Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ РОЗФАРБОВУВАННЯ ЗОБРАЖЕННЯ НА ОСНОВІ НЕЙРОМЕРЕЖІ

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гречкосій М. П.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., проф. каф. КСУ
Биков М. М.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри _____
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гречкосію Марку Петровичу

1. Тема роботи: Програмний модуль розфарбовування зображення на основі нейромережі.
керівник роботи: Колесницький Олег Костянтинович, к.т.н., доц.
затверджені наказом вищого навчального закладу "20" березня 2025 року №

2. Термін подання студентом роботи 30" травня 2025 р.

3. Вихідні дані до роботи :
Розмір вхідного зображення – не менше 224x224; обсяг навчальної вибірки – не менше 50000 зображень; обсяг тестової вибірки – не менше 5000 зображень; вид кодувальника параметрів зображень – нейронна мережа, використання об'єктно-орієнтованої мови програмування.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
проаналізувати існуючі методи розфарбовування зображення та вибрати найбільш ефективний;
вибрати нейронну мережу, спроектувати її структуру та метод навчання
розробити алгоритм роботи модуля розфарбовування зображення;
спроектувати та реалізувати модуль розфарбовування зображення за допомогою нейронної мережі;
протестувати роботу програми розфарбовування зображення на основі нейронної мережі.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Схема алгоритму роботи програми
Структура нейронної мережі
Формалізація задачі розфарбовування сірого зображення
Результати роботи програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Колесницький О. К., к.т.н., проф. каф. КН		

7. Дата видачі завдання 05 травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Аналіз предметної області	05.05.25	08.05.25	
2	Обгрунтування методу розв'язання задачі, проектування програмного модуля розфарбовування зображення	09.05.25	13.05.25	
3	Розробка алгоритму роботи	14.05.25	16.05.25	
4	Програмна реалізація модуля розфарбовування зображення	17.05.25	28.05.25	
5	Аналіз результатів тестування модуля розфарбовування зображення	29.05.25	01.06.25	
6	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ **Гречкосій М.П.**
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ **Колесницький О. К**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Гречкосій М. П. Програмний модуль розфарбовування зображення на основі нейромережі. Бакалаврська кваліфікаційна робота складається з 73 сторінок формату А4, на яких є 19 рисунків, 2 таблиці, список використаних джерел містить 33 найменування.

Метою роботи є підвищення якості розфарбовування чорно-білих зображень за рахунок використання згорткової нейронної мережі.

У роботі розроблено метод розфарбовування зображення на основі нейромережі, який відноситься до автоматичних методів мультимодального розфарбовування зображень у градаціях сірого. Метод базується на згорткових нейронних мережах типу енкодер-декодер, однак включає попередньо навчену додаткову модель Inception-ResNet-v2 згорткової нейромережі, та використовує колірний простір Lab*. Метод програмно реалізовано на мові програмування Python з використанням спеціалізованих бібліотек Keras та TenzorFlow. Набір даних у кількості 60000 зображень для навчання і тестування модуля сформовано на основі відомої бази даних ImageNet, з яких 54000 було взято як навчальні та 6000 як тестові. Для якісної оцінки процесу розфарбовування зображень було використано таку метрику як середній відсоток сприйняття розфарбованих зображень як справжніх групою користувачів (експертів). Для кількісної оцінки було використано таку метрику як усереднена середньоквадратична помилка прогнозування кольору. Розроблений програмний модуль розфарбовування зображень має порівняно з програмою-аналогом збільшену на 11÷19% якість розфарбовування чорно-білих зображень.

Ключові слова: розфарбовування, чорно-біле зображення, кольорове зображення, згорткова нейронна мережа.

ABSTRACT

Hrechkosii M. P. Neural network-based image coloring software module. The bachelor thesis consists of 73 pages of A4 format, on which there are 19 figures, 2 tables, the list of used sources contains 33 names.

The aim of the work is to improve the quality of coloring black and white images by using a convolutional neural network.

The work developed a method for coloring images based on a neural network, which refers to automatic methods for multimodal coloring of grayscale images. The method is based on convolutional neural networks of the encoder-decoder type, but includes a pre-trained additional model of the Inception-ResNet-v2 convolutional neural network, and uses the Lab* color space. The method is implemented programmatically in the Python programming language using specialized libraries Keras and TenzorFlow. A dataset of 60,000 images for training and testing the module was formed based on the well-known ImageNet database, of which 54,000 were taken as training and 6,000 as test. For a qualitative assessment of the image coloring process, a metric such as the average percentage of perception of colored images as real by a group of users (experts) was used. For quantitative assessment, a metric such as the averaged root mean square error of color prediction was used. The developed image coloring software module has an increased quality of coloring black and white images by 11÷19% compared to the analog program.

Keywords: coloring, black and white image, color image, convolutional neural network.

ЗМІСТ

ВСТУП

Актуальність досліджень. Колір відіграє дуже важливу роль у процесі пізнання світу людиною, а насичені кольори можуть не лише виражати більше інформації, але й покращувати візуальний досвід людини. Розфарбовування зображень було дуже активною темою досліджень у галузі цифрової обробки зображень і є міждисциплінарною областю, що охоплює такі дисципліни, як комп'ютерний зір, комп'ютерна графіка, розпізнавання образів та взаємодія людини з комп'ютером. Розфарбовування широко використовується в багатьох галузях, таких як колоризація фотографій у градаціях сірого, відновлення кольорів старих плівок, автоматичне розфарбовування мультфільмів тощо.

Розфарбовування зображень у градаціях сірого оживляє зображення та надає їм більш реалістичного вигляду об'єктів, присутніх у реальному світі. Його можна застосовувати для реставрації історичних монохромних зображень та фільмів, а також допомагати художникам-мультиплікаторам. Попередні підходи до розфарбовування зображень включали нанесення кольорових підказок на вхідне зображення у градаціях сірого або вибір еталонного кольорового зображення для можливості перенесення кольорових рис на вхідне зображення у градаціях сірого. У цьому дослідженні тягар необхідності втручання користувача в процес розфарбовування усувається за допомогою згорткової нейронної мережі (CNN), яка навчається на великому наборі кольорових зображень. Якість процесу розфарбовування оцінюється за допомогою L2-регресійних втрат, де розфарбоване зображення порівнюється з еталонним зображенням для досягнення відносно близького розфарбовування. Попередні результати розфарбовування показують ефективне розфарбовування глобальної семантики зображення, такої як небо, трава та дерева, але страждають від розмивання кольорів, коли певні кольори перевищують їхні призначені області. Покращити процес розфарбовування може новий підхід з використанням згорткової нейронної мережі (ЗНМ), де

використовується ансамблеве розфарбовування, коли окремі розфарбовування виконуються двома різними ЗНМ, а потім мережа уточнення об'єднує та уточнює результати. Ефективність підходу додатково оцінюється шляхом виконання «тесту Тюрінга на розфарбовування», де учасникам доручено визначити, які з заданого набору зображень є розфарбованими, а які – реальними.

Метою дослідження є підвищення якості розфарбовування чорно-білих зображень за рахунок використання згорткової нейронної мережі.

Об’єктом дослідження є процес комп’ютеризованого розфарбовування чорно-білих зображень на основі нейромережі.

Предметом дослідження є алгоритми та програмні засоби розфарбовування чорно-білих зображень на основі згорткової нейронної мережі та якість їх роботи.

Задачі дослідження:

1. Проаналізувати відомі методи розфарбовування чорно-білих зображень та обрати напрямок досліджень;
2. Розробити метод розфарбовування чорно-білих зображень з використанням згорткових нейронних мереж;
3. Розробити архітектуру згорткової нейронної мережі для розфарбовування чорно-білих зображень;
4. Проаналізувати процес навчання згорткової нейронної мережі для розфарбовування чорно-білих зображень;
5. Розробити алгоритм роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі;
6. Здійснити програмну реалізацію модуля розфарбовування зображень на основі згорткової нейронної мережі;
7. Провести тестування програмного модуля розфарбовування зображень на основі згорткової нейронної мережі.

За матеріалами бакалаврської кваліфікаційної роботи опубліковано тези доповіді [1] на конференції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗФАРБОВУВАННЯ ЗОБРАЖЕНЬ

1.1 Постановка задачі

Спочатку визначимо проблему розфарбовування з точки зору колірного простору CIE Lab. Як і колірний простір RGB, це 3-канальний колірний простір, але на відміну від RGB, інформація про колір кодується лише в каналах a (зелено-червоний компонент) та b (синьо-жовтий компонент). Канал L (яскравість) кодує лише інформацію про інтенсивність.

Зображення у градаціях сірого, яке ми хочемо розфарбувати, можна розглядати як L-канал зображення в колірному просторі Lab, а наша мета – знайти компоненти a та b. Отримане таким чином зображення Lab можна перетворити в колірний простір RGB за допомогою стандартних перетворень колірного простору. Наприклад, в OpenCV цього можна досягти за допомогою cvtColor з опцією COLOR_BGR2Lab.

Для спрощення обчислень, простір ab колірного простору Lab квантується на 313 інтервалів, як показано на рисунку 1.1.

