	++	++	++	+++	++++
Обчисл. витрати	++++	+	++	++	+++	+++	++++
Складн. налаштув.	+++	+++	+++	+++	++	++	++

Таким чином, у якості алгоритму оптимізації для всіх ШНМ, які розроблюються у роботі, було обрано алгоритм Adam. Значення кроку навчання η та швидкостей затухання β_1 та β_2 дорівнюють 0,0002, 0,9 і 0,999 відповідно.

3.2 Аналіз результатів навчання мереж

3.2.1 Результати колоризації

ГЗМ навчалася 88 епох (250000 ітерацій по 16 зображень у пакеті). На рис. 3.1 зображено результати колоризації.

Рисунок 3.1 – Результаті колоризації

З результатів роботи видно, що ГЗМ навчилася використовувати такі базові кольори, як зелений і синій, особливо на зображеннях з пейзажами. Крім того, ГЗМ може колоризувати зображення людей, достатньо великі і виразні об’єкти.

З іншої сторони, ГЗМ схильна робити помилки у наступних випадках:

– наявність малих деталей. Це пов’язано з тим, що малі об’єкти на зображенні несуть менше інформації, ніж великі. Таким чином, ГЗМ складно виділити залежності між кольором і такими ознаками об’єктів, як форма, контур, текстура та ін.;

– невелика кількість зображень з певним кольором чи об’єктом. У такому випадку, у ГЗМ недостатньо знань, що призводить до неправильної колоризації;

– варіативність у освітленні. ГЗМ залежна від різної освітленості вхідного зображення, що призводить до різної колоризації одного об’єкту.

Розрахована метрика SSIM дорівнює 0,79.

3.2.2 Результати шумозниження

ГЗМ для шумозниження навчалася 44 епохи (125000 ітерацій по 16 зображень у пакеті).

ГЗМ було порівняно з такими алгоритмами, як медіанний фільтр, білатеральний фільтр і метод нелокальних середніх.

Результати шумозниження зображено на рис. 3.2.

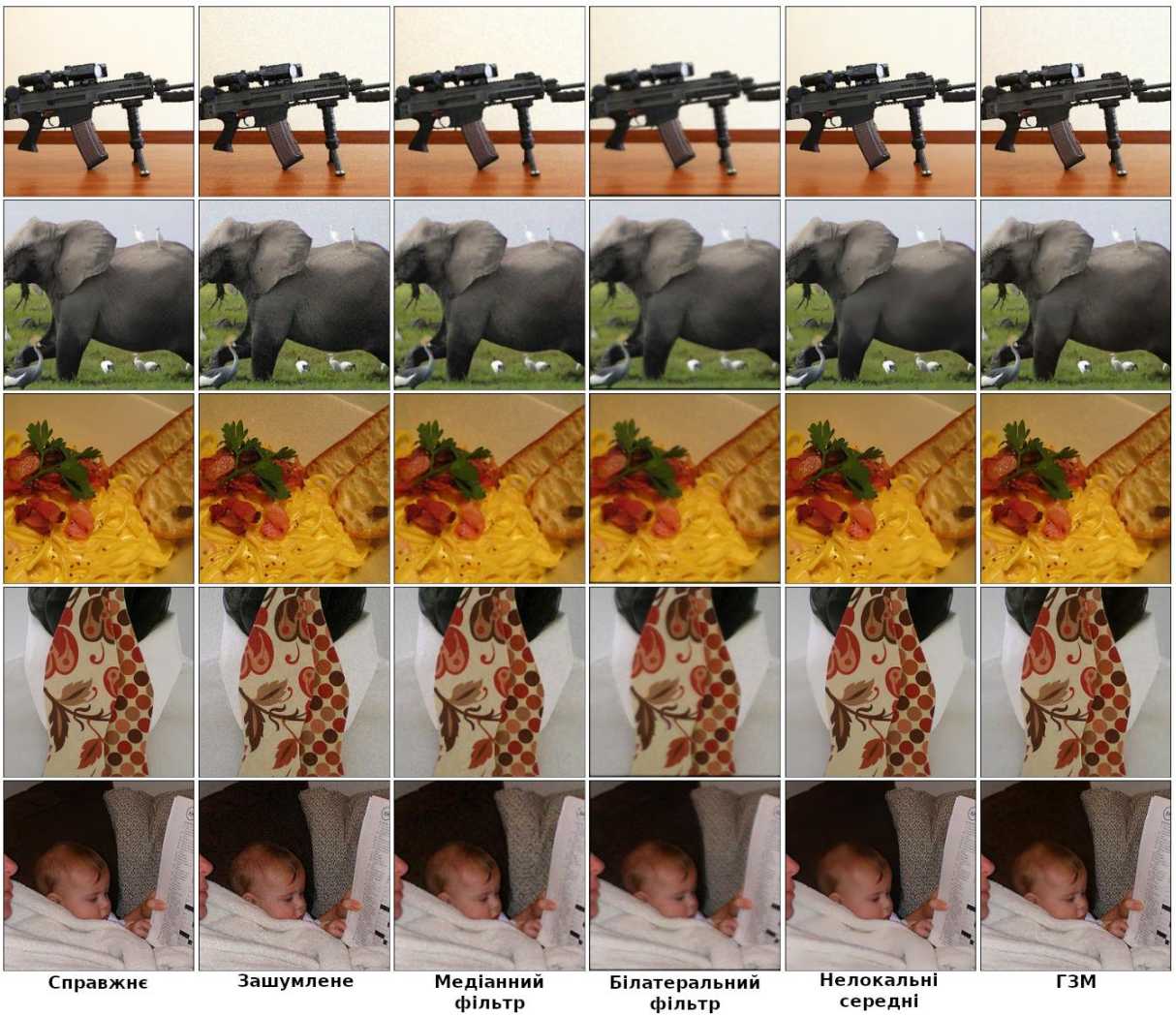


Рисунок 3.2 – Результати шумозниження

У табл. 3.2 наведено значення метрики SSIM методів, що використовувалися для порівняння результатів шумозниження.

Таблиця 3.2 – Значення метрики SSIM методів шумозниження

Метрика	Медіанний фільтр	Білатеральний фільтр	Нелокальні середні	ГЗМ
SSIM	0,565	0,273	0,698	0,749

За результатами порівняння видно, що ГЗМ знижує шум на зображеннях краще, ніж методи фільтрації. Головний недолік – розмиття невеликих деталей.

3.2.3 Результати збільшення роздільної здатності

ГЗМ для збільшення роздільної здатності навчалася 44 епохи (250000 ітерацій по 8 зображень у пакеті).

ГЗМ було порівняно з такими алгоритмами збільшення роздільної здатності, як метод найближчого сусіда, лінійна інтерполяція і бікубічна інтерполяція.

На рис. 3.3 зображено результати збільшення роздільної здатності.

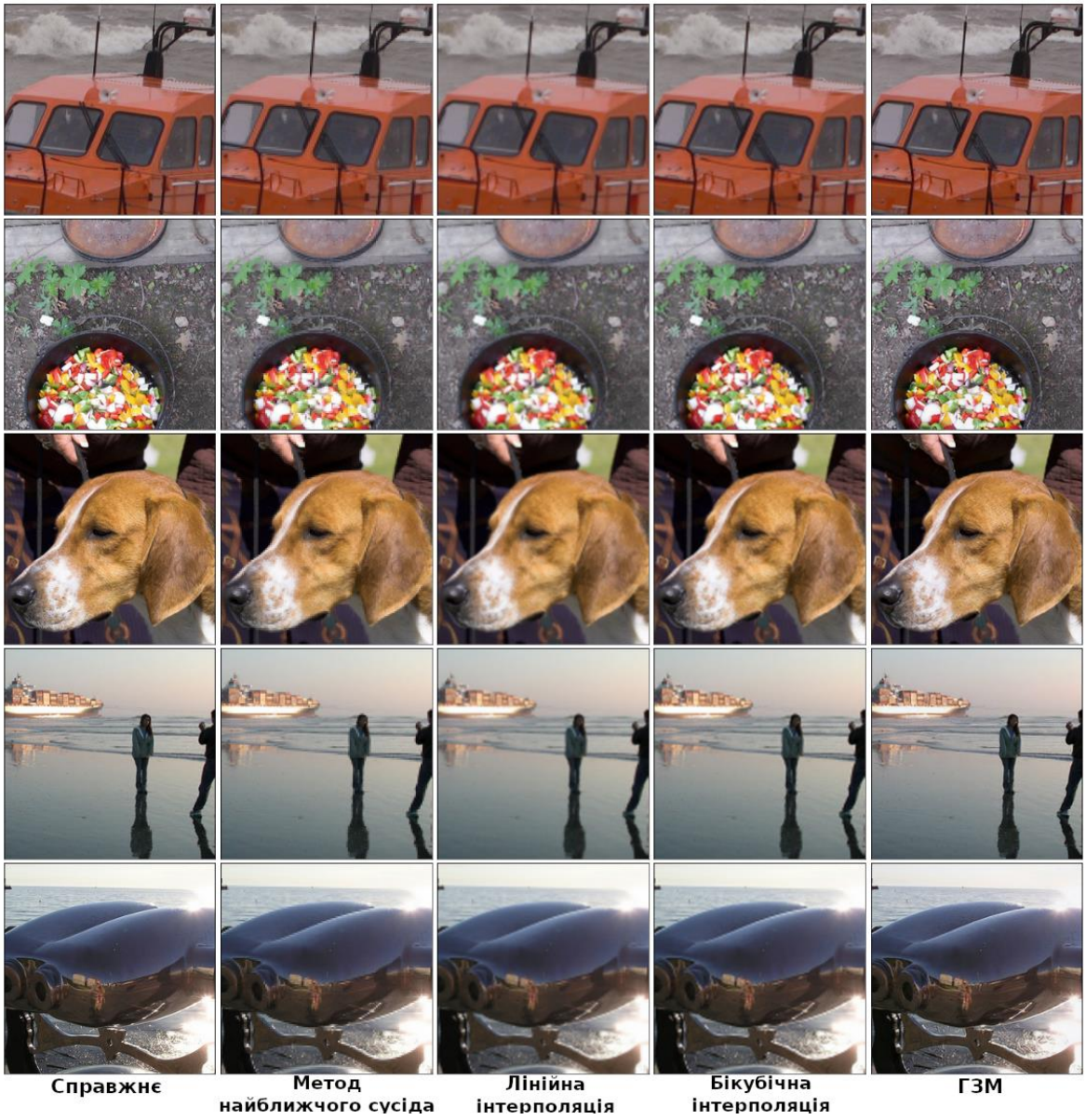


Рисунок 3.3 – Результати збільшення роздільної здатності

У табл. 3.3 наведено значення метрики SSIM методів, що використовувалися для порівняння результатів шумозниження.

Таблиця 3.3 – Значення метрики SSIM методів шумозниження

Метрика	Метод найближчого сусіда	Лінійна інтерполяція	Бікубічна інтерполяція	ГЗМ
SSIM	0,828	0,802	0,833	0,804

Хоча результати збільшення роздільної здатності зображення використовуючи ГЗМ кращі за методи інтерполяції, значення метрики SSIM у ГЗМ нижче. Це пов’язано з тим, що під час обробки зображення ГЗМ деякі відтінки кольорів пошкоджуються. Незважаючи на те, що різниця у кольорах не помітна оку, накопичування цієї різниці по всьому зображенню призводить до зниження метрики SSIM.

3.3 Реалізація компонентів програмного забезпечення

Розробку програмного забезпечення було виконано мовою програмування Python. Такий вибір було зроблено, щоб зменшити кількість додаткових залежностей між розробленими ГЗМ та іншими компонентами системи, оскільки мова програмування Python є основною мовою розробки ШНМ у фреймворках Keras і Tensorflow.

Під час реалізації програмного використовувались методи ООП та структурного програмування. На рис. 3.4 зображено створені модулі системи.

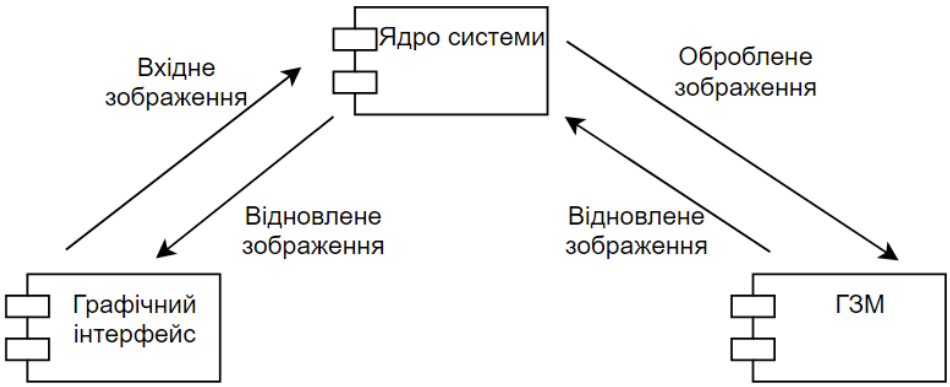


Рисунок 3.4 – Створені модулі програми

За допомогою модулю графічного інтерфейсу здійснюється комунікація між системою і користувачем. У ядрі системи реалізовано попередню обробку і післяобробку зображення, інтерфейси зв'язку з графічним інтерфейсом і створеними ГЗМ.

Основними даними, якими оперують компоненти програми, є зображення. На рис. 3.5 зображено діаграму потоку даних у системі.

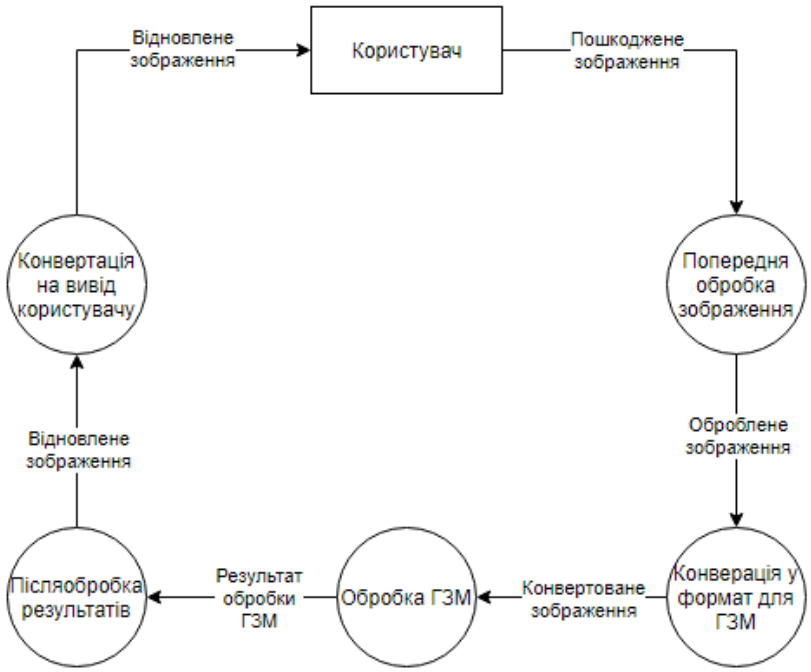


Рисунок 3.5 – Діаграма потоку даних у системі

Джерелом даних у системі є користувач, який завантажує зображення у систему. Зображення проходить попередню обробку, яка включає в себе масштабування, конвертацію у чорно-білу колірну модель і додання шуму (за вибором), нормалізація значень пікселів. Потім оброблене зображення конвертується у необхідний для ГЗМ формат. Вихідні результати ГЗМ проходять через етап післяобробки: комбінація каналів L та ab у випадку колоризації, денормалізація значень пікселів, конвертація у RGB. Вихідне відновлене зображення виводиться користувачу у графічному інтерфейсі.

Під час розробки програмного забезпечення було використано методи ООП та структурного програмування. У табл. 3.2 наведено опис реалізованих класів.

Таблиця 3.2 – Опис створений класів

Клас	Опис
OptionsWindow	Клас графічного інтерфейсу, що реалізує форму, де користувач може обрати необхідні опції обробки
ResultsWindow	Клас графічного інтерфейсу, що реалізує форму виводу результатів
GAN	Абстрактний клас ядра, що слугує базовим класом для обгорток ГЗМ
ColorizationGAN	Клас ядра, що реалізує зв’язок між ядром та ГЗМ для колоризації
DenoisingGAN	Клас ядра, що реалізує зв’язок між ядром та ГЗМ для шумозниження
SRGAN	Клас ядра, що реалізує зв’язок між ядром та ГЗМ для збільшення роздільної здатності

На рис. 3.6 Зображено структурну схему розроблених класів.

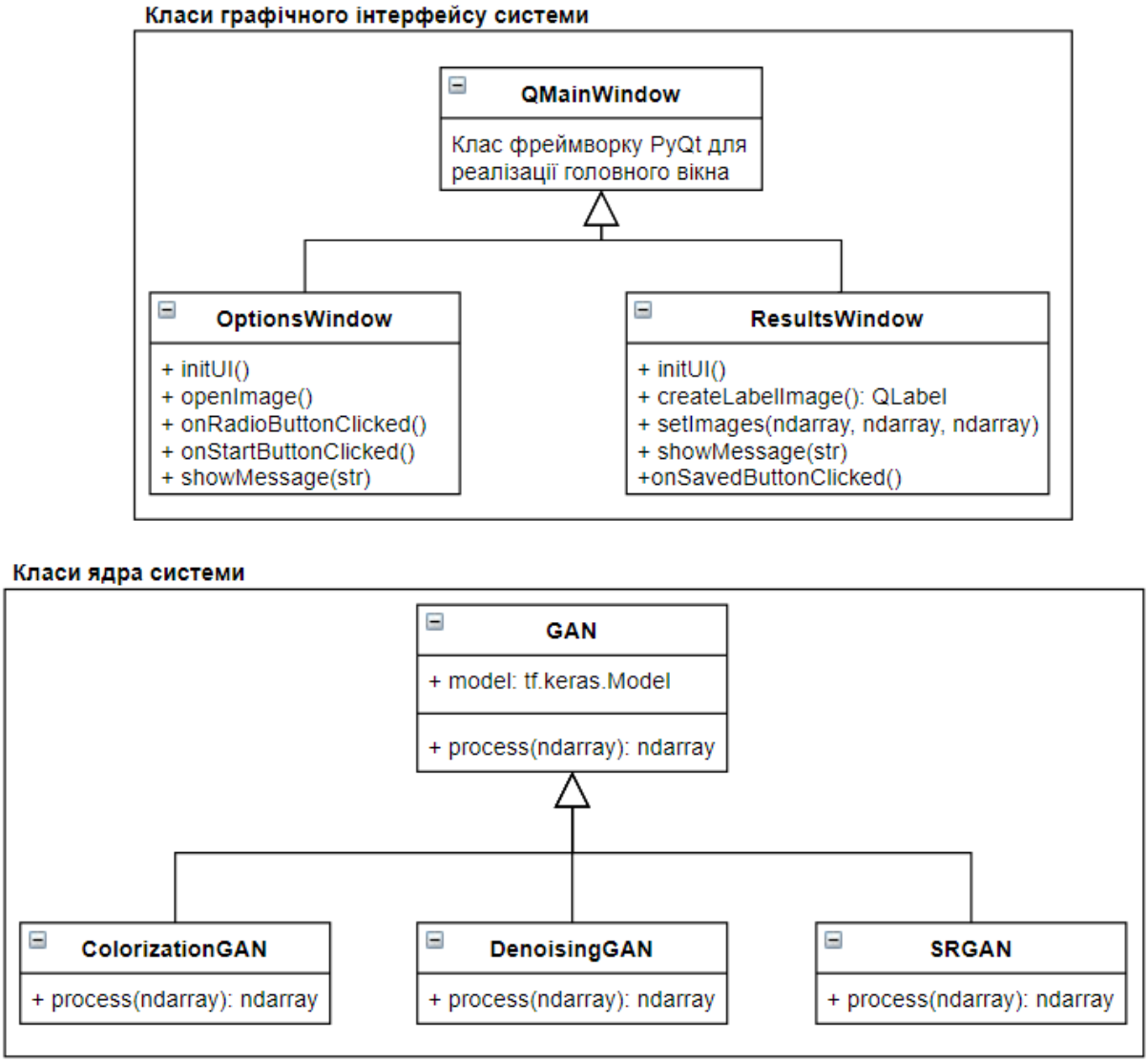


Рисунок 3.6 – Структурна схема розроблених класів

3.4 Висновки за розділом 3

У даному розділі було розглянуто і обґрунтовано вибір гіперпараметів навчання моделі. Було розглянуто та проаналізовано результати роботи створених ГЗМ. Також було наведено кроки для подальшого вдосконалення ГЗМ. Крім того, було описано архітектуру розробленої програми, наведено схеми і діаграми роботи, структури створених класів.

4 ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

4.1 Призначення програми

Розроблений програмний продукт призначений для відновлення пошкоджених зображень. Основними функціями програми є:

– колоризація чорно-білих зображень;

– шумозниження на зображеннях;

– збільшення роздільної здатності зображень.

4.2 Умови виконання програми

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

– 2 ГБ оперативної пам’яті;

– відеокарта з 2 ГБ пам’яті;

– 100 МБ вільного місця на диску;

– процесор з тактовою частотою 2 ГГц;

– платформа x64;

– операційна система Linux;

– встановлені бібліотеки Tensorflow, numpy, scikit-image, PyQt5;

– встановлені драйвера CUDA для виконання розрахунків нейронних мереж за допомогою відеокарти;

– наявність моделей створених ГЗМ на комп’ютері.

4.3 Виконання програми

4.3.1 Запуск програми

Для запуску програми потрібно упевнитися в відповідності характеристик комп’ютера до системних вимог програми. Другим кроком потрібно упевнитися в наявності всіх бібліотек для роботи програми та моделей ГЗМ.

Звертання до програми відбувається через запуск файлу main.py з командного рядка.

Після запуску файлу main.py з’являється головна форма програми, що свідчить про початок роботи з програмою. Головна форма програми зображена на рис. 4.1.

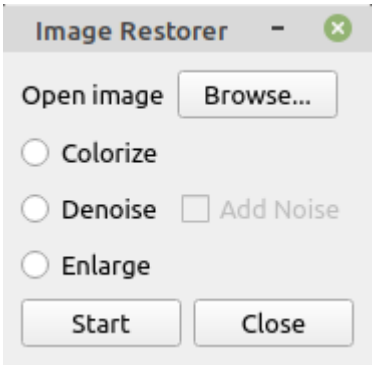


Рисунок 4.1 – Головна форма програми

4.3.2 Робота з програмою

Після запуску програми користувачу необхідно обрати зображення. Для цього необхідно натиснути на кнопку «Browse…» і у вікні файлового менеджеру, що відкриється, обрати необхідне зображення. Потім необхідно обрати один з режимів відновлення: «Colorize» (колоризація), «Denoise» (шумозниження), «Enlarge» (збільшення роздільної здатності). Режим шумозниження має додатковий параметр «Add noise» (додати шум), що використовується для демонстрації шумозниження. Після обрання певного

режиму користувач натискає кнопку «Start» для початку обробки зображення. По завершенню відновлення відкриється вікно результатів (рис. 4.2).

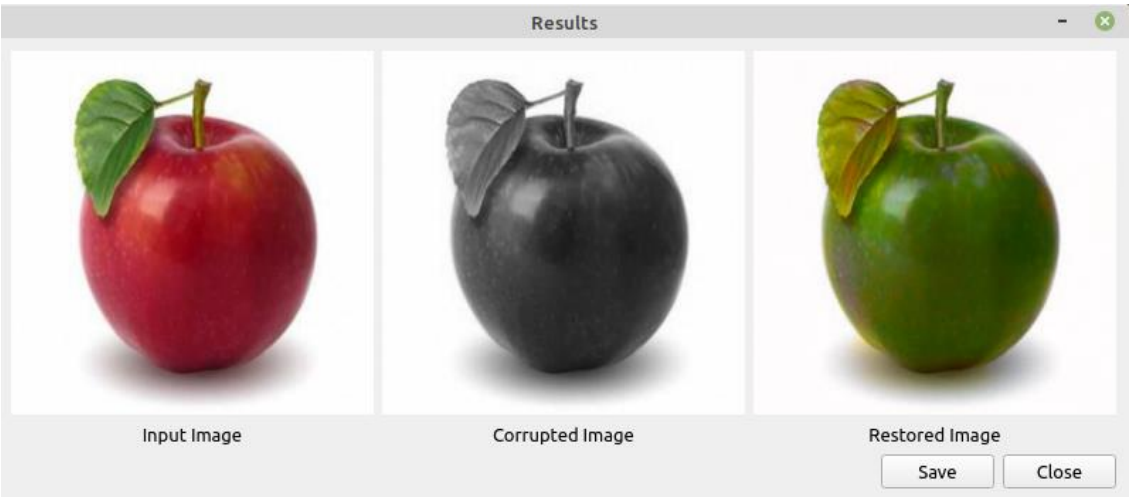


Рисунок 4.2 – Вікно результатів

Користувач може зберегти результат, натиснувши кнопку «Save» або закрити вікно результатів.

4.4 Повідомлення користувачу

Під час роботи програма може видавати певні повідомлення, що сповіщають користувача про помилки або процес виконання роботи.

Програма може видавати наступні повідомлення:

– «Image will be resized to 256×256 pixels» - повідомлення про те, що вхідне зображення буде масштабовано до розміру 256×256 пікселів. Це пов’язано з тим, що ГЗМ були навчені на датасеті, зображення якого мали розмір 256×256 пікселів, тому вхідний шар ГЗМ також має розмір 256×256 пікселів;

– «Image wasn’t selected!» – повідомлення про те, що вхідне зображення не було обрано;

– «Choose operation type!» – повідомлення про необхідність обрати режим відновлення;

– «Image was successfully saved!» – повідомлення про успішне збереження результатів відновлення.

4.5 Висновки за розділом 4

У даному розділі було описано призначення програми, наведено мінімальні вимоги до комп’ютера, на якому запускається програма. Також було описано процес запуску програми, процес роботи із програмою і перелік повідомлень програми. Розроблена програма є інтуїтивно зрозумілою для користувача, вимагає лише базові навички роботи з комп’ютером і повністю задовольняє поставленим вимогам.

ВИСНОВКИ

Під час виконання роботи було розглянуто предметну область, проаналізовано існуючі методи відновлення зображень та інструменти розробки програмного забезпечення.

На основі аналізу існуючих методів було розроблено архітектури і навчено ГЗМ для колоризації, шумозниження і збільшення роздільної здатності.

Виходячи з результатів роботи створених ГЗМ можна виділити наступні кроки підвищення якості відновлення зображень:

– збільшення розміру навчальної вибірки, що покращує її репрезентативність. Таким чином, ГЗМ зможе краще узагальнювати інформацію про такі ознаки зображень, як колір, контури, текстура та ін.;

– використання іншої архітектури ГЗМ. Глибша чи, навпаки, менша архітектура ГЗМ може дати кращі результати, але для цього необхідно проводити відповідні експерименти;

– збільшення кількості епох навчання. Завдяки цьому ГЗМ має більше часу узагальнити вхідні дані та структурувати набуті знання.

Було створено програму, яка дозволяє користувачу колоризувати, знижувати шум і збільшувати роздільну здатність зображення за допомогою ГЗМ. Розроблена програма повністю відповідає всім вимогам технічного завдання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Black and white image colorization with OpenCV and Deep Learning [Electronic resource]. – Access mode : https://www.pyimagesearch.com/2019/02/25/black-and-white-image-colorization-with-opencv-and-deep-learning/.

2. Зниження шуму [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Зниження_шуму.

3. Super-resolution imaging [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Super-resolution_imaging.

4. Image restoration [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Image_restoration.

5. Цифрове зображення [Електрон. ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/Цифрове_зображення.

6. Петров, М. Н. Компьютерная графика [Текст] / М. Н. Петров, В. П. Молочков. – СПб. : Питер, 2002. – 736 с.

7. Субботін, С. О. Нейронні мережі : теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

8. Conzalez, R. C. Digital Image Processing Third Edition [Text] / R. C. Conzalez, R. E. Woods. – New Jersey: Pearson Prentice Hall, 2008. – 977 p.

9. Bilateral filter [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Bilateral_filter.

10. Non-local means [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Non-local_means.

11. Interpolation [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Interpolation.

12. Nearest neighbor interpolation [Electronic resource]. – Access mode : https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html.

13. Bilinear interpolation [Electronic resource]. – Access mode : https://www.wikiwand.com/en/Bilinear_interpolation.

14. Bicubic interpolation [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Bicubic_interpolation.

15. Convolution [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Convolution.

16. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [Electronic resource]. – Access mode : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

17. O'Shea, K. An Introduction to Convolutional Neural Networks [Electronic resource] / K. O'Shea, R. Nash. – Access mode : https://arxiv.org/pdf/1511.08458.pdf.

18. Autoencoder: Downsampling and Upsampling [Electronic resource]. – Access mode : https://kharshit.github.io/blog/2019/02/15/autoencoder-downsampling-and-upsampling.

19. Batch normalization: theory and how to use it with Tensorflow [Electronic resource]. – Access mode : https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad.

20. Zhang, R. Colorful Image Colorization [Electronic resource] / R. Zhang, A. A. Efros. – Access mode : https://arxiv.org/pdf/1603.08511.pdf.

21. Simonyan, K. Very Deep Convolutional Networks for Large-Sace Image Recognition [Electronic resource] / K. Simonyan, A. Zisserman. – Access mode : https://arxiv.org/pdf/1409.1556.pd.

22. Auto-Encoder: What Is It? And What Is It Used For? (Part 1) [Electronic resource]. – Access mode : https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

23. Autoencoder [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Autoencoder.

24. Lee, D. Performance evaluation of image denoising developed using convolutional denoising autoencoders in chest radiography [Electronic resource] /

D. Lee, S. Choi, H. Kim. – Access mode : https://www.sciencedirect.com/science/article/abs/pii/S0168900217314560.

25. Goodfellow, I. J. Generative Adversarial Networks [Electronic resource] / [I. J. Goodfellow, J. Pouget-Abadie, M. Mirza et al.] – Access mode : http://www.arxiv.org/abs/1406.2661.

26. Mirza, M. Conditional Generative Adversarial Nets Conditional Generative Adversarial Nets [Electronic resource] / M. Mirza, S. Osindero. – Access mode : https://arxiv.org/pdf/1411.1784.pdf.

27. Nazeri, K. Image Colorization using Generative Adversarial Networks [Electronic resource] / K. Nazeri, E. Ng, M. Ebrahimi. – Access mode : https://arxiv.org/pdf/1803.05400.pdf.

28. Ronnenberger, O. U-Net: Convolutional Networks for Biomedical Image Segmentation [Electronic resource] / O. Ronnenberger, P. Fischer, T. Brox. – Access mode: https://arxiv.org/pdf/1505.04597.pdf.

29. Yan, Q. DCGANs for image super-resolution, denoising and debluring [Electronic resource] / Q. Yan, W. Wang. – Access mode : http://stanford.edu/class/ee367/Winter2017/yan_wang_ee367_win17_report.pdf;

30. Ledig, C. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network [Electronic resource] / [C. Ledig, L. Theis, F. Huszár et al.] – Access mode : https://arxiv.org/pdf/1609.04802.pdf.

31. Wang, Z. Image quality assessment: from error visibility to structural similarity [Electronic resource] / [Z. Wang, A. C. Bovik, H. R. Sheikh et al.] – Access mode : https://ieeexplore.ieee.org/document/1284395.

32. Schildt, H. C++: A Beginner’s Guide, 3rd Edition [Text] / H. Schildt. – Osborne : McGraw-Hill Education, 2011. – 656p.

33. Shildt, H. Java: A Beginner’s Guide, Eighth Edition [Text] / H. Shildt. – Osborne : McGraw-Hill Education, 2018. – 720p.

34. Лутц, М. Изучаем Python / М. Лутц. – Т. 1 : Изучаем Python, пятое издание. – СПб. : ООО «Диалектика», 2019. – 832с.

35. Tensorflow [Electronic resource]. – Access mode : https://www.tensorflow.org/.

36. Keras [Electronic resource]. – Access mode : https://keras.io/.

37. PyTorch [Electronic resource]. – Access mode : https://pytorch.org/.

38. Theano (Software) [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Theano_(software)

39. Caffe [Electronic resource]. – Access mode : http://caffe.berkeleyvision.org/.

40. ONNX [Electronic resource]. – Access mode : https://onnx.ai/

41. Інтелектуальні системи : навч. посіб. / С. О. Субботін, А. О. Олійник; за ред. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2014. – 219 с.

42. Large-scale CelebFaces Attributes (CelebA) Dataset [Electronic resource]. – Access mode : http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

43. Places: A 10 million Image Database for Scene Recognition [Electronic resource]. – Access mode : http://places2.csail.mit.edu/index.html.

44. ImageNet [Electronic resource]. – Access mode : http://www.image-net.org/index.

45. ResNet (34, 50, 101): «остаточные» CNN для классификации изображений [Электрон. ресурс]. – Режим доступа : https://neurohive.io/ru/vidy-nejrosetej/resnet-34-50-101/.

46. Image-to-Image Translation with Conditional Adversarial Networks [Electronic resource] / P.Isola, J. Zhu, T. Zhou, A. Efros. – 2016. – Access mode : https://arxiv.org/abs/1611.07004.

47. Hyperparameter (machine learning) [Electronic resource]. – Access mode : https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning).

48. Difference Between a Batch and an Epoch in a Neural Network [Electronic resource]. – Access mode : https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

49. Loss and Loss Functions for Training Deep Learning Neural Networks [Electronic resource]. – Access mode : https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

50. Ruder, S. An overview of gradient descent optimization algorithms [Electronic resource] / S. Ruder. – Access mode : https://arxiv.org/pdf/1609.04747.pdf.

51. Various Optimization Algorithms for Training Neural Network [Electronic resource]. – Access mode : https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

ДОДАТОК А

Технічне завдання

А.1 Вступ

У даному розділі описано призначення, область застосування і вимоги до програми.

А.1.1 Назва програми

Програма називається «Image Restorer».

А.1.2 Призначення та область застосування

Програма призначена для відновлення зображень шляхом колоризації, шумозниження і збільшення роздільної здатності.

Програма представляє графічний інтерфейс і застосовується як обгортка для зручного використовування навчених генеративно-змагальних мереж (ГЗМ).

А.2 Підстави для розробки

Підставою для розробки є завдання на диплому роботу на тему «Програмне забезпечення для відновлення зображень з використанням генеративно-змагальних мереж».

Актуальність роботи полягає в потребі швидких автоматизованих інструментів для відновлення зображень.

А.3 Призначення розробки

Програмний продукт надає можливості для відновлення зображення шляхом колоризації, шумозниження і збільшення роздільної здатності з використанням ГЗМ.

А.4 Вимоги до програми

В даному розділі описані вимоги до програмного продукту.

А.4.1 Вимоги до функціональних характеристик

Програмний продукт повинен забезпечувати виконання наступних функцій:

– користувач повинен мати можливість колоризувати чорно-біле зображення;

– користувач повинен мати можливість знизити шум на зображенні;

– користувач повинен мати можливість збільшити роздільну здатність зображення;

– користувач повинен мати можливість зберегти отриманий результат відновлення зображення;

– програма повинна виконуватись на комп’ютері без доступу до мережі.

А.4.2 Вимоги до надійності

Програма призначена для виконання на персональному комп’ютері і має наступні вимоги до надійності:

– забезпечення безперебійного виконання процесу програми до її завершення;

– захист від помилкових дій користувача.

А.4.3 Вимоги до технічних та програмних засобів

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

– 2 ГБ оперативної пам’яті;

– відеокарта з 2 ГБ пам’яті;

– 100 МБ вільного місця на диску;

– процесор з тактовою частотою 2 ГГц;

– платформа x64;

– операційна система Linux;

– встановлені бібліотеки Tensorflow, numpy, scikit-image, PyQt5;

– встановлені драйвера CUDA для виконання розрахунків нейронних мереж за допомогою відеокарти;

– наявність моделей створених ГЗМ на комп’ютері.

А.5 Вимоги до програмної документації

Склад програмної документації до програмного продукту:

– технічне завдання;

– опис розробки програми;

– опис експлуатації програми.

А.6 Стадії і етапи розробки

Розробка має бути проведена в наступні стадії:

– створення та узгодження технічного завдання;

– аналіз методів колоризації, шумозниження і збільшення роздільної здатності зображень;

– пошук і обробка датасету для ГЗМ;

– створення архітектури і навчання ГЗМ для колоризації, шумозниження і збільшення роздільної здатності;

– аналіз результатів роботи створених ГЗМ;

– створення графічного інтерфейсу для взаємодії користувача із створеними ГЗМ;

– створення супровідної документації.

А.7 Порядок контролю та приймання

Продукту та супутній документації необхідно пройти наступні стадії контролю та приймання:

– тестування програмного продукту;

– аналіз роботи створених ГЗМ;

– затвердження працездатності продукту та супровідної документації науковим керівником;

– нормоконтроль документації до програмного продукту.

ДОДАТОК Б

Текст програми

Б.1 Код графічного інтерфейсу

Код файлу OptionsWindow.py

```
from PyQt5.QtWidgets import (QLabel,
                             QMainWindow,
                             QFileDialog,
                             QPushButton,
                             QHBoxLayout,
                             QVBoxLayout,
                             QWidget,
                             QRadioButton,
                             QCheckBox,
                             QGridLayout,
                             QMessageBox)
from PyQt5.QtCore import Qt
import skimage
from ResultsWindow import ResultsWindow
from GAN import *


class OptionsWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super(QMainWindow, self).__init__(*args, **kwargs)

        self.imagePath = ""
        self.currentOperation = ""
        self.noiseAdded = False
        self.resizedBefore = False

        self.initUI()
        self.resultsWindow = ResultsWindow()
        self.colorizator = ColorizationGAN("../GANs/colorization")
        self.denoisator = DenoisingGAN("../GANs/denoising")
        self.srgan = SRGAN("../GANs/super_resolution")


    def initUI(self):
        self.setWindowTitle("Image Restorer")
        self.setWindowIcon(QtGui.QIcon('logo.png'))
        self.openImageButton = QPushButton("Browse...")
        self.openImageButton.clicked.connect(self.openImage)
        self.openImageLabel = QLabel("Open image")
        self.openImageLayout = QHBoxLayout()
        self.openImageLayout.addWidget(self.openImageLabel)
        self.openImageLayout.addWidget(self.openImageButton)
        self.openImageLayout.addStretch(1)
        self.colorizeRadioButton = QRadioButton("Colorize")
        self.colorizeRadioButton.operation = "colorization"
        self.colorizeRadioButton.toggled.connect(self.onRadioButtonClicked)

        self.denoiseRadioButton = QRadioButton("Denoise")
        self.denoiseRadioButton.operation = "denoising"
        self.denoiseRadioButton.toggled.connect(self.onRadioButtonClicked)
```

```
        self.enlargeRadioButton = QRadioButton("Enlarge")
        self.enlargeRadioButton.operation = "enlarging"
        self.enlargeRadioButton.toggled.connect(self.onRadioButtonClicked)

        self.addNoiseCheckBox = QCheckBox("Add Noise")
        self.addNoiseCheckBox.operation = "noise"
        self.addNoiseCheckBox.setEnabled(False)

        self.optionsGridLayout = QGridLayout()
        self.optionsGridLayout.addWidget(self.colorizeRadioButton, 0, 0)
        self.optionsGridLayout.addWidget(self.denoiseRadioButton, 1, 0)
        self.optionsGridLayout.addWidget(self.enlargeRadioButton, 2, 0)
        self.optionsGridLayout.addWidget(self.addNoiseCheckBox, 1, 1)


        self.startButton = QPushButton("Start")
        self.startButton.clicked.connect(self.onStartButtonClicked)
        self.closeButton = QPushButton("Close")
        self.closeButton.clicked.connect(self.close)
        self.buttonLayout = QHBoxLayout()
        self.buttonLayout.addStretch(1)
        self.buttonLayout.addWidget(self.startButton)
        self.buttonLayout.addWidget(self.closeButton)
        self.buttonLayout.addStretch(1)

        self.mainWidget = QWidget()
        self.setCentralWidget(self.mainWidget)
        self.mainLayout = QVBoxLayout()
        self.mainLayout.addLayout(self.openImageLayout)
        self.mainLayout.addLayout(self.optionsGridLayout)
        self.mainLayout.addLayout(self.buttonLayout)
        self.mainWidget.setLayout(self.mainLayout)

        self.setFixedSize(self.minimumSize().width(),
self.minimumSize().height())

    def openImage(self):
        self.showMessage("Image will be resized to 256x256 pixels")
        fname = QFileDialog.getOpenFileName(self, 'Open file', 'd:\\', "Image
files (*.jpg *.png *.jpeg *.JPEG)")
        self.imagePath = fname[0]

    def showMessage(self, message):
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Warning)
        msg.setText(message)
        msg.setWindowTitle("Warning")
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()

    def onRadioButtonClicked(self):
        radioButton = self.sender()
        if radioButton.isChecked():
            self.currentOperation = radioButton.operation
            if radioButton.operation == "denoising":
                self.addNoiseCheckBox.setEnabled(True)
            else:
                self.addNoiseCheckBox.setEnabled(False)

    def onStartButtonClicked(self):
```

```
        if self.imagePath == "":
            self.showMessage("Image wasn't selected!")
            return
        if self.currentOperation == "":
            self.showMessage("Choose operation type!")
            return
        trueImage = skimage.io.imread(self.imagePath)
        trueImage = resize_image(trueImage, (256, 256))
        if trueImage.ndim != 3:
            trueImage = skimage.color.gray2rgb(trueImage)
            trueImage = (trueImage * 255).astype(np.uint8)
        if trueImage.shape[2] == 4:
            trueImage = skimage.color.rgba2rgb(trueImage)
            trueImage = (trueImage * 255).astype(np.uint8)

        if self.currentOperation == "colorization":
            inputImage = skimage.color.rgb2lab(trueImage)
            inputImage = inputImage[:, :, 0]
            inputImage = inputImage[..., np.newaxis]
            resultImage = self.colorizator.process(inputImage)
            inputImage                                                    =
skimage.color.gray2rgb(skimage.color.rgb2gray(trueImage)) * 255

        elif self.currentOperation == "denoising":
            if self.addNoiseCheckBox.isChecked():
                inputImage = add_noise(trueImage, 0, 10)
            else:
                inputImage = trueImage.copy()
            resultImage = self.denoisator.process(inputImage)

        elif self.currentOperation == "enlarging":
            inputImage = resize_image(trueImage, (128, 128))
            resultImage = self.srgan.process(inputImage)
            pass

        inputImage = inputImage.astype(np.uint8)
        inputImage = inputImage.astype(np.uint8)

        self.resultsWindow.setImages(trueImage, inputImage, resultImage)
        self.resultsWindow.setWindowModality(Qt.ApplicationModal)
        self.resultsWindow.show()
```

Код файлу ResultsWindow.py

```
from PyQt5.QtWidgets import (QLabel,
                             QMainWindow,
                             QFileDialog,
                             QPushButton,
                             QHBoxLayout,
                             QVBoxLayout,
                             QWidget,
                             QMessageBox)
from PyQt5.QtCore import Qt
from skimage.io import imsave
from image_processing import *


class ResultsWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super(QMainWindow, self).__init__(*args, **kwargs)
```

```
            self.initUI()
            self.resImage = None

        def initUI(self):
            self.setWindowTitle("Results")
            self.setWindowIcon(QtGui.QIcon('logo.png'))

            self.inputImage = self.createLabelImage()
            self.corruptedImage = self.createLabelImage()
            self.corruptedImage.setAlignment(Qt.AlignCenter)
            self.restoredImage = self.createLabelImage()

            imagesLayout = QHBoxLayout()
            imagesLayout.addWidget(self.inputImage)
            imagesLayout.addWidget(self.corruptedImage)
            imagesLayout.addWidget(self.restoredImage)

            inputImageLabel = QLabel("Input Image")
            inputImageLabel.setAlignment(Qt.AlignCenter)
            corruptedImageLabel = QLabel("Corrupted Image")
            corruptedImageLabel.setAlignment(Qt.AlignCenter)
            restoredImageLabel = QLabel("Restored Image")
            restoredImageLabel.setAlignment(Qt.AlignCenter)

            labelsLayout = QHBoxLayout()
            labelsLayout.addWidget(inputImageLabel)
            labelsLayout.addWidget(corruptedImageLabel)
            labelsLayout.addWidget(restoredImageLabel)

            self.saveButton = QPushButton("Save")
            self.saveButton.clicked.connect(self.onSaveButtonClicked)
            self.closeButton = QPushButton("Close")
            self.closeButton.clicked.connect(self.close)
            buttonLayout = QHBoxLayout()
            buttonLayout.addStretch(1)
            buttonLayout.addWidget(self.saveButton)
            buttonLayout.addWidget(self.closeButton)

            self.mainWidget = QWidget()
            self.setCentralWidget(self.mainWidget)
            self.mainLayout = QVBoxLayout()
            self.mainLayout.addLayout(imagesLayout)
            self.mainLayout.addLayout(labelsLayout)
            self.mainLayout.addLayout(buttonLayout)
            self.mainWidget.setLayout(self.mainLayout)

            self.setFixedSize(self.minimumSize().width(),
    self.minimumSize().height())

        def createLabelImage(self):
            label = QLabel()
            label.setFixedSize(256, 256)
            return label

        def onSaveButtonClicked(self):
            name = QFileDialog.getSaveFileName(self, 'Save file')[0]
            if name != "":
                imsave(name, self.resImage)
                self.showMessage("Image was successfully saved!")

        def showMessage(self, message):
```

```
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Information)
        msg.setText(message)
        msg.setWindowTitle("Image saving")
        msg.setStandardButtons(QMessageBox.Ok)
        msg.exec_()


    def setImages(self, inputImage, corruptedImage, restoredImage):
        self.inputImage.setPixmap(image2pixmap(inputImage))
        self.corruptedImage.setPixmap(image2pixmap(corruptedImage))
        self.restoredImage.setPixmap(image2pixmap(restoredImage))
        self.resImage = restoredImage
```

Код файлу main.py

```
import sys
from PyQt5.QtWidgets import QApplication
from OptionsWindow import OptionsWindow

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = OptionsWindow()
    window.show()
    app.exec_()
```

Б.2 Код ядра програми

Код файлу image_processing.py

```
from skimage import color, transform
from PyQt5 import QtGui
import numpy as np


def add_noise(image, m, std):
    noise = np.random.normal(m, std, image.shape)
    noise = noise.reshape(image.shape)
    noisy_image = image + noise
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image


def image2pixmap(image):
    height, width, channels = image.shape
    bytesPerLine = channels * width
    qImg    =    QtGui.QImage(image.data,    width,    height,    bytesPerLine,
QtGui.QImage.Format_RGB888)
    pixmap = QtGui.QPixmap(qImg)
    return pixmap


def resize_image(image, shape):
    res = transform.resize(image, shape)
    res = (res * 255).astype(np.uint8)
    return res
```

```
def image2tensor(image):
    image = (image - 127.5) / 127.5
    image = image[np.newaxis, ...]
    return image


def output2image(output):
    image = (output[0] + 127.5) * 127.5
    image = image.astype(np.uint8)
    return image


def lab_output2image(L, ab):
    image = np.zeros(shape=(256, 256, 3), dtype=np.float32)
    image[:, :, 0] = (L[:, :, 0] + 1.) * 50
    image[:, :, 1:] = ab[0, :, :, :] * 128
    image = color.lab2rgb(image) * 255
    image = image.astype(np.uint8)
    return image
```

Код файлу GAN.py

```
import tensorflow as tf
from image_processing import *


class GAN():
    def __init__(self, model_path):
        self.model = tf.keras.models.load_model(model_path)

    def process(self, input_image):
        raise NotImplementedError()


class ColorizationGAN(GAN):
    def __init__(self, model_path):
        super().__init__(model_path)

    def process(self, input_image):
        if input_image.shape[2] != 1:
            raise ValueError("Image shape should be (h, w, 1)")

        input_image = input_image / 50. - 1.
        input_tensor = input_image[np.newaxis, ...]
        ab = self.model.predict(input_tensor)

        output_image = lab_output2image(input_image, ab)
        return output_image


class DenoisingGAN(GAN):
    def __init__(self, model_path):
        super().__init__(model_path)

    def process(self, input_image):
        input_tensor = image2tensor(input_image)
        output_tensor = self.model.predict(input_tensor)
        output_image = output2image(output_tensor)
        return output_image
```

```
class SRGAN(GAN):
    def __init__(self, model_path):
        super().__init__(model_path)

    def process(self, input_image):
        input_tensor = image2tensor(input_image)
        output_tensor = self.model.predict(input_tensor)
        output_image = output2image(output_tensor)
        return output_image
```

Б.3 Код створення архітектур генеративно-змагальних мереж

Код файлу colorizationGAN.py

```
from skimage import color, io
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras import Model
import os

os.environ["SM_FRAMEWORK"] = "tf.keras"
import segmentation_models as sm


def generate_real_samples(batch):
    X = list()
    y = list()
    for i in range(batch.shape[0]):
        lab_image = color.rgb2lab(batch[i])
        lab_image[:,:,0] = lab_image[:,:,0] / 50. - 1.
        lab_image[:,:,1:] = lab_image[:,:,1:] / 128.
        X.append(lab_image[:,:,0])
        y.append(lab_image[:,:,1:])
    X = np.asarray(X)
    y = np.asarray(y)
    return X, y


def vizualize_perfomance(g_model, images, n_samples=3):
    real_grayscale, real_ab = generate_real_samples(images)
    fake_ab = g_model.predict(real_grayscale)

    fig, ax = plt.subplots(n_samples, 3, figsize=(12, 4 * n_samples))
    if ax.ndim == 1:
        ax = np.array([ax])
    for i in range(n_samples):
        ax[i, 0].imshow(color.rgb2gray(images[i, :, :, :]), cmap='gray')

        ax[i, 1].imshow(images[i, :, :, :])
```

```
        fake_image = np.zeros(shape=(256, 256, 3), dtype=np.float32)
        fake_image[:, :, 0] = (real_grayscale[i, :, :] + 1.) * 50
        fake_image[:, :, 1:] = fake_ab[i, :, :, :] * 128
        fake_image = color.lab2rgb(fake_image)
        ax[i, 2].imshow(fake_image)

    for i in range(n_samples):
        for j in range(3):
            ax[i, j].set_xticks([])
            ax[i, j].set_yticks([])

    labels = ['grayscale', 'true', 'colorized']
    for i in range(3):
        ax[n_samples - 1, i].set_xlabel(labels[i], fontsize=14)

    fig.subplots_adjust(wspace=0, hspace=0.05)
    fig.savefig('results.png', dpi=fig.dpi)
    plt.close()


def downsample(filters, kernel_size, strides=2, apply_batchnorm=True):
    initializer = tf.random_normal_initializer(0., 0.02)

    result = tf.keras.Sequential()
    result.add(tf.keras.layers.Conv2D(filters,      kernel_size,      strides,
padding='same',
                                      kernel_initializer=initializer,
use_bias=False))
    if apply_batchnorm:
        result.add(tf.keras.layers.BatchNormalization())
    result.add(tf.keras.layers.LeakyReLU(0.2))
    return result


def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)

    inp = tf.keras.layers.Input(shape=[256, 256, 1], name='input_image')
    tar = tf.keras.layers.Input(shape=[256, 256, 2], name='target_image')

    x = tf.keras.layers.concatenate([inp, tar])

    down1 = downsample(64, 4, apply_batchnorm=False)(x)
    down2 = downsample(128, 4)(down1)
    down3 = downsample(256, 4)(down2)

    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)

    conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                  kernel_initializer=initializer,
                                  use_bias=False)(zero_pad1)
    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)

    last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                  kernel_initializer=initializer)(zero_pad2)
    last = tf.keras.layers.LeakyReLU()(last)
```

```
    return tf.keras.Model(inputs=[inp, tar], outputs=last)


def Generator():
    base_model = sm.Unet(backbone_name='resnet50',
                         encoder_weights='imagenet',
                         input_shape=(256, 256, 3),
                         classes=2,
                         activation='tanh',
                         encoder_freeze=True,
                         decoder_block_type='transpose',
                         decoder_use_batchnorm=True)

    inp = Input(shape=(None, None, 1))
    l1 = Conv2D(3, (1, 1))(inp)
    out = base_model(l1)

    return Model(inp, out, name=base_model.name)


def generator_loss(disc_generated_output, gen_output, target):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    gan_loss        =        crossentropy(tf.ones_like(disc_generated_output),
disc_generated_output)
    gan_loss = tf.cast(gan_loss, dtype=tf.float64)

    l1_loss = tf.reduce_mean(tf.abs((target - gen_output)))

    total_gen_loss = gan_loss + (100 * l1_loss)

    return total_gen_loss, gan_loss, l1_loss


def discriminator_loss(disc_real_output, disc_generated_output):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss          =          crossentropy(tf.ones_like(disc_real_output),
disc_real_output) * 0.9
    real_loss = tf.cast(real_loss, dtype=tf.float64)

    generated_loss    =    crossentropy(tf.zeros_like(disc_generated_output),
disc_generated_output)
    generated_loss = tf.cast(generated_loss, dtype=tf.float64)
    total_disc_loss = real_loss + generated_loss

    return total_disc_loss
```

Код файлу denoisingGAN.py

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf


def add_noise(batch):
    samples, row, col, ch = batch.shape
    mean = 0
    var = 10
    noise = np.random.normal(mean,var,(samples, row, col, ch))
    noise = noise.reshape(samples, row, col, ch)
    noisy_batch = batch + noise
```

```
    noisy_batch = np.clip(noisy_batch, 0, 255)
    return noisy_batch


def vizualize_perfomance(real_images, noisy_images, n_samples):
    denoised_images = generator(noisy_images)

    real_images = (real_images + 1) * 0.5
    noisy_images = (noisy_images + 1) * 0.5
    gan_denoised_images = (denoised_images + 1) * 0.5

    fig, ax = plt.subplots(n_samples, 3, figsize=(12, 4 * n_samples))
    if ax.ndim == 1:
        ax = np.array([ax])
    for i in range(n_samples):
        ax[i, 0].imshow(real_images[i, :, :, :])

        ax[i, 1].imshow(noisy_images[i, :, :, :])
        ax[i, 2].imshow(gan_denoised_images[i, :, :, :])

    for i in range(n_samples):
        for j in range(3):
            ax[i, j].set_xticks([])
            ax[i, j].set_yticks([])

    labels = ['true', 'noisy', 'denoised']
    for i in range(3):
        ax[n_samples - 1, i].set_xlabel(labels[i], fontsize=14)

    fig.subplots_adjust(wspace=0, hspace=0.05)
    fig.savefig('results.png', dpi=fig.dpi)
    plt.close()


def downsample(filters, kernel_size, strides=2, apply_batchnorm=True):
    initializer = tf.random_normal_initializer(0., 0.02)

    result = tf.keras.Sequential()
    result.add(tf.keras.layers.Conv2D(filters,      kernel_size,      strides,
padding='same',
                              kernel_initializer=initializer, use_bias=False))

    if apply_batchnorm:
        result.add(tf.keras.layers.BatchNormalization())

    result.add(tf.keras.layers.LeakyReLU(0.2))

    return result


def upsample(filters, size, apply_dropout=False):
    initializer = tf.random_normal_initializer(0., 0.02)

    result = tf.keras.Sequential()
    result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))

    result.add(tf.keras.layers.BatchNormalization())
```

```
    if apply_dropout:
        result.add(tf.keras.layers.Dropout(0.5))

    result.add(tf.keras.layers.ReLU())

    return result


def Generator():
    inputs = tf.keras.layers.Input(shape=[256, 256, 3])

    down_stack = [
        downsample(64, 4, strides=1, apply_batchnorm=False),
        downsample(64, 4),
        downsample(128, 4),
        downsample(256, 4),
        downsample(512, 4),
        downsample(512, 4),
        downsample(512, 4),
        downsample(512, 4)
    ]

    up_stack = [
        upsample(512, 4, apply_dropout=True),
        upsample(512, 4, apply_dropout=True),
        upsample(512, 4),
        upsample(256, 4),
        upsample(128, 4),
        upsample(64, 4),
        upsample(64, 4)
    ]

    initializer = tf.random_normal_initializer(0., 0.02)
    last = tf.keras.layers.Conv2D(3, 1,
                                  strides=1,
                                  kernel_initializer=initializer,
                                  activation='tanh')

    x = inputs

    skips = []
    for down in down_stack:
        x = down(x)
        skips.append(x)

    skips = reversed(skips[:-1])

    for up, skip in zip(up_stack, skips):
        x = up(x)
        x = tf.keras.layers.Concatenate()([x, skip])

    x = last(x)

    return tf.keras.Model(inputs=inputs, outputs=x)


def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)

    inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
```

```
    tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')

    x = tf.keras.layers.concatenate([inp, tar])

    down1 = downsample(64, 4, apply_batchnorm=False)(x)
    down2 = downsample(128, 4)(down1)
    down3 = downsample(256, 4)(down2)

    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)

    conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                  kernel_initializer=initializer,
                                  use_bias=False)(zero_pad1)
    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)

    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)

    last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                  kernel_initializer=initializer)(zero_pad2)
    last = tf.keras.layers.LeakyReLU()(last)
    return tf.keras.Model(inputs=[inp, tar], outputs=last)


def generator_loss(disc_generated_output, gen_output, target):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    gan_loss        =        crossentropy(tf.ones_like(disc_generated_output),
disc_generated_output)
    gan_loss = tf.cast(gan_loss, dtype=tf.float64)

    l1_loss = tf.reduce_mean(tf.abs((target - gen_output)))

    total_gen_loss = gan_loss + (100 * l1_loss)

    return total_gen_loss, gan_loss, l1_loss


def discriminator_loss(disc_real_output, disc_generated_output):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss          =          crossentropy(tf.ones_like(disc_real_output),
disc_real_output) * 0.9
    real_loss = tf.cast(real_loss, dtype=tf.float64)

    generated_loss    =    crossentropy(tf.zeros_like(disc_generated_output),
disc_generated_output)
    generated_loss = tf.cast(generated_loss, dtype=tf.float64)
    total_disc_loss = real_loss + generated_loss

    return total_disc_loss
```

Код файлу SRGAN.py

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg19 import VGG19
import tensorflow.keras.backend as K
from tensorflow.keras import Model
```

```
def resize_images(images):
    lr_images = list()
    for i in range(images.shape[0]):
        lr_image    =    cv2.resize(images[i,:,:,:],    (images.shape[1]//2,
images.shape[2]//2))
        lr_images.append(lr_image)
    lr_images = np.asarray(lr_images)
    return lr_images


def vizualize_perfomance(hr_images, lr_images, n_samples):
    sr_images = generator(lr_images)

    hr_images = (hr_images + 1) * 0.5
    lr_images = (lr_images + 1) * 0.5
    sr_images = (sr_images + 1) * 0.5

    fig, ax = plt.subplots(n_samples, 5, figsize=(20, 4*n_samples))
    if ax.ndim == 1:
        ax = np.array([ax])
    for i in range(n_samples):
        ax[i, 0].imshow(hr_images[i, :,:,:])

        ax[i,   1].imshow(cv2.resize(lr_images[i,   :,:,:],   (256,   256),
interpolation=cv2.INTER_NEAREST))
        ax[i,   2].imshow(cv2.resize(lr_images[i,   :,:,:],   (256,   256),
interpolation=cv2.INTER_LINEAR))
        ax[i,   3].imshow(cv2.resize(lr_images[i,   :,:,:],   (256,   256),
interpolation=cv2.INTER_CUBIC))
        ax[i, 4].imshow(sr_images[i, :,:,:])

    for i in range(n_samples):
        for j in range(5):
            ax[i, j].set_xticks([])
            ax[i, j].set_yticks([])

    labels  =  ['true',  'INTER_NEAREST',  'INTER_LINEAR',  'INTER_CUBIC',
'SRGAN']
    for i in range(3):
        ax[n_samples-1, i].set_xlabel(labels[i], fontsize = 14)

    fig.subplots_adjust(wspace=0, hspace=0.05)
    fig.savefig('esults.png', dpi=fig.dpi)
    plt.close()


def res_block_gen(model, kernal_size, filters, strides):
    gen = model

    model = Conv2D(filters=filters, kernel_size=kernal_size, strides=strides,
padding="same")(model)
    model = BatchNormalization(momentum=0.5)(model)
    model = LeakyReLU(0.2)(model)

    model = Conv2D(filters=filters, kernel_size=kernal_size, strides=strides,
padding="same")(model)
    model = BatchNormalization(momentum=0.5)(model)

    model = add([gen, model])
```

```
    return model


def up_sampling_block(model, kernal_size, filters, strides):
    model = Conv2DTranspose(filters, kernal_size, strides=strides,
                            padding='same',
                            use_bias=False)(model)
    model = LeakyReLU(alpha=0.2)(model)
    model = BatchNormalization()(model)

    return model


def Generator(input_shape):
    gen_input = Input(shape=input_shape)

    model      =      Conv2D(filters=64,      kernel_size=9,      strides=1,
padding="same")(gen_input)
    model = LeakyReLU(0.2)(model)

    gen_model = model

    for index in range(10):
        model = res_block_gen(model, 3, 64, 1)

    model      =      Conv2D(filters=64,      kernel_size=3,      strides=1,
padding="same")(model)
    model = BatchNormalization(momentum=0.5)(model)
    model = add([gen_model, model])

    for index in range(1):
        model = up_sampling_block(model, 3, 128, 2)

    model      =      Conv2D(filters=3,      kernel_size=9,      strides=1,
padding="same")(model)

    model = Activation('tanh')(model)

    generator_model = Model(inputs=gen_input, outputs=model)

    return generator_model


def downsample(filters, kernel_size, strides=2, apply_batchnorm=True):
    initializer = tf.random_normal_initializer(0., 0.02)

    result = tf.keras.Sequential()

    result.add(
        tf.keras.layers.Conv2D(filters, kernel_size, strides, padding='same',
                               kernel_initializer=initializer,
use_bias=False))

    if apply_batchnorm:
        result.add(tf.keras.layers.BatchNormalization())

    result.add(tf.keras.layers.LeakyReLU(0.2))

    return result
```

```
def Discriminator():
    initializer = tf.random_normal_initializer(0., 0.02)

    inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')

    down1 = downsample(64, 4, apply_batchnorm=False)(inp)  # (bs, 128, 128,
64)
    down2 = downsample(128, 4)(down1)  # (bs, 64, 64, 128)

    down3 = downsample(256, 4)(down2)  # (bs, 32, 32, 256)

    zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)  # (bs, 34, 34, 256)

    conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                  kernel_initializer=initializer,
                                  use_bias=False)(zero_pad1)  # (bs, 31, 31,
512)

    batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
    leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
    # downsample(512, 4, 1)(zero_pad1)

    zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)  # (bs, 33, 33,
512)

    last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                  kernel_initializer=initializer)(zero_pad2)
# (bs, 30, 30, 1)
    last = tf.keras.layers.LeakyReLU()(last)

    return tf.keras.Model(inputs=inp, outputs=last)


vgg19 = VGG19(include_top=False, weights='imagenet', input_shape=(256, 256,
3))
vgg19.trainable = False
for l in vgg19.layers:
    l.trainable = False
vgg                         =                         Model(inputs=vgg19.input,
outputs=vgg19.get_layer('block5_conv4').output)
vgg.trainable = False


def vgg_loss(y_true, y_pred):
    return K.mean(K.square(vgg(y_true) - vgg(y_pred)))


def generator_loss(disc_generated_output, gen_output, target):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    gan_loss        =        crossentropy(tf.ones_like(disc_generated_output),
disc_generated_output)
    gan_loss = tf.cast(gan_loss, dtype=tf.float64)

    content_loss = vgg_loss(target, gen_output)

    content_loss = tf.cast(content_loss, dtype=tf.float64)

    total_gen_loss = (1e-3 * gan_loss) + content_loss

    return total_gen_loss, gan_loss, content_loss
```

```
def discriminator_loss(disc_real_output, disc_generated_output):
    crossentropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss          =          crossentropy(tf.ones_like(disc_real_output),
disc_real_output) * 0.9
    real_loss = tf.cast(real_loss, dtype=tf.float64)

    generated_loss    =    crossentropy(tf.zeros_like(disc_generated_output),
disc_generated_output)
    generated_loss = tf.cast(generated_loss, dtype=tf.float64)
    total_disc_loss = real_loss + generated_loss

    return total_disc_loss
```

Б.4 Код обробки датасету

Код файлу dataset_utils.py

```
import glob
import tensorflow as tf
import numpy as np


def get_dataset(records_path, shuffle_size, batch_size):
    tf_records = glob.glob(records_path)
    dataset = tf.data.TFRecordDataset(tf_records)
    dataset = dataset.map(parse_dataset)
    dataset = dataset.shuffle(shuffle_size).repeat().batch(batch_size)
    return dataset


def parse_dataset(record):
    name_to_features = {
        'image': tf.io.FixedLenFeature([], tf.string),
    }
    features = tf.io.parse_example([record], features=name_to_features)
    image = tf.io.decode_raw(features['image'], np.uint8)
    image = tf.reshape(image, (256, 256, 3))
    return image
```

Б.4 Код тренування генеративно-змагальних мереж

Код файлу colorizationGAN_training.py

```
from models.colorizationGAN import *
from utils.dataset_utils import *
import tensorflow as tf
import time
import datetime


BATCH_SIZE = 16
ITERATIONS = 125000
```

```
ECHO = 1000
SHUFFLE = 1000

generator_optimizer = tf.keras.optimizers.Adam(2e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4)


@tf.function
def train_step(input_image, target, iteration):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        gen_output = generator(input_image, training=True)

        gen_output = tf.cast(gen_output, dtype=tf.float64)

        disc_real_output      =      discriminator([input_image,      target],
training=True)
        disc_generated_output   =   discriminator([input_image,   gen_output],
training=True)
        gen_total_loss,          gen_gan_loss,          gen_l1_loss          =
generator_loss(disc_generated_output, gen_output, target)
        disc_loss            =            discriminator_loss(disc_real_output,
disc_generated_output)

    generator_gradients = gen_tape.gradient(gen_total_loss,
                                            generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss,

discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(generator_gradients,
                                            generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(discriminator_gradients,

discriminator.trainable_variables))

    if iteration % 100 == 0:
        with summary_writer.as_default():
            tf.summary.scalar('gen_total_loss',               gen_total_loss,
step=iteration)
            tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=iteration)
            tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=iteration)
            tf.summary.scalar('disc_loss', disc_loss, step=iteration)


def fit(train_dataset, iterations, echo):
    start_time = time.time()
    iteration = 0
    for batch in train_dataset:
        real_source, real_target = generate_real_samples(batch)
        train_step(real_source,  real_target,  tf.convert_to_tensor(iteration,
dtype=tf.int64))
        iteration += 1
        if iteration % echo == 0:
            print (f"Iteration {iteration}, {time.time() - start_time}s")
            start_time = time.time()
            vizualize_perfomance(generator, batch, BATCH_SIZE)
            manager.save()
        if iteration >= iterations:
            break
```

```
log_dir="logs/"
summary_writer    =    tf.summary.create_file_writer(log_dir    +    "fit/"    +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))


if __name__ == "__main__":
    physical_devices = tf.config.list_physical_devices('GPU')
    try:
        tf.config.experimental.set_memory_growth(physical_devices[0], True)
    except:
        pass

    train_dataset = get_dataset("/tf_dataset/train*", SHUFFLE, BATCH_SIZE)

    generator = Generator()
    discriminator = Discriminator()

    checkpoint_dir = '/checkpoints'
    checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                     generator=generator,
                                     discriminator=discriminator)
    manager    =    tf.train.CheckpointManager(checkpoint,    checkpoint_dir,
max_to_keep=1)

    fit(train_dataset, ITERATIONS, ECHO)

    generator.save("/resnet50GAN/generator")
    discriminator.save("/resnet50GAN/discriminator")
```

Код файлу denoisingGAN_training.py

```
from models.denoisingGAN import *
from utils.dataset_utils import *
import tensorflow as tf
import numpy as np
import time
import datetime


BATCH_SIZE = 16
ITERATIONS = 125000
ECHO = 1000
SHUFFLE = 1000

generator_optimizer = tf.keras.optimizers.Adam(2e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4)


log_dir="logs/"
summary_writer    =    tf.summary.create_file_writer(log_dir    +    "fit/"    +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))


@tf.function
def train_step(input_image, target, iteration):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        gen_output = generator(input_image, training=True)
```

```
        gen_output = tf.cast(gen_output, dtype=tf.float64)

        disc_real_output      =      discriminator([input_image,      target],
training=True)
        disc_generated_output   =   discriminator([input_image,   gen_output],
training=True)
        gen_total_loss,          gen_gan_loss,          gen_l1_loss          =
generator_loss(disc_generated_output, gen_output, target)
        disc_loss            =            discriminator_loss(disc_real_output,
disc_generated_output)

    generator_gradients = gen_tape.gradient(gen_total_loss,
                                            generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss,

discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(generator_gradients,
                                            generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(discriminator_gradients,

discriminator.trainable_variables))

    if iteration % 100 == 0:
        with summary_writer.as_default():
            tf.summary.scalar('gen_total_loss',               gen_total_loss,
step=iteration)
            tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=iteration)
            tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=iteration)
            tf.summary.scalar('disc_loss', disc_loss, step=iteration)


def fit(train_dataset, iterations, echo):
    start_time = time.time()
    iteration = 0
    for batch in train_dataset:
        real_images = batch.numpy().astype(np.float64)
        noisy_images = add_noise(real_images)

        real_images = (real_images - 127.5) / 127.5
        noisy_images = (noisy_images - 127.5) / 127.5

        train_step(noisy_images, real_images, tf.convert_to_tensor(iteration,
dtype=tf.int64))

        iteration += 1
        if iteration % echo == 0:
            print(f"Iteration {iteration}, {time.time() - start_time}s")
            start_time = time.time()
            vizualize_perfomance(real_images, noisy_images, BATCH_SIZE)
            manager.save()
        if iteration >= iterations:
            break


if __name__ == "__main__":
    physical_devices = tf.config.list_physical_devices('GPU')
    try:
        tf.config.experimental.set_memory_growth(physical_devices[0], True)
    except:
        pass
```

```
    train_dataset = get_dataset("/tf_dataset/train*", SHUFFLE, BATCH_SIZE)

    generator = Generator()
    discriminator = Discriminator()

    checkpoint_dir = '/checkpoints'
    checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                     generator=generator,
                                     discriminator=discriminator)
    manager    =    tf.train.CheckpointManager(checkpoint,    checkpoint_dir,
max_to_keep=1)

    fit(train_dataset, ITERATIONS, ECHO)

    generator.save("/denoisingGAN/generator")
    discriminator.save("/denoisingGAN/discriminator")
```

Код файлу SRGAN_training.py

```
from models.SRGAN import *
from utils.dataset_utils import *
import tensorflow as tf
import numpy as np
import time
import datetime


BATCH_SIZE = 16
ITERATIONS = 125000
ECHO = 1000
SHUFFLE = 1000


lr_input_shape = (128, 128, 3)
hr_input_shape = (256, 256, 3)


generator_optimizer = tf.keras.optimizers.Adam(2e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4)


log_dir="logs/"
summary_writer   =   tf.summary.create_file_writer(log_dir   +   "fit/"   +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))


@tf.function
def train_step(lr_images, hr_images, iteration):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        sr_images = generator(lr_images, training=True)

        sr_images = tf.cast(sr_images, dtype=tf.float64)

        disc_real_output = discriminator(hr_images, training=True)
        disc_generated_output = discriminator(sr_images, training=True)
```

```
        gen_total_loss,          gen_gan_loss,          gen_l1_loss          =
generator_loss(disc_generated_output, sr_images, hr_images)
        disc_loss           =           discriminator_loss(disc_real_output,
disc_generated_output)

    generator_gradients = gen_tape.gradient(gen_total_loss,
                                            generator.trainable_variables)
    discriminator_gradients = disc_tape.gradient(disc_loss,

discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(generator_gradients,
                                            generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(discriminator_gradients,

discriminator.trainable_variables))

    if iteration % 100 == 0:
        with summary_writer.as_default():
            tf.summary.scalar('gen_total_loss',               gen_total_loss,
step=iteration)
            tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=iteration)
            tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=iteration)
            tf.summary.scalar('disc_loss', disc_loss, step=iteration)


def fit(train_dataset, iterations, echo):
    start_time = time.time()
    iteration = 0
    for batch in train_dataset:
        hr_images = batch.numpy().astype(np.float64)
        lr_images = resize_images(hr_images)

        hr_images = (hr_images - 127.5) / 127.5
        lr_images = (lr_images - 127.5) / 127.5

        train_step(lr_images,    hr_images,    tf.convert_to_tensor(iteration,
dtype=tf.int64))

        iteration += 1
        if iteration % echo == 0:
            print (f"Iteration {iteration}, {time.time() - start_time}s")
            start_time = time.time()
            vizualize_perfomance(hr_images, lr_images, BATCH_SIZE)
            manager.save()
        if iteration >= iterations:
            break


if __name__ == "__main__":
    physical_devices = tf.config.list_physical_devices('GPU')
    try:
        tf.config.experimental.set_memory_growth(physical_devices[0], True)
    except:
        pass

    train_dataset = get_dataset("/tf_dataset/train*", SHUFFLE, BATCH_SIZE)

    generator = Generator(lr_input_shape)
    discriminator = Discriminator()
```

```
    checkpoint_dir = '/checkpoints'
    checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                     generator=generator,
                                     discriminator=discriminator)
    manager    =    tf.train.CheckpointManager(checkpoint,    checkpoint_dir,
max_to_keep=1)

    fit(train_dataset, ITERATIONS, ECHO)

    generator.save("/SRGAN/generator")
    discriminator.save("/SRGAN/discriminator")
```

ДОДАТОК В

Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для відновлення зображень з використанням генеративно-змагальних мереж

Виконав:
ст. гр. КНТ-127 А. Є. Діденко

Керівник:
доцент, к.т.н. А. О. Олійник

2021

Рисунок В.1 – Слайд №1

Мета, об’єкт, предмет дослідження

Мета роботи – програмна реалізація методів відновлення зображень з використанням генеративно-змагальних мереж.

Об’єкт дослідження – процес відновлення зображень.

Предмет дослідження – методи відновлення зображень з використанням глибокого навчання.

2

Рисунок В.2 – Слайд №2

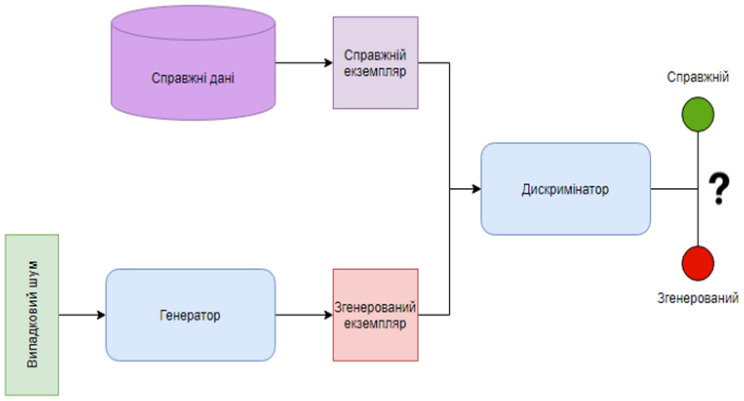
Порівняння існуючих методів

Характеристика	Класичні алгоритми	Згорткові нейронні мережі	Генеративно-змагальні мережі
Складність реалізації	Низька	Помірна	Висока
Точність роботи	Низька	Помірна	Висока
Швидкість роботи	Висока	Помірна	Помірна
Необхідність навчання	Ні	Так	Так
Здатність до узагальнення	Низька	Помірна	Висока
Залежність від обчислювальної потужності	Низька	Висока	Висока

3

Рисунок В.3 – Слайд №3

Генеративно-змагальні мережі

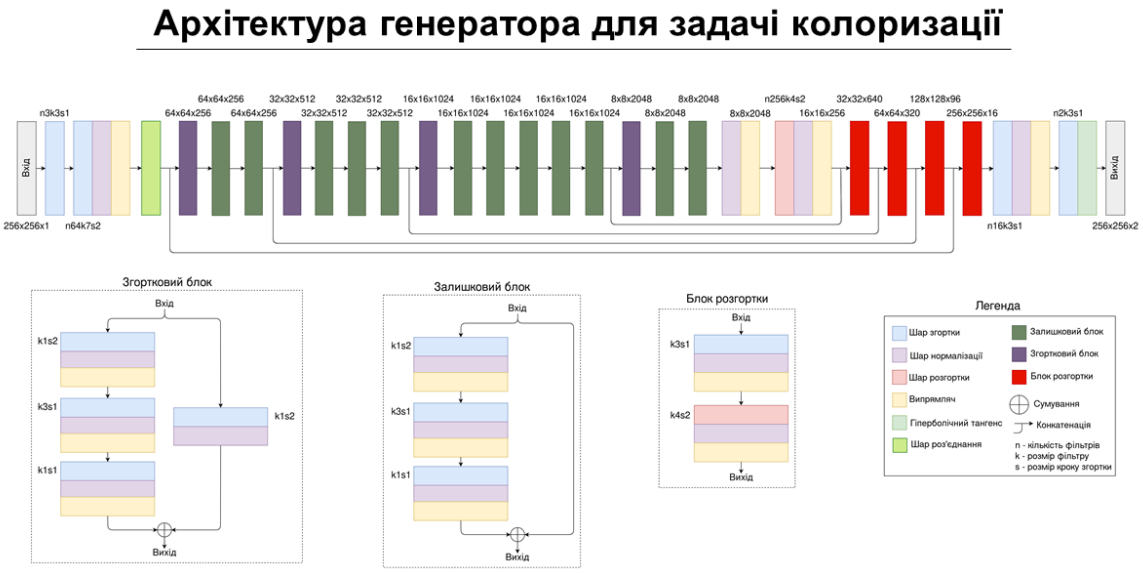


Формальна постановка задачі генеративно-змагальних мереж:

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

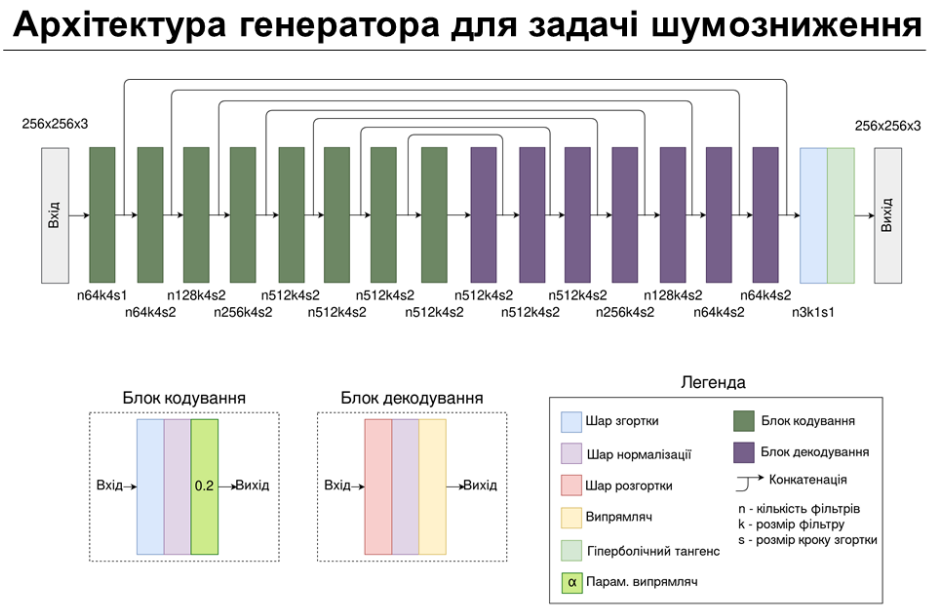
4

Рисунок В.4 – Слайд №4



5

Рисунок В.5 – Слайд №5



6

Рисунок В.6 – Слайд №6

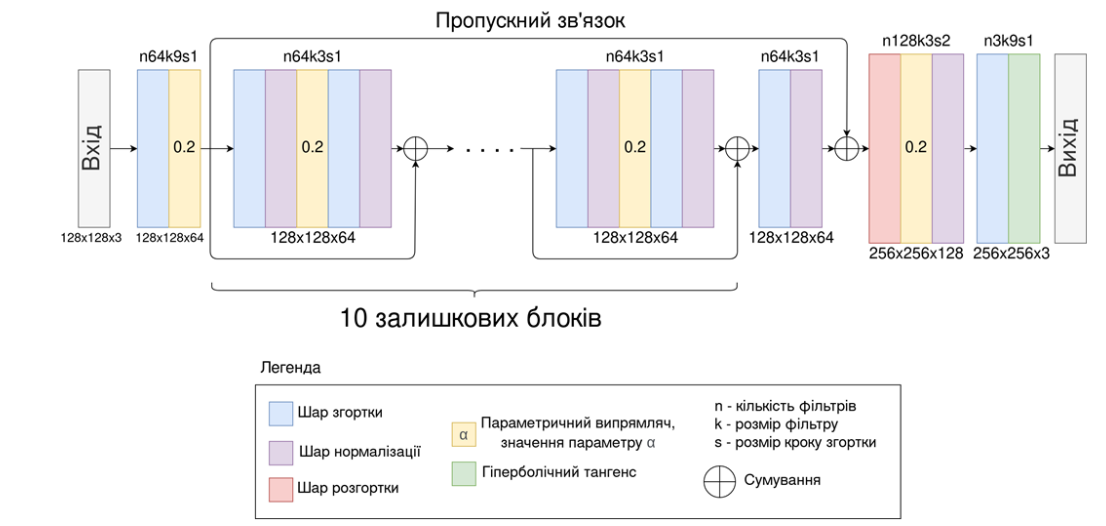


Рисунок В.7 – Слайд №7

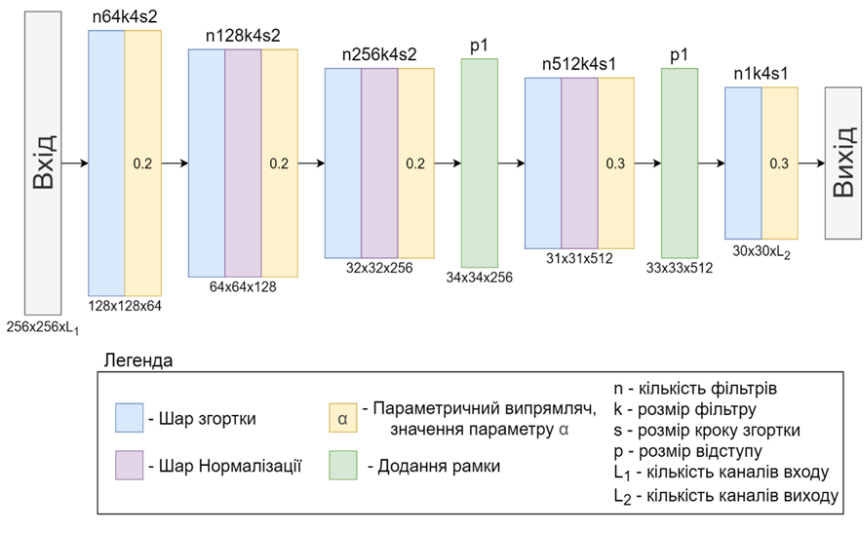


Рисунок В.8 – Слайд №8

Вибір мови програмування

Критерій	C++	Java	Python
Кросплатформність	Так	Так	Так
Типізація	Статична	Статична	Динамічна
Швидкість виконання	+++	++	+
Простота і читабельність	+	++	+++
Наявність сторонніх бібліотек	+	++	+++
Підтримка аналізу данних	+	+	+++

Обрана мова програмування - Python

9

Рисунок В.9 – Слайд №9

Вибір фреймворку глибинного навчання

Критерій	Tensorflow	Keras	PyTorch	Theano	Caffe	ONNX
Швидкодія	++++	+++	+++	++	++	++
Інтуїтивна зрозумілість	++	++++	+++	+	++	++
Документація	++++	+++	+++	++	+	+
Спільнота	++++	+++	++	+	+	+
Підтримка GPU	++++	+++	+++	+	+	+
Гнучкість	+++	++	++	+	+	+
Відлагодження	++	+	+++	+	+	+
Оновлення	+++	+++	+++	+	+	++

Обраний фреймворк – Tensorflow з обгорткою Keras

10

Рисунок В.10 – Слайд №10

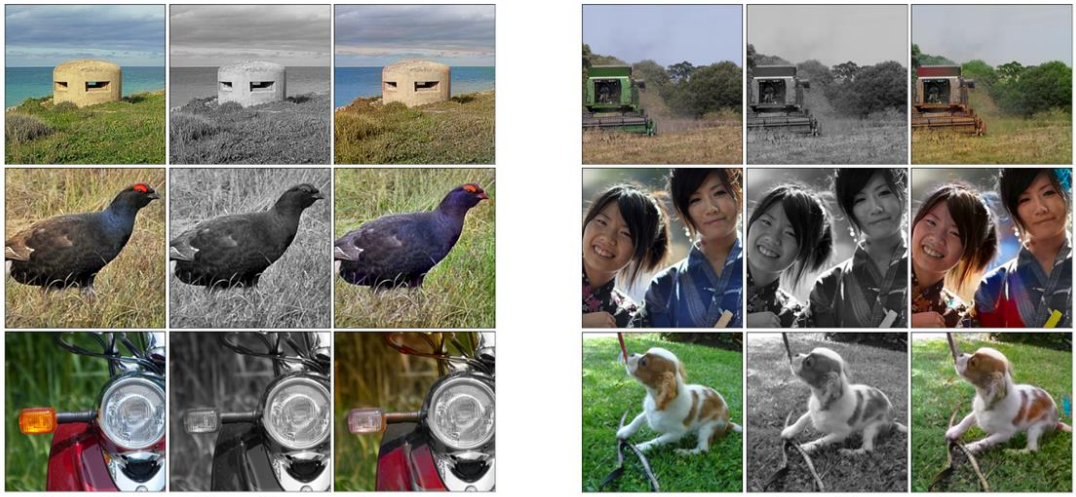
Вибір гіперпараметрів навчання

Параметр	Значення
Датасет	ImageNet
Розмір навчальної вибірки	45000
Розмір тестової вибірки	5000
Розмір пакету даних	16
Кількість епох	40
Метод оптимізації	Adam
Крок навчання	0,002
Функція втрат генератора колоризації	BC + 100*MAE
Функція втрат генератора шумозниження	BC + 100*MAE
Функція втрат генератора ЗРЗ	0,001*BC + VGGloss
Функція втрат дискримінатора	BC(real) + BC(gen)

11

Рисунок В.11 – Слайд №11

Результати колоризації



Розрахована метрика SSIM дорівнює 0,79

12

Рисунок В.12 – Слайд №12

Рисунок В.13 – Слайд №13

Рисунок В.14 – Слайд №14

Висновки

- Проаналізовано предметну область та порівняно існуючі методи;
- Розроблено архітектури і навчено генеративно-змагальні мережі;
- Створено графічний інтерфейс взаємодії користувача із ГЗМ.

15

Рисунок В.15 – Слайд №15

Подальші вдосконалення

- Збільшення розміру навчальної вибірки;
- Проведення експериментів з іншими архітектурами;
- Збільшення кількості епох навчання;
- Розширення списку задач відновлення.

16

Рисунок В.16 – Слайд №16