Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж

Виконала: студентка IV курсу, групи 2ПІ-20б

спеціальності

121 – Інженерія програмного забезпечення

(шифр і назва напряму підготовки, спеціальності)

Хортюк Діана Сергіївна

(прізвище та ініціали)

Керівник: д.т.н., професор каф. ПЗ Романюк О.Н.

(прізвище та ініціали)

Рецензент: к.т.н., доцент каф. ОТ Черняк О.І.

(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О.Н. Романюк

«__» _____ 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 121 – «Інженерія програмного забезпечення»
Освітньо-професійна програма – «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
О. Н. Романюк _____
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Хортюк Діані Сергіївні

1. Тема роботи – розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж

Керівник роботи: Романюк Олександр Никифорович, д.т.н., професор, завідувач кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: методи реставрації зображень – нейронні мережі з використанням згорткових та рекурентних шарів; моделі – архітектури ResNet, U-Net, та їх комбінації; автоматизація виявлення та виправлення артефактів, відновлення кольору та роздільної якості зображень; кольоровий режим – TrueColor, мова програмування – Java, середовище – IntelliJ IDEA.

4. Зміст текстової частини: вступ, аналіз підходів до процесу реставрації зображень з використанням нейронних мереж, розробка методів для реставрації зображень з використанням нейронних мереж, розробка програмного забезпечення для реставрації зображень, тестування програмного забезпечення, висновки, список використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: титульний аркуш, аналіз предметної галузі, аналіз використання нейронних мереж у задачах реставрації зображень, мета дослідження, розробка методів реставрації зображень, розробка модулів програмного забезпечення, інтерфейс програмного забезпечення, наукова новизна та практичне значення дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата			
		Завдання видав		Завдання прийняв	
1-4	Романюк О.Н., д.т.н., професор каф ПЗ	13.03.24		03.06.24	

7. Дата видачі завдання 13 березня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з\п	Назва етапів бакалаврської дипломна роботи	Строк виконання етапів роботи	Примітка
1	Аналіз підходів до процесу реставрації зображень з використанням нейронних мереж	14.03.24 – 22.03.24	Вик.
2	Розробка методів для реставрації зображень з використанням нейронних мереж	22.03.24 – 19.04.24	Вик.
3	Розробка алгоритмів роботи програми	19.04.24 – 10.05.24	Вик.
4	Розробка програмного продукту	10.05.24 – 24.05.24	Вик.
5	Тестування роботи програми	24.05.24 – 27.05.24	Вик.
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24	Вик.

Студент _____ **Д. С. Хортюк**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **О. Н. Романюк**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 00492

Хортюк Д. С. Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж : бакалаврська дипломна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 61 с.

На укр. мові. Бібліогр. : 23 назв ; рис. : 24; табл. 3.

У бакалаврській дипломній роботі було розроблено методи та програмні модулі для реставрації зображень з використанням нейронних мереж. Проведено аналіз відомих підходів до процесу реставрації зображень з використанням нейронних мереж. Обгрунтовано необхідність розробки методів реставрації зображень з використанням нейронних мереж з метою підвищення роздільної якості пошкоджених зображень з можливістю реставрації за рахунок використанням нейронної мережі, що дозволяє знаходити та аналізувати пошкоджені частини фотографії.

Розроблено методи та програмні модулі реставрації вхідних зображень використовуючи засоби нейронних мереж, що дозволяють виконати попередню обробку та аналіз вхідного зображення та виявити дефекти.

На основі проведених у БДР досліджень розроблено алгоритми та програмні засоби для реставрації зображень. Проведене тестування підтвердило достовірність теоритичних досліджень методів реставрації зображень та засобів їх реалізації.

Розроблений програмний застосунок було написано, використовуючи мови програмування Java і Python. Для розробки використовувалися середовища IntelliJ IDEA 2023. Розроблений програмний продукт відповідає поставленій меті та завданням.

Ключові слова: реставрація зображення, відновлення кольору, обробка зображень, нейронна мережа, програмне забезпечення.

ABSTRACT

UDC 00492

Khortiuk D. S. Development of software for image restoration using neural networks : bachelor's thesis in specialty 121 - Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2024. 61 c.

In Ukrainian. Bibliography: 23 titles; Figures: 24; Table 3.

In the bachelor's thesis, methods and program modules for image restoration using neural networks were developed. The analysis of known approaches to the process of image restoration using neural networks was carried out. The necessity of developing methods of image restoration using neural networks in order to improve the resolution quality of damaged images with the possibility of restoration by using a neural network that allows you to find and analyze damaged parts of the photo is substantiated.

Methods and software modules for restoration of input images using neural networks have been developed, which allow pre-processing and analysis of the input image and detection of defects.

Based on the research conducted at the GDR, algorithms and software tools for image restoration were developed. The testing confirmed the reliability of the theoretical studies of image restoration methods and their implementation tools.

The developed software application was written using the Java and Python programming languages. The IntelliJ IDEA 2023 environment was used for development. The developed software product meets the goal and objectives.

Keywords: image restoration, color restoration, image processing, neural network, software.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

У сучасному світі, де візуальна інформація є однією з основних форм спілкування, розробка та вдосконалення методів реставрації зображень стає надзвичайно актуальною та важливою задачею. Завдяки стрімкому прогресу в області обробки зображень та штучного інтелекту, можливості у сфері реставрації зображень набувають значного підвищення потужності та ефективності.

Високороздільна реставрація зображень, яка полягає в відновленні високоякісного зображення з початкового низькороздільного зображення, стає важливим завданням у сфері комп'ютерного зору [1]. Ця технологія знаходить широке застосування в різних галузях, включаючи медичну кіноіндустрію, сферу культури, а також області спостереження та безпеки [2]. Покращення візуальної якості зображень через високороздільну реставрацію може сприяти виконанню різноманітних завдань у сфері комп'ютерного зору [3, 4].

У останні роки для вирішення цієї проблеми було запропоновано безліч методів, що базуються на глибокому машинному навчанні, які демонструють значну перевагу над традиційними підходами [5].

Використання глибоких нейронних мереж для виконання завдання високороздільної реставрації зображень забезпечує ефективне відновлення роздільної якості зображень та відповідає вимогам різноманітних застосувань у практичній діяльності. Такі мережі можуть застосовувати різноманітні архітектури, включаючи згорткові та рекурентні шари, активуючі функції та оптимізатори, щоб здійснювати складні обчислення та вдосконалювати якість відновлення зображень.

На даний момент існує значна кількість архівних зображень, які мають різноманітні дефекти. Процес ручної реставрації цих зображень є надзвичайно трудомістким та малоефективним, що обумовлює великі витрати часу і ресурсів. Використання комп’ютерних технологій дозволить підвищити якість реставрації

зображень. У зв'язку з цим, розробка програмних засобів для автоматизованої реставрації зображень набуває особливої актуальності.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності реставрації архівних зображень за рахунок використанням нейронної мережі.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **задачі**:

- аналіз підходів до реставрації зображень з використанням нейронних мереж;
- розробка алгоритму навчання нейронної мережі, який буде спроможний аналізувати вхідні данні та навчати нейронну мережу виявляти дефекти на зображеннях;
- розробити алгоритм покращення роздільної здатності зображення з використанням засобів нейронної мережі;
- розробити алгоритм реставрації кольору зображення з використанням засобів нейронної мережі;
- розробити алгоритм використання нейронної мережі, який буде спроможний виявляти пошкодження, реставрувати та покращувати роздільну здатність зображення;
- тестування розроблених програмних продуктів.

Об'єкт дослідження – процес реставрації зображень з використанням нейронних мереж.

Предмет дослідження – методи та засоби реставрації зображень з використанням нейронних мереж.

Методи дослідження. У процесі дослідження застосовувалися: теорія нейронних мереж; теорія алгоритмів для обробки та аналізу зображень та методи і засоби нейронних мереж для реставрації пошкоджених зображень,

комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Вперше запропоновано метод для реставрації зображень з використанням нейронних мереж, особливість полягає у використанні комбінації різних архітектурних моделей нейронних мереж для автоматичного виявлення та виправлення артефактів, відновлення кольору та підвищення роздільної якості зображень, що дозволяє автоматизувати процес реставрації.

Практичне значення отриманих результатів полягає у розробці програмного забезпечення, яке дозволить виконувати реставрацію та покращувати роздільну здатність зображень з використанням нейронних мереж.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, студенту належать: [6] – проаналізовано можливості застосування нейронних мереж у задачі реставрації зображень.

Апробація матеріалів бакалаврської дипломної роботи. Основні положення БДР доповідалися та обговорювалися на Міжнародних і Всеукраїнських конференціях: XXXII Міжнародна науково-практична конференція «Інформаційні технології: наука, техніка, технологія, освіта, здоров’я (MicroCAD-2024)» (м. Харків, 2024).

Публікації. За темою бакалаврської роботи надруковано 1 праця у матеріалах міжнародної науково-практичної конференції.

Структура та обсяг БДР. Бакалаврська дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ ПІДХОДІВ ДО ПРОЦЕСУ РЕСТАВРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

1.1 Аналіз предметної галузі

У сучасному інформаційному просторі використання графічних зображень для відображення інформаційного змісту набуло великої популярності. Зображення стали невід'ємною частиною веб-сайтів, додатків та медіа-контенту, які споживачі все більше вимагають у якісному та естетичному вигляді.

Зображення дозволяють нам надати більш детальний опис продуктів та послуг. Проте разом з підвищенням якості контенту зросли й вимоги до швидкості передачі даних і комфорту користувача. Тому зображення для веб-сторінок і додатків оптимізуються для швидкодії. Нерідко така оптимізація може призводити до втрати якості зображень, що може стати проблемою, якщо використовувати їх окремо. Саме тому технології реставрації зображень здійснюють стрімкий розвиток.

Процес реставрації зображення – це комплексний набір методів і технік, спрямованих на відновлення пошкоджених або втрачених частин зображення з метою поліпшення його якості, чіткості, кольору та візуальної привабливості. Цей процес може включати в себе ряд операцій, таких як видалення шуму, реставрація деталей, відновлення текстури, усунення артефактів або роботу з кольором.

Традиційні методи реставрації зображень, такі як фільтри та інтерполяція, зазвичай базуються на математичних моделях або евристичних підходах до відновлення пошкоджених або втрачених даних на зображенні. Хоча ці методи можуть бути ефективними у деяких ситуаціях, вони часто не забезпечують достатньої якості відновлення для вимог сучасних застосувань. Наприклад, фільтри можуть впливати на розмиття або втрату деталей, особливо в складних або контрастних областях зображення, тоді як інтерполяція може призводити до збоїв у формі або структурі зображення, особливо коли відсутня достатня

інформація про контекст.

У зв'язку з цим, постає необхідність у розробці нових методів, які були б більш точними та надійними у відновленні зображень. Одним із напрямків у цьому відношенні є використання нейронних мереж, заснованих на глибокому навчанні. Вони демонструють значні переваги у порівнянні з традиційними методами, оскільки можуть автоматично вивчати складні залежності між пікселями та контекстуальною інформацією на зображенні, що дозволяє отримувати більш точні та природні результати відновлення. Крім того, нейронні мережі здатні адаптуватися до різноманітних умов та пошкоджень, що робить їх ефективними для різноманітних застосувань у сучасних системах реставрації зображень. Тому, застосування нейронних мереж у реставрації зображень відкриває нові можливості, оскільки вони здатні враховувати складні шаблони та контекстну інформацію при відновленні

Останні дослідження в галузі реставрації зображень свідчать про широке використання нейронних мереж з різноманітними архітектурами, що включають глибокі згорткові нейронні мережі (CNN) та генеративні адверсаріальні мережі (GAN). Ці архітектури виявилися особливо ефективними в роботі з відновленням зображень, оскільки вони дозволяють враховувати складні шаблони та контекстну інформацію під час відновлення, що призводить до покращення якості результуючих зображень [6].

Глибокі згорткові мережі, зокрема, використовуються для автоматичного виявлення та відновлення деталей та текстур на зображеннях. Вони можуть ефективно працювати зі складними взаємозв'язками між пікселями та контекстуальною інформацією, що дозволяє отримувати більш точні та реалістичні результати. Генеративні адверсаріальні мережі (GAN) є ще однією потужною архітектурою, яка використовується для реставрації зображень. У GAN відбувається конкуренція між двома нейронними мережами - генератором і дискримінатором, що сприяє створенню більш реалістичних та природних зображень.

Крім того, наявність відкритих джерел та бібліотек програмного

забезпечення, таких як TensorFlow, PyTorch та Keras, робить доступ до цих технологій більш простим та доступним для дослідників та розробників. Ці інструменти сприяють швидкому поширенню та розвитку досліджень у галузі реставрації зображень з використанням нейронних мереж, створюючи умови для подальших інновацій та покращення якості та ефективності відновлення зображень.

Використання нейронних мереж у системах реставрації зображень дозволяє отримувати значні переваги порівняно з традиційними методами. Однією з головних переваг є їх здатність до врахування складних взаємозв'язків між пікселями та контекстуальною інформацією на зображенні. Нейронні мережі здатні виявляти і аналізувати різноманітні шаблони та структури в зображенні, враховуючи їхні взаємозв'язки та контекст, що дозволяє їм глибше розуміти зміст та сутність зображення. Пікселі на зображенні мають складні взаємозв'язки як у просторовому, так і в часовому вимірах.

Крім того, нейронні мережі можуть адаптуватися до різноманітних структур та шаблонів зображення. Вони можуть розпізнавати і адаптуватися до різних типів текстур, контрастів та особливостей освітлення, що робить їх ефективними в різних сценаріях відновлення зображень. Ця гнучкість дозволяє нейронним мережам здійснювати більш точне та природне відновлення зображень, ніж традиційні методи.

Ще одною перевагою використання нейронних мереж у області реставрації зображень є поліпшена якість отриманих даних. Застосування глибоких нейронних мереж дозволяє відтворювати деталі та текстури на зображенні, що втратилися через стиснення або шум.

Отже, аналіз сучасних технологій реставрації зображень з використанням нейронних мереж свідчить про їх значний потенціал у вдосконаленні процесу відновлення зображень та покращенні якості результуючих зображень. Розробка програмних засобів для впровадження реставрації зображень з використанням нейронних мереж стає необхідним для забезпечення процесу реставрації зображень.

1.2 Аналіз методів для реставрації зображень та обгрунтування вибору технологій для реалізації задачі

Розробка програмних засобів для реставрації зображень з використанням нейронних мереж потребує ретельного аналізу доступних методів, які можуть бути використані для досягнення оптимальних результатів поліпшення якості та реставрації зображень з використанням нейронних мереж. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких програмних модулів.

Один з основних аспектів розробки програмних засобів для реставрації зображень – це тренування нейронної мережі на великому об’ємі даних. Це передбачає використання наборів зображень, які містять як пошкоджені, так і відновлені версії, для навчання моделі розрізняти шум, дефекти та відновлені області.

Один із можливих методів – це використання глибоких згорткових нейронних мереж для реконструкції пошкоджених зображень [7]. Цей підхід полягає у використанні заздалегідь навченої моделі для відновлення дефектних частин зображення на основі зразків, зібраних під час тренування. Перевагою цього методу є висока швидкодія та здатність до відновлення якісних зображень навіть у реальному часі. Однак, точність відновлення може залежати від якості та кількості тренувальних даних.

Інший підхід включає використання генеративно-зустрічних мереж (GAN) [8], які здатні генерувати реалістичні відновлені зображення з пошкоджених вхідних даних. Модель складається з двох частин: генератора, який створює відновлені зображення, та дискримінатора, який намагається відрізнити справжні від вигаданих зображень. Цей підхід дозволяє отримувати деталізовані та реалістичні результати відновлення, але може вимагати більше обчислювальних ресурсів та часу для тренування моделі.

Ще одним підходом до реставрації зображень є використання автоенкодерів [9] – тип нейронних мереж, які навчаються репрезентувати вхідні

дані в компактному коді, після чого відтворювати їх назад в оригінальний вигляд. Для реставрації зображень автоенкодери використовуються таким чином, що вхідні зображення пошкоджуються шумом або дефектами, а потім передаються через автоенкодер для відновлення їх оригінального вигляду. Перевагою цього методу є його здатність до відновлення дефектних зображень з високою точністю, навіть при наявності значного рівня шуму. Автоенкодери можуть ефективно виявляти та видаляти шум, підсилюючи лише корисну інформацію на зображенні. Крім того, цей підхід є досить ефективним у використанні та не вимагає великої кількості обчислювальних ресурсів. Однак автоенкодери можуть страждати від обмежень у відновленні складних дефектів або втрати деталей у великих областях зображення.

Розглядаючи різні методи розв’язання задачі реставрації зображень з використанням нейронних мереж, виявлено, що кожен з них має свої переваги та недоліки. У ході аналізу можливих рішень було вирішено використовувати комбінацію методів для досягнення найкращих результатів у відновленні зображень. Зокрема, планується використання глибоких згорткових нейронних мереж для реконструкції пошкоджених зображень у поєднанні з генеративно-зустрічними мережами для отримання реалістичних результатів та видалення шумів. Такий підхід має потенціал забезпечити оптимальні результати реставрації зображень з використанням нейронних мереж.

При розробці програмного забезпечення для реставрації зображень з використанням нейронних мереж важливо обрати найбільш оптимальні технології для розробки кожного модуля програмних засобів.

Java – [10] це об'єктно-орієнтована, кросплатформена мова програмування, яка використовується для розробки різноманітних програмних додатків, від веб-додатків до мобільних застосунків та великих корпоративних систем. Вона відома своєю простотою, надійністю та широким спектром функцій, які полегшують розробку та підтримку програм.

JavaFX – [11] це фреймворк для розробки графічного інтерфейсу користувача (GUI) у програмах, написаних на мові програмування Java. Він

надає набір інструментів та бібліотек для створення сучасних та естетичних GUI додатків для різних платформ, включаючи настільні комп'ютери, мобільні пристрої та вбудовані системи. JavaFX дозволяє розробникам легко створювати вікна, кнопки, таблиці, графіки та інші елементи інтерфейсу з допомогою XML-подібної мови розмітки (FXML) або програмного коду. Він також підтримує анімацію, мультимедіа, веб-відображення та інші сучасні можливості для покращення взаємодії з користувачем.

Python – [12] це високорівнева інтерпретована мова програмування, що має велику кількість бібліотек і інструментів для роботи із комп’ютерним зором та нейронними мережами.

SceneBuilder – [13] це графічний інструмент для швидкої та зручної розробки інтерфейсів користувача на основі технології JavaFX. Він дозволяє розміщувати та налаштовувати різноманітні візуальні елементи, такі як кнопки, поля введення, таблиці тощо, шляхом простого перетягування та розміщення на сцені. SceneBuilder забезпечує зручне візуальне моделювання інтерфейсу, що спрощує процес розробки програмних додатків на основі JavaFX.

PyTorch – [14] це фреймворк для машинного навчання та глибокого навчання, який дозволяє створювати, навчати та використовувати нейронні мережі. Він відрізняється своєю гнучкістю, швидкістю та простотою використання, що робить його популярним інструментом для дослідження та розробки алгоритмів штучного інтелекту.

TensorFlow – [15] це відкрита платформа для машинного навчання та глибокого навчання, розроблена компанією Google. Вона надає широкі можливості для розробки та навчання нейронних мереж, включаючи різноманітні алгоритми для розпізнавання образів, обробки природних мов, рекомендаційних систем і багато іншого. TensorFlow забезпечує гнучкий та ефективний інструментарій для роботи з великими обсягами даних та розгортання навчених моделей на різних платформах.

Keras – [16] це високорівнева бібліотека для мови програмування Python, яка спрощує процес розробки та навчання глибоких нейронних мереж. Вона

надає простий та інтуїтивно зрозумілий інтерфейс для побудови та навчання нейронних мереж, що дозволяє швидко створювати та експериментувати з різними архітектурами моделей. Keras також підтримує інтеграцію з іншими відомими бібліотеками для глибокого навчання, такими як TensorFlow і OpenCV, що робить її потужним інструментом для розробки і використання нейронних мереж.

OpenCV (Open Source Computer Vision Library) – [17] це бібліотека комп'ютерного зору та машинного навчання з відкритим вихідним кодом, яка надає широкі можливості для обробки зображень та відео. Вона включає в себе реалізації багатьох алгоритмів обробки зображень, від простих операцій до складних завдань, таких як розпізнавання облич та виявлення об'єктів.

Поєднання цих технологій дозволяє розробити програмні засоби для реставрації зображень з використанням нейронних мереж. Python та бібліотеки PyTorch і Keras забезпечують ефективну кодову базу для навчання нейронних мереж, в той час як Java та фреймворки JavaFx й SceneBuilder надають розширені інструменти для створення зручного інтерфейсу користувача. Використанням бібліотеки OpenCV спростить процес аналізу зображення та дозволить акцентувати увагу реставрації на необхідних ділянках зображення. Бібліотека TensorFlow надасть функціонал для функціоналу з відновлення кольорової гами зображення.

1.3 Аналіз використання нейронних мереж у задачах обробки зображень

Нейронні мережі стали фундаментальним інструментом у сучасній обробці зображень, завдяки їхній здатності впроваджувати складні математичні алгоритми для аналізу та інтерпретації великих обсягів зображень. Їх використання призводить до вражаючої точності в класифікації зображень, де найкращі моделі можуть досягати помилок на рівні кількох відсотків при обробці тисяч зображень, які не були використані для навчання мережі. Одним з

ключових напрямків застосування нейронних мереж є обробка зображень, що полягає у визначенні категорії, до якої належить кожне зображення. Цей процес знайшов широке застосування у таких галузях, як медицина, технології безпеки, автомобільна промисловість та інші, де він використовується для автоматизації виявлення та аналізу об'єктів на зображеннях.

Цифрове зображення, зазвичай, представлене у вигляді матриці, де кожен елемент відповідає певному пікселю на зображенні. Кожен піксель кодується числовим значенням, що відображає його інтенсивність або колір. Наприклад, у випадку чорно-білого зображення кожен піксель може мати значення від 0 до 255, де 0 відповідає чорному кольору, а 255 - білому. У зображеннях кольору кожен піксель може бути кодований трьома числовими значеннями, що відповідають інтенсивності червоного, зеленого та синього кольорів (формат RGB).

Нейронні мережі оперують цими числовими значеннями пікселів, виконуючи різні математичні операції, такі як множення та додавання, над вхідними даними. Кожен шар мережі приймає вхідні дані з попереднього шару, обробляє їх і передає до наступного шару. Різні типи шарів можуть виконувати різні операції, такі як згорткові операції, операції пулінгу, повністю зв'язані шари тощо.

Процес обробки зображення нейронною мережею включає кілька ключових кроків, що відбуваються в різних шарах мережі:

1. Згорткові операції (Convolutional Operations): перший шар зазвичай виконує згорткові операції над вхідним зображенням. Кожен фільтр (ядро) згортки обробляє невеликий регіон вхідного зображення та виконує згортку для виділення певних ознак, таких як контури або текстури. Результатом є карта ознак, яка відображає наявність цих ознак у різних частинах зображення.

2. Функція активації (Activation Function): після кожної згорткової операції застосовується нелінійна функція активації, така як ReLU (Rectified Linear Unit) або Sigmoid, для введення нелінійності в мережу та покращення її здатності до вивчення складних залежностей у вхідних даних.

3. Пулінг (Pooling): після кожного шару згортки може бути виконана операція пулінгу, яка зменшує розмір карти ознак та сприяє збереженню важливих ознак. Зазвичай використовується максимальне або середнє пулінг, де вибирається максимальне або середнє значення в кожному підмасиві з карти ознак.

4. Повністю зв'язаний шар (Fully Connected Layer): після кількох шарів згортки та пулінгу зазвичай додається один або кілька повністю зв'язаних шарів. Вони приймають вектор ознак з попереднього шару та виконують операції множення на ваги та додавання зсуву. Це дозволяє моделі генерувати складніші взаємозв'язки між ознаками та вибирати найважливіші аспекти зображення для їх обробки та аналізу.

5. Функція втрат (Loss Function): після останнього повністю зв'язаного шару використовується функція втрат для оцінки того, наскільки добре мережа виконує обробку зображення. Це може бути середньоквадратична помилка для задач регресії або категоріальна крос-ентропія для задач класифікації.

6. Оптимізація (Optimization): після обчислення функції втрат використовується алгоритм оптимізації, такий як стохастичний градієнтний спуск (SGD) або його варіанти, для оновлення ваг моделі з метою мінімізації втрати.

Обробка зображень за допомогою нейронних мереж для виявлення дефектів, втрат кольору, артефактів та низької роздільної здатності виконується шляхом застосування алгоритмів комп'ютерного зору. Під час цього процесу, нейронна мережа отримує на вхід цифрове зображення та аналізує його піксель за пікселем, використовуючи математичні операції та вагові коефіцієнти, щоб визначити різноманітні аспекти якості та цілісності зображення.

Одним з основних завдань є виявлення дефектів на зображенні. Це може включати виявлення розривів, подряпин, розмивання або інших дефектів, які можуть впливати на якість зображення. Для цього нейронні мережі використовують методи виявлення зразків та аномалій, які дозволяють виявляти навіть незначні зміни у зображенні, що можуть вказувати на наявність дефекту.

Також, нейронні мережі можуть використовуватися для виявлення втрати кольору або артефактів на зображенні. Це включає в себе виявлення змін у колірній палітрі, неправильне відтворення кольорів, а також появу небажаних артефактів, таких як шум, розмиття або аномальні піксельні значення.

Додатково, нейромережі можуть виявляти низьку роздільну здатність зображень, що означає недостатню чіткість та деталізацію зображення. Це може бути викликано різними факторами, такими як низька якість обладнання, стиснення зображень або інші фактори, що призводять до втрати деталей. Нейронні мережі можуть аналізувати структуру зображення та виявляти області з низькою роздільною здатністю, щоб підвищити якість зображення шляхом відновлення втрачених деталей або усунення артефактів.

Усі ці завдання виконуються за допомогою складних алгоритмів обробки зображень, які вбудовані в нейронні мережі та оптимізовані для виконання на великих обсягах даних. Використання нейронних мереж у цих задачах дозволяє автоматизувати процес виявлення та виправлення дефектів на зображеннях.

1.4 Аналіз аналогів та постановка задачі

В останні роки використання технологій реставрації зображень з нейронними мережами набуває все більшого попиту та застосування у різних галузях. Ці технології інтегруються у різноманітні програмні засоби та системи, які стикаються з проблемою відновлення якості зображень. Розробка програмних засобів для реставрації зображень з використанням нейронних мереж дозволить підвищити якість та ефективність аналізу та обробки зображень, що є важливим для швидкого виявлення дефектів, зниження шуму та покращення загального візуального враження. До найбільш відомих рішень відносять:

- Deep-Image;
- Topaz Gigapixel AI;
- Let’s Enhance;

- AI Image Enlarge;
- DeepArt.

Одним із прикладів сучасного програмного засобу для реставрації зображень з використанням нейронних мереж є сервіс Deep-Image (рис. 1.1). Цей онлайн-сервіс спеціалізується на збільшенні роздільної здатності зображень і використовує глибокі згорткові нейронні мережі для досягнення цієї мети [18]. Однією з основних функцій Deep-Image є збільшення зображень у 2, 3, або навіть 4 рази, з обмеженням у 25 мегапікселів. Крім того, сервіс пропонує додаткові опції, такі як придушення шуму та зниження артефактів стиснення JPEG, що робить його дуже зручним для поліпшення якості зображень у різних умовах. Користувачі можуть завантажити зображення на сервіс як зі свого комп'ютера, так і з хмарного сховища Google Drive, що робить його доступним та зручним для широкого кола користувачів. Будучи онлайн-сервісом, Deep-Image не потребує встановлення додаткового програмного забезпечення та дозволяє використовувати свої можливості безпосередньо з веб-браузера. Такий підхід забезпечує максимальну доступність та зручність використання для користувачів з різних країн та платформ.

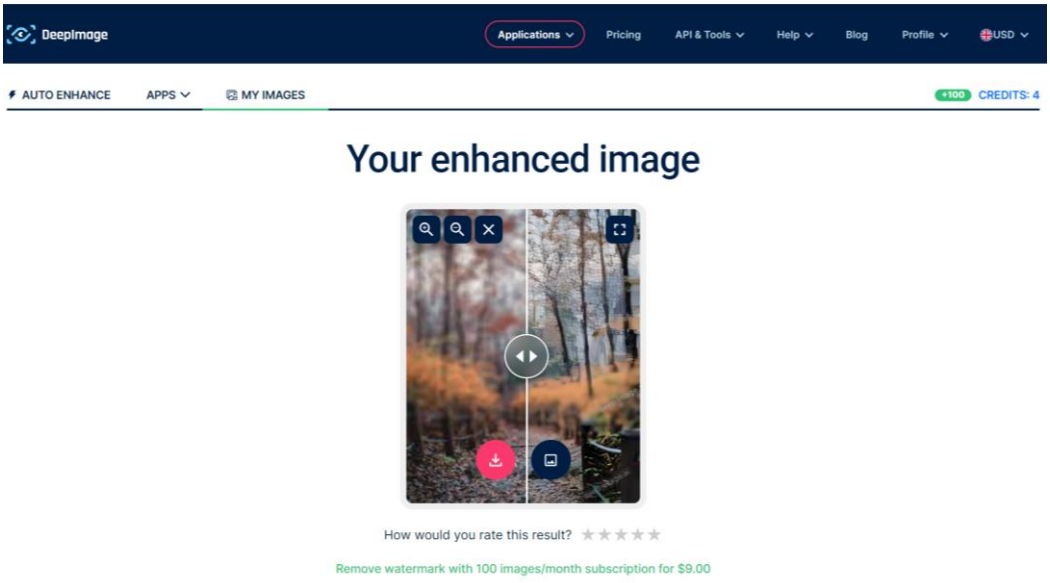

Рисунок 1.1 – Приклад роботи застосунку Deep-Image

Інший приклад – Topaz Gigapixel AI [19] є програмним інструментом, який

використовує надточні нейронні мережі для збільшення роздільної здатності зображень. Він доступний для операційних систем Windows та macOS. Це програмне забезпечення дозволяє збільшувати розмір зображень до 6 разів за допомогою нейронних мереж, а також застосовувати класичні алгоритми для ще більшого збільшення. Крім того, воно має можливість встановлення конкретного розміру в пікселях як по ширині, так і по висоті.

Окрім стандартного режиму, Topaz Gigapixel AI (рис. 1.2) має три додаткових режими:

- режим "Архітектурний", який оптимізований для міських пейзажів та зображень будівель, у якому особлива увага приділяється збереженню прямих ліній на зображенні під час збільшення;

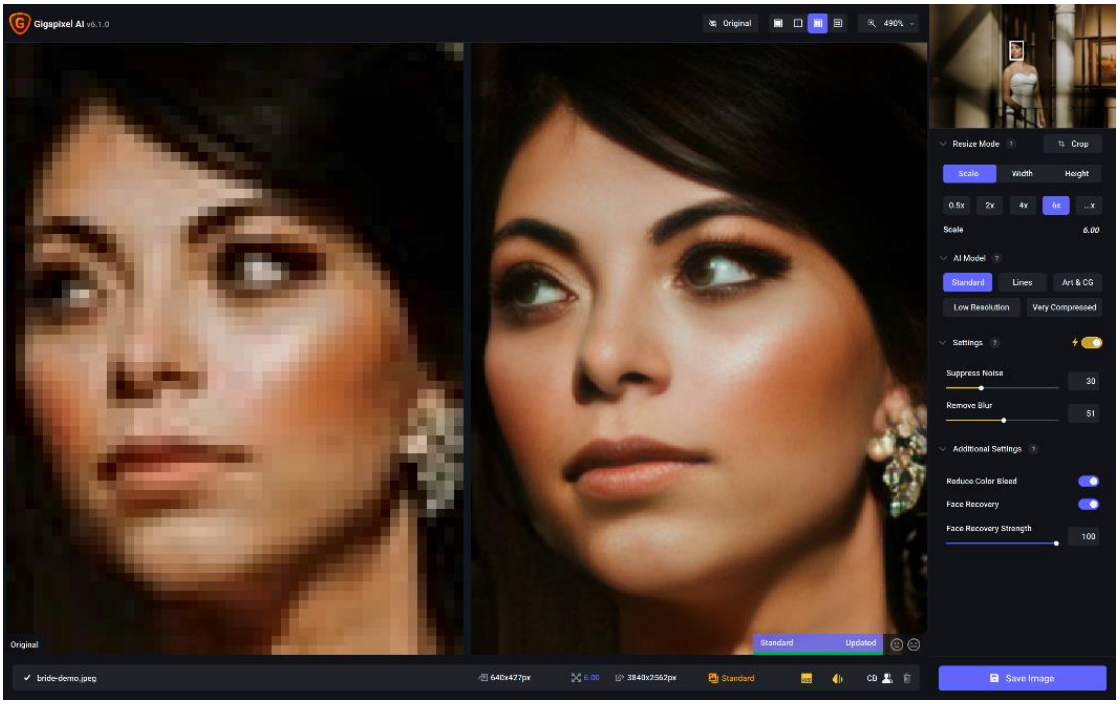

Рисунок 1.2 – Приклад роботи застосунку Topaz Gigapixel AI

- режим "Дуже стиснений", спеціально розроблений для виправлення артефактів, що зазвичай виникають під час стиснення зображень;

- режим "Художнє мистецтво", який оптимізований для збільшення зображень картин та анімацій, зберігаючи їхні деталі та текстури.

Ще одним аналогом є онлайн-сервіс Let's Enhance [20], який базується на

глибоких згорткових мережах для збільшення якості зображень (рис. 1.3). Цей сервіс дозволяє збільшувати розмір зображення до 16 разів з обмеженням у 30 мегапікселів. Перевагою Let's Enhance є можливість встановлення певного розміру зображення у пікселях як по ширині, так і по висоті, що робить його дуже гнучким у використанні. Сервіс пропонує три режими: фото, призначений для зображень архітектури, природи та людей; цифрове мистецтво, спеціально розроблений для малюнків, ілюстрацій, картин та анімацій; режим розумного поліпшення, який спрямований на покращення якості зображення. Способи завантаження зображень на сервіс включають комп'ютер, хмарне сховище Google Drive та за URL-посиланням, що забезпечує зручну і просту можливість використання для користувачів з різних пристроїв та платформ.

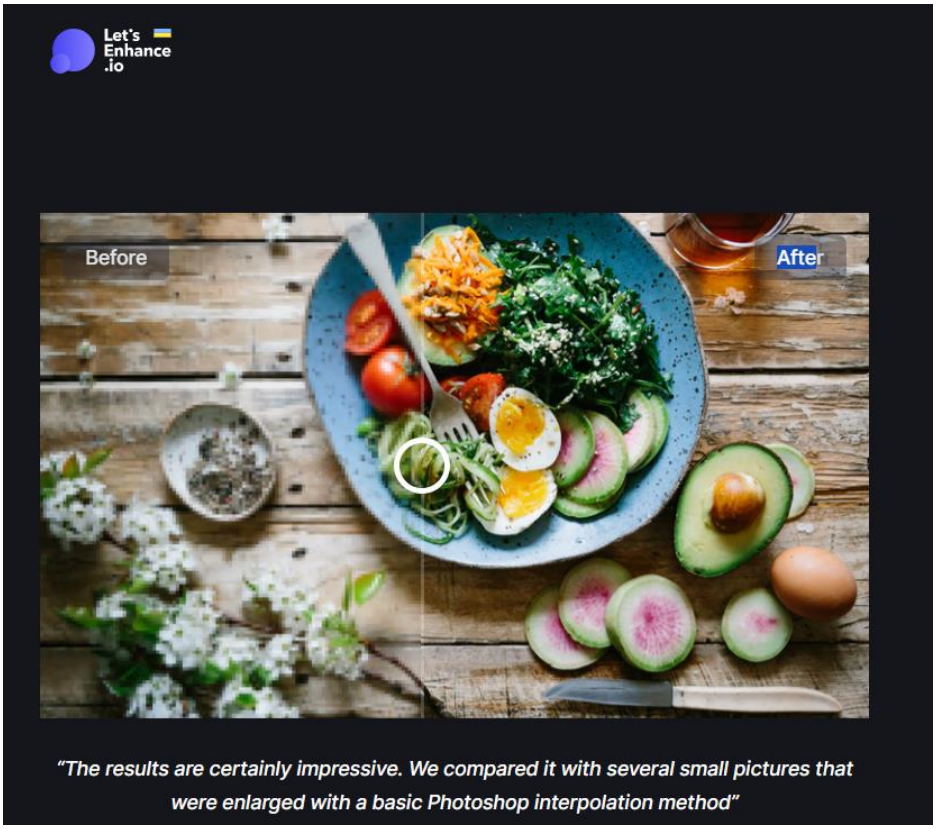

Рисунок 1.3 – Приклад роботи застосунку Let's Enhance

Ще одним аналогом, який варто розглянути, є AI Image Enlarge [21]. Цей сервіс також використовує глибокі згорткові нейронні мережі для збільшення якості зображень (рис. 1.4). AI Image Enlarge надає користувачам можливість

збільшити розмір зображень із збереженням деталей та якості. Завдяки своїй потужній технології, він дозволяє збільшувати зображення без значних втрат чіткості, а також враховує контекст та деталі зображення для оптимального результату. Особливістю AI Image Enlarge є можливість збільшення розміру зображення до 800%, забезпечуючи високу роздільну здатність та якість.

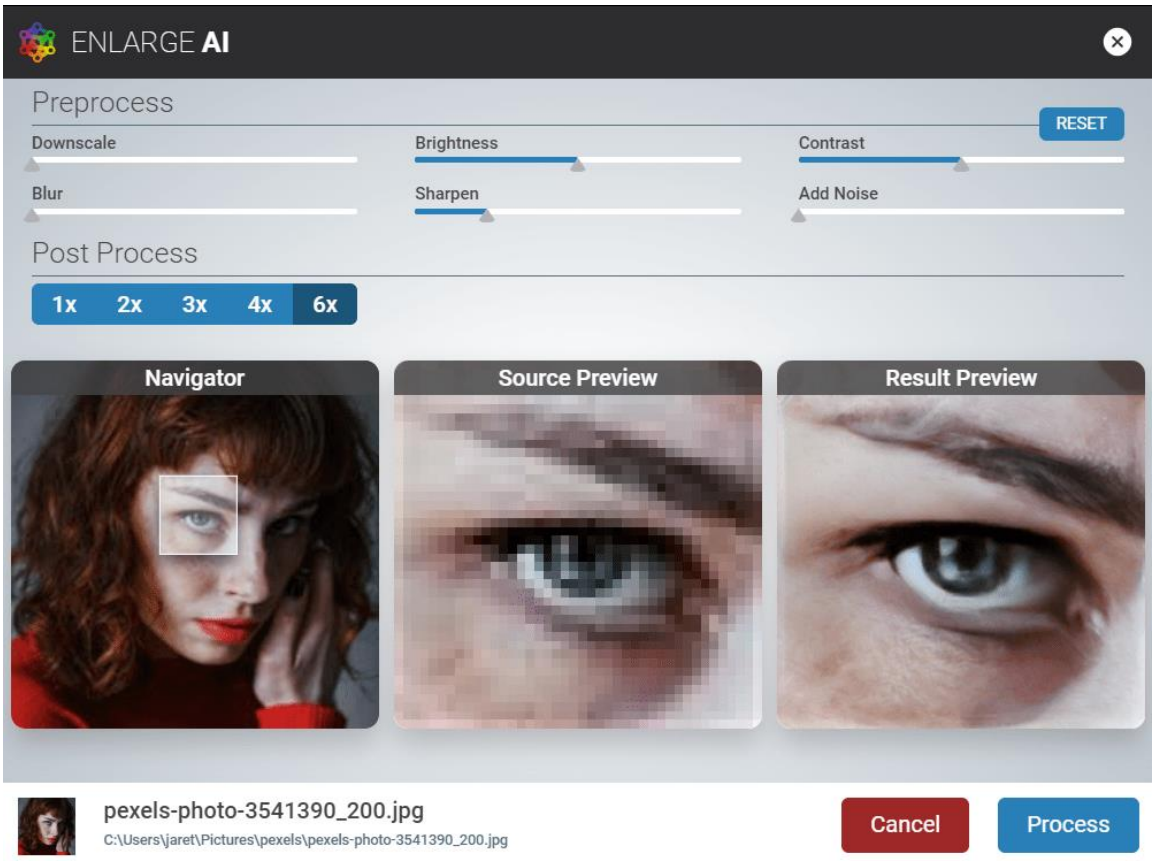

Рисунок 1.4 – Приклад роботи застосунку AI Image Enlarge

DeepArt – [22] платформа, яка використовує складні алгоритми глибокого навчання для аналізу і розуміння мистецького стилю, властивого конкретному зображенню (рис. 1.5). Вона дозволяє користувачам не лише перетворювати зображення в мистецькі шедеври, але й налаштовувати рівень впливу стилю на кінцевий результат. Крім того, DeepArt постійно оновлюється та доповнюється новими мистецькими стилями та алгоритмами, що робить його надзвичайно цікавим для художників та творчих особистостей. Користувачі можуть експериментувати з різними налаштуваннями, включаючи контраст, яскравість

та текстуру, щоб створювати унікальні та естетично привабливі композиції.

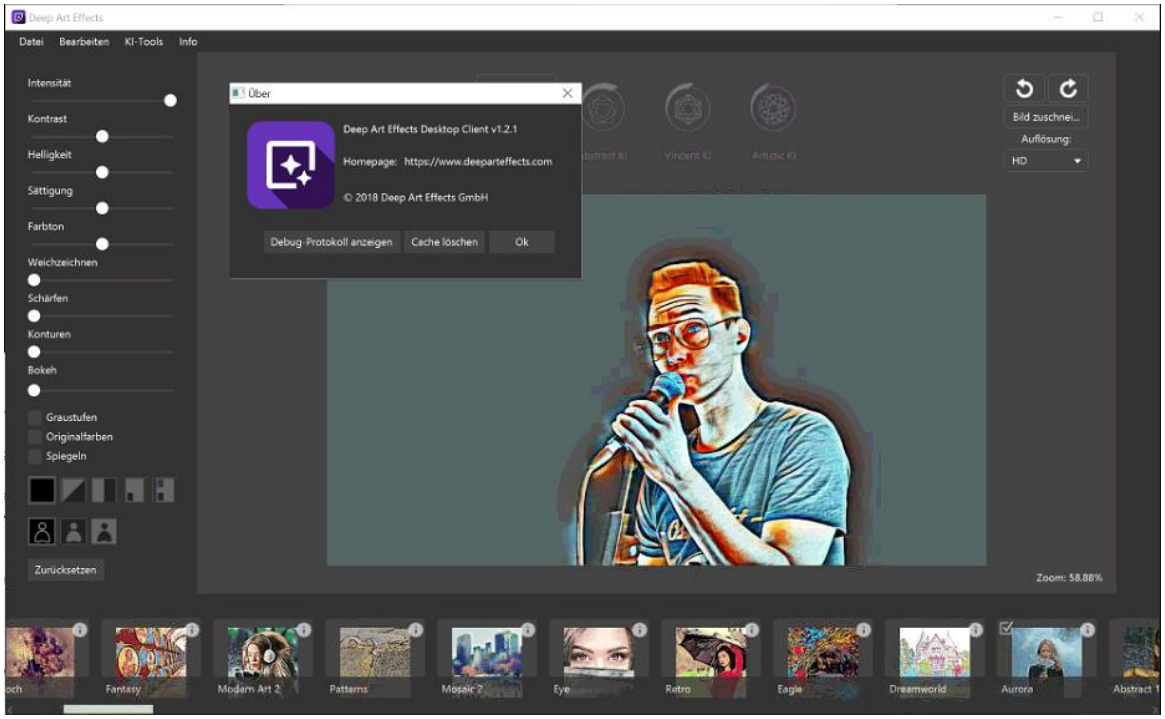

Рисунок 1.5 – Приклад роботи застосунку DeepArt

Розробка програмних засобів реставрації зображень з використанням нейронних мереж є складним та багатоаспектним процесом, що потребує ретельного планування та реалізації. Для успішної розробки таких програмних продуктів необхідно мати досвід роботи з штучним інтелектом, зокрема з нейронними мережами, а також розуміння принципів їх функціонування та навчання. Першим кроком у розробці є створення внутрішньої системи, яка буде здатна навчати нейронну мережу. Наступним кроком необхідно провести тренування мережі на великому обсязі даних. Крім того, програмне забезпечення повинно бути здатне виконувати реставрацію та поліпшення якості вхідного зображення за допомогою навченої нейронної мережі.

Для виконання описаних кроків необхідно виконати розробку ефективних алгоритмів обробки зображень, які використовують результати роботи нейронної мережі для відновлення деталей, зменшення шуму та покращення загальної якості.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Deep-Image	Topaz Gigapixel AI	Let's Enhance	AI Image Enlarge	Deep Art	Власний модуль
Можливість збільшення роздільної здатності	+	+	+	+	-	+
Можливість додаткової обробки зображень	-	-	+	+	+	+
Швидкість обробки	+	+	-	-	-	+
Можливість реставрації зображень	+	+	-	-	-	+
Зручність інтерфейсу	+	-	+	+	+	+
Загальна оцінка	80%	60%	60%	60%	40%	100%

Відповідно до таблиці порівняльних характеристик розробка власних програмних засобів для реставрації зображень з використанням нейронних мереж є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити розширені можливості для поліпшення якості вхідних зображень з використанням нейронних мереж.

Отже, розробка програмних засобів реставрації зображень з використанням нейронних мереж відкриває широкі можливості для покращення роздільної здатності зображень, видалення артефактів з зображення та має додаткові можливості роботи із зображенням.

Проаналізувавши переваги та недоліки існуючих програмних засобів для реставрації зображень з використанням нейронних мереж, було визначено наступні завдання, які необхідно виконати для успішної розробки програмного забезпечення:

- розробити модуль навчання нейронної мережі, який буде спроможний аналізувати вхідні данні та навчати нейронну мережу виявляти дефекти на зображеннях;

- розробити алгоритми реставрації, який забезпечить високу точність та швидкість відновлення роздільної якості вхідного зображення та дозволить прибирати артефакти із вхідного зображення;
- розробити алгоритм та програмний модуль для використання нейронної мережі, який буде спроможний виявляти пошкодження, реставрувати та покращувати роздільну здатність зображення;
- розробити додатковий модуль управлінням параметрами відреставрованого зображення;
- розробка модуля графічного інтерфейсу, який буде зручним у використанні для користувачів програмного забезпечення, забезпечуючи зручність і швидкість взаємодії з розробленим функціоналом;
- реалізувати програмний продукт з використанням засобів нейронних мереж;
- провести тестування програмного забезпечення згідно поставлених задач.

1.5 Висновки

У першому розділі було розглянуто стан предметної галузі реставрації зображень з використанням нейронних мереж. Були розглянуті програмні засоби такі як Deep-Image, Topaz Gigapixel AI, Let’s Enhance. AI Image Enlarge, DeepArt. Кожен з цих засобів має свої переваги і недоліки, що були проаналізовані. На основі даного аналізу було сформовано завдання для подальшої розробки власного програмного забезпечення для реставрації зображень. Ці завдання включають розробку алгоритмів реставрації зображень з використанням нейронної мережі, тренування та використанням нейронної мережі для поліпшення роздільної якості зображення.

Було проведено аналіз використання нейронних мереж для задач попередньої обробки зображення, що показав процеси, які використовує нейронна мережа.

2 РОЗРОБКА МЕТОДІВ ДЛЯ РЕСТАВРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

2.1 Розробка методу реставрації кольору зображення

Відновлення кольору на зображеннях належить до категорії обернених задач, що вимагає застосування різних підходів, одним з яких є використання підходу регуляризації. Даний процес передбачає додавання члена регуляризації до функції асоційованих витрат (дефектів) разом зі звичайним членом найменшої квадратичної похибки. Параметр регуляризації контролює відносний внесок цих двох змінних.

Загалом модель розпізнавання дефектів зображення, придатна для більшості практичних цілей, формується як лінійний процес з адитивним шумом, для якого матрична форма має такий вигляд:

$$g = H\,f + n \tag{2.1}$$

, де g і f – вироджений і вихідний вектори відповідно, H – матриця виродження, а n – шум. Регуляризовані методи відновлення зображень мають на меті мінімізувати обмежену міру найменшої квадратичної похибки:

$$E = \frac{1}{2}||y - Hx||^2 + \frac{1}{2}\lambda||Dx||^2 \tag{2.2}$$

, де вектор y представляє розмиті пікселі зображення, вектор x – оцінку відновленого пікселя, λ – параметр регуляризації, а D – матрицю регуляризації, яка, як правило, є оператором високих частот.

Ключем до вирішення цієї проблеми є мінімізація міри асоційованих витрат, яка представлена $E_{i,j}$, визначеною для кожного пікселя (i, j) на зображенні M X N і задається формулою:

$$E_{i,j} = \frac{1}{2}(y_{i,j} - h * x)^2 + \frac{1}{2}\lambda_{i,j}(d * x)^2 \quad (2.3)$$

, де $h * x$ позначає згортку між фільтром розмиття h з центром в точці (i, j) і відновленим зображенням x, $d * x$ - згортка між фільтром високих частот d з центром у точці (i, j) і відновленим зображенням x, змінна $\lambda_{i,j}$ представляє локальні параметри регуляризації.

У загальному випадку великі значення λ потрібні для гладких областей, а малі λ - для країв. Ітеративні підходи до оновлення зазвичай використовуються для мінімізації функції асоційованих витрат за допомогою підходів градієнтної оптимізації:

$$\frac{\partial E_{i,j}}{\partial x_{p,q}} = \big(h * x - y_{i,j}\big)h_{-a.-b} + \lambda(d * x)h_{-a.-b} \quad (2.4)$$

, де $\mathrm{p} = \mathrm{i} + \mathrm{a}, \mathrm{q} = \mathrm{j} + \mathrm{b},\ \mathrm{p} \in \{\mathrm{i} - \frac{\mathrm{W}}{2}, \dots., \mathrm{i} + \frac{\mathrm{W}}{2}\}\ \mathrm{q} \in \{\mathrm{j} - \frac{\mathrm{H}}{2}, \dots., \mathrm{j} + \frac{\mathrm{H}}{2}\}$.

Отже, отримано функцію асоційованих витрат, що дозволяє отримати втрати кольору. Враховуючи, що маска згортки зображення має вигляд W X H, для того щоб мінімізувати отриману функцію асоційованих витрат, необхідно прирівняти вищевказану похідну до нуля для оновленого пікселя, в якому попереднє значення пікселя $x_{p,q}$ замінюється на $x`_{p,q}$. Таким чином, необхідна кількість оновлення кольору для пікселя визначається як:

$$\Delta x_{p,q} = x`_{p,q} - x_{p,q} \quad (2.5)$$

Таким чином, видно, що необхідна кількість оновлень залежить від параметра локальної регуляризації. Щоб мінімізувати функцію асоційованих витрат, нам потрібно прирівняти до нуля похідну для оновленого пікселя $x`_{p,q}$, в якому попереднє значення кольору $x_{p,q}$ замінено. Таким чином, необхідна кількість оновлень для пікселя визначається як

$$(x_{p,q}\ , \Delta x^{d}{}_{p,q}) \tag{2.6}$$

Усі зображення мають канали кольорів RGB(Red-Green-Blue), тому було вирішено розробити формули для знаходження регулятизатора для кожного значення кольору:

$$\Delta x_{(p,q)_r} = x`_{(p,q)_r} - x_{(p,q)_r} \tag{2.7}$$

$$\Delta x_{(p,q)_g} = x`_{(p,q)_g} - x_{(p,q)_g} \tag{2.8}$$

$$\Delta x_{(p,q)_b} = x`_{(p,q)_b} - x_{(p,q)_b} \tag{2.9}$$

Після знаходження параметрів регулятизатора для кожного складового кольору, нейронна мережа застосовує зафарбовування ділянки зображення, використовуючи маску.

2.2 Розробка методу реставрації роздільної якості зображення

В сучасному світі велика увага приділяється вдосконаленню методів відновлення зображень з метою підвищення їхньої якості та роздільної здатності. Одним із ключових аспектів цього завдання є необхідність виявлення та розрізнення розмитих та не розмитих пікселів у зображенні.

На сьогоднішній день найпоширенішими об'єктивними методами оцінки якості зображення є метрики, засновані на різниці між пікселями. Ці методи відрізняються низькою обчислювальною складністю та можуть бути легко інтегровані в інші алгоритми обробки зображень. Вони також не залежать від умов перегляду та окремих спостерігачів.

Запропоновано вимірювати універсальний індекс якості зображення, методом порівняння між еталонним і тестовим зображеннями, шляхом розбиття на три різні порівняння: порівняння яскравості, контрастності та структурне

порівняння. Нехай параметр яскравості буде відображатись як функція $L(x, y)$, де x еталонне зображення, а y тестове зображення. Тоді вимірювання функції порівняння яскравості між зображеннями буде описуватись наступною формулою:

$$\mathrm{L}(x, y) = \frac{2\mu_x\mu_y}{{\mu_x}^2 + {\mu_y}^2} \tag{2.10}$$

, де μ_x та μ_y позначають середні значення яскравості зображень x та y відповідно. Тоді порівняння контрастності $\mathrm{c}(x, y)$ між еталонним та тестовим зображенням буде описуватися наступною формулою:

$$\mathrm{c}(x, y) = \frac{2\eta_x\eta_y}{{\eta_x}^2 + {\eta_y}^2} \tag{2.11}$$

, де η_x та η_y – стандартні відхилення контрастності зображень x та y відповідно. Структурне порівняння еталонного і тестового зображення задається формулою:

$$\mathrm{s}(x, y) = \frac{\eta_{xy}}{\eta_x\eta_y} \tag{2.12}$$

, де η_{xy} – коваріація зображень та x та y.

Наведені формули порівняльних показників універсального індексу якості зображення, розробимо формулу, що на основі цих трьох порівняльних показників буде обраховувати загальний показник якості зображення (UIQI):

$$\mathrm{UIQI}(x, y) = L(x, y) * c(x, y) + s(x, y) \tag{2.13}$$

UIQI (Universal Image Quality Index) є простим показником, який залежить

виключно від показників порівняння першого та другого порядку еталонного та тестового зображень. Однак він є дещо нестабільним, особливо на однорідних ділянках, де член у знаменнику дуже малий. Крім того, ретельні тести показали, що UIQI погано корелює з суб'єктивною оцінкою. Для того, щоб полегшити проблему стабільності та покращити кореляцію між об'єктивними та суб'єктивними показниками, розробимо формулу, на основі даного показника, що буде обраховувати структурну схожість (SSIM) двох зображень:

$$SIMM(x,y) = -\frac{(2\mu_x\mu_y + (K_1L)^2)(2\eta_{xy} + (K_2L)^2)}{(\mu_x{}^2 + \mu_y{}^2 + (K_1L)^2)(\eta_x{}^2 + \eta_y{}^2 + (K_2L)^2)} \quad (2.14)$$

, де L – динамічний діапазон значень пікселів (255 для 8-бітових зображень), а K_1 і K_2 – малі додатні константи.

Знайшовши формулу для обрахування структурної схожості тестового та оригінального зображення, виведемо формулу для порівняння якості та схожості кожного пікселя (i,j). Для цього будемо обраховувати локальний індекс SSIM(i,j), шляхом оцінки середнього значення, стандартного відхилення та коваріації у певному пікселі. Так як загальна якість зображення вимірюється структурним індексом схожості SSIM, для поліпшення точності визначення якості зображення, необхідно обрахувати значення структурної схожості для кожного пікселя, що входять до зображення. Для цього знайдемо формулу усередненого значення структурної схожості пікселів для еталонного та тестового зображення:

$$MSSIM = \frac{1}{M}\sum_i\sum_j SSIM(i,j) \quad (2.15)$$

, де M – загальна кількість пікселів зображення.

Після знаходження формули, що буде порівнювати якість зображення для кожного пікселю та формули загальної якості зображення, необхідною дією

постає програмування нейронної мережі. Для реставрації роздільної якості зображення, нейронна мережа повинна пройти навчання. У якості ваг значень та зміщень буде використано розроблені формули $MSSIM$ та $SIMM$. Нейромережа буде приймати два зображення – еталонне та тестове з погіршеною якістю. Наступним етапом буде обрахування ваг значень, а саме усереднене значення структурної схожості пікселів двох зображень та автоматично буде обраховувати зміщення. Після чого буде обраховуватись структурна схожість зображення. Для досягнення максимальної ефективності в реставрації роздільної якості зображення, вирішено задати мінімальну похибку у $SIMM$ – 0.001, тобто структурна схожість зображень повинна відрізнятися не більше ніж 0.001.

2.3 Особливості навчання нейронної мережі

Зображення для експериментів були взяті з бази даних зображень USC-SIPI, а також зображення з галереї MIDAS та JPEG. Ці зображення були обрані, оскільки вони містять готові набори тестових зображень і мають поєднання областей текстури та країв. Зображення з обраних тестових наборів були піддані трьом типам погіршення:

– розмиття: на кожне зображення з тестових наборів було додано точку розмиття, що в подальшому дозволить навчити нейронну мережу ідентифікувати та реставрувати розмиті пікселі на зображенні;

– мультиплікативний шум: додавання на кожну копію зображення з тестових наборів ефекту зернистості та пошкоджень, що в подальшому дозволить навчити нейромережу ідентифікувати та реставрувати пошкоджені ділянки зображення;

– колір: позбавлення кольору кожної копії зображення з тестових наборів, шляхом застосування фільтрів «Відтінки сірого», що в подальшому дозволить навчити нейромережу виявляти та відновлювати колір чорно-білих зображень.

Загалом, у всіх тестових було більше 5000 різних зображень. Було обрано

такі тестові набори зображень, які містили найбільшу кількість різних об’єктів, що забезпечить найефективніше тренування нейромережі. Для навчання нейронної мережі використовувались зображення що пройшли один з етапів погіршення та оригінальне зображення.

Нейронна мережа зі зворотним поширенням зображується як така, що має три або максимум чотири шари. Вхідний шар подається на вхід після нормалізації значень пікселів. Кількість вузлів у вхідному шарі дорівнює кількості пікселів на зображенні. Якщо зображення, що розглядається, представлено у вигляді M x N пікселів, ми маємо M x N вхідних вузлів.

Зображення, що надходять до нейронної мережі під час її навчання, мають розміри 128 x 128 та 256 x 256 пікселів. Якщо розглядати другий випадок, то кількість вхідних вузлів становить 65536 для зображень, що позбавлені кольору і 65536 x 3 для інших зображень, що пройшли перші 2 етапи погіршення.

Оскільки кожне значення пікселя має бути регуляризоване з використанням підходу регуляризації, описаного вище, кількість вихідних вузлів у вихідному шарі дорівнює кількості вхідних вузлів. Рішення про кількість вузлів у прихованому шарі залежить від різних факторів, таких як

1. Кількість вхідних та вихідних вузлів.
2. Кількість навчальних прикладів.
3. Кількість шуму у мішенях.
4. Регуляризація.

У більшості випадків не існує стандартного способу визначити оптимальну кількість вузлів у прихованому шарі без навчання мережі та оцінки помилки узагальнення. Рішення про кількість прихованих вузлів має бути збалансованим. Якщо використовується занадто мало прихованих вузлів, помилка навчання буде високою, як і помилка узагальнення через недостатню відповідність даних. З іншого боку, якщо прихованих вузлів більше, навіть якщо досягнута помилка навчання є меншою, існує ймовірність високої помилки узагальнення через надмірне припасування та високу дисперсію.

Враховуючи вищесказане, під час навчання нейромережі кількість прихованих вузлів залишалася в межах від 10 до 100. Це також пов'язано з тим, що кількість прихованих вузлів безпосередньо впливає на час обчислень, необхідний для навчання мережі. Також кількість прихованих шарів обмежена максимум двома. Це пов'язано з тим, що узагальнення мережі зменшується, а запам'ятовування збільшується, що не представляє інтересу.

Після створення нейронної мережі зі зворотним поширенням та ініціалізації ваг і зсувів мережі, її можна навчати. Під час експериментів використовувався пакетний режим або режим навчання по епохах (рис. 2.1).

Рисунок 2.1 – Процес навчання нейромережі

Набір даних зображень, що використовувався для навчання, було розділено на три підмножини. Це було зроблено випадковим чином таким чином, що 60% набору даних було віднесено до навчальної множини, 20% - до перевірочної множини і 20% - до тестової множини. Навчальна множина використовується для обчислення градієнта та оновлення ваг і зміщень мережі. Похибка, отримана в результаті використання даних валідаційної множини, постійно контролюється в процесі навчання. Якщо помилка на валідаційному наборі зростає, це означає, що мережа надмірно підлаштовується під дані. У

запропонованому дизайні, якщо помилка валідації зростає протягом п'яти ітерацій, навчання зупиняється, а ваги та зсуви на ітерації, на якій помилка валідації почала зростати, встановлюються заново.

2.4 Висновки

У другому розділі було проведено розробку методів реставрації кольору зображення, методу реставрації роздільної здатності зображення та описано метод навчання нейронної мережі. Усі методи використовують засоби використання нейронних мереж.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕСТАВРАЦІЇ ЗОБРАЖЕНЬ

3.1 Розробка алгоритмів програмного забезпечення

Розробка програмних засобів для реставрації зображень з використанням нейронних мереж передбачає кілька етапів. По-перше, потрібно створити модуль, який буде відповідати за зручну та інтуїтивно зрозумілу взаємодію із системою. Цей модуль буде відповідати за взаємодію користувача із програмним продуктом та навігацією із функціоналом застосунку. Далі буде розроблений модуль для імпорту та додаткової обробки зображень, який дозволить змінювати основні параметри зображення, такі як яскравість, виокремлення тіней, контраст та інші.

Наступним кроком буде розроблено модуль тренування нейромережі, використовуючи фреймворк PyTorch та мову програмування Python модуль навчання нейронної мережі. Для цього буде використовуватись набір даних із пошкоджений і відреставрованих зображень. Також будуть застосовані додаткові бібліотеки: Open Neural Network Exchange (ONNX) – це відкритий формат для обміну нейронними мережами між різними фреймворками та платформами; ONNX runtime – це виконавча система для запуску моделей, представлених у форматі ONNX, на різних пристроях та платформах.

Після навчання нейромережі буде розроблений модуль для автоматичного визначення пошкоджених ділянок та артефактів зображення. Цей модуль використовуватиме особливості навченої нейронної мережі, процеси поліпшення роздільної якості вхідного зображення, видалення артефактів та ефекту шуму із зображення, для реставрації зображення.

Наступним буде модуль для порівняння результатів реставрації. Цей модуль дозволить користувачам переглядати різницю між відреставрованим зображенням та вхідним зображенням. Для візуалізації порівняння зображень буде використано слайдер.

Розробка цих програмних модулів потребує глибокого розуміння роботи із нейронними мережами, графічного програмування та програмування на Python та JavaFX. Для реалізації цих модулів будуть використані різноманітні інструменти та бібліотеки програмування, зокрема IntelliJ IDEA, Open Neural Network Exchange та ONNX runtime.

Для кращого розуміння роботи модулів програмних засобів реставрації зображень з використанням нейронних мереж було вирішено розробити модель роботи системи на прикладі UML діаграми діяльності (рис. 3.1).

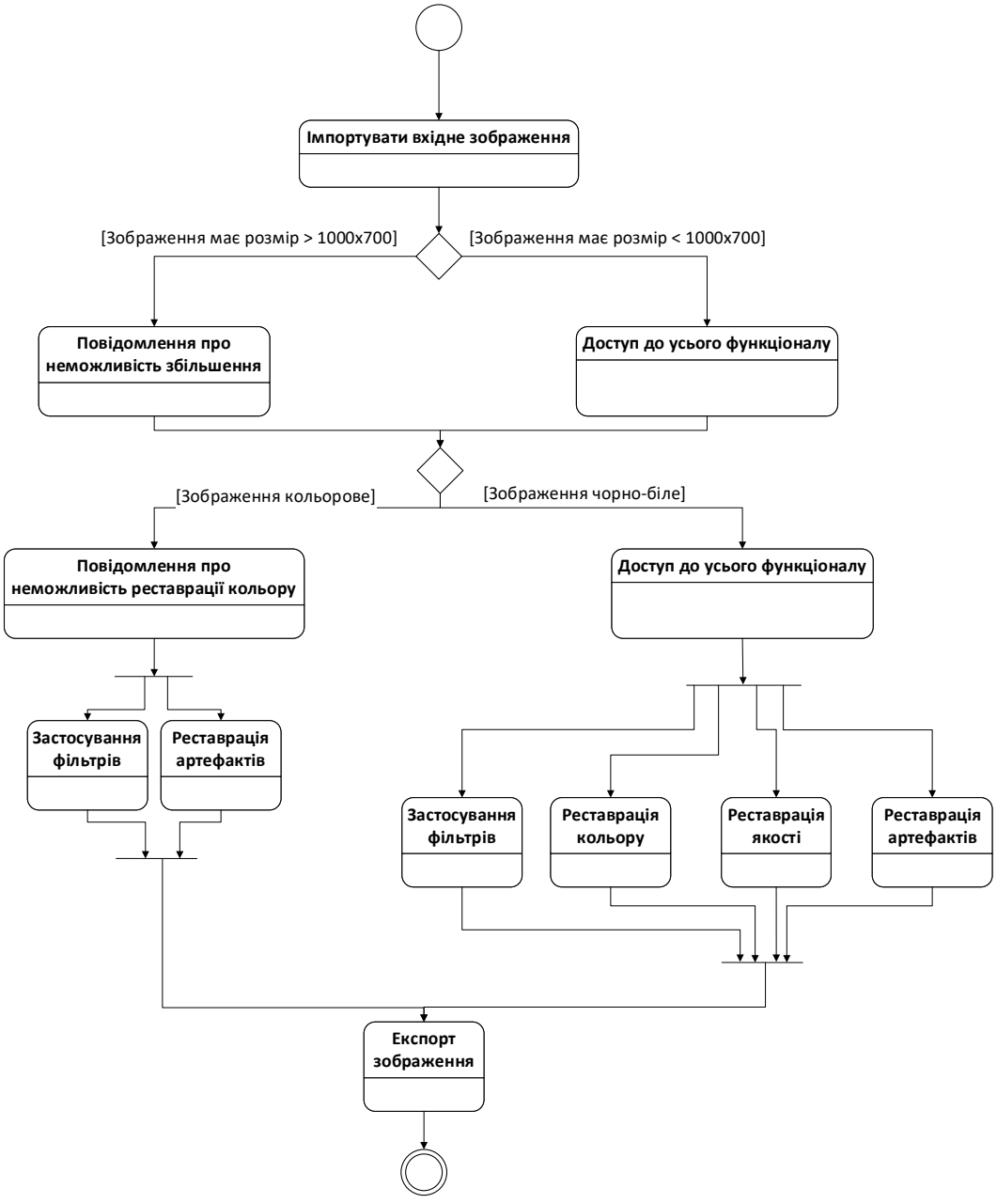

Рисунок 3.1 – Діаграма діяльності програмних засобів

Згідно вказаної діаграми користувач для початку взаємодії із програмним застосунком повинен імпортувати вхідне зображення будь-якого формату. Наступним кроком буде перевірка імпортованого зображення на перевищення допустимого розміру для покращення якості зображення шляхом збільшення його розмірів.

Якщо розмір зображення перевищує 1000x700 пікселів, то користувачу виведеться помилка про неможливість використання функції реставрації якості зображення, але при цьому буде доступний весь інший функціонал. При умові що імпортоване зображення не перевищує вище вказаний розмір, користувачу буде доступний для використання весь розроблений функціонал. Наступним кроком виконується перевірка вхідного зображення на наявність кольорової палітри, а саме чи є імпортоване зображення «чорно-білим». При умові, що зображення є «чорно-білим» користувачу буде доступний увесь розроблений функціонал, інакше програмний застосунок виведе повідомлення про неможливість використання функції реставрації кольору зображення. У разі аналізу зображення та виявлені «чорно-білої» палітри зображення та розміру, що не перевищує 1000x700 пікселів, користувач зможе виконувати реставрацію вхідного зображення, а саме використовувати функції реставрації кольору, якості та артефактів (пошкоджень), а також йому будуть доступні додаткові функції, такі як застосування фільтрів до зображення. У іншому випадку, а саме при перевищені розміру зображення 1000x700 пікселів та кольоровій палітрі зображення, користувачу буде доступний функціонал реставрації артефактів зображення та застосування фільтрів. Варто зазначити, що програмний застосунок дозволяє двом станам користувача виконати експорт обробленого зображення.

Для розробки програмного застосунку, який реалізує функціонал реставрації вхідних зображень з використанням нейронних мереж, потрібно спочатку розробити загальний алгоритм, згідно з яким працюватиме програмний застосунок (рис. 3.2). Згідно з цим алгоритмом, першим кроком є завантаження вітального вікна програмного застосунку, де користувач бачить назву застосунку

та короткий опис його призначення.

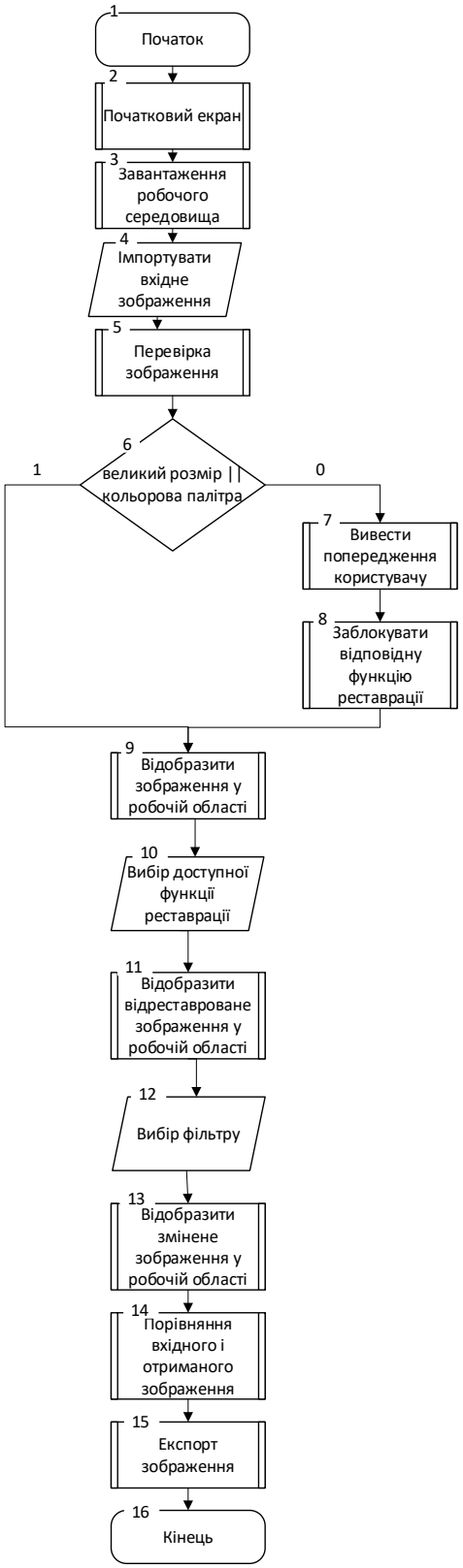

Рисунок 3.2 – Алгоритм роботи програмного застосунку

Після цього відбувається завантаження робочого середовища програми,

яке надає доступ до функціоналу програмного застосунку. Далі, користувачу потрібно імпортувати вхідне зображення, яке потребує реставрації.

Обравши зображення, програмний застосунок запускає модуль перевірки імпортованого зображення на наявність кольорів та перевищення допустимого розміру для якісної реставрації. Якщо в модулі виявляються проблеми, програмний застосунок виводить користувачу попередження про це та блокує відповідні функції реставрації, щоб уникнути відмови роботи програми. При відсутності проблем зображення, програмне забезпечення відображає вхідне зображення у робочому середовищі. Після цього програмний застосунок очікує вибору доступної функції реставрації зображення користувачем. Обравши функцію, застосунок автоматично виконує реставрацію та відображає відреставроване зображення. Далі, користувач може обрати фільтр для застосування до відреставрованого зображення або пропустити цей крок. У разі вибору фільтру, він автоматично застосовується до зображення. Після цього користувач може скористатися модулем порівняння вхідного і відреставрованого зображення за допомогою слайдера. На завершення, користувач має можливість експортувати отримане зображення вказавши необхідну директорію.

Для кращого розуміння процесу реставрації вхідних зображень нейронною мережею, було розроблено додатковий алгоритм (див. рис. 3.3), який демонструє процес навчання нейронної мережі для реставрації зображень. В загальному визначенні під навчанням розуміється ітеративний процес представлення нейронній мережі графічного образу із набору тренувальних даних, де отримана класифікація порівнюється з бажаним виходом. Згідно з даною алгоритмічною схемою, спочатку необхідно обрати нейронну мережу. У розробленому додатку були обрані такі нейронні мережі, як GPEN-BFN-512, ParseNet, realesrnet та RetinaFace. Усі ці мережі підтримують технології TensorFlow та OpenCV для роботи із зображеннями. Далі важливим етапом є знаходження тестового набору даних, що включає в себе перелік пар з більш ніж 1000 зображень. Цей набір даних складається з пар пошкоджених/низької якості/чорно-білих зображень та

відповідних звичайних зображень. Ці пари зображень становлять важливу основу для навчання нейронних мереж, дозволяючи їм вивчати зв'язок між вхідними і вихідними даними та навчитися ефективно відновлювати пошкоджені або низькоякісні зображення. Такий підхід дозволяє покращити якість та ефективність роботи нейронної мережі під час реставрації зображень в реальному часі.

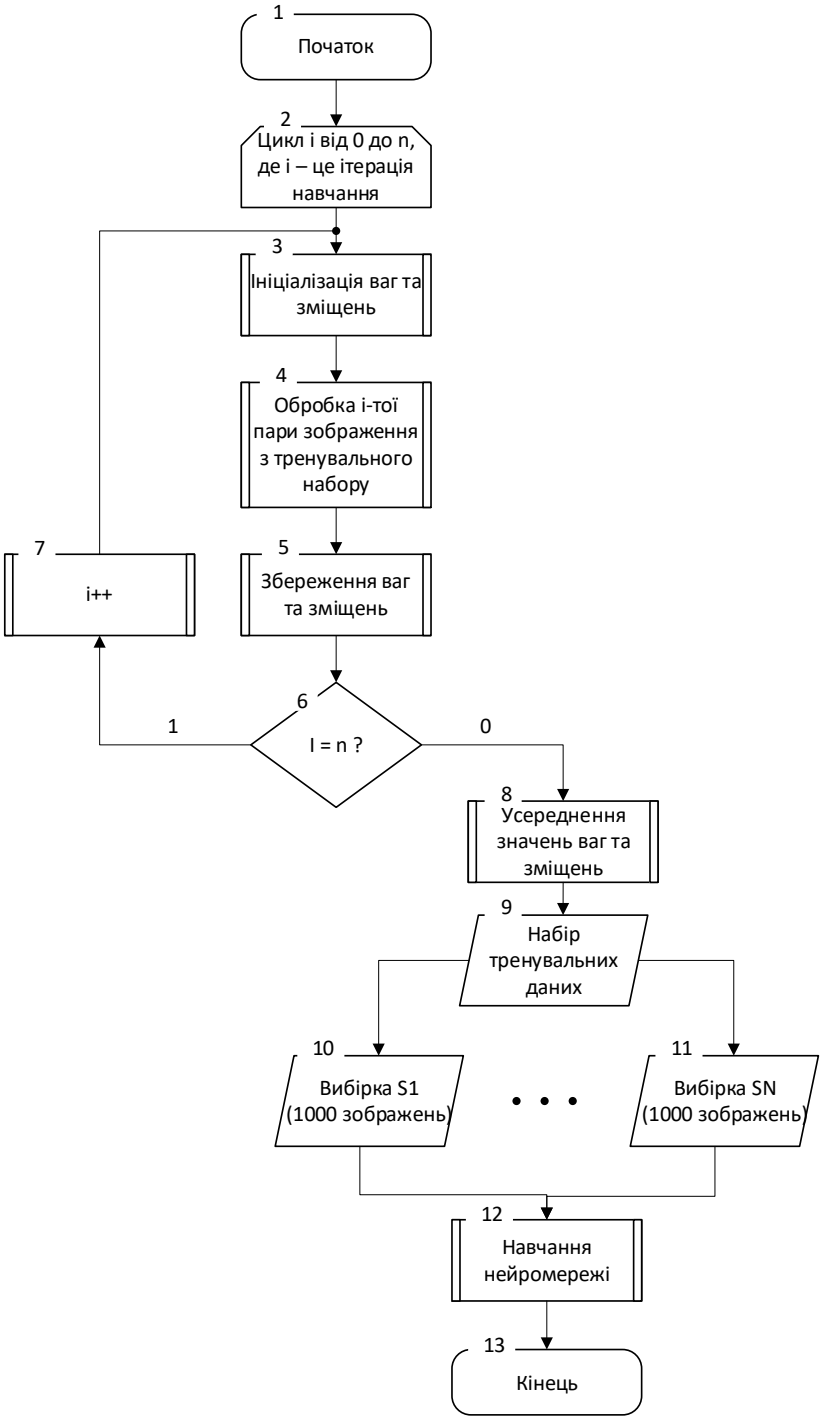

Рисунок 3.3 – Алгоритм навчання нейромережі

Під час навчання нейронних мереж використовується ітеративний підхід для визначення початкових значень ваг та зміщень, що є ключовим етапом у процесі навчання. Ці значення ініціалізуються випадковими під час кількох ітерацій, щоб забезпечити різноманітність та випадковість. Після кожної ініціалізації нейронна мережа обробляє певну кількість зображень, зазвичай 1000, для початку процесу навчання.

На наступному етапі, після обробки кожної пари зображень з тренувального набору, розпочинається процес визначення ваг та зміщень для реставрації пошкодженого зображення на основі отриманих результатів. Ці значення зберігаються для подальшого використання у процесі навчання. Після збереження цих параметрів, процес починається знову з ініціалізації новими випадковими значеннями. Це важливо для того, щоб нові значення ваг та зміщень відхилялися на значну величину від попередніх, що допомагає уникнути відмов в локальних мінімумах та забезпечує більш ефективний процес навчання. Після завершення кожного циклу ітерацій, проводиться усереднення ваг та зміщень шляхом знаходження середнього арифметичного значення для кожного параметра. Це сприяє зменшенню варіативності та забезпечує стабільність навчального процесу, допомагаючи моделі краще усвідомити загальні закономірності в даних і підвищуючи її здатність до узагальнення на нові, раніше не бачені дані.

Подальший процес навчання проводиться за невеликими групами зображень з тренувального набору, які зазвичай містять по 1000 зображень у кожній групі. Кожна група складається з зображень різних класів, що сприяє підвищенню стійкості мережі до помилок та забезпечує адаптабельність до реальних даних. Цей підхід дозволяє мережі краще розуміти різноманіття вхідних даних та здійснювати більш точні прогнози. Після обробки кожної вибірки зображень нейронною мережею, значення ваг та зміщень усереднюються між усіма групами. Це сприяє збалансованому навчанню та покращує загальну ефективність мережі.

Усереднення ваг та зміщень допомагає уникнути перенавчання та робить

модель більш стійкою до різноманітності даних, що може зустрічатися у реальних умовах. Такий підхід до навчання сприяє здатності мережі адаптуватися до нових ситуацій та забезпечує більш точні та надійні результати на практиці.

Отже, було проведено розробку та опис модулів, які необхідні для розробки програмного забезпечення. Також було проведено розробку діаграми станів застосунку, що показує можливі стани програмного забезпечення. Було проведено розробку загального алгоритму роботи програмного застосунку, а також, розробка алгоритму навчання нейронної мережі.

3.2 Розробка модуля навчання нейронної мережі

При розробці модуля для навчання нейронної мережі виявляти та виконувати реставрацію пошкоджень на зображеннях важливо враховувати ряд технічних аспектів. Варто зауважити, що для тренування було обрано чотири набори даних: DIV2K dataset, що містить в собі 800 зображень з роздільною здатністю до 2K для задач реставрації зображень; набір даних Flickr2K, що містить 2650 пар зображень, які складаються із зображень високої та низької роздільної здатності; набір даних RealSR, що містить 500 пар зображень, які складаються з чорно-білих зображень та кольорових зображень; набір даних OutdoorSceneTraining, що містить в собі зображення текстур води, неба, дерев, гір, трави.

Під час формування навчальних пар зображень високої та низької роздільної здатності для навчання та зображень з наявністю артефактів та без них, а також зображень без кольору та кольорових зображень, була використана випадкова вибірка частин зображень розміром 128 на 128 пікселів. Для зображень низької роздільної здатності розміром 32 на 32 пікселів застосовувалася бікубічна знижувальна дискретизація. Для розширення кількості навчальних пар використовувалися аугментації, такі як повороти і горизонтальні відображення.

Для початку тренування необхідно було розробити попередній алгоритм обробки пар зображень. Для цього було використано бібліотеку комп'ютерного зору OpenCV та бібліотеку, що реалізує навчання нейронних мереж Kernel.

На рисунку 3.4 зображено блок-схему функції попередньої обробки пар зображень з наборів даних, що буде автоматично визначати вхідне зображення, що потребує реставрації та очікуване вихідне зображення.

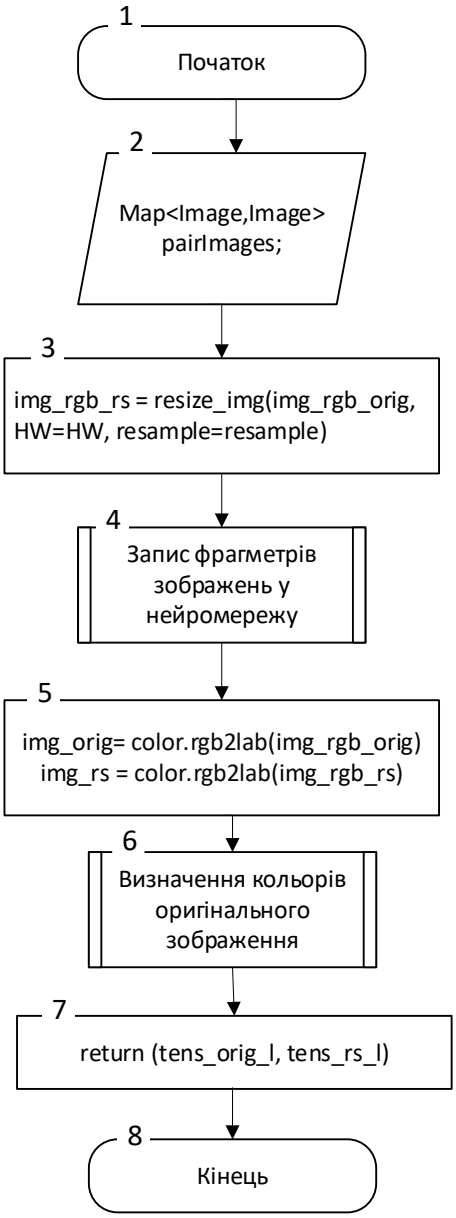

Рисунок 3.4 – Функція попередньої обробки пар зображень

Після визначення вхідного зображення і очікуваного зображення після реставрації, необхідно розробити функцію, що буде обробляти та аналізувати

різницю ваг між зображенями. Для цього необхідно використати бібліотеку PyTorch, що дозволяє взаємодіяти з розробленими нейронними мережами та програмувати нейронні мережі. Реалізація даної функції наведена на рисунку 3.5.

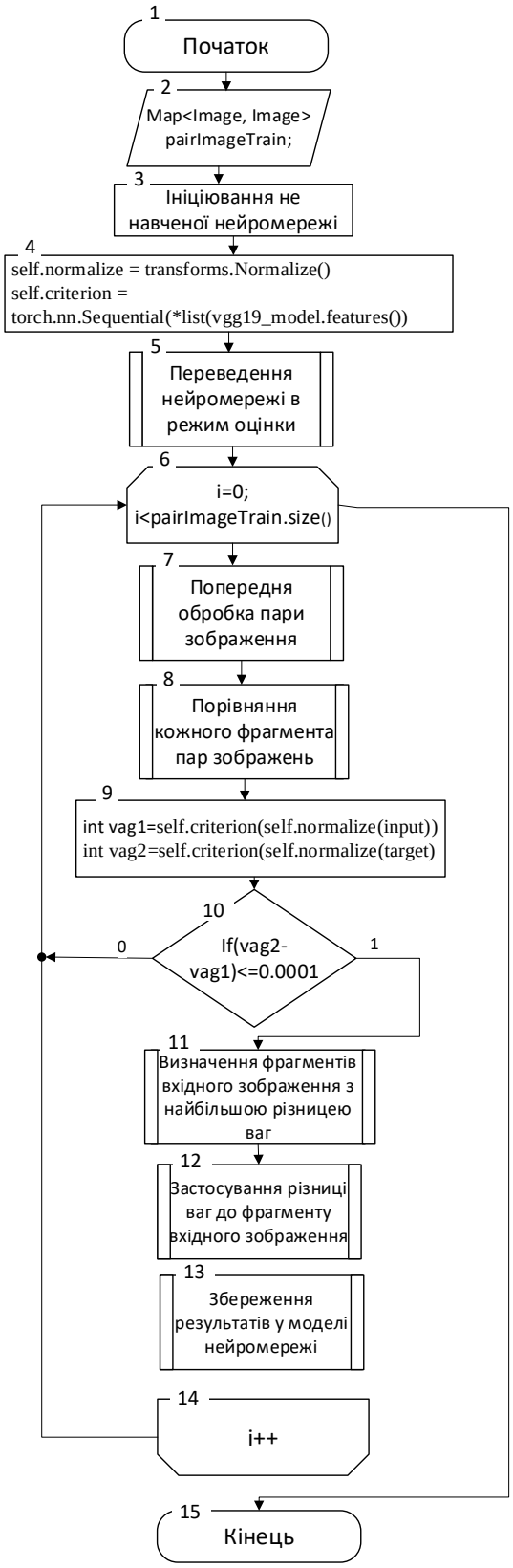

Рисунок 3.5 – Функція визначення різниці ваг

Основною задачею функції є визначення різниці ваг між обраними фрагментами кожної пари зображень. Наступною задачею даної функції є програмування нейронної мережі, а саме навчання мережі визначати пошкоджені фрагменти зображень та застосування визначених ваг. Також варто зазначити, що дана функція використовує бібліотеку TensorFlow для визначення ділянок зображень, що потребують реставрації.

Наступним кроком є усереднення отриманих ваг та зміщень з кожної пари зображень конкретного набору даних. Для цього усі визначені ваги та зміщення, що були визначенні при порівнянні фрагментів зображень з кожної пари, були записані у відповідний блок нейронної мережі. Після чого відбувається ідентифікація кожної ваги та зміщення та обраховується усереднене значення показників, що дозволяє покращити та збільшити точність навчання нейронної мережі. На рисунку 3.6 відображено клас та функції що реалізують описаний алгоритм.

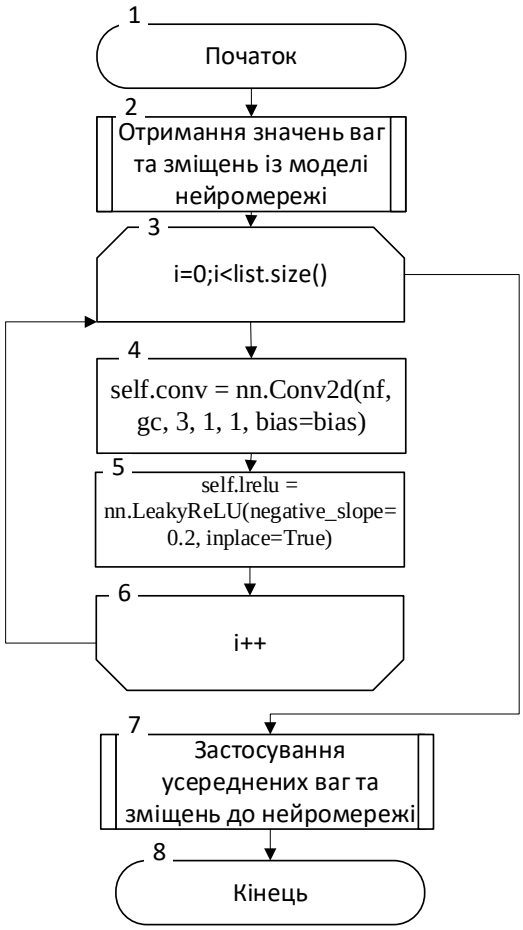

Рисунок 3.6 – Функція усереднення показників зміщення

Наступним етапом є створення класу нейромережі, що буде використовуватись при реставрації та задання усереднених показників зміщення та ваг. Також для збільшення точності та покращення результату реставрації проводиться додаткове навчання нейромережі на відповідному наборі даних з усередненими значеннями. На рисунку 3.7 відображено процес нейромережі, що реалізує додаткове навчання.

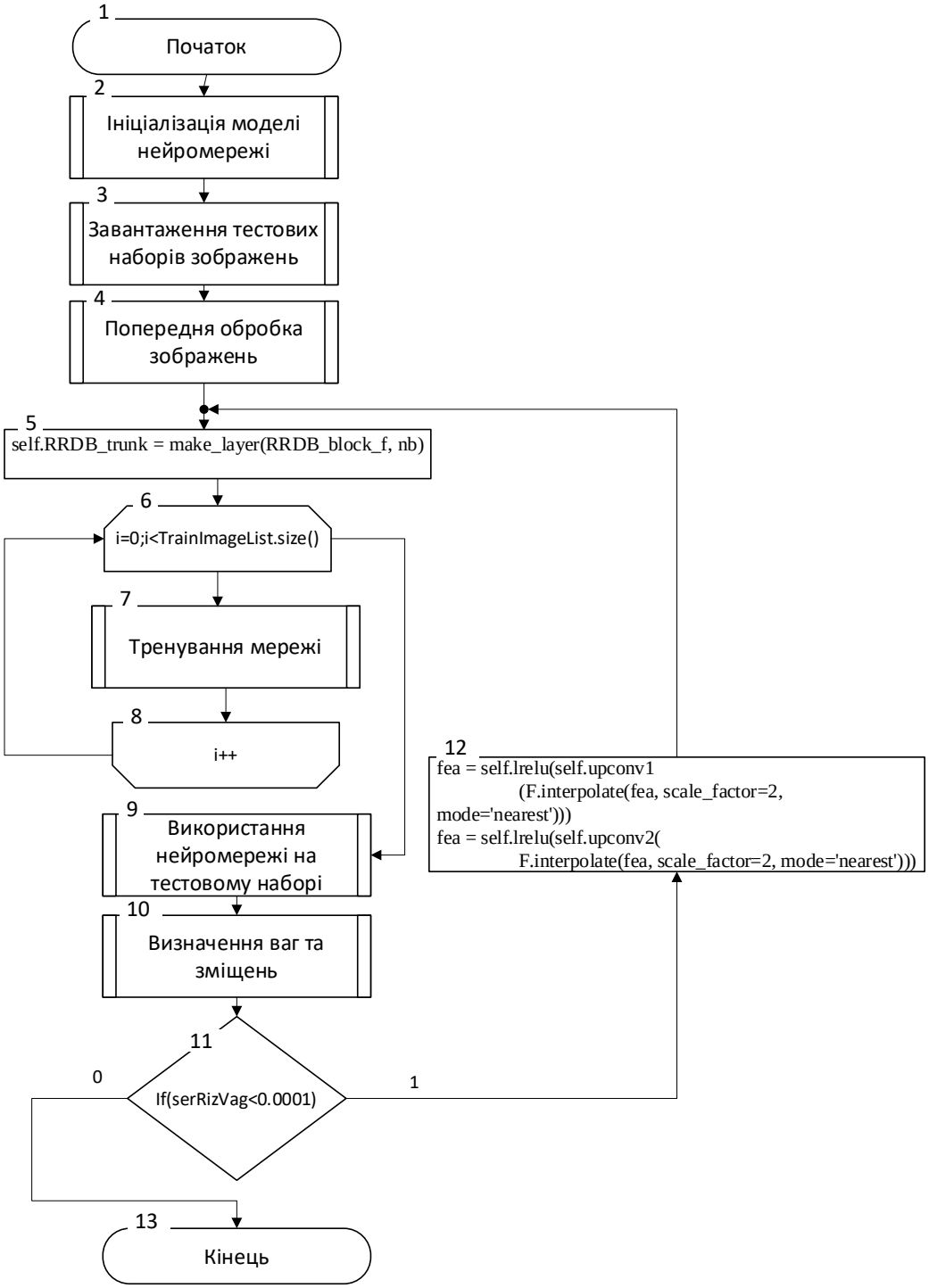

Рисунок 3.7 – Процес навчання нейромережі

3.3 Розробка модуля використання нейронних мереж для реставрації кольору зображення

Модуль реставрації кольору для чорно-білих зображень є важливою частиною процесу відновлення зображень. Однією з ключових задач модуля є точне визначення основних об’єктів, які присутні на вхідному зображенні. Для досягнення цієї мети було розроблено клас SIGGRAPHGenerator (рис. 3.8).

Рисунок 3.8 – Клас для реставрації кольору фрагменту зображення

Цей клас призначений для приймання основного кольору певного фрагменту зображення як параметра. Його завдання полягає у використанні цього кольору для відновлення кольорів інших областей чорно-білого зображення. Цей процес забезпечує збереження правильного кольору об’єктів та їхній відтвір у відновленому зображенні з високою точністю.

Далі виконується аналізування даного фрагменту та за допомогою навченої нейронної мережі та бібліотек OpenCV та Kernel виконується ідентифікація ділянки зображення. Далі використовуються отримані усереднені значення ваг та зміщень, що дозволяють визначити об’єкт на фрагменті та відобразити кольоровий варіант цього фрагменту.

Додатково до розробки класу для застосування кольорової маски до фрагментів зображення, в процесі роботи було створено окремий клас, спрямований на визначення основного кольору кожного фрагменту фотографії. Цей клас (рис. 3.9) відіграє ключову роль у визначенні та наданні відповідного кольору для кожного об'єкту або фрагменту зображення.

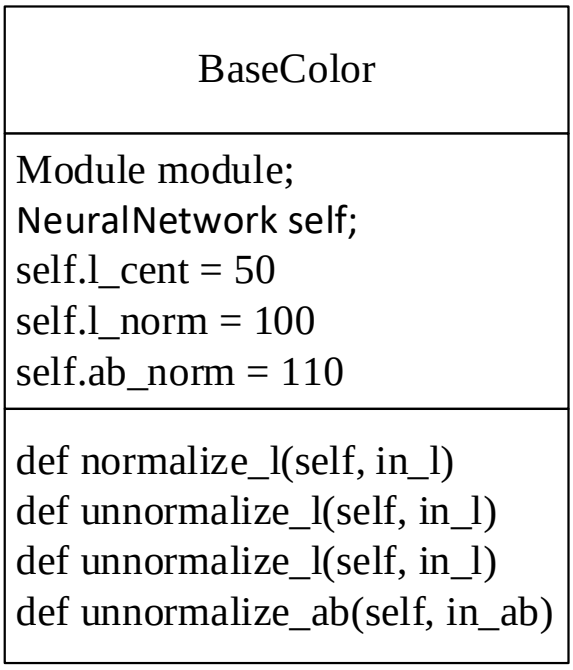

Рисунок 3.9 – Клас ідентифікації кольору

На відміну від класу, що відповідає за застосування кольорової маски, клас

BaseColor спрямований на використання нейронної мережі для аналізу та обробки кожного вхідного фрагменту зображення з метою ідентифікації об'єктів на ньому. Після успішної ідентифікації клас надає необхідну інформацію про основний колір, який потрібно застосувати до відповідного фрагменту. Такий підхід дозволяє точно визначити кольори та застосувати їх до кожного елементу фотографії, забезпечуючи високу якість та точність обробки зображення.

Після розробки нейронної мережі для обробки зображень, наступним кроком у процесі є налагодження алгоритмів застосування кольорів до фрагментів зображення та використання масок. Цей процес відбувається через розробку спеціального класу, який відповідає за застосування кольорів до зображення. Реалізація даного класу наведена на рисунку 3.10.

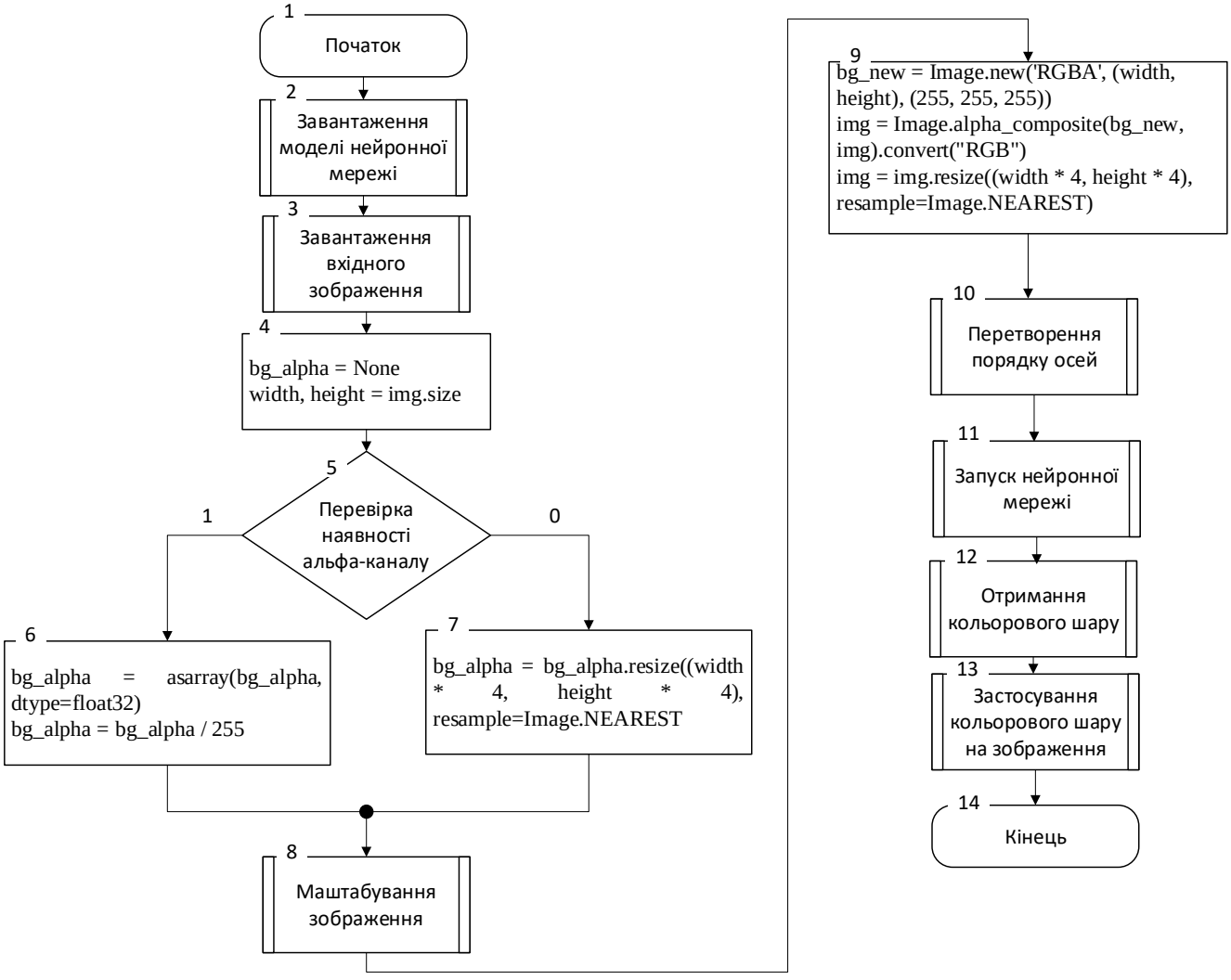

Рисунок 3.10 – Клас застосування кольору до зображення

Даний клас містить метод inference, який є ключовим для виконання цих завдань. Метод inference приймає вхідне чорно-біле зображення та кольорову маску, застосовує маску до зображення, перетворює зображення у формат, необхідний для подальшої обробки нейронною мережею, виконує обробку зображення з використанням нейронної мережі та повертає оброблене зображення з накладеною маскою.

3.4 Висновки

У третьому розділі було проведено розробку алгоритмів програмного забезпечення. Також було описано та обгрунтовано модулі, які необхідні для розробки програмного забезпечення. Також було проведено розробку діаграми станів застосунку, що показує можливі стани програмного забезпечення.

У розділі також було описано розробку основних програмних модулів для навчання нейронної мережі та використання нейронних мереж для реставрації кольору зображення.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування програмного забезпечення

Процес тестування програмного забезпечення (ПЗ) полягає в перевірці системи на наявність помилок, прогалин або невідповідності вимогам порівняно з фактичною вимогою. Це важлива складова кожного проекту, яка здійснюється на всіх етапах розробки системи. Використання ефективної методології тестування є ключовим для стабільності продукту і економії часу та коштів. Тестування поділяється на два основних типи: статичне та динамічне.

Статичне тестування, переважно виконуване на початкових етапах розробки, спрямоване на перевірку відповідності вимог ПЗ або документації, не звертаючи увагу на його код.

Динамічне тестування, з іншого боку, ставить за мету переконатися, що ПЗ функціонує згідно з вимогами та очікуваннями користувачів. Цей тип тестування включає різні методи, такі як модульні, тестування "білої скриньки" і "чорної скриньки". Модульні тести проводяться над окремими частинами програми для перевірки їхньої правильної роботи.

Тестування "білої скриньки" перевіряє внутрішні структури та роботу програми, враховуючи код, що використовується для розробки тестових випадків. Тестування "чорної скриньки" оцінює функціональність програми, не звертаючи увагу на її внутрішню структуру. Також використовуються регресійне тестування, яке перевіряє роботу програми після внесення змін, і димове тестування, яке включає швидкі сценарії для перевірки основної функціональності продукту перед його релізом. Обрано методику "чорної скриньки", оскільки вона дозволяє виявити помилки в логіці функціонування програми та неполадки у роботі її основних алгоритмів.

Перед початком процесу тестування необхідно спочатку встановити тестову версію програми на комп'ютер з операційною системою Windows, яка буде використовуватись під час виконання тестів. Також перед стартом тестування важливо забезпечити належну документацію, включаючи

специфікації вимог та план тестування.

Під час виконання першого етапу тестування розробленого програмного продукту проводиться перевірка правильності виконання його функцій. Один із таких тестів включає імпортування вхідного зображення (рис. 4.1). Результатом цього тестування є імпортоване вхідне зображення, яке пройшло обробку та аналіз і було відображене у робочому середовищі програмного застосунку.

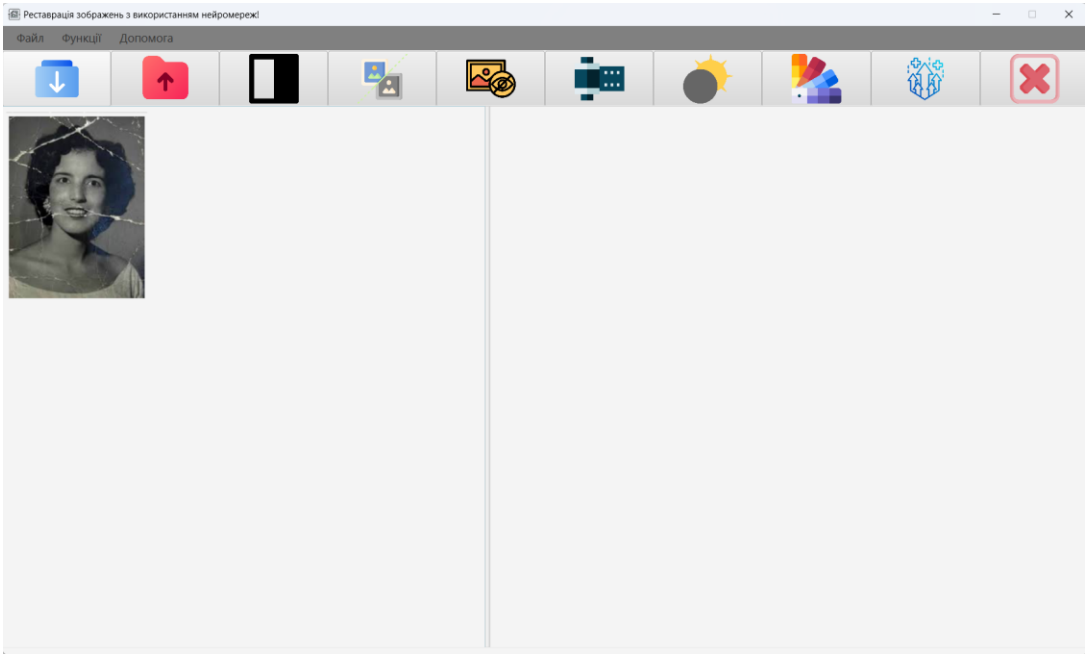

Рисунок 4.1 – Тестування функції імпорту зображення

Після успішного імпортування зображення користувачеві пропонується скористатися функцією реставрації зображення, щоб покращити його якість та відновити деталі. Реставрація зображення включає в себе ряд процесів, зокрема відновлення кольору, поліпшення якості та видалення артефактів, які можуть бути присутні на вихідних зображеннях.

Наприклад, функція відновлення кольору може бути застосована до чорно-білого зображення з метою надання йому кольорових аспектів та природного вигляду (рис. 4.2).

Результатом цього процесу реставрації є відновлене кольорове зображення, яке пройшло через обробку за допомогою нейронної мережі.

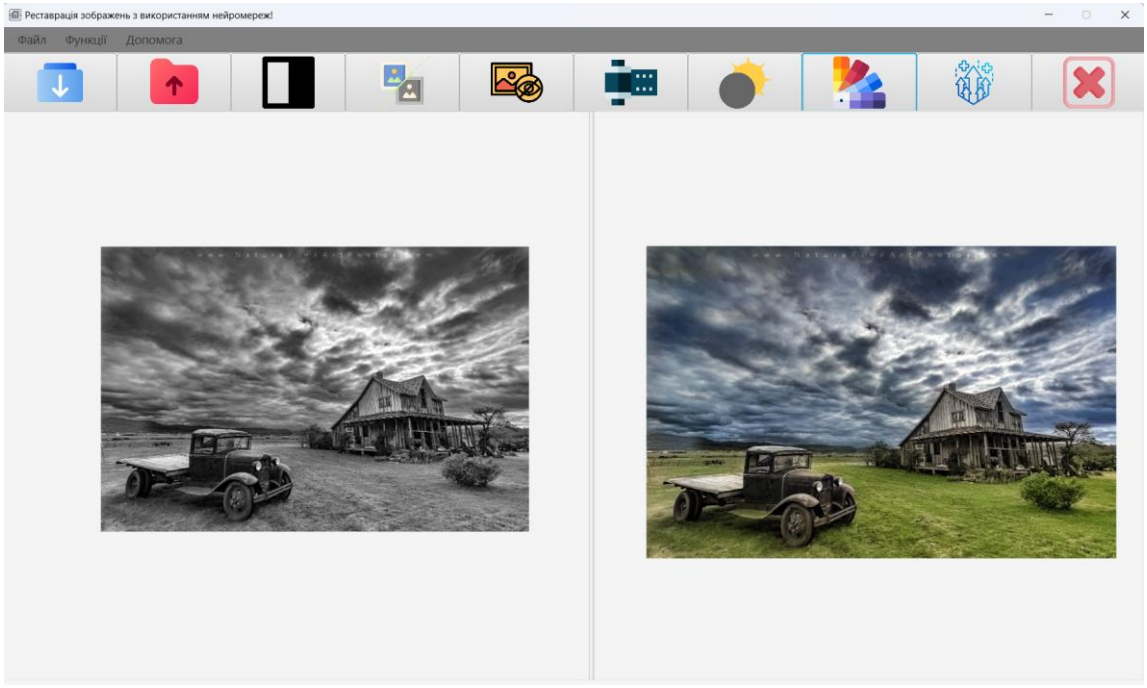

Рисунок 4.2 – Тестування функції реставрації кольору

Наступним етапом функціонального тестування є перевірка ефективності функції знищення артефактів з метою покращення якості зображення. Це означає, що програма має здатність виявляти та коригувати пошкодження на зображенні, такі як розмиття чи спотворення, для покращення його роздільної здатності та загальної якості. Результати цього тестування представлені на рисунку 4.3.

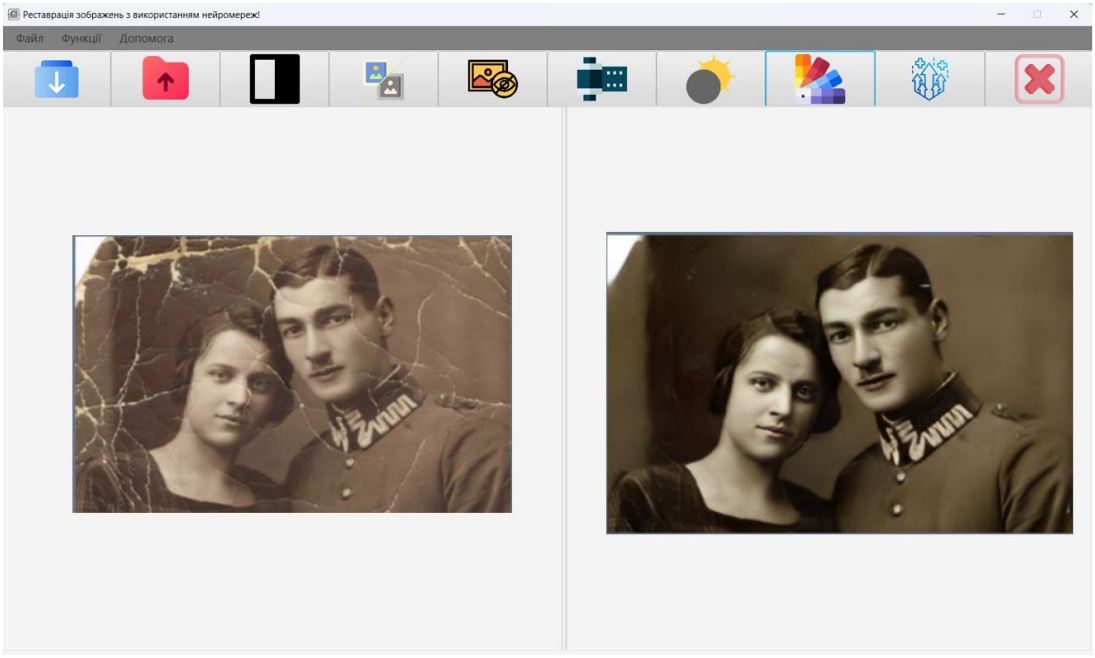

Рисунок 4.3 – Тестування функцій реставрації

Після цього проводиться тестування функції збільшення роздільної здатності вхідного зображення. Це може бути особливо корисно для зображень низької роздільної здатності, де програма намагається поліпшити якість та деталізацію зображення. Результати цього тестування показані на рисунку 4.4.

Рисунок 4.4 – Тестування збільшення роздільної здатності зображення

Наступним етапом є перевірка функції застосування фільтрів до зображень. Це може включати в себе застосування різних ефектів, таких як відтінки сірого, сепія, негатив та інші, до вихідного зображення. Результати цього тестування можна побачити на рисунку 4.5.

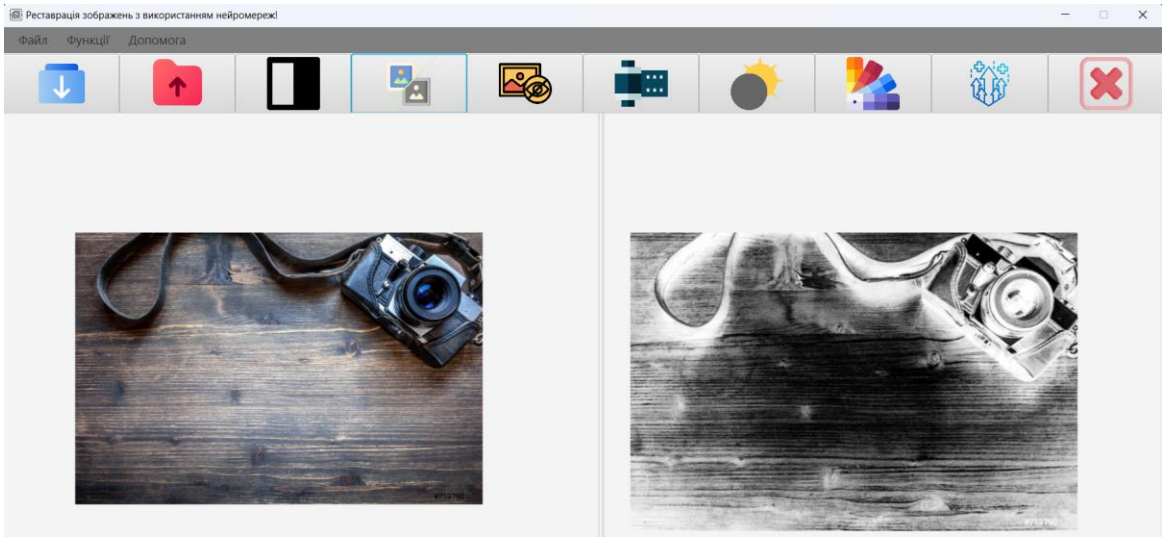

Рисунок 4.5 – Тестування функцій застосування фільтрів

Результати тестування функціонального тестування показали, що програмне забезпечення чітко та правильно виконує усі реалізовані функції, відповідає запрограмованій бізнес-логікі.

Другим етапом тестування є перевірка відповіді програми на введення чи імпортування невірних або ж пошкоджених даних. Для цього у програмне забезпечення було завантажено графічне зображення розміром 2500 на 1200 пікселів. Після чого, було протестовано реакція програми на спробу відновити якість зображення та збільшити його роздільну здатність. Результати тестування (рис. 4.6) показали, що програмне забезпечення успішно оброблює такі випадки та дозволяє уникнути ситуацій відмови ПЗ.

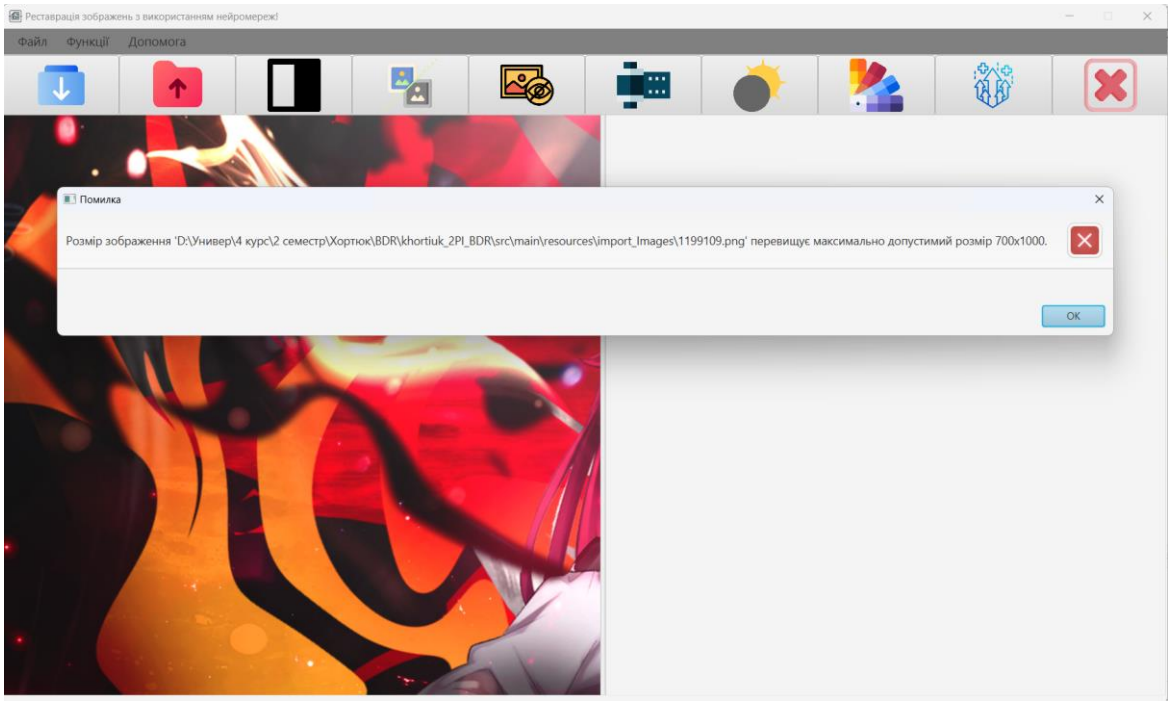

Рисунок 4.6 – Тестування сценарію відмови функції реставрації якості

Наступним кроком тестування є перевірка функції реставрації кольору на кольоровому зображені. Для цього у програмне забезпечення було імпортовано кольорове зображення та натиснута кнопка «Реставрувати колір».

Результати тестування (рис. 4.7) показали, що програмне забезпечення успішно оброблює такі випадки та дозволяє уникнути ситуацій відмови ПЗ.

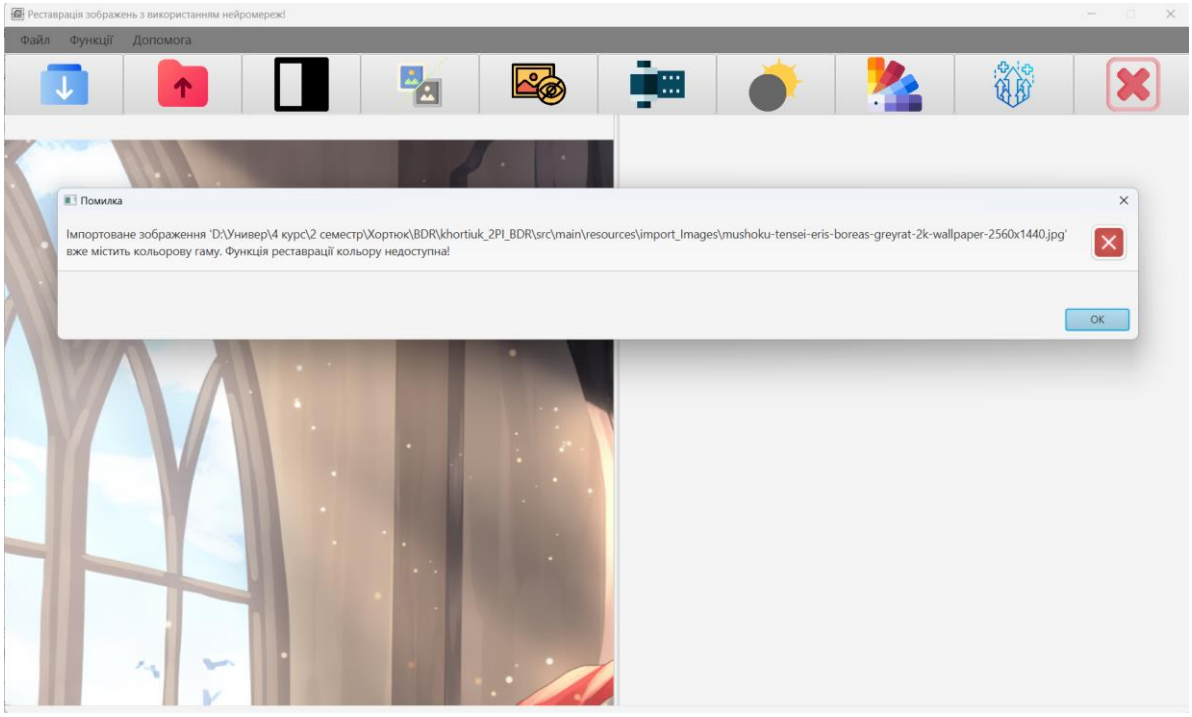

Рисунок 4.7 – Тестування сценарію відмови функції реставрації кольору

Отже, після закінчення тестування можливих випадків відмови програмного забезпечення, було виведено результати тестування, що показали відмовостійкість та надійність використання програмного забезпечення.

Наступним кроком тестування було перевірка швидкодії і продуктивності програмного забезпечення. Проведене тестування показало, що швидкість відклику програми на будь-який запит користувача не перевищує 500 мілісекунд. Також було проведено аналіз його сумісності з різними версіями операційних систем та мінімальною конфігурацією апаратного забезпечення. Зокрема, програмний продукт був перевірений на сумісність з наступними операційними системами: Windows 10 та Windows 8.1. Результати тестування показали успішну роботу програмного забезпечення як на потужних, так і на менш потужних пристроях, що вказує на його високу оптимізацію та ефективність роботи.

4.2 Розробка інструкції користувача

Програмний продукт призначений для реставрації пошкоджених зображень шляхом використання засобів нейронних мереж. Для цього

користувачу необхідно завантажити вхідне зображення у робоче середовище програмного застосунку та обрати одну з можливих дій: реставрація кольору, реставрація якості, реставрація артефактів, застосування фільтрів. Також у програмі передбачено експортування результатів, а також розширені можливість взаємодії із програмним забезпеченням, а саме: довідка та вікно «Про програму».

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту [23]. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Мінімальна конфігурація:

Процесор	Intel Core i3
Оперативна пам'ять	8 ГБ RAM
Вільний простір на диску	1,5 ГБ
Відеокарта	NVIDIA GeForce GTX 1060
Операційна система	Windows 8.1

Таблиця 4.2 – Рекомендована конфігурація:

Процесор	Intel Core i7
Оперативна пам'ять	16 ГБ RAM
Вільний простір на диску	1,5 ГБ
Відеокарта	NVIDIA GeForce RTX 2070Ti
Операційна система	Windows 10

Процес інсталяції програмного забезпечення для реставрації зображень передбачає процес завантаження виконавчого файлу та встановлення програмного забезпечення у відповідну директорію на персональному комп’ютері користувача.

До основних функцій, що виконує програмне забезпечення відносяться:

– реставрація кольору: передбачає імпортування чорно-білого зображення та автоматичне відновлення його кольору;

– реставрація артефактів: передбачає автоматичну ідентифікацію пошкоджень на зображенні та їх видалення;

– реставрація якості зображення: передбачає автоматичне визначення пікселів із низькою роздільною здатністю та подальша їх обробка;

– застосування фільтрів: передбачає застосування доступних фільтрів до зображення;

– експорт результатів: передбачає збереження зображення після обробки.

Після інсталяції та запуску програмного забезпечення, користувач переходить на вітальний екран програмного застосунку та очікує на завантаження робочого середовища. Після завантаження робочого середовища, користувач отримує доступ до широкого спектру функціоналу програми. Першим кроком взаємодії з програмними засобами є імпортування вхідного зображення, незалежно від його формату. Після завантаження зображення, модуль аналізу автоматично перевіряє його та визначає доступні функції для реставрації, які відображаються для користувача. Після цього зображення автоматично відображається у робочому середовищі, готове до подальшої обробки.

Далі користувач має можливість взаємодіяти з різними пунктами меню, такими як "Довідка" або "Про програму", або розпочати процес реставрації зображення, обравши потрібну функцію з панелі інструментів. Після обрання функції реставрації, користувач може спостерігати за процесом в реальному часі та переглядати результати на льоту. Після завершення обраної функції реставрації, відреставроване зображення автоматично відображається у правій частині робочого середовища.

Після цього користувач має можливість порівняти вихідне і відреставроване зображення, використовуючи спеціальний слайдер. У разі, якщо відреставрований варіант не відповідає очікуванням користувача, він може використовувати інші доступні функції реставрації або накладати різні фільтри для досягнення бажаного ефекту.

Після завершення процесу реставрації, користувач має можливість експортувати відредаговане зображення до вказаної директорії. Варто відзначити, що всі зображення експортуються у форматі .png, що забезпечує максимальну якість та роздільну здатність зображень після обробки.

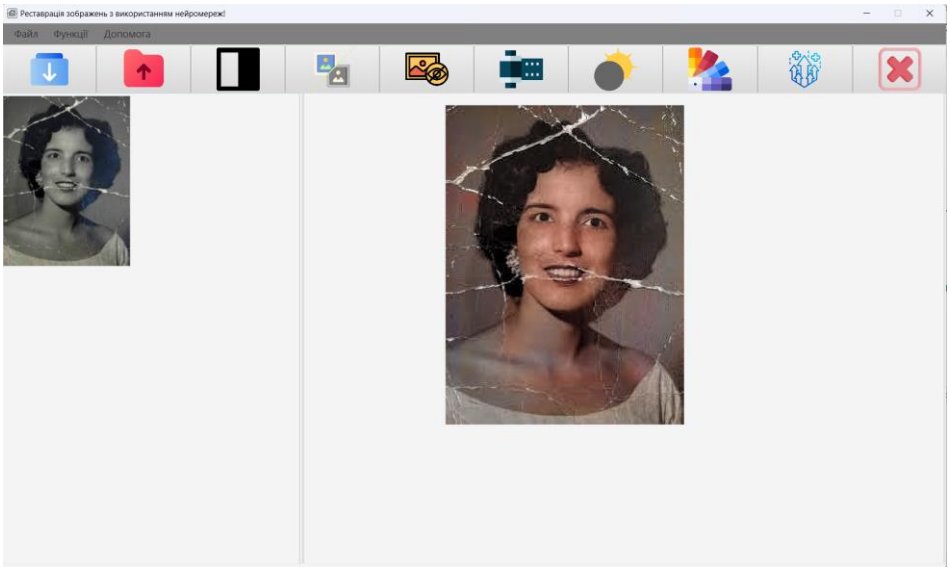

Рисунок 4.8 – Робоче середовище програмного забезпечення

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів програмних засобів реставрації зображень з використанням нейронних мереж. Було проведено функціональне тестування, тестування відмовостійкості програмного забезпечення та тестування продуктивності роботи додатку. Також було розроблено інструкцію користувача по початковій експлуатації програмного продукту.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено методи та програмні модулі для реставрації зображень з використанням нейронних мереж. Проведено аналіз відомих підходів до процесу реставрації зображень з використанням нейронних мереж. Обгрунтовано необхідність розробки методів реставрації зображень з використанням нейронних мереж з метою підвищення роздільної якості пошкоджених зображень з можливістю реставрації за рахунок використанням нейронної мережі, що дозволяє знаходити та аналізувати пошкоджені частини фотографії.

Розроблено методи та програмні модулі реставрації вхідних зображень використовуючи засоби нейронних мереж, що дозволяють виконати попередню обробку та аналіз вхідного зображення та виявити дефекти.

На основі проведених у БДР досліджень розроблено алгоритми та програмні засоби для реставрації зображень. Проведене тестування підтвердило достовірність теоритичних досліджень методів реставрації зображень та засобів їх реалізації.

Розроблений програмний застосунок було написано, використовуючи мови програмування Java і Python. Для розробки використовувалися середовища IntelliJ IDEA 2023. Розроблений програмний продукт відповідає поставленій меті та завданням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Irani, M., Peleg, S. Improving resolution by image registration. // Graphical Models and Image Processing 53 (3), 1991. – 231–239 pp.

2. Isaac J. S., Kulkarni R. Super resolution techniques for medical image processing. // International Conference on Technologies for Sustainable Development, ICTSD, 2015. – 1–6 pp.

3. Huang Y., Shao L., Frangi A. F. Simultaneous superresolution and cross-modality synthesis of 3d medical images using weakly-supervised joint convolutional sparse coding. // Conference on Computer Vision and Pattern Recognition, CVPR, 2017. – 5787–5796 pp.

4. Rasti P., Uiboupin T., Escalera S., Anbarjafari G. Convolutional neural network super resolution for face recognition in surveillance monitoring. // Articulated Motion and Deformable Objects, AMDO, Lecture Notes in Computer Science, vol 9756. Springer, 2016.

5. Dai D., Wang Y., Chen Y., Van Gool L. Is image superresoltion helpful for other vision tasks? // Winter Conference on Applications of Computer Vision, WACV, IEEE, 2016. – 1–9 pp.

6. Хортюк Д.С., Романюк О.Н., Роль нейронних мереж у високороздільній реставрації зображень // Матеріали XXXII Міжнародної науково-практичної конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров'я (MicroCAD-2024)», 22-25 травня 2024 року, Харків, - ХПІ.

7. Згорткові нейронні мережі (частина 1) [Електронний ресурс] – Режим доступу до ресурсу: https://itmaster.biz.ua/programming/vision/cnns1.html (дата звернення: 21.03.2024)

8. Gan.ai: Personalized videos at scale [Електронний ресурс] – Режим доступу до ресурсу: https://gan.ai/ (дата звернення: 21.03.2024)

9. Автоенкодери в Keras [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/(дата звернення: 21.03.2024)

10. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/(дата звернення: 21.03.2024)

11. JavaFx [Електронний ресурс] – Режим доступу до ресурсу: https://openjfx.io/ (дата звернення: 21.03.2024)

12. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/ (дата звернення: 21.03.2024)

13. Scene Builder [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html (дата звернення: 21.03.2024)

14. PyTorch [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/ (дата звернення: 21.03.2024)

15. TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/learn? (дата звернення: 21.03.2024)

16. Keras [Електронний ресурс] – Режим доступу до ресурсу: https://keras.io/ (дата звернення: 21.03.2024)

17. OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://opencv.org/ (дата звернення: 21.03.2024)

18. DeepImage [Електронний ресурс] – Режим доступу до ресурсу: https://deep-image.ai/app/tools/auto-enhance/ (дата звернення: 21.03.2024)

19. Topaz Gigapixel AI [Електронний ресурс] – Режим доступу до ресурсу: https://www.topazlabs.com/gigapixel (дата звернення: 21.03.2024)

20. Let's Enhance [Електронний ресурс] – Режим доступу до ресурсу: https://letsenhance.io/ (дата звернення: 21.03.2024)

21. AI Image Enlarge [Електронний ресурс] – Режим доступу до ресурсу: https://imglarger.com/Enhancer (дата звернення: 21.03.2024)

22. DeepArt [Електронний ресурс] – Режим доступу до ресурсу: https://www.deeparteffects.com/ (дата звернення: 21.03.2024)

23. ЩО ТАКЕ ТЕСТУВАННЯ ПЗ, ЙОГО ЕТАПИ, ВИДИ, ІНСТРУМЕНТИ [Електронний ресурс] – Режим доступу до ресурсу:

https://university.sigma.software/what-is-software-testing/ (дата звернення: 21.03.2024)

ДОДАТКИ

Додаток А – Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., професор О. Н. Романюк
«12» березня 2024 р.
Виконала:
_____ студентка гр. 2ПІ-20б Д. С. Хортюк
«12» березня 2024 р.

Вінниця – 2024

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж».

Галузь застосування – архівна реставрація зображень.

2. Підстава для розробки

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки

Метою дослідження є підвищення роздільної якості пошкоджених зображень з можливістю реставрації за рахунок використанням нейронної мережі, що дозволяє знаходити та аналізувати пошкоджені частини фотографії.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Irani, M., Peleg, S. Improving resolution by image registration. // Graphical Models and Image Processing 53 (3), 1991. – 231–239 pp.

2. Хортюк Д.С., Романюк О.Н., Роль нейронних мереж у високороздільній реставрації зображень // Матеріали XXXII Міжнародної науково-практичної конференції «Інформаційні технології: наука, техніка, технологія, освіта, здоров’я (MicroCAD-2024)», 22-25 травня 2024 року, Харків, - ХПІ.

5. Технічні вимоги

Вихідні дані до роботи: методи реставрації зображень – глибокі нейронні мережі з використанням згорткових та рекурентних шарів; моделі – архітектури ResNet, U-Net; графічний режим – TrueColor; нові методи – автоматичне

виявлення та виправлення артефактів, відновлення кольору та роздільної якості зображень; максимальна якість реставрації зображення; вихідні дані – результати реставрації зображень у вигляді відновлених зображень з високою якістю та точністю.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ з\п	Назва етапів бакалаврської дипломна роботи	Строк виконання етапів роботи
1	Аналіз підходів до процесу реставрації зображень з використанням нейронних мереж	14.03.24 –22.03.24
2	Розробка методів для реставрації зображень з використанням нейронних мереж	22.03.24 – 19.04.24
3	Розробка алгоритмів роботи програми	19.04.24 –10.05.24
4	Розробка програмного продукту	10.05.24 –24.05.24
5	Тестування роботи програми	24.05.24 – 27.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 03.06.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.Н.

Оригінальність	77,2%
Схожість	22,8%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Хортюк Д. С.

Керівник роботи Романюк О.Н.

Додаток В – Лістинг програмного забезпечення

```
package com.example.khortiuk_2pi_bdr;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class Black {
    public static void setWhiteAndBlack(String filePath) throws IOException {
        BufferedImage img = null;
        File inputFile = new File(filePath);
        File outputFile = null;

        try {
            img = ImageIO.read(inputFile);
        } catch (IOException e) {
            System.out.println(e);
        }

        int width = img.getWidth();
        int height = img.getHeight();

        BufferedImage bwImage = new BufferedImage(width, height,
BufferedImage.TYPE_BYTE_GRAY);

        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                int rgb = img.getRGB(x, y);

                int r = (rgb >> 16) & 0xFF;
                int g = (rgb >> 8) & 0xFF;
                int b = rgb & 0xFF;

                int gray = (int) (0.299 * r + 0.587 * g + 0.114 * b);

                bwImage.setRGB(x, y, (gray << 16) | (gray << 8) | gray);
            }
        }
        inputFile.delete();
        try {
            outputFile = new File(filePath);
            ImageIO.write(bwImage, "jpg", outputFile);
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
package com.example.khortiuk_2pi_bdr;

import javafx.fxml.FXML;
```

```
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.util.Objects;

public class GreetingWindowController {
    @FXML
    Button button;
    @FXML
    public void openMainEnvironment() {
        try {
            Stage start = (Stage) button.getScene().getWindow(); start.close();

            Parent root = FXMLLoader.load(Objects.requireNonNull(getClass().getResource("hello-
view.fxml")));
            Scene scene = new Scene(root);

            Stage stage = new Stage();
            stage.setScene(scene);
            stage.setTitle("Реставрація зображень з використанням нейромереж!");

            stage.setFullScreen(false);
            stage.getIcons().add(new Image("D:\\Универ\\4 курс\\2
семестр\\Хортюк\\BDR\\khortiuk_2PI_BDR\\src\\main\\resources\\com\\example\\khortiuk_2pi_b
dr\\icons\\icon.png"));
            stage.setResizable(false);
            stage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
package com.example.khortiuk_2pi_bdr;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class Histequal {

    public static void setGray(String path){
        try {
            // Load the input image
            BufferedImage inputImage = ImageIO.read(new File(path));

            // Perform histogram equalization
            BufferedImage outputImage = equalizeHistogram(inputImage);
```

```
            File d=new File(path);
            d.delete();

            // Save the output image
            File output = new File(path);
            ImageIO.write(outputImage, "jpg", output);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static BufferedImage equalizeHistogram(BufferedImage inputImage) {
        int width = inputImage.getWidth();
        int height = inputImage.getHeight();

        // Create a new BufferedImage for the output image
        BufferedImage outputImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        // Calculate the histogram of the input image
        int[] histogram = new int[256];
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int rgb = inputImage.getRGB(x, y);
                int gray = (int) (0.299 * ((rgb >> 16) & 0xFF) + 0.587 * ((rgb >> 8) & 0xFF) + 0.114 *
(rgb & 0xFF));
                histogram[gray]++;
            }
        }

        // Calculate the cumulative distribution function (CDF) of the histogram
        int[] cdf = new int[256];
        cdf[0] = histogram[0];
        for (int i = 1; i < 256; i++) {
            cdf[i] = cdf[i - 1] + histogram[i];
        }

        // Normalize the CDF to the range [0, 255]
        int cdfMin = cdf[0];
        for (int i = 0; i < 256; i++) {
            cdf[i] = (int) (255.0 * (cdf[i] - cdfMin) / (width * height - cdfMin));
        }

        // Apply histogram equalization to each pixel
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int rgb = inputImage.getRGB(x, y);
                int gray = (int) (0.299 * ((rgb >> 16) & 0xFF) + 0.587 * ((rgb >> 8) & 0xFF) + 0.114 *
(rgb & 0xFF));
                int equalizedGray = cdf[gray];
```

```
                int equalizedRGB = (equalizedGray << 16) | (equalizedGray << 8) | equalizedGray;
                outputImage.setRGB(x, y, equalizedRGB);
            }
        }

        return outputImage;
    }
}
package com.example.khortiuk_2pi_bdr;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.stage.DirectoryChooser;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.Objects;

public class MainActivityController {
    @FXML
    private ImageView image_before, image_after;
    @FXML
    SplitPane splitPane;
    private String nameFile;
    private Path path_to_original_image, path_to_save_directory= Path.of("D:\\Универ\\4 курс\\2
семестр\\Хортюк\\BDR\\khortiuk_2PI_BDR\\src\\main\\resources\\import_Images");
    @FXML
    Button white,gray,sepia,negative,shadow,retouch,quality,importBut,exportBut,exit;
    @FXML
    private ScrollPane scrollPane_before, scrollPane_after;
    @FXML
    public void initialize(){
        setButtonStyle(white, "black-and-white.png", "Застосувати чорно-білий фільтр до
зображення");
        setButtonStyle(gray, "greyscale.png", "Застосувати фільтр Відтінки сірого");
        setButtonStyle(sepia, "image_974876.png", "Застосувати фільтр Сепія");
        setButtonStyle(negative, "photo-film.png", "Застосувати фільтр Негатив");
        setButtonStyle(exit, "rejected.png", "Завершити роботу з програмою!");
        setButtonStyle(shadow, "eclipse_10497841.png", "Застосувати фільтр Поліпшені тіні!");
        setButtonStyle(retouch, "color-palette_3214398.png", "Відновити колір фото!");
        setButtonStyle(quality, "buff.png", "Покращити якість фото!");
```

```
        setButtonStyle(importBut, "download.png", "Імпортувати нове фото!");
        setButtonStyle(exportBut, "folder.png", "Експортувати фото!");
        image_before.setOnMouseEntered(event -> setMouseScaleHandler(image_before));
        image_after.setOnMouseEntered(event -> setMouseScaleHandler(image_after));
    }
    private void setMouseScaleHandler(ImageView imageView) {
        imageView.setOnScroll(event -> {
            if (event.isControlDown()) {
                double delta = event.getDeltaY() * 0.001;
                imageView.setScaleX(imageView.getScaleX() + delta);
                imageView.setScaleY(imageView.getScaleY() + delta);
                event.consume();
            }
        });
    }
    private void setButtonStyle(Button button, String iconName, String tooltipText) {
        ImageView imageView = new ImageView(new
Image(getClass().getResourceAsStream("/com/example/khortiuk_2pi_bdr/icons/" + iconName)));
        imageView.setFitWidth(64);
        imageView.setFitHeight(64);
        button.setGraphic(imageView);
        button.setTooltip(new Tooltip(tooltipText));
        button.getStyleClass().add("button-with-icon");
    }
    @FXML
    private void exit(){
        System.exit(0);
    }

    @FXML
    private void import_image(){
        deleteFilesInDirectory(String.valueOf(path_to_save_directory));
        deleteFilesInDirectory("C:\\Users\\Dell\\PycharmProjects\\ESRGAN\\LR");
        File newFile=new File("C:\\Users\\Dell\\PycharmProjects\\ESRGAN\\results\\result.png");
        newFile.delete();
        FileChooser fileChooser = new FileChooser();

        FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("Image Files",
"*.png", "*.jpg", "*.jpeg", "*.gif");
        fileChooser.getExtensionFilters().add(extFilter);

        File file = fileChooser.showOpenDialog(null);

        if (file != null) {
            try {
                nameFile=file.getName();
                System.out.println(nameFile);
                copyFile(file.getAbsolutePath(), String.valueOf(path_to_save_directory));
                path_to_original_image= Path.of(path_to_save_directory + "\\" + nameFile);

                Image image = new Image(String.valueOf(path_to_original_image));
```

```
                image_before.setImage(image);
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }

    }

    @FXML
    private void setBlackAndWhite() throws IOException {
            Black.setWhiteAndBlack(String.valueOf(path_to_original_image));
            image_after.setImage(new Image(String.valueOf(path_to_original_image)));
    }

    @FXML
    private void setGray() throws IOException {
        Histequal.setGray(String.valueOf(path_to_original_image));
        image_after.setImage(new Image(String.valueOf(path_to_original_image)));
    }
    @FXML
    private void setShadow() throws IOException {
        Mean.setShadow(String.valueOf(path_to_original_image));
        image_after.setImage(new Image(String.valueOf(path_to_original_image)));
    }
    @FXML
    private void setSepia() throws IOException {
        LogTrans.setSepia(String.valueOf(path_to_original_image));
        image_after.setImage(new Image(String.valueOf(path_to_original_image)));
    }
    @FXML
    private void setNegative() throws IOException {
        Neg.setNegative(String.valueOf(path_to_original_image));
        image_after.setImage(new Image(String.valueOf(path_to_original_image)));
    }
    @FXML
    private void export_image() {
        // Путь к директории с исходными изображениями
        String sourceDirectoryPath = "D:\\Универ\\4 курс\\2
семестр\\Хортюк\\BDR\\khortiuk_2PI_BDR\\src\\main\\resources\\import_Images";

        // Папка, в которую будут экспортированы изображения (пользовательский выбор)
        DirectoryChooser directoryChooser = new DirectoryChooser();
        directoryChooser.setTitle("Оберіть папку для експорту");
        File selectedDirectory = directoryChooser.showDialog(null);

        if (selectedDirectory != null) {
            try {
                // Создание пути для выбранной пользователем папки
                Path exportDirectory = Paths.get(selectedDirectory.getPath());
```

```
            // Перебор всех файлов в директории с исходными изображениями
            Files.walk(Paths.get(sourceDirectoryPath))
                    .filter(Files::isRegularFile)
                    .forEach(sourceFilePath -> {
                        try {
                            // Создание пути для экспорта изображения в новую папку
                            Path destinationFilePath =
exportDirectory.resolve(sourceFilePath.getFileName());

                            // Копирование файла из исходной директории в новую
                            Files.copy(sourceFilePath, destinationFilePath);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    });
            // Уведомление пользователя об успешном экспорте
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setTitle("Експорт завершено");
            alert.setHeaderText(null);
            alert.setContentText("Усі зображення було успішно експортовано до обраної
папки.");
            alert.showAndWait();
        } catch (IOException e) {
            e.printStackTrace();
            // Уведомление пользователя об ошибке при экспорте
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Помилка експорту");
            alert.setHeaderText(null);
            alert.setContentText("Виникла помилка під час експорту зображень.");
            alert.showAndWait();
        }
    }
}
@FXML
private void retouch_color(){
    String[] command={"cmd", "/c", "python", "C:\\Users\\Dell\\PycharmProjects\\Reviving-
History-Bringing-Old-Photographs-to-Life\\demo_release.py", "-i",
String.valueOf(path_to_original_image)};
    File workingDirectory = new File("C:\\Users\\Dell\\PycharmProjects\\Reviving-History-
Bringing-Old-Photographs-to-Life");
    try
    {
        Process process = Runtime.getRuntime().exec(command, null, workingDirectory);

        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream(), "CP1251"));

        int exitcode=process.waitFor();
    }
    catch (IOException | InterruptedException e)
    {
        throw new RuntimeException(e);
```

```
        }

        copyFile("C:\\Users\\Dell\\PycharmProjects\\Reviving-History-Bringing-Old-Photographs-to-Life\\image.png", String.valueOf(path_to_save_directory));
        File file=new File(String.valueOf(path_to_original_image));
        file.delete();
        renameFile(path_to_save_directory+"\\image.png", nameFile);

        copyFile("C:\\Users\\Dell\\PycharmProjects\\Reviving-History-Bringing-Old-Photographs-to-Life\\image_eccv16.png", String.valueOf(path_to_save_directory));
        renameFile(path_to_save_directory+"\\image_eccv16.png", "eccv16_"+nameFile);

        image_after.setImage(new Image(String.valueOf(new File(path_to_original_image.toUri()))));
    }
    @FXML
    private void restore_quality(){
        Image image=new Image(String.valueOf(path_to_original_image));
        if (image.getHeight()>=700 && image.getWidth()>=1000) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Ошибка");
            alert.setHeaderText("Невозможно обработать изображение");
            alert.setContentText("Размер изображения '" + image.getUrl() + "' превышает максимально допустимый размер " + 700 + "x" + 1000 + ".");
            alert.showAndWait();
        }else {
            String directoryPath = "C:\\Users\\Dell\\PycharmProjects\\ESRGAN\\LR";
            copyFile(String.valueOf(path_to_original_image), directoryPath);

            String[] command = {"cmd", "/c", "python", "C:\\Users\\Dell\\PycharmProjects\\ESRGAN\\test.py"};

            File workingDirectory = new File("C:\\Users\\Dell\\PycharmProjects\\ESRGAN");
            try {
                Process process = Runtime.getRuntime().exec(command, null, workingDirectory);


                BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream(), "CP1251"));
                String line;

                while ((line = reader.readLine()) != null) {
                    System.out.println(line);
                }

                int exitcode = process.waitFor();

            } catch (IOException | InterruptedException e) {
                throw new RuntimeException(e);
            }
            copyFile("C:\\Users\\Dell\\PycharmProjects\\ESRGAN\\results\\result.png", String.valueOf(path_to_save_directory));
            File file = new File(String.valueOf(path_to_original_image));
```

```
            file.delete();
            renameFile(path_to_save_directory + "\\result.png", nameFile);

            image_after.setImage(new Image(String.valueOf(new
File(path_to_original_image.toUri()))));
        }
    }
    public void copyFile(String originalFilePath, String targetDirectory) {
        try {
            File originalFile = new File(originalFilePath);
            File targetDir = new File(targetDirectory);

            // Проверяем существование исходного файла и целевой директории
            if (!originalFile.exists()) {
                System.err.println("Исходный файл не существует.");
                return;
            }
            if (!targetDir.exists()) {
                System.err.println("Целевая директория не существует.");
                return;
            }



            // Копируем файл
            Path targetPath = new File(targetDir, originalFile.getName()).toPath();
            Files.copy(originalFile.toPath(), targetPath, StandardCopyOption.REPLACE_EXISTING);


            System.out.println("Файл скопирован успешно.");
        } catch (IOException e) {
            System.err.println("Ошибка при копировании файла: " + e.getMessage());
        }
    }
    public void renameFile(String filePath, String newFileName) {
        File file = new File(filePath);

        // Проверяем существование файла
        if (!file.exists()) {
            System.err.println("Файл не существует.");
            return;
        }

        // Получаем путь к директории и создаем новый файл с новым именем
        String directoryPath = file.getParent();
        File newFile = new File(directoryPath, newFileName);

        // Переименовываем файл
        if (file.renameTo(newFile)) {
            System.out.println("Файл успешно переименован.");
        } else {
            System.err.println("Не удалось переименовать файл.");
```

```
        }
    }
    public static void deleteFilesInDirectory(String directoryPath) {
        File directory = new File(directoryPath);

        // Проверяем существование директории
        if (!directory.exists() || !directory.isDirectory()) {
            System.err.println("Директория не существует или не является директорией.");
            return;
        }

        // Получаем список файлов в директории
        File[] files = directory.listFiles();

        // Удаляем каждый файл
        if (files != null) {
            for (File file : files) {
                if (file.isFile()) {
                    if (file.delete()) {
                        System.out.println("Файл " + file.getName() + " удален.");
                    } else {
                        System.err.println("Не удалось удалить файл " + file.getName());
                    }
                }
            }
        }
    }
    @FXML
    public void showAbout(){
        try{
            Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("about.fxml")));
            Scene scene = new Scene(root);

            Stage stage = new Stage();
            stage.setScene(scene);
            stage.setTitle("RestoreVision - Про програму");

            stage.setFullScreen(false);
            stage.getIcons().add(new Image("D:\\Универ\\4 курс\\2
семестр\\Хортюк\\BDR\\khortiuk_2PI_BDR\\src\\main\\resources\\com\\example\\khortiuk_2pi_b
dr\\icons\\icon.png"));
            stage.initStyle(StageStyle.UNDECORATED);
            stage.setResizable(false);
            stage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @FXML
    public void showHelp(){
        try{
```

```
            Parent root =
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("help.fxml")));
            Scene scene = new Scene(root);

            Stage stage = new Stage();
            stage.setScene(scene);
            stage.setTitle("RestoreVision - Довідка");

            stage.setFullScreen(false);
            stage.getIcons().add(new Image("D:\\Универ\\4 курс\\2
семестр\\Хортюк\\BDR\\khortiuk_2PI_BDR\\src\\main\\resources\\com\\example\\khortiuk_2pi_b
dr\\icons\\icon.png"));
            stage.initStyle(StageStyle.UNDECORATED);
            stage.setResizable(false);
            stage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import torch
import torch.nn as nn
import numpy as np
from IPython import embed

from .base_color import *

class ECCVGenerator(BaseColor):
    def __init__(self, norm_layer=nn.BatchNorm2d):
        super(ECCVGenerator, self).__init__()

        model1=[nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1, bias=True),]
        model1+=[nn.ReLU(True),]
        model1+=[nn.Conv2d(64, 64, kernel_size=3, stride=2, padding=1, bias=True),]
        model1+=[nn.ReLU(True),]
        model1+=[norm_layer(64),]

        model2=[nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        model2+=[nn.ReLU(True),]
        model2+=[nn.Conv2d(128, 128, kernel_size=3, stride=2, padding=1, bias=True),]
        model2+=[nn.ReLU(True),]
        model2+=[norm_layer(128),]

        model3=[nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1, bias=True),]
        model3+=[nn.ReLU(True),]
        model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
        model3+=[nn.ReLU(True),]
        model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=2, padding=1, bias=True),]
        model3+=[nn.ReLU(True),]
        model3+=[norm_layer(256),]
```

```
model4=[nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[norm_layer(512),]

model5=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[norm_layer(512),]

model6=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[norm_layer(512),]

model7=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[norm_layer(512),]

model8=[nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=True),]
model8+=[nn.ReLU(True),]
model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
model8+=[nn.ReLU(True),]
model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
model8+=[nn.ReLU(True),]

model8+=[nn.Conv2d(256, 313, kernel_size=1, stride=1, padding=0, bias=True),]

self.model1 = nn.Sequential(*model1)
self.model2 = nn.Sequential(*model2)
self.model3 = nn.Sequential(*model3)
self.model4 = nn.Sequential(*model4)
self.model5 = nn.Sequential(*model5)
self.model6 = nn.Sequential(*model6)
self.model7 = nn.Sequential(*model7)
self.model8 = nn.Sequential(*model8)

self.softmax = nn.Softmax(dim=1)
self.model_out = nn.Conv2d(313, 2, kernel_size=1, padding=0, dilation=1, stride=1,
```

```
bias=False)
        self.upsample4 = nn.Upsample(scale_factor=4, mode='bilinear')

    def forward(self, input_l):
        conv1_2 = self.model1(self.normalize_l(input_l))
        conv2_2 = self.model2(conv1_2)
        conv3_3 = self.model3(conv2_2)
        conv4_3 = self.model4(conv3_3)
        conv5_3 = self.model5(conv4_3)
        conv6_3 = self.model6(conv5_3)
        conv7_3 = self.model7(conv6_3)
        conv8_3 = self.model8(conv7_3)
        out_reg = self.model_out(self.softmax(conv8_3))

        return self.unnormalize_ab(self.upsample4(out_reg))

def eccv16(pretrained=True):
    model = ECCVGenerator()
    if(pretrained):
      import torch.utils.model_zoo as model_zoo
      model.load_state_dict(model_zoo.load_url('https://colorizers.s3.us-east-
2.amazonaws.com/colorization_release_v2-9b330a0b.pth',map_location='cpu',check_hash=True))
    return model
import torch
import torch.nn as nn

from .base_color import *

class SIGGRAPHGenerator(BaseColor):
    def __init__(self, norm_layer=nn.BatchNorm2d, classes=529):
        super(SIGGRAPHGenerator, self).__init__()

        # Conv1
        model1=[nn.Conv2d(4, 64, kernel_size=3, stride=1, padding=1, bias=True),]
        model1+=[nn.ReLU(True),]
        model1+=[nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1, bias=True),]
        model1+=[nn.ReLU(True),]
        model1+=[norm_layer(64),]
        # add a subsampling operation

        # Conv2
        model2=[nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        model2+=[nn.ReLU(True),]
        model2+=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        model2+=[nn.ReLU(True),]
        model2+=[norm_layer(128),]
        # add a subsampling layer operation

        # Conv3
        model3=[nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1, bias=True),]
        model3+=[nn.ReLU(True),]
        model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
```

```
model3+=[nn.ReLU(True),]
model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
model3+=[nn.ReLU(True),]
model3+=[norm_layer(256),]
# add a subsampling layer operation

# Conv4
model4=[nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model4+=[nn.ReLU(True),]
model4+=[norm_layer(512),]

# Conv5
model5=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model5+=[nn.ReLU(True),]
model5+=[norm_layer(512),]

# Conv6
model6=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1, padding=2, bias=True),]
model6+=[nn.ReLU(True),]
model6+=[norm_layer(512),]

# Conv7
model7=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1, bias=True),]
model7+=[nn.ReLU(True),]
model7+=[norm_layer(512),]

# Conv7
model8up=[nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=True)]
model3short8=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]

model8=[nn.ReLU(True),]
model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
model8+=[nn.ReLU(True),]
model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1, bias=True),]
model8+=[nn.ReLU(True),]
model8+=[norm_layer(256),]
```

```
        # Conv9
        model9up=[nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1, bias=True),]
        model2short9=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        # add the two feature maps above

        model9=[nn.ReLU(True),]
        model9+=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        model9+=[nn.ReLU(True),]
        model9+=[norm_layer(128),]

        # Conv10
        model10up=[nn.ConvTranspose2d(128, 128, kernel_size=4, stride=2, padding=1, bias=True),]
        model1short10=[nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1, bias=True),]
        # add the two feature maps above

        model10=[nn.ReLU(True),]
        model10+=[nn.Conv2d(128, 128, kernel_size=3, dilation=1, stride=1, padding=1, bias=True),]
        model10+=[nn.LeakyReLU(negative_slope=.2),]

        # classification output
        model_class=[nn.Conv2d(256, classes, kernel_size=1, padding=0, dilation=1, stride=1,
bias=True),]

        # regression output
        model_out=[nn.Conv2d(128, 2, kernel_size=1, padding=0, dilation=1, stride=1, bias=True),]
        model_out+=[nn.Tanh()]

        self.model1 = nn.Sequential(*model1)
        self.model2 = nn.Sequential(*model2)
        self.model3 = nn.Sequential(*model3)
        self.model4 = nn.Sequential(*model4)
        self.model5 = nn.Sequential(*model5)
        self.model6 = nn.Sequential(*model6)
        self.model7 = nn.Sequential(*model7)
        self.model8up = nn.Sequential(*model8up)
        self.model8 = nn.Sequential(*model8)
        self.model9up = nn.Sequential(*model9up)
        self.model9 = nn.Sequential(*model9)
        self.model10up = nn.Sequential(*model10up)
        self.model10 = nn.Sequential(*model10)
        self.model3short8 = nn.Sequential(*model3short8)
        self.model2short9 = nn.Sequential(*model2short9)
        self.model1short10 = nn.Sequential(*model1short10)

        self.model_class = nn.Sequential(*model_class)
        self.model_out = nn.Sequential(*model_out)

        self.upsample4 = nn.Sequential(*[nn.Upsample(scale_factor=4, mode='bilinear'),])
        self.softmax = nn.Sequential(*[nn.Softmax(dim=1),])

    def forward(self, input_A, input_B=None, mask_B=None):
```

```
        if(input_B is None):
            input_B = torch.cat((input_A*0, input_A*0), dim=1)
        if(mask_B is None):
            mask_B = input_A*0

        conv1_2 =
self.model1(torch.cat((self.normalize_l(input_A),self.normalize_ab(input_B),mask_B),dim=1))
        conv2_2 = self.model2(conv1_2[:,:,::2,::2])
        conv3_3 = self.model3(conv2_2[:,:,::2,::2])
        conv4_3 = self.model4(conv3_3[:,:,::2,::2])
        conv5_3 = self.model5(conv4_3)
        conv6_3 = self.model6(conv5_3)
        conv7_3 = self.model7(conv6_3)

        conv8_up = self.model8up(conv7_3) + self.model3short8(conv3_3)
        conv8_3 = self.model8(conv8_up)
        conv9_up = self.model9up(conv8_3) + self.model2short9(conv2_2)
        conv9_3 = self.model9(conv9_up)
        conv10_up = self.model10up(conv9_3) + self.model1short10(conv1_2)
        conv10_2 = self.model10(conv10_up)
        out_reg = self.model_out(conv10_2)

        conv9_up = self.model9up(conv8_3) + self.model2short9(conv2_2)
        conv9_3 = self.model9(conv9_up)
        conv10_up = self.model10up(conv9_3) + self.model1short10(conv1_2)
        conv10_2 = self.model10(conv10_up)
        out_reg = self.model_out(conv10_2)

        return self.unnormalize_ab(out_reg)

def siggraph17(pretrained=True):
    model = SIGGRAPHGenerator()
    if(pretrained):
        import torch.utils.model_zoo as model_zoo
        model.load_state_dict(model_zoo.load_url('https://colorizers.s3.us-east-
2.amazonaws.com/siggraph17-df00044c.pth',map_location='cpu',check_hash=True))
    return model

from fastai import *
from fastai.core import *
from fastai.vision.transform import get_transforms
from fastai.vision.data import ImageImageList, ImageDataBunch, imagenet_stats


def get_colorize_data(
    sz: int,
    bs: int,
    crappy_path: Path,
    good_path: Path,
    random_seed: int = None,
    keep_pct: float = 1.0,
    num_workers: int = 8,
```

```
    stats: tuple = imagenet_stats,
    xtra_tfms=[],
) -> ImageDataBunch:

    src = (
        ImageImageList.from_folder(crappy_path, convert_mode='RGB')
        .use_partial_data(sample_pct=keep_pct, seed=random_seed)
        .split_by_rand_pct(0.1, seed=random_seed)
    )

    data = (
        src.label_from_func(lambda x: good_path / x.relative_to(crappy_path))
        .transform(
            get_transforms(
                max_zoom=1.2, max_lighting=0.5, max_warp=0.25, xtra_tfms=xtra_tfms
            ),
            size=sz,
            tfm_y=True,
        )
        .databunch(bs=bs, num_workers=num_workers, no_check=True)
        .normalize(stats, do_y=True)
    )

    data.c = 3
    return data


def get_dummy_databunch() -> ImageDataBunch:
    path = Path('./dummy/')
    return get_colorize_data(
        sz=1, bs=1, crappy_path=path, good_path=path, keep_pct=0.001
    )
```

Додаток Г – Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РЕСТАВРАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

Вінницький Національний Технічний Університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА

на тему:

«Розробка програмного забезпечення для реставрації зображень з використанням нейронних мереж»

Виконала:
ст. групи 2ПІ-20б
Хортюк Д. С.

Науковий керівник:
д.т.н., професор каф. ПЗ
Романюк О.Н.

Рисунок Г.1 – Титульний аркуш

Реставрація зображень

Процес реставрації зображення – це комплексний набір методів і технік, спрямованих на відновлення пошкоджених або втрачених частин зображення з метою поліпшення його якості, чіткості, кольору та візуальної привабливості.

Рисунок Г.2 – Аналіз предметної галузі

Використання нейронних мереж у задачах обробки зображень

Процес обробки зображення нейронною мережею включає кілька ключових кроків, що відбуваються в різних шарах мережі:

1. Згорткові операції
2. Функція активації
3. Пулінг
4. Функція втрат
5. Оптимізація

Рисунок Г.3 – Використання нейронних мереж у задачах реставрації зображень

Мета дослідження

- **Мета та завдання дослідження.** Метою дослідження є підвищення ефективності реставрації архівних зображень за рахунок використанням нейронної мережі.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі **задачі**:

- аналіз підходів до реставрації зображень з використанням нейронних мереж;
- розробка алгоритму навчання нейронної мережі, який буде спроможний аналізувати вхідні данні та навчати нейронну мережу виявляти дефекти на зображеннях;
- розробити алгоритм покращення роздільної здатності зображення з використанням засобів нейронної мережі;
- розробити алгоритм реставрації кольору зображення з використанням засобів нейронної мережі;
- розробити алгоритм використання нейронної мережі, який буде спроможний виявляти пошкодження, реставрувати та покращувати роздільну здатність зображення;
- тестування розроблених програмних продуктів.

- **Об'єкт дослідження** – процес реставрації зображень з використанням нейронних мереж.

- **Предмет дослідження** – методи та засоби реставрації зображень з використанням нейронних мереж.

- **Методи дослідження.** У процесі дослідження застосовувалися: теорія нейронних мереж; теорія алгоритмів для обробки та аналізу зображень та методи і засоби нейронних мереж для реставрації пошкоджених зображень, комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Рисунок Г.4 – Мета дослідження

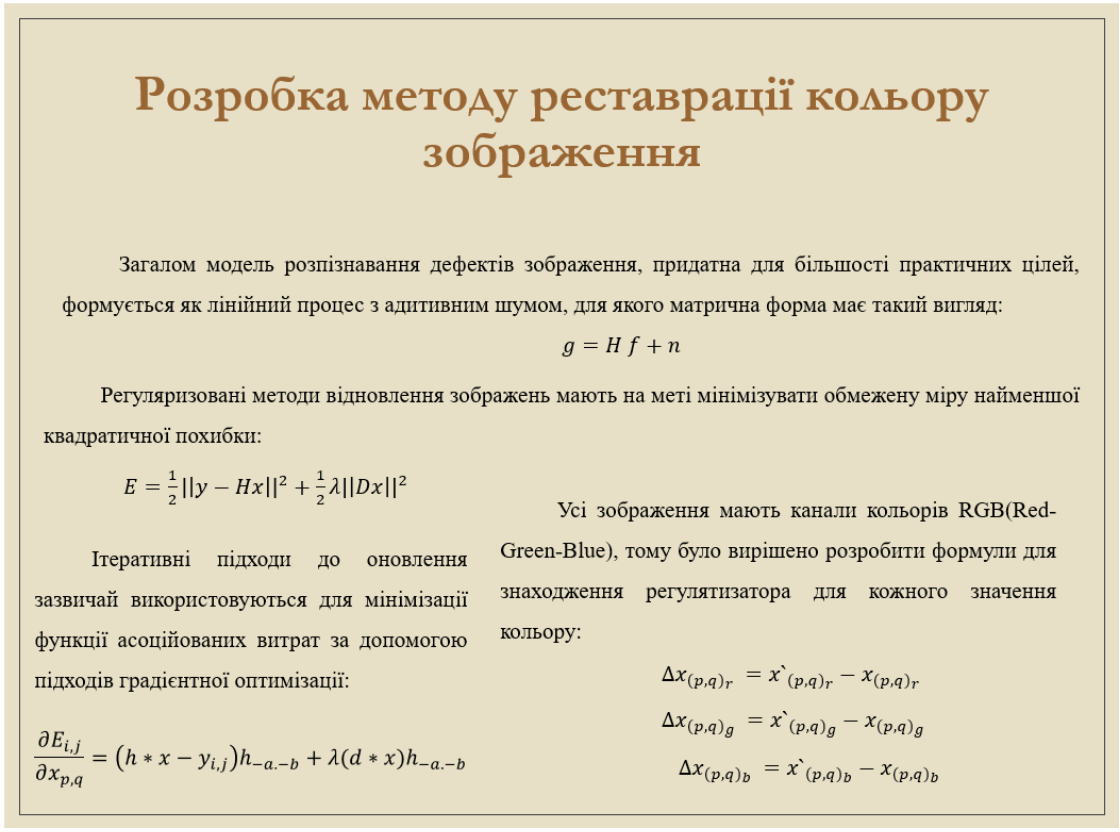

Рисунок Г.5 – Розробка методу реставрації кольору зображення

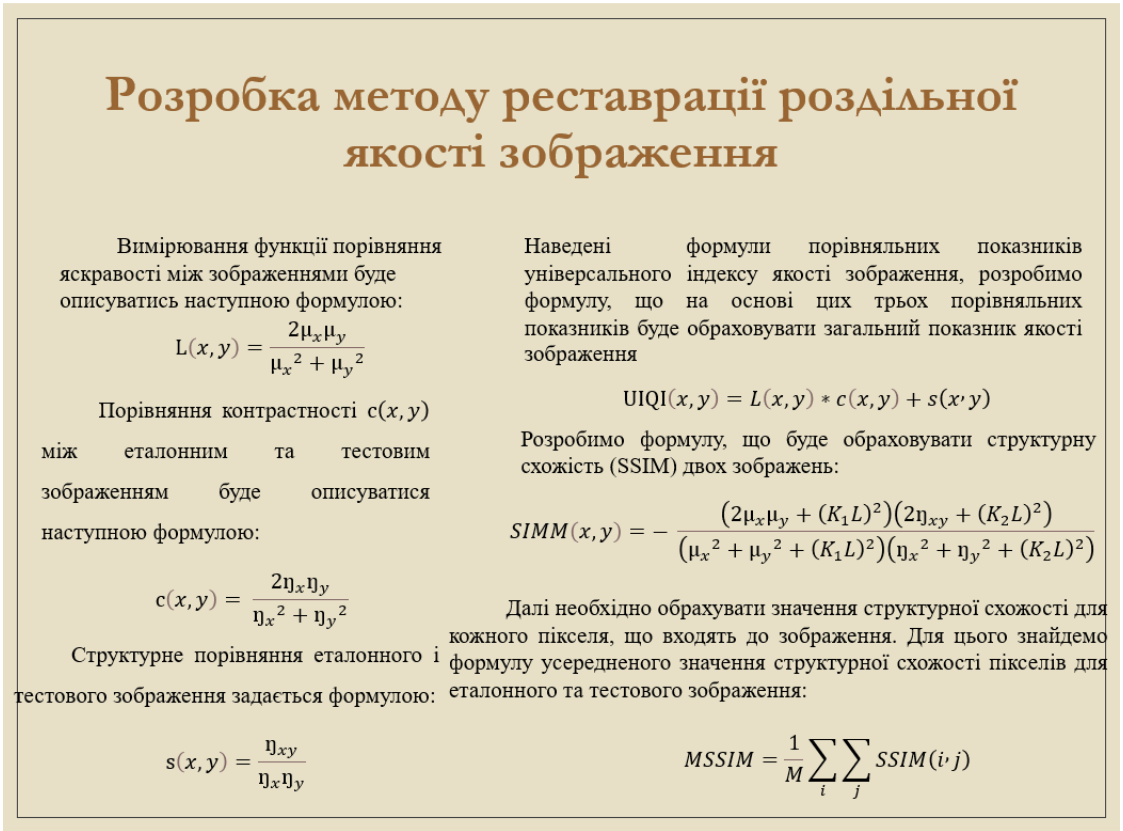

Рисунок Г.6 – Розробка методу реставрації роздільної якості зображення

Рисунок Г.7 – Особливості навчання нейронної мережі

Рисунок Г.8 – Розробка модуля реставрації кольору

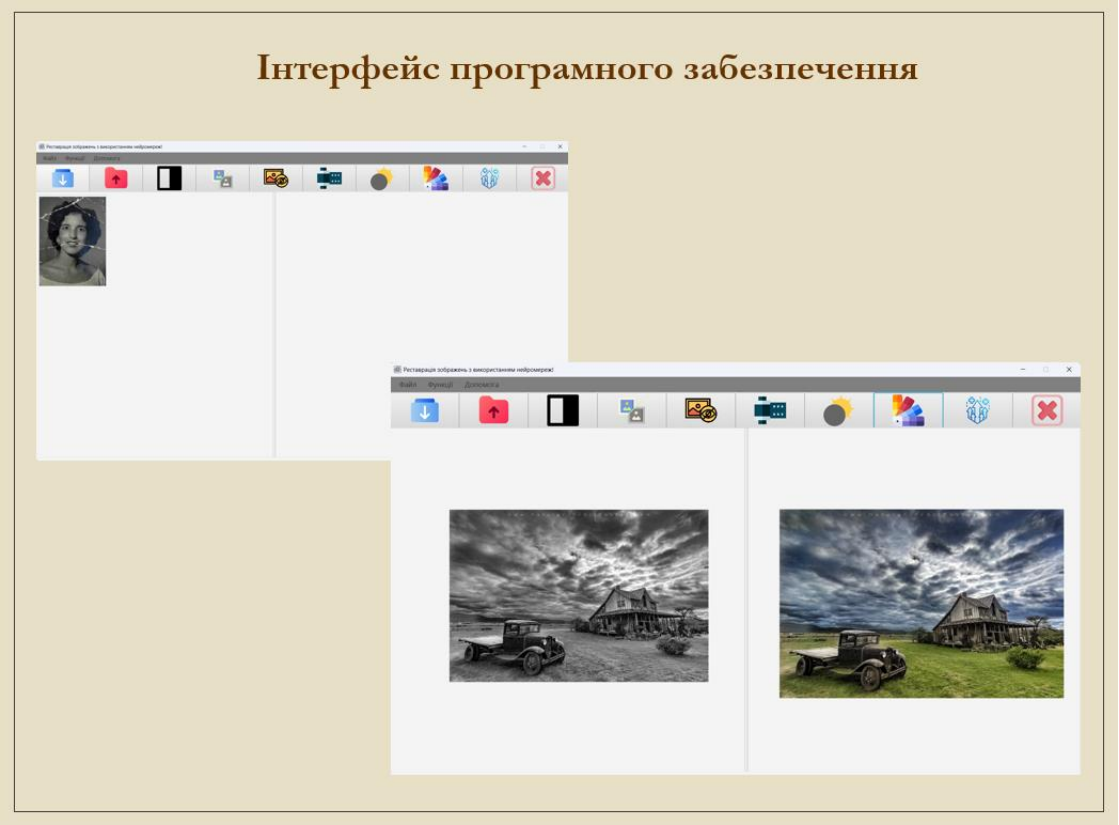

Рисунок Г.9 – Інтерфейс програмного забезпечення

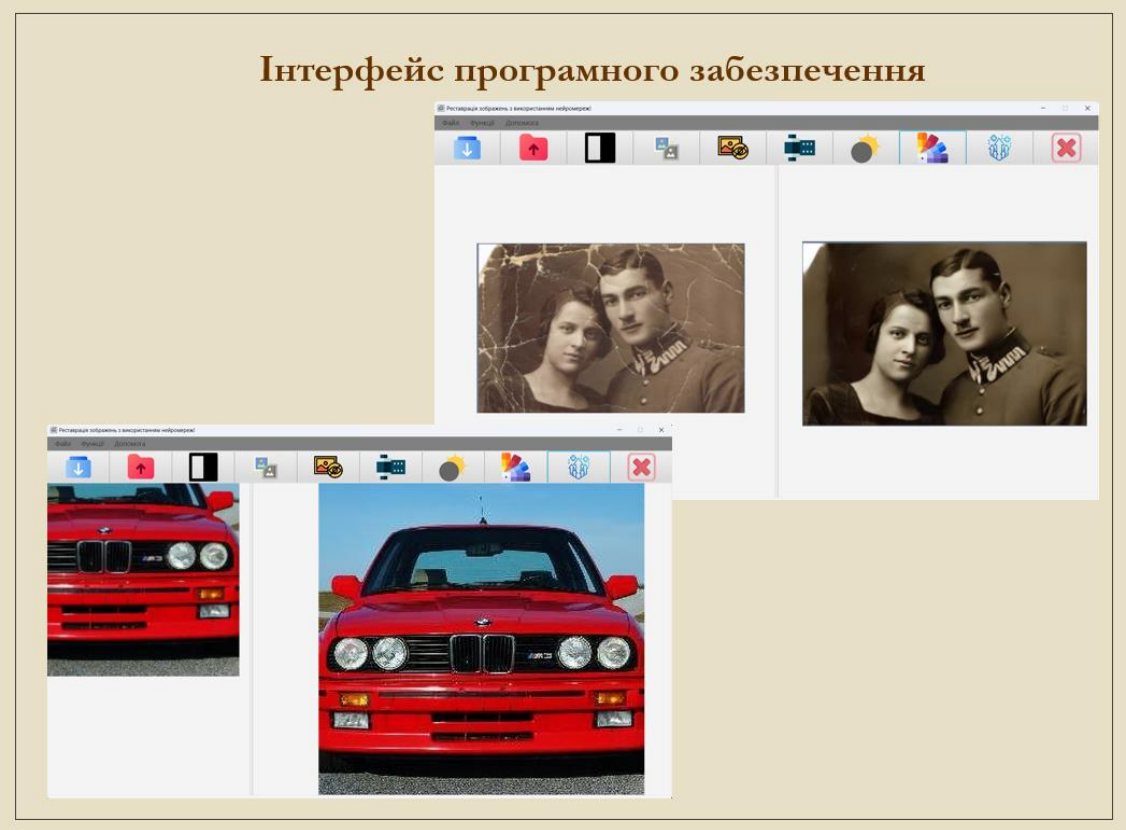

Рисунок Г.10 – Інтерфейс програмного забезпечення

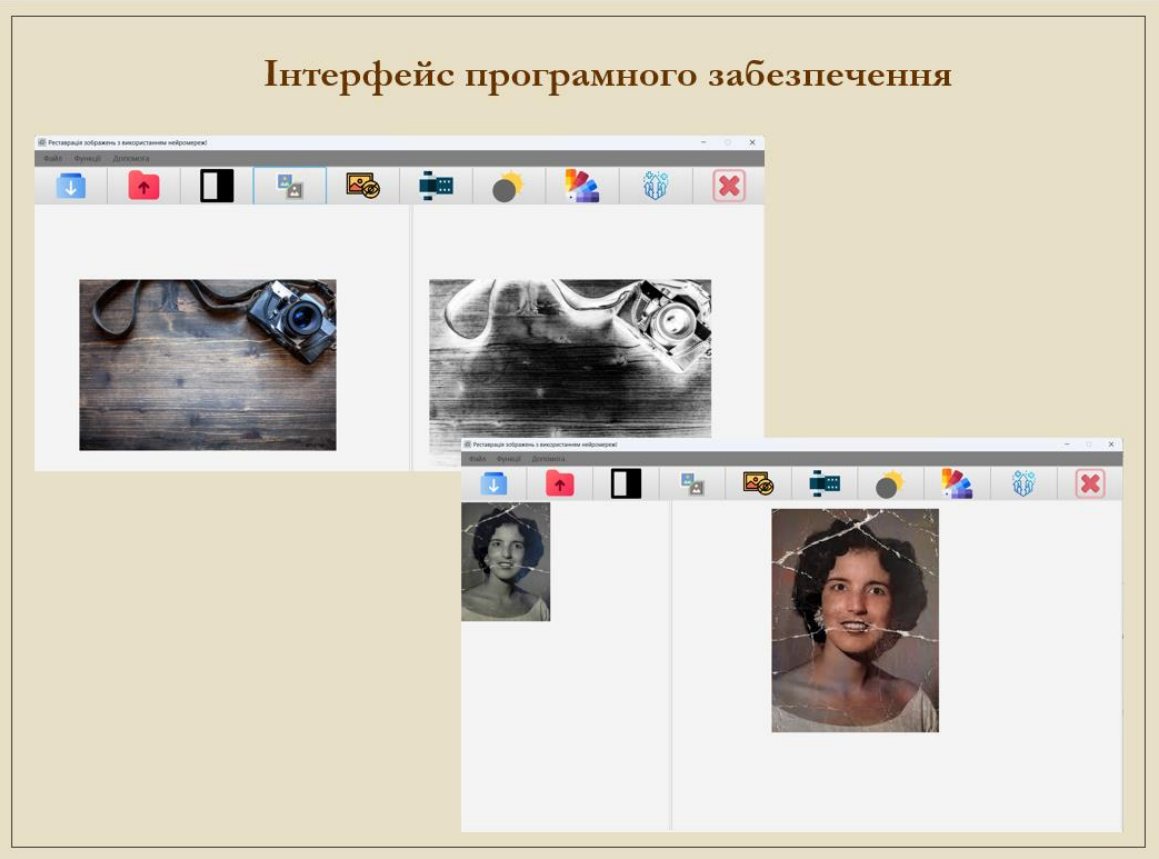

Рисунок Г.11 – Інтерфейс програмного забезпечення

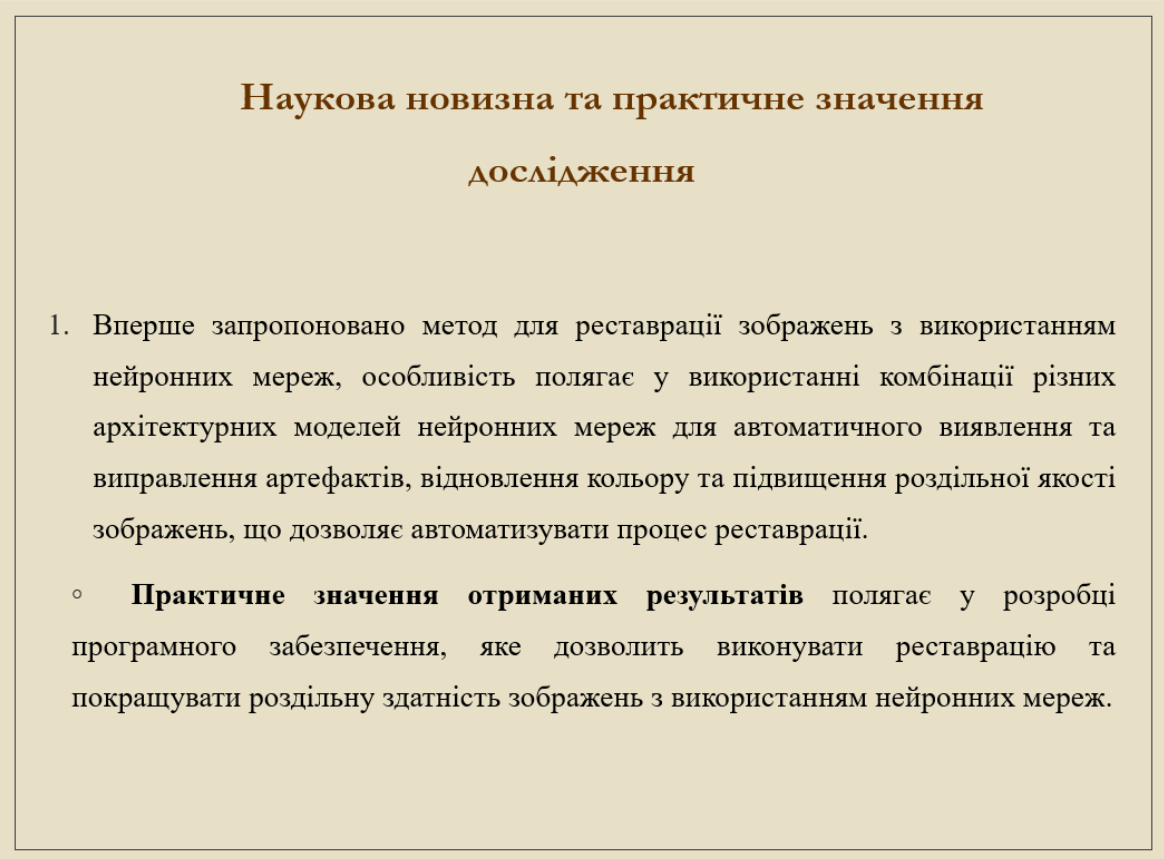

Рисунок Г.12 – Наукова новизна та практичне значення дослідження

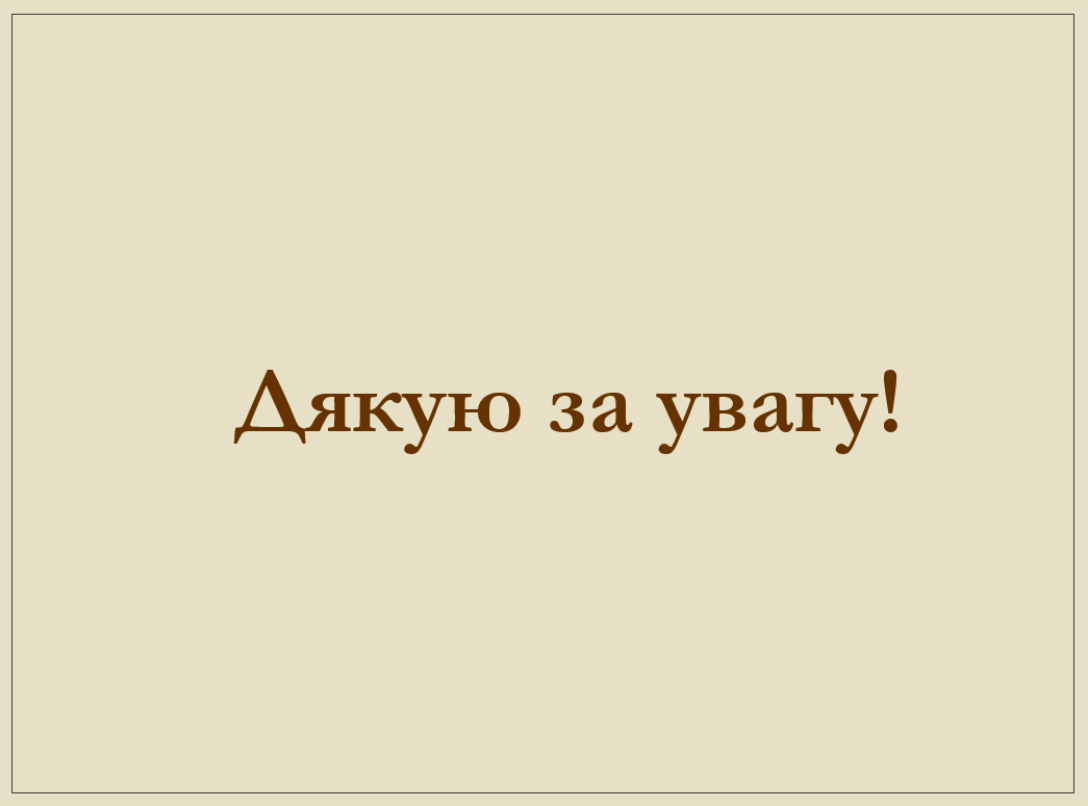

Рисунок Г.13 – Закінчення презентації