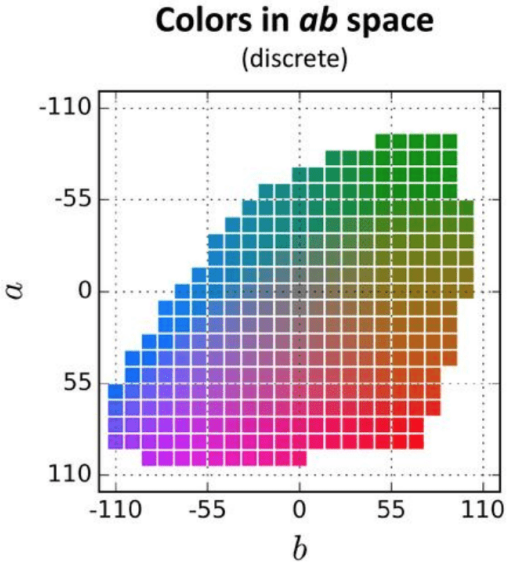


Рисунок 1.1 – Вид 313 квантованих кольорів в ab-просторі

Замість того, щоб знаходити значення a та b для кожного пікселя, через це квантування нам просто потрібно знайти номер інтервалу від 0 до 312. Ще один спосіб мислення про проблему полягає в тому, що у нас вже є канал L, який приймає значення від 0 до 255, і нам потрібно знайти канал ab, який приймає значення від 0 до 312. Таким чином, задача прогнозування кольору тепер перетворюється на задачу поліноміальної класифікації, де для кожного сірого пікселя є 313 класів на вибір.

Розфарбовування зображень у градаціях сірого може мати великий вплив у широкому спектрі галузей, наприклад, на ремастеринг історичних зображень та покращення потоків спостереження. Інформаційний вміст зображення у градаціях сірого досить обмежений, тому додавання кольорових компонентів може дати більше інформації про його семантику. У контексті глибокого навчання такі моделі, як Inception [2], ResNet [3] або VGG [4], зазвичай навчаються з використанням кольорових наборів даних зображень. Під час застосування цих мереж до зображень у градаціях сірого попередній крок розфарбовування може допомогти покращити результати. Однак, розробка та впровадження ефективної та надійної системи, яка автоматизує цей процес, досі залишається складним завданням. Складність ще більше зростає, якщо ми прагнемо обдурити людське око.

У зв'язку з цим ми пропонуємо модель, яка здатна певною мірою розфарбовувати зображення, поєднуючи глибоку архітектуру згорткової нейронної мережі та останню випущену на сьогодні модель Inception, а саме Inception-ResNet-v2 [5], яка базується на Inception v3 [2] та ResNet від Microsoft [3,6]. Хоча глибока ЗНМ навчається з нуля, Inception-ResNet-v2 використовується як високорівневий екстрактор ознак, який надає інформацію про вміст зображення, що може допомогти в їх розфарбовуванні.

Через часові обмеження розмір навчального набору даних доволі малий, що призводить до обмеження використовуваної моделі невеликим розмаїттям зображень. Тим не менш, отримані результати досліджують деякі підходи,

використані іншими дослідниками, та підтверджують можливість автоматизації процесу розфарбовування.

1.2 Огляд відомих методів розфарбовування зображень

Розфарбовування зображень – це класична та важлива тема в комп'ютерній графіці, метою якої є додавання кольору до монохроматичного вхідного зображення для отримання барвистого результату.

Ранні роботи з розфарбовування в основному зосереджувалися на розробці методів покращення якості розфарбовування. За останні кілька років дослідники розглядали більше можливостей, таких як поєднання розфарбовування з NLP (обробкою природної мови), та більше зосереджувалися на промисловому застосуванні. Для кращого контролю кольору розроблені різні типи керування кольором, такі як надання опорних зображень.

Для розфарбовування зображень у градаціях сірого більшість методів перетворюють зображення в колірний простір YUV або Lab та відновлюють значення каналів кольоровості зображення, що підлягає розфарбуванню, на основі подібності каналу яскравості.

Методи розфарбовування зображень у градаціях сірого можуть бути згруповані в три категорії:

1) повністю автоматичні методи розфарбовування,
2) напівавтоматичні методи розфарбовування на основі кольорових штрихів або еталонних зображень
3) методи розфарбовування зображень на основі тексту.

Значення кольору кожного пікселя на зображенні у градаціях сірого знаходиться між чорним та білим. Канал градацій сірого можна витягти з кольорових зображень, але такі зображення, як фотографії, зроблені в минулому, та багато коміксів, мають лише інформацію про градації сірого та можуть отримати користь від розфарбовування. Методи розфарбовування

зображень у градаціях сірого можна розділити на дві групи залежно від того, чи використовується взаємодія, а саме: автоматичні методи розфарбовування та напівавтоматичні методи розфарбовування. У першій групі дослідники використовували технологію глибокого навчання на основі даних для автоматичного розфарбовування зображень у градаціях сірого на основі навчальних даних [7, 8, 9]. Наприклад, існує автоматична модель з відкритим кодом DeOldify (Antic, 0000), яку можна використовувати безкоштовно. Для другої групи методи розфарбовування часто отримують певну інформацію від користувачів, наприклад, шляхом малювання кольорових штрихів [10], надання опорних кольорових зображень або надання певної кольорової теми. Включення вказівок користувача зазвичай підвищує ефективність та правильність моделі розфарбовування. Це також допомагає вирішити притаманні неоднозначності для некоректно поставленої задачі розфарбовування (наприклад, листя дерев може бути зеленим навесні та жовтим восени). Крім того, у деяких недавніх дослідженнях дослідники також вивчали використання семантичної інформації для керування розфарбовуванням зображень, наприклад, розфарбовування у градаціях сірого на основі текстових сценаріїв.

У даному дослідженні нас цікавить тільки автоматичне розфарбовування зображень у градаціях сірого.

На відміну від методів розфарбовування, заснованих на вказівках, в методах автоматичного розфарбовування зображень дослідники можуть розробити модель, яка забезпечує кілька кольорів для одного пікселя, щоб вирішити проблему мультимодального розфарбовування монохромних зображень. Наприклад, моделі розфарбовування генеруватимуть зображення зеленого або жовтого листя. Метод автоматичного розфарбовування розділимо на дві категорії залежно від різноманітності згенерованих результатів;

– унімодальне розфарбовування, в якому методи можуть генерувати лише один результат,
– мультимодальне розфарбовування, де методи можуть генерувати кілька різноманітних результатів та вводити їх відповідно.

1.2.1. Унімодальне розфарбовування зображень

Ларссон та ін. (2016) [11] запропонували метод автоматичної розфарбовки, заснований на самокерованому процесі навчання візуального представлення. Мережа побудована на повністю згортковій мережі VGG-16 [4] з видаленим шаром класифікації та доданим шаром фільтра. Крім того, модель використовує з'єднання пропущених шарів для об'єднання ознак різних згорткових шарів, щоб забезпечити вхідні дані для шару класифікації, який прогнозує гістограму кольорів кожного пікселя. Їїзука та ін. (2016) [8] запропонували вивчати глобальні та локальні ознаки окремо від зображення, а потім об'єднувати їх разом для остаточного процесу розфарбовування (див. рис. 1.2). Однак для об'єктів з кількома різними кольорами результат, найімовірніше, дасть домінантні кольори, які вивчаються під час навчання, як-от листя зеленого кольору.

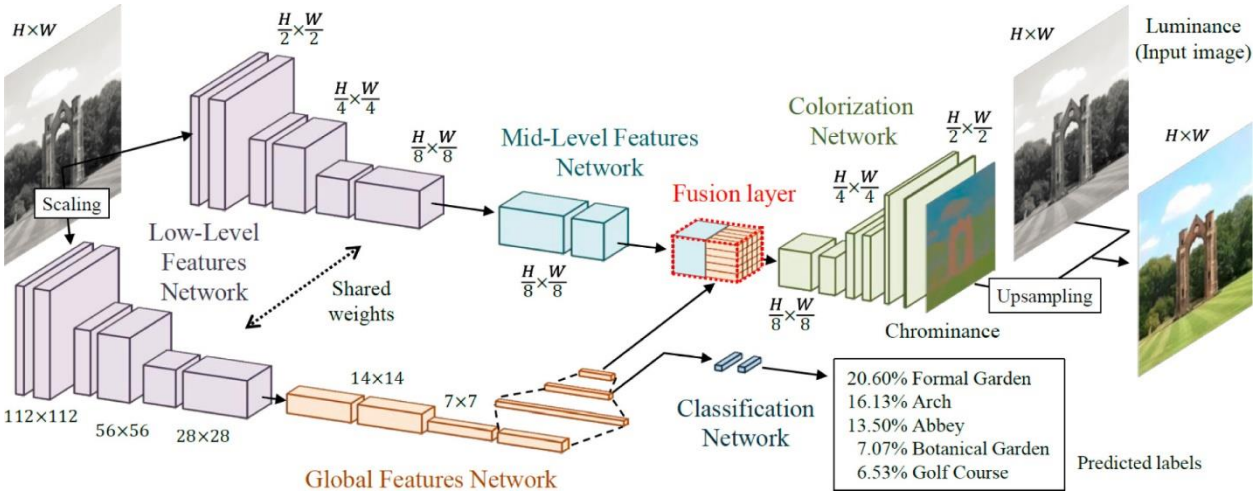


Рисунок 1.2 – Огляд наскрізної мережі для розфарбовування зображень у градаціях сірого

Попередні методи розфарбовування зазвичай навчаються розфарбовувати все зображення, і тому вони часто не підходять для розфарбовування екземплярів на зображенні. Су та ін. (2020) [12] запропонували метод реалізації розфарбовування з урахуванням екземплярів. Спочатку вони визначають розташування цільових об'єктів, а потім розфарбовують об'єкт та зображення загалом відповідно. Коли в одному екземплярі класу відбувається виявлення промахів або перекриття, результат буде змінено, як показано на рис. 1.3. Враховуючи обмеження навчальних даних, в роботі [13] намагаються розфарбувати зображення з невеликою кількістю даних. Вся їхня мережа складається з мережі пам'яті та мереж розфарбовування. Мережа пам'яті навчається без нагляду, щоб допомогти отримати найбільш схожі кольорові ознаки, які відповідають вхідному зображенню, а потім задавати їх як умову для мереж розфарбовування.

(a) Missing detections (b) Superimposed detections

Рисунок 1.3 – Випадки невдачі розфарбовування зображень на основі екземплярів

Унімодальні автоматичні методи розфарбовування можуть генерувати зображення з прийнятним кольором, коли задані довільні зображення у градаціях сірого, але для вхідного зображення у градаціях сірого може бути згенеровано лише одне відповідне кольорове зображення. Один згенерований

результат може не відповідати очікуванням користувача, і користувач не може вказати локальні або специфічні для об'єкта кольори. Різні результати можуть бути згенеровані шляхом введення випадкових шумових ознак або випадкових векторів колірних умов у мережевий модуль.

1.2.2. Мультимодальне розфарбовування зображень

Прагнучи вирішити проблему, що розфарбовування вимагає багато взаємодії з користувачем, а насиченість кольорів розфарбованих зображень, як правило, низька, Чжан та ін. (2016) запропонували повністю автоматичний метод розфарбовування, який може генерувати насичені та реалістичні зображення розфарбовування. Цей метод перетворює завдання розфарбовування на завдання самостійного навчання виразів шляхом вивчення семантичного та текстурного відображення між зображенням у градаціях сірого та кольоровим зображенням. Водночас задача розфарбовування новим чином перетворюється на завдання класифікації, і розподіл кольорів прогнозується для кожного пікселя, щоб вирішити задачу мультимодального розфарбовування зображення, що підтримує різноманітність результатів розфарбовування. Чжан та ін. (2016) [14] були натхненні методом імітованого відпалу і запропонували операцію, яка обчислює відпалене середнє значення розподілу, щоб оцінити значення кольору простору з розподілу кольорів кожного пікселя. Значення кожного пікселя в завданні розфарбовування зображення у градаціях сірого не є фіксованим, і той самий об'єкт у реальному світі може бути розфарбований по-різному.

На відміну від [14], у роботі [15] не лише розглядають оцінку значення кольору кожного пікселя, але й враховують загальну просторову безперервність результатів розфарбовування. Цей метод використовує варіаційний автоенкодер (variational autoencoder - VAE) для вивчення низьковимірного латентного змінного вбудовування колірного поля та використовує мережу змішаної щільності (Mixed Density Network - MDN) для

вивчення мультимодальної моделі, обумовленої зображенням у градаціях сірого. Нарешті, з MDN беруться кілька зразків, які об'єднуються з декодером VAE для отримання кількох результатів розфарбовування для кожного зразка, щоб забезпечити багатий набір результатів розфарбовування.

Хоча модель класифікації, заснована на розподілі кольорів, та генеративна модель, заснована на варіаційних автоенкодерах, можуть отримати різноманітні схеми розфарбовування, результатам розфарбовування бракує узгодженості просторової структури та керованості кольору користувачем. Іноді в одній семантичній області в результаті розфарбовування з'являються плями різних кольорів. Щоб забезпечити глобальну узгодженість розфарбовування та керованість користувачем, Мессауд та ін. (2018) [9] запропонували умовне випадкове поле на основі VAE та використовувати гауссове умовне марковське випадкове поле (G-CRF) для збору глобальної статистики зображення, моделюючи вихідний простір декодера VAE та кодування інформації про редагування користувачем.

Зі швидким розвитком нейромереж типу Transformer [16] у галузі комп'ютерного зору, Kumar et al. (2020) [17] запропонували архітектуру мережі розфарбовування у градаціях сірого (Colorization Transformer, ColTran) на основі блоків Transformer. ColTran в основному складається з авторегресивного колоризатора, семплера кольору та просторового семплера. Авторегресивний колоризатор дозволив зіставити кольорову інформацію з вхідними зображеннями у градаціях сірого з низькою роздільною здатністю, а потім семплер кольору та просторовий семплер перетворювали кольорові зображення низької роздільної здатності на зображення високої роздільної здатності повністю паралельним чином. Завдяки кращій здатності трансформера до зіставлення, цей метод може забезпечити різноманітні кольорові сірі зображення відповідно до різних опорних кольорових зображень.

Порівняно з унімодальним розфарбовуванням, багатомодальні методи розфарбовування можуть генерувати кілька кольорових результатів для

заданого вхідного сигналу у градаціях сірого. Хоча ці автоматичні методи не потребують взаємодії з користувачем, згенеровані результати спираються на попередньо навчені мережеві моделі. Користувач не може налаштувати згенеровані результати, такі як загальний стиль розфарбовування або кольори деталей, що ускладнює отримання результатів, яких він очікує.

1.3 Обгрунтування вибору аналогу до розроблюваного модуля розфарбовування зображень

В останні роки експериментально було доведено, що звичайні нейронні мережі (CNN) майже вдвічі знижують рівень помилок під час розпізнавання об'єктів [18], що призвело до масового переходу до глибокого навчання в спільноті комп'ютерного зору. У зв'язку з цим у роботі [7] запропонували глибоку нейронну мережу, яка використовує дескриптори зображень (яскравість, ознаки DAISY [19] та семантичні ознаки) як вхідні дані. У 2016 році Іїзука та ін. [8] запропонували метод, заснований на використанні ознак глобального та середнього рівнів для кодування зображень та їх розфарбовування.

Розроблена модель базується на їхньому підході [8] та також служить для валідації. Однак ми вводимо в рівняння попередньо навчену модель. Варто зазначити, що подібні підходи також були представлені останнім часом. Наприклад, у роботі [14] запропонували мультимодальну схему, де кожному пікселю надавалося значення ймовірності для кожного можливого кольору.

Ще один цікавий підхід був розроблений Ларссоном та ін. [11], в якому повністю згорткова версія VGG-16 [20] з відкинутим шаром класифікації була використана для побудови розподілу ймовірності кольору для кожного пікселя. Нещодавно Чжан та ін. [21] представили комплексний підхід ЗНМ, що включає користувацькі «підказки» в дусі методів на основі малюнків, забезпечуючи систему рекомендацій кольорів для допомоги початківцям та

стверджуючи, що дозволив використовувати свою систему розфарбовування в режимі реального часу.

Таким чином, аналогом до розробленого модуля, оберемо програму розфарбовування зображень по роботі [8]. Її недоліком є не дуже висока якість розфарбовування чорно-білих зображень. Так, сприйняття користувачами розфарбованих зображень як справжніх досягає у них у середньому 45,87%. Тобто треба підвищувати цей показник. Звідси випливає і мета даної бакалаврської роботи - підвищення якості розфарбовування чорно-білих зображень.

1.4 Висновок до розділу 1

У розділі описано детальну постановку задачі розфарбовування чорно-білих зображень. Також розглядаються різні методи розв'язання задачі розфарбовування зображень і оцінюються їх переваги та недоліки. Розроблюваний метод відноситься до автоматичних методів мультимодального розфарбовування зображень у градаціях сірого. Було обрано напрямок досліджень, що базується на згорткових нейронних мережах типу енкодер-декодер, однак буде включати попередньо навчену додаткову модель згорткової нейромережі. Крім цього, було обгрунтовано вибір аналогу до розроблюваного модуля розфарбовування зображень.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ РОЗФАРБОВУВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

2.1 Метод розфарбовування зображень на основі згорткової нейронної мережі

Чорно-білі зображення можуть бути представлені сіткою пікселів. Кожен піксель має значення, яке відповідає його яскравості. Значення варіюються від 0 до 255, від чорного до білого (рис. 2.1).

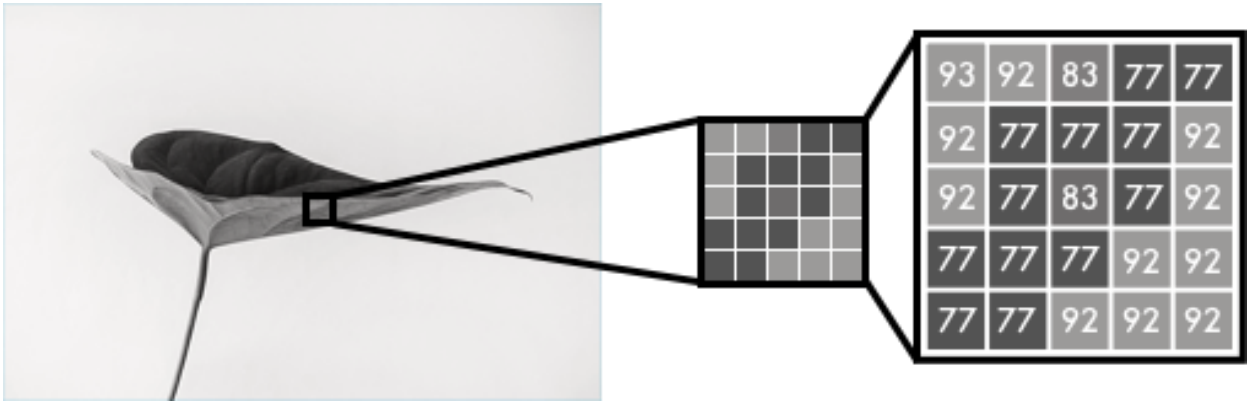


Рисунок 2.1 – Значення яскравостей чорно-білих зображень (варіюються від 0 до 255)

Кольорові зображення складаються з трьох шарів: червоного, зеленого та синього. Уявімо, що ми розділили зелений листок на білому тлі на три канали. Інтуїтивно можна подумати, що рослина присутня лише в зеленому шарі. Але, як видно з рис. 2.2 нижче, листок присутній у всіх трьох каналах. Шари визначають не тільки колір, але й яскравість.

Для досягнення білого кольору, наприклад, потрібно рівномірний розподіл усіх кольорів. Додаючи однакову кількість червоного і синього, зелений стає яскравішим. Таким чином, кольорове зображення кодує колір і контраст за допомогою трьох шарів (рис. 2.3)

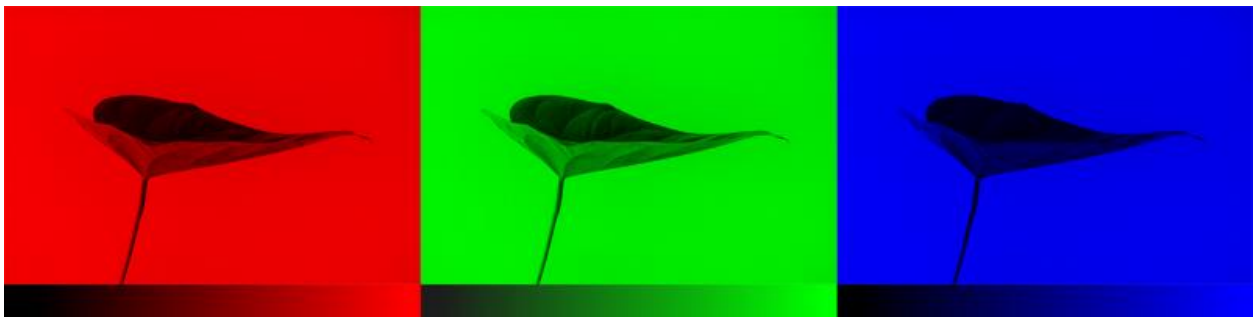


Рисунок 2.2 – Представлення зображення зеленого листка трьома каналами R, G, B

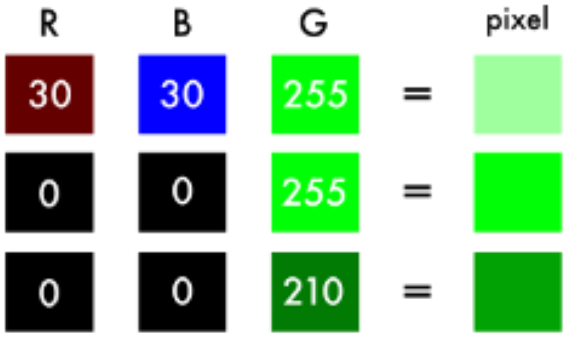


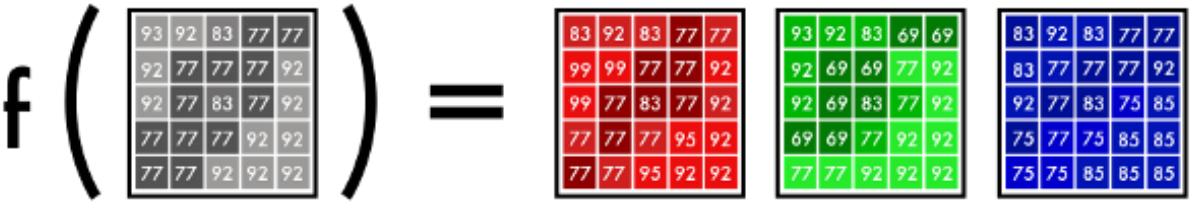
Рисунок 2.3 – Приклад яскравості зеленого кольору при різних значеннях яскравості у каналах R, G, B

Як і чорно-білі зображення, кожен шар у кольоровому зображенні має значення від 0 до 255. Значення 0 означає, що в цьому шарі немає кольору. Якщо значення дорівнює 0 для всіх каналів кольору, то піксель зображення є чорним.

Будь-яка нейронна мережа створює зв’язок між вхідним значенням і вихідним значенням. Щоб бути більш точним у нашій задачі розфарбовування, мережа має знайти ознаки, які пов’язують зображення у відтінках сірого з кольоровими.

Загалом, ми шукаємо функції, які пов’язують сітку значень градацій сірого з трьома кольоровими сітками (рис. 2.4).

На рис. 2.4 f() – це нейронна мережа, [B&W] – чорно-білий вхід, а [R], [G], [B] – три колірних канали виходу.



B&W R G B

Рисунок 2.4 – Формалізація задачі розфарбовування сірого зображення у випадку застосування колірного простору RGB

Ми розглядаємо зображення розміром H × W у колірному просторі CIE L*a*b*. [22]. Починаючи з компонента яскравості $\mathbf{X_L} \in \mathbb{R}^{H \times W \times 1}$, мета моделі полягає в оцінці решти компонентів для створення повнокольорової версії $\tilde{\mathbf{x}} \in \mathbb{R}^{H \times W \times 3}$. Коротко кажучи, припускаємо, що існує відображення F, таке що:

$$F : \mathbf{X_L} \rightarrow (\tilde{\mathbf{X}}\,\mathbf{a}, \tilde{\mathbf{X}}\,\mathbf{b}), \qquad (2.1)$$

де $\tilde{\mathbf{X}}\,\mathbf{a}, \tilde{\mathbf{X}}\,\mathbf{b}$ - це компоненти a*, b* реконструйованого зображення, які в поєднанні з вхідними даними дають оцінене кольорове зображення $\tilde{\mathbf{X}} = (\mathbf{X_L}, \tilde{\mathbf{X}}\mathbf{a}, \tilde{\mathbf{X}}\mathbf{b})$.

Щоб бути незалежними від розміру вхідного зображення, архітектуру повністю потрібно базувати на згорткових нейронних мережах (ЗНМ), які останнім часом широко вивчалися та застосовувалися в літературі [23].

Згорткова нейронна мережа (CNN) – це клас штучних нейронних мереж, який став домінуючим у різних завданнях комп’ютерного зору і привертає увагу в різних сферах. CNN призначена для автоматичного та адаптивного вивчення просторових ієрархій ознак через зворотне поширення помилки, використовуючи кілька основних компонентів, таких як згорткові шари, шари підбору (пулінгу) та повнозв’язані шари.

CNN є математичною конструкцією, яка зазвичай складається з трьох основних типів шарів (або будівельних блоків):

– Згортковий шар (convolution layer)
– Шар пулінгу (pooling layer)
– Повнозв’язаний шар (fully connected layer)

Перші два – згортковий і пулінг-шари – виконують вилучення ознак, тоді як третій, повнозв’язаний шар, перетворює отримані ознаки на кінцевий вихідний результат, наприклад, класифікацію.

Згортковий шар відіграє ключову роль у CNN. Він складається з набору математичних операцій, таких як згортка – спеціалізована форма лінійної операції. У цифрових зображеннях значення пікселів зберігаються у двовимірній (2D) сітці, тобто у вигляді масиву чисел (Рис. 2.5). Маленька сітка параметрів, яка називається ядром (kernel), виконує функцію оптимізованого екстрактора ознак і застосовується до кожної позиції зображення. Завдяки цьому CNN є надзвичайно ефективною для обробки зображень, оскільки однакові ознаки можуть зустрічатися в будь-якій частині зображення.

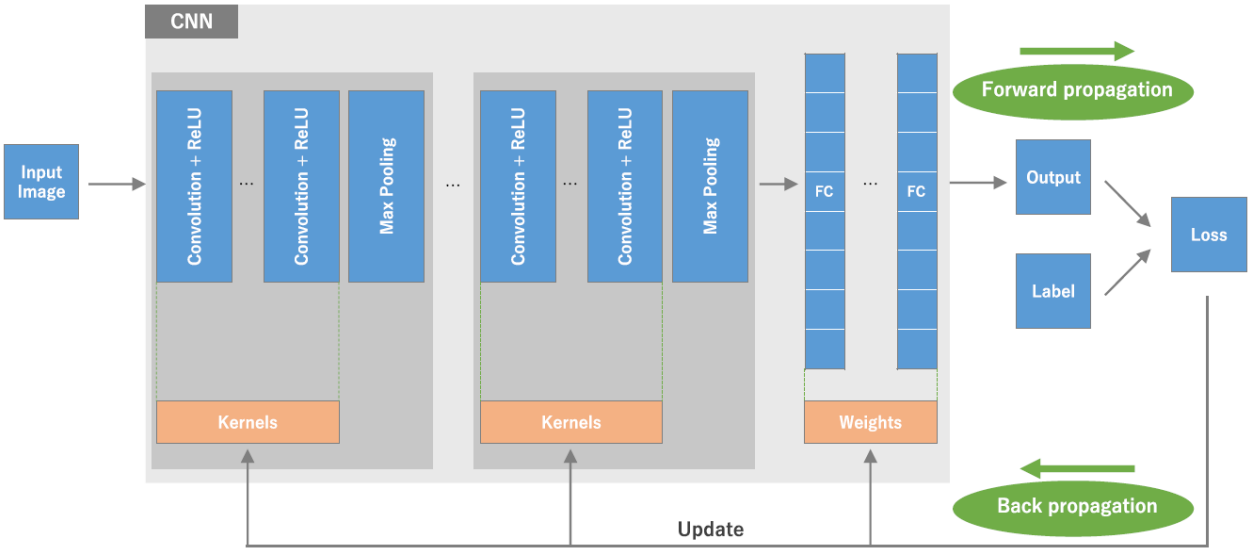


Рисунок 2.5 – Огляд архітектури згорткової нейронної мережі (CNN) та процесу навчання

Оскільки один шар передає свої результати наступному, виділені ознаки можуть поступово ставати дедалі складнішими. Процес оптимізації параметрів, таких як ядра, називається тренуванням, і він спрямований на мінімізацію різниці між вихідними результатами та реальними мітками (ground truth) за допомогою алгоритмів оптимізації, таких як зворотне поширення помилки (backpropagation) та градієнтний спуск (gradient descent).

CNN відрізняється від інших методів. У більшості сучасних досліджень використовуються методи вилучення ознак вручну, наприклад, аналіз текстури, за яким слідує класифікація за допомогою традиційних методів машинного навчання, таких як випадковий ліс (random forest) та метод опорних векторів (support vector machines).

Однак CNN суттєво відрізняється від цих методів:

1. CNN не вимагає ручного вилучення ознак.
2. CNN-архітектури не обов’язково потребують сегментації об’єктів експертами.
3. CNN набагато більш вимоглива до даних, оскільки містить мільйони параметрів, що піддаються навчанню, через що вона є значно більш обчислювально витратною. Тому для навчання моделей CNN зазвичай використовують графічні процесори (GPU).

Продуктивність моделі при використанні певних ядер (kernels) і ваг (weights) оцінюється за допомогою функції втрат (loss function) у процесі прямого поширення (forward propagation) на навчальному наборі даних. Навчаються параметри моделі (ядра та ваги) шляхом зворотного поширення помилки (backpropagation) із використанням алгоритму градієнтного спуску (gradient descent optimization algorithm). Функція активації ReLU (Rectified Linear Unit) використовується для внесення нелінійності в модель.

Основні будівельні блоки архітектури CNN.

Архітектура CNN включає кілька основних будівельних блоків, таких як згорткові шари (convolution layers), шари пулінгу (pooling layers) та повнозв’язані шари (fully connected layers). Типова архітектура складається з

повторюваних стеків із декількох згорткових шарів і шару пулінгу, за якими слідує один або кілька повнозв'язаних шарів. Процес, у якому вхідні дані проходять через ці шари і перетворюються на вихідні значення, називається прямим поширенням (forward propagation) (Рис. 2.5). Хоча операції згортки та пулінгу, описані в цьому розділі, стосуються 2D-CNN, аналогічні операції також можуть виконуватися для тривимірних (3D)-CNN.

Згортковий шар є фундаментальним компонентом архітектури CNN, що виконує вилучення ознак (feature extraction). Він зазвичай складається з комбінації лінійних та нелінійних операцій, таких як операція згортки (convolution operation) та функція активації (activation function).

Конволюція – це спеціалізований тип лінійної операції, що використовується для витягнення ознак, де невеликий масив чисел, який називається ядром, застосовується до вхідних даних, які є масивом чисел, званим тензором. Виконується покроковий добуток між кожним елементом ядра та елементом вхідного тензора на кожній позиції тензора, і результати додаються, щоб отримати вихідне значення на відповідній позиції вихідного тензора, що називається карти ознак (рис. 2.6a–c). Цю процедуру повторюють, застосовуючи кілька ядер для формування довільної кількості карт ознак, що представляють різні характеристики вхідних тензорів; таким чином, різні ядра можна вважати різними засобами витягнення ознак. Два основні гіперпараметри, які визначають операцію конволюції, – це розмір і кількість ядер. Перший зазвичай дорівнює 3 × 3, але інколи може бути 5 × 5 або 7 × 7. Другий є довільним і визначає глибину вихідних карт ознак.

Описана вище операція конволюції не дозволяє центру кожного ядра перекривати найзовнішній елемент вхідного тензора та зменшує висоту і ширину вихідної карти ознак порівняно з вхідним тензором. Паддінг, зазвичай нульовий паддінг, є технікою для вирішення цієї проблеми, де ряди та стовпці нулів додаються з кожного боку вхідного тензора, щоб центр ядра знаходився на найзовнішньому елементі і щоб зберігати однакові розміри в площині під час виконання операції конволюції. Сучасні архітектури CNN зазвичай

використовують нульовий паддінг для збереження розмірів площини, щоб можна було застосовувати більше шарів. Без нульового паддінгу кожна наступна карта ознак ставала б меншою після операції конволюції.

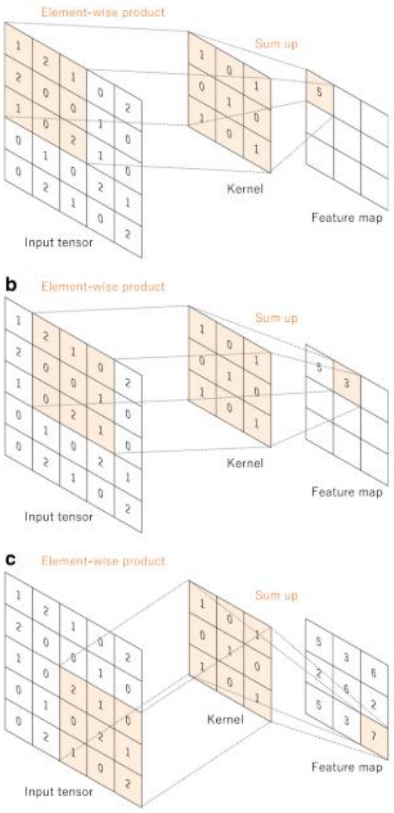


Рисунок 2.6 - Приклад операції згортки з ядром розміром 3 × 3, без доповнення (padding) та кроком 1.

Відстань між двома послідовними позиціями ядра називається кроком, що також визначає операцію конволюції. Зазвичай вибір кроку дорівнює 1, однак інколи застосовують крок більше 1 для досягнення зменшення розміру карт ознак. Альтернативною технікою для виконання зменшення розміру є операція пулингу.

Ключовою характеристикою операції конволюції є спільне використання ваг: ядра використовуються на всіх позиціях зображення. Спільне використання ваг створює наступні характеристики операцій конволюції: 1) дозволяє локальним патернам ознак, що витягуються ядрами,

бути інваріантними до їх переміщення по всіх позиціях зображення та виявляти навчені локальні патерни, 2) навчання просторових ієрархій патернів ознак через зменшення розміру в поєднанні з операцією пулингу, що призводить до захоплення все більшого поля зору, та 3) підвищення ефективності моделі шляхом зменшення кількості параметрів, які необхідно навчити, порівняно з повністю з'єднаними нейронними мережами.

Процес навчання моделі CNN стосовно шару конволюції полягає в ідентифікації ядер, які найкраще працюють для певного завдання на основі наданого навчального набору даних. Ядра – це єдині параметри, які автоматично навчаються під час процесу навчання в шарі конволюції; з іншого боку, розмір ядер, кількість ядер, паддінг та крок є гіперпараметрами, які потрібно встановити до початку навчання.

Коротко, згортковий шар – це набір невеликих навчаних фільтрів, які відповідають специфічним локальним шаблонам у вхідному зображенні. Шари, ближчі до входу, шукають прості шаблони, такі як контури, тоді як шари, ближчі до виходу, витягують більш складні ознаки [24].

Як уже зазначалося, обираємо колірний простір CIE Lab* для представлення вхідних зображень, оскільки він відокремлює колірні характеристики від яскравості, яка містить основні ознаки зображення [25, 26]. Поєднання яскравості з передбаченими колірними компонентами забезпечує високий рівень деталізації на кінцевому реконструйованому зображенні.

Якщо зображення представлено у колірному просторі RGB, то ми використаємо алгоритм для зміни каналів кольору з RGB на Lab. L позначає освітленість, a і b позначають колірні спектри зелено-червоного і синьо-жовтого.

Як ви можете бачити нижче на рис. 2.7, зображення, закодоване в Lab, має один шар для відтінків сірого та містить три кольорові шари об'єднані у два. Це означає, що ми можемо використовувати вихідне зображення у градаціях сірого в нашому остаточному прогнозі. Крім того, у нас є лише два канали для прогнозування.

Рисунок 2.7 – Приклад зображення, закодованого в Lab*

Науковий факт – 94% клітин наших очей визначають яскравість. Це залишає лише 6% наших рецепторів для сприйняття кольорів. Як ви можете бачити на рис. 2.7, зображення у відтінках сірого набагато чіткіше, ніж кольорові шари. Це ще одна причина залишити зображення в градаціях сірого в нашому остаточному прогнозі.

Таким чином, ми подаємо на вхідний шар нейромережі зображення у відтінках сірого, а хочемо, щоб нейромережа передбачила два кольорові шари a і b у Lab* (рис.2.8). Щоб створити остаточне кольорове зображення, ми додамо зображення L (відтінки сірого), яке ми використовували на вході. Результатом буде створення зображення в Lab*.

Щоб перетворити один шар на два, ми використовуємо згорткові фільтри. Кожен фільтр визначає те, що ми бачимо на зображенні. Вони можуть виділяти або видаляти щось, щоб витягнути інформацію із зображення. Мережа може або створити нове зображення з фільтра, або об’єднати кілька фільтрів в одне зображення.

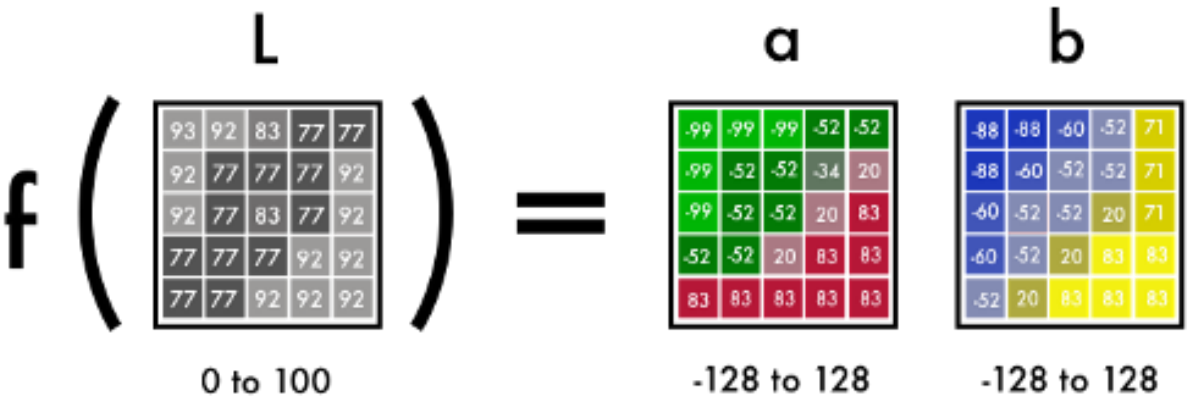


Рисунок 2.8 – Формалізація задачі розфарбовування сірого зображення у випадку застосування колірного простору Lab*

Для згорткової нейронної мережі кожен фільтр автоматично налаштовується в процесі навчання, щоб допомогти отримати очікуваний результат. Згорткова нейромережа складається із сотень фільтрів і до виходу нейромережі їх кількість звужується до двох шарів, a і b .

2.2 Архітектура нейронної мережі

Архітектура використовуваної ЗНМ базується на роботі [8]: за заданою компонентою яскравості зображення модель оцінює його компоненти a*b* і поєднує їх із вхідними даними для отримання остаточної оцінки кольорового зображення. Замість того, щоб навчати гілку вилучення ознак з нуля, було використано нейромережу Inception-ResNet-v2 (надалі називану Inception) і отримуємо вбудоване представлення сірого зображення з її останнього шару. Використовувана архітектура нейромережі показана на рис. 2.9

Мережа логічно поділена на чотири основні компоненти. Компоненти кодування та вилучення ознак отримують ознаки середнього та високого рівня відповідно, які потім об’єднуються у шарі злиття. Нарешті, декодер використовує ці ознаки для оцінки вихідних даних. Таблиця 2.1 детальніше описує шари мережі, в якій ліворуч показана мережа кодера, посередині - мережа злиття, праворуч - мережа декодера. Кожен згортковий шар

використовує функцію активації ReLU, за винятком останнього, який застосовує функцію гіперболічного тангенсу. Гілка вилучення ознак має таку ж архітектуру, як Inception-ResNet-v2, за винятком останнього шару softmax.

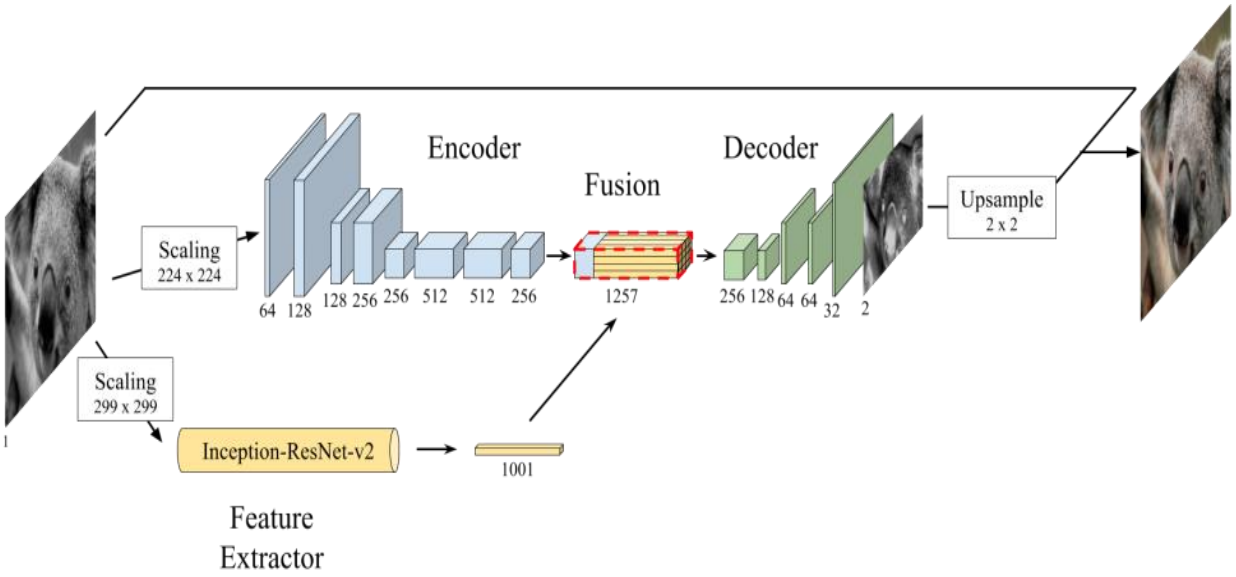


Рисунок 2.9 – Архітектура використовуваної згорткової нейромережі

Таблиця 2.1 - Структура шарів використовуваної згорткової нейромережі

Шар (Layer)	Ядро (Kernel)	Крок (Stride)	Шар (Layer)	Ядро (Kernel)	Крок (Stride)	Шар (Layer)	Ядро (Kernel)	Крок (Stride)
conv	64×(3×3)	2 × 2	fusion	-	-	conv	128×(3×3)	1 × 1
conv	128×(3×3)	1 × 1	conv	256×(1×1)	1 × 1	upsamp	-	-
conv	128×(3×3)	2 × 2				conv	64×(3×3)	1 × 1
conv	256×(3×3)	1 × 1				conv	64×(3×3)	1 × 1
conv	256×(3×3)	2 × 2				upsamp	-	-
conv	512×(3×3)	1 × 1				conv	32×(3×3)	1 × 1
conv	512×(3×3)	1 × 1				conv	2×(3×3)	1 × 1
conv	256×(3×3)	1 × 1				upsamp	-	-

Попередня обробка.

Для забезпечення правильного навчання значення пікселів усіх трьох компонентів зображення центровано та масштабується (відповідно до їхніх діапазонів [27]), щоб отримати значення в інтервалі [−1, 1].

Енкодер.

Енкодер обробляє сірі зображення розміром H × W і видає представлення ознак розміром H/8 × W/8 × 512. Для цього він використовує 8 згорткових шарів із ядрами 3 × 3. Застосовується заповнення (padding), щоб зберегти розмір вхідних даних шару. Крім того, перший, третій і п’ятий шари використовують крок (stride) 2, що призводить до зменшення розміру їхнього виходу вдвічі, а отже, зменшує кількість необхідних обчислень [28].

Вилучення ознак.

Ознаки високого рівня, наприклад, «підводна сцена» або «інтер’єр», передають інформацію про зображення, яка може бути використана в процесі кольоризації. Для вилучення вбудованого представлення зображення було використано попередньо навчену модель Inception. Спочатку вхідне зображення масштабується до розміру 299 × 299. Далі дублюється зображення саме на себе, щоб отримати триканальне зображення (як показано на рис. 2.10), щоб відповідати вимогам розмірності Inception. Потім отримане зображення подається в мережу та вилучається вихід останнього шару перед функцією softmax. Це дає вбудоване представлення розміром 1001 × 1 × 1.

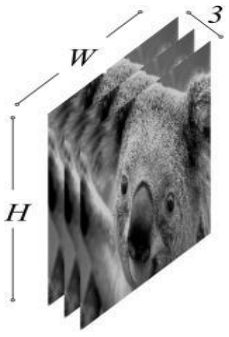


Рисунок 2.10 – Трьохкратне накладання компоненти яскравості

Злиття.

Шар злиття бере вектор ознак від Inception, повторює його $HW/8^2$ разів і приєднує до об’єму ознак, отриманого від кодера, вздовж осі глибини. Цей метод був представлений у [8] і проілюстрований на рис. 2.11. Такий підхід дозволяє отримати єдиний об’єм із закодованим зображенням і ознаками середнього рівня форми H/8 × H/8 × 1257. Повторюючи вектор ознак і конкатенуючи його кілька разів, ми забезпечуємо рівномірний розподіл семантичної інформації, що передається вектором ознак, по всіх просторових областях зображення. Крім того, це рішення також стійке до довільних розмірів вхідного зображення, що підвищує гнучкість моделі. Нарешті, застосовується 256 згорткових ядер розміром 1 × 1, що в результаті генерує об’єм ознак розміром H/8 × W/8 × 256.

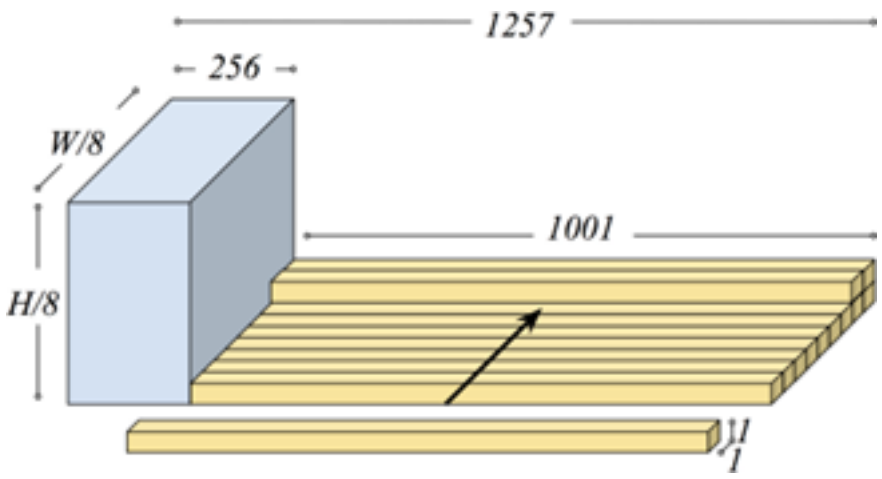


Рисунок 2.11 – Об’єднання вбудованого представлення Inception із виходом згорткових шарів кодера

Декодер.

Нарешті, декодер бере цей об’єм розміром H/8 × W/8 × 256 і застосовує серію згорткових шарів і шарів апсемплінгу (масштабування вгору), щоб отримати кінцевий шар розміром H × W × 2. Масштабування вгору виконується за допомогою базового методу найближчого сусіда, так що висота і ширина виходу вдвічі більші за вхідні.

2.3 Навчання згорткової нейронної мережі для розфарбовування зображень

Оптимальні параметри моделі знаходять шляхом мінімізації цільової функції, визначеної над передбачуваним виходом і цільовим виходом. З метою кількісної оцінки втрати моделі, ми використовуємо середню квадратичну похибку (MSE - Mean Square Error) між передбаченими нейромережею кольорами пікселів у просторі a*b* та їх реальними значеннями. Для картинки **X**, MSE задається за формулою:

$$C(X,\theta) = \frac{1}{2HW} \sum_{k\in\{a,b\}} \sum_{i=1}^{H} \sum_{j=1}^{W} (X_{k,i,j} - \tilde{X}_{k,i,j})^2, \tag{2.2}$$

де θ представляє всі параметри моделі, $X_{k,i,j}$ та $\tilde{X}_{k,i,j}$ позначають значення ij-го пікселя k-ї компоненти цільового та реконструйованого зображення відповідно. Це можна легко розширити на пакет B, усереднюючи втрати між усіма зображеннями в пакеті, тобто $1/|\mathcal{B}| \sum_{\mathbf{X}\in\mathcal{B}} C(\mathbf{X},\theta)$.

Під час навчання ці втрати зворотно поширюються для оновлення параметрів моделі θ за допомогою оптимізатора Adam [29] з початковою швидкістю навчання η = 0.001.

```
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)
```

Метод навчання Adam - це адаптивний алгоритм градієнтного спуску для оптимізації функції втрат в штучних нейронних мережах. Він являє собою комбінацією інших двох методів оптимізації - Momentum і RMSprop.

Оптимізатор Adam використовує експоненційне середнє зважене градієнтів та квадратів градієнтів для визначення коефіцієнтів оптимізації. Він

використовує також bias-корекцію для того, щоб уникнути невірного зміщення на початковому етапі.

Формула для оновлення параметрів за методом Adam має такий вид:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$
$$\theta_t = \theta_{t-1} - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon},$$

де m_t та v_t - експоненційно зважені середні відповідно градієнтів та квадратів градієнтів, g_t – градієнт від функції втрат у час t, α – швидкість навчання, β_1, β_2 - змінні параметри плавального середнього, $\widehat{m_t}, \widehat{v_t}$ - змінені значення m_t та v_t , які враховують bias-колекцію, а ϵ - вельми мала величина (для уникнення ділення на нуль).

Оптимізатор Adam добре працює з тими нейронними мережами, які мають велике число параметрів і складні вирази функції втрат.

Під час навчання ми встановлюємо фіксований розмір вхідного зображення, щоб уможливити пакетну обробку.

Параметри процесу навчання наведено в такому фрагменті:

```
# PARAMETERS
run_id = args.run_id
val_number_of_images = 10
batch_size = 100
learning_rate = 0.001
```

Наступний фрагмент показує, що для кількісної оцінки втрат моделі, ми використовуємо середню квадратичну похибку (MSE - Mean Square Error)

```
    def loss_with_metrics(img_ab_out, img_ab_true, name=""):
        # Loss is mean square erros
        cost   =   tf.reduce_mean(tf.squared_difference(img_ab_out,   img_ab_true),
name="mse")
        # Metrics for tensorboard
        summary = tf.summary.scalar("cost " + name, cost)
        return cost, summary
```

Також під час навчання записуються певні чек поінти (контрольні точки) з тим, щоб потім можна було аналізувати процес навчання:

```
    def checkpointing_system(dir_checkpoints):
        # Add ops to save and restore all the variables.
        saver = tf.train.Saver()
        latest_checkpoint = tf.train.latest_checkpoint(dir_checkpoints)
        checkpoint_paths = join(dir_checkpoints, "weights")
        return saver, checkpoint_paths, latest_checkpoint
```

Лістінг класу train.py наведено у додатку Б.

2.4 Розробка алгоритму роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі

Розроблювана програма повинна перетворювати чорно-біле зображення у кольорове (розфарбовувати чорно-біле зображення). Зокрема, вона має робити наступне:

1. Завантажує попередньо навчану нейронну мережу Inception-ResNet-v2, яка буде використовуватись для вилучення ознак вхідного зображення.
2. Завантажує датасет зображень, сформований на основі ImageNet.
3. Змінює розміри зображень до 224x224 для подачі в енкодер і до 299x299 для подачі в нейронну мережу Inception-ResNet-v2.

4. Виконує попередню обробку зображень завантаженого датасету, яка полягає у їх нормалізації до діапазону [-1,1] та отриманні ембеддінгів.

5. Виконує навчання згорткової нейромережі задачі розфарбовування зображень на основі навчальної вибірки протягом заданої кількості епох.

6. Оцінює продуктивність навченої нейромережі шляхом підрахунку середньоквадратичної помилки розфарбовування на зображеннях тестової вибірки.

7. При потребі розфарбовує невідоме мережі зображення користувача, яке записано у відповідну папку

Згідно з метою роботи та постановкою завдання було розроблено алгоритм програмного модуля розфарбовування зображень на основі згорткової нейронної мережі, представлений на рис. 2.12.

Першим кроком в програмному модулі розфарбовування зображень на основі згорткової нейронної мережі буде завантаження необхідних бібліотек (блок 1), зокрема TenzorFlow та Keras.

Другим кроком в буде завантаження попередньо навченої нейронної мережі Inception-ResNet-v2, яка буде використовуватись для вилучення ознак вхідного зображення (блок 2).

Потім здійснюється завантаження набору даних зображень, сформованого на основі ImageNet (блок 3).

Після цього виконується resizing, тобто зміна розмірів зображень (блок 4) до 224x224 для подачі в енкодер і до 299x299 для подачі в нейронну мережу Inception-ResNet-v2

Далі виконується попередня обробка зображень завантаженого датасету, яка полягає нормалізації їх значень яскравості до діапазону [-1,1] та отриманні ембеддінгів (блок 5).

Потім відбувається процес навчання згорткової нейромережі (блок 7) задачі розфарбовування зображень на основі навчальної вибірки протягом заданої кількості епох. Кількість епох контролюється умовним блоком 6.

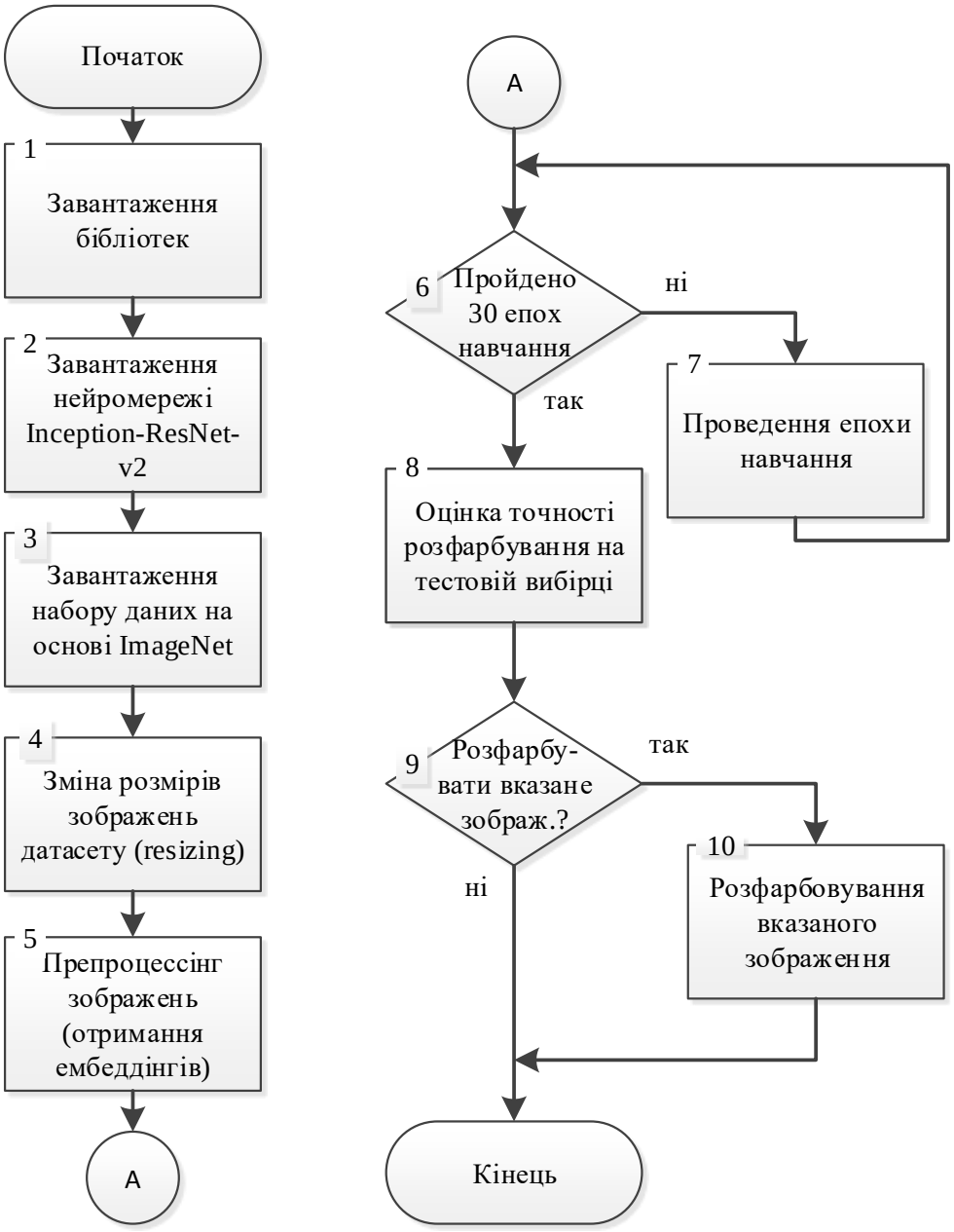


Рисунок 2.12 – Схема алгоритму роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі

Після цього оцінюється продуктивність навченої нейромережі шляхом підрахунку середньоквадратичної помилки розфарбовування на зображеннях із тестової вибірки (блок 8).

Далі при потребі розфарбувати зображення користувача (визначає блок 9), яке записано у відповідну папку, програма розфарбовує його (блок 10) і записує у папку вихідних зображень.

На цьому робота алгоритму завершується. Згідно цього алгоритму було проведено програмну реалізацію модуля (п. 3.3).

2.5 Висновок до розділу 2

У розділі було розроблено метод розфарбовування зображень на основі згорткової нейронної мережі з використанням колірного простору Lab*. Було розроблено архітектуру згорткової нейронної мережі, яка лежить в основі розробленого методу. Архітектура нейромережі побудована по схемі енкодер-декодер та включає гілку вилучення ознак на основі попередньо навченої нейронної мережі Inception-ResNet-v2. Розроблено послідовність та параметри процесу навчання згорткової нейронної мережі для розфарбовування зображень. Розроблено алгоритм роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ РОЗФАРБОВУВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

3.1 Обґрунтування вибору мови програмування та спеціалізованих бібліотек

Для реалізації програмного модуля розфарбовування зображень на основі згорткової нейронної мережі було обрано мову програмування Python [30].

Python – це об'єктно-орієнтована мова програмування високого рівня із динамічною типізацією. Вона має розвинені структури даних та динамічну семантику. Це робить її придатною для швидкої розробки програм. Python підтримує модулі та пакети модулів, що сприяє широкому повторному використанню коду. Інтерпретатор мови Python та його множинні стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх платформах. У мові Python підтримується кілька парадигм програмування: об'єктно-орієнтована, аспектно-орієнтована, функціональна та процедурна.

Таким чином, Python – це високорівнева мова програмування, яка здобула авторитет у розробників своєю простотою, легкою читабельністю та зручністю використання. Завдяки простому синтаксису, мова Python є чудовим варіантом для початківців, а також вона надає потужні функції, які задовольнять потреби також і досвідчених розробників.

Головні переваги мови Python:

– Читабельність коду: Синтаксис Python легко читається та легко розуміється. Код на Python виглядає природно, майже як псевдокод, що спрощує процес розробки, підвищує рівень взаєморозуміння та ефективність співпраці між розробниками.

– Легкість у вивченні: Новачки швидко засвоюють основи і починають генерувати код. Для допомоги початківцям у вільному доступі є багато навчальних матеріалів і ресурсів.

– Розширюваність: Мова Python може розширювати свої можливості на основі зовнішніх бібліотек та модулів, що значно розширює функціональність мови. Python має велику кількість зовнішніх пакетів, які допомагають у вирішенні різних задач.

– Кросплатформеність: Мова Python підтримується багатьма операційними системами, зокрема Windows, macOS та Linux. Це дозволяє створювати програми, що працюють на різних платформах без необхідності суттєво змінювати код.

– Багатофункціональність: Мова Python підтримує різні типи програмування, у тому числі з об'єктно-орієнтоване програмування, процедурне програмування та функціональне програмування. Це надає розробникам можливість вибору найкращого підходу при розв`язанні конкретних завдань.

– Активна професійна мережа: Мова Python має велику та активну мережу розробників, які внесли та вносять суттєвий внесок у збагачення мови та створення нових корисних інструментів. Тобто завжди можна отримати підтримку, допомогу та величезну кількість ресурсів для розробників на мові Python.

Мова Python застосовується для різноманітних задач, зокрема до веб-розробки, наукових досліджень, аналізу даних, штучного інтелекту, робототехніки та багатьох інших. Вона є потужним і гнучким інструментом для розробників, що прагнуть простоти, поєднаної із широкою функціональністю.

Оскільки ми використовуємо нейронні мережі, то для цієї роботи знадобляться бібліотеки TenzorFlow [31] та Keras [32].

У листопаді 2015 року Google випустила свій фреймворк з відкритим кодом для машинного навчання під назвою TensorFlow. Він підтримує глибоке навчання, нейронні мережі та загальні числові обчислення на процесорах, графічних процесорах та кластерах графічних процесорів. Однією з найбільших переваг TensorFlow є спільнота розробників, фахівців з обробки даних та інженерів даних з відкритим кодом, які роблять внесок у його репозиторій. Поточну версію TensorFlow можна знайти на GitHub разом із примітками до випуску. TensorFlow є найпопулярнішим двигуном штучного інтелекту, який використовується сьогодні.

TensorFlow – це бібліотека з відкритим кодом для числових обчислень, великомасштабного машинного навчання, глибокого навчання та інших статистичних і прогнозних аналітичних завдань. Цей тип технології дозволяє розробникам швидше та простіше впроваджувати моделі машинного навчання, оскільки він допомагає в процесі отримання даних, надання прогнозів у великих масштабах та уточнення майбутніх результатів.

TensorFlow може навчати та запускати глибокі нейронні мережі для таких речей, як класифікація рукописних цифр, розпізнавання зображень, вбудовування слів та обробка природної мови (NLP). Код, що міститься в його програмних бібліотеках, можна додати до будь-якої програми, щоб допомогти їй навчитися цим завданням.

Програми TensorFlow можуть працювати як на звичайних процесорах (центральних процесорах), так і на графічних процесорах (графічних процесорах високої продуктивності). Оскільки TensorFlow був розроблений Google, він також працює на власних тензорних процесорах (TPU) компанії, які спеціально розроблені для пришвидшення завдань TensorFlow.

Хоча TensorFlow використовує Python як фронтенд API для створення додатків за допомогою фреймворку, він насправді має обгортки кількома мовами, включаючи C++ та Java. Це означає, що ви можете швидко навчати та розгортати свою модель машинного навчання, незалежно від мови чи платформи, яку ви використовуєте.

TensorFlow розроблено для оптимізації процесу розробки та виконання передових аналітичних програм для таких користувачів, як фахівці з обробки даних, статистики та прогнозні моделісти.

Компанії різних типів та розмірів широко використовують цей фреймворк для автоматизації процесів та розробки нових систем, і він особливо корисний для дуже масштабних програм з паралельною обробкою, таких як нейронні мережі. Він також використовувався в експериментах та тестах для безпілотних транспортних засобів.

Як і слід було очікувати, материнська компанія TensorFlow, Google, також використовує його для внутрішніх операцій, таких як покращення можливостей пошуку інформації своєї пошукової системи та забезпечення роботи додатків для автоматичного генерування відповідей на електронні листи, класифікації зображень та оптичного розпізнавання символів.

Однією з переваг TensorFlow є те, що він забезпечує абстракцію, що означає, що розробники можуть зосередитися на загальній логіці програми, поки фреймворк піклується про дрібні деталі. Це також зручно для розробників, яким потрібно налагоджувати та отримувати самоаналіз у додатки TensorFlow.

Пакет візуалізації TensorBoard має інтерактивну веб-панель інструментів, яка дозволяє перевіряти та профілювати спосіб відображення графіків. Також існує режим швидкого виконання, який дозволяє оцінювати та змінювати кожну операцію з графом окремо та прозоро, замість того, щоб створювати весь граф як єдиний непрозорий об'єкт та оцінювати його все одразу.

Keras – відкрита нейромережева бібліотека, написана мовою програмування Python. Вона може працювати поверх TensorFlow, R, Microsoft Cognitive Toolkit, Theano та PlaidML. Її було спроектовано для уможливлення швидких експериментів з нейронними мережами глибокого навчання, та зосереджено на тому, щоб вона була зручною у використанні, модульною та розширюваною. Keras було створено як частину дослідницьких зусиль

проекту ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System).

2017 року команда TensorFlow Google вирішила підтримувати Keras в основній бібліотеці TensorFlow. Було пояснено, що Keras замислювався радше як інтерфейс, аніж як самостійна система машинного навчання.

3.2 Опис набору даних для навчання та тестування модуля

Не менш важливим, ніж сама архітектура нейромережі, є вибір набору даних. У більшості підходів до автоматичного перефарбування зображень наразі широко використовувався ImageNet [11, 14]. Окрім вражаючого розміру ImageNet (понад 14 000 000 зображень), детальна документація та вільний доступ роблять його привабливим для наших цілей. Набір даних складається з мільйонів зображень у широкому розмаїтті категорій. Зокрема, він базується на вузлах назв, що містяться в наборі даних WordNet. Для спрощення навчання та скорочення часу виконання використовується лише невелика підмножина приблизно з 60 000 зображень.

Зображення ImageNet мають різноманітні розміри, тому всі зображення в навчальному наборі масштабуються до 224 × 224 для входу в гілку кодування та до 299 × 299 для Inception. Кожне зображення розтягується або стискається за потреби, але його співвідношення сторін зберігається шляхом додавання білого заповнення, якщо це необхідно..

Тренування.

З приблизно 60 000 оригінальних зображень ми залишили 10% для використання як валідаційні дані під час навчання. Результати, представлені в цій роботі, отримані з цього валідаційного набору, тому мережа ніколи не мала можливості бачити ці зображення під час навчання. Оптимізатор Adam використовувався протягом приблизно 23 годин навчання.

Мережа навчалася та тестувалася з використанням прискорювача GPU для прискорення обчислень. Розмір пакету навчання становив 100, що унеможливило ризик переповнення пам’яті GPU.

Усі кроки підготовки даних є незалежними та зберігаються на диску, структура папок за замовчуванням (і рекомендована):

```
./data
├── fall11_urls.txt
├── imagenet1000_clsid_to_human.pkl
├── inception_resnet_v2_2016_08_30.ckpt
├── original
├── resized
└── tfrecords
```

Зображення IMAGENET.

Для завантаження зображень з ImageNet надається скрипт, який приймає на вхід файл, що містить URL-адреси зображень (по одній на рядок). Ми жодним чином не обмежені у використанні зображень з ImageNet, можна надати скрипту будь-який список URL-адрес, і він подбає про завантаження. Приклад :

```
# Photo by Anders Wideskott on Unsplash
https://unsplash.com/photos/_QnPkc4C6E4/download?force=true&w=640

# Photo by Matt Jones on Unsplash
https://unsplash.com/photos/9CPAjGVB378/download?force=true&w=640
```

Також, якщо вже є колекція зображень для навчання, їх треба помістити у папку `data/original` та перейти до наступного кроку.

Зміна розміру зображення для навчання.

Щоб мати можливість навчання пакетами, ми змінюємо розмір усіх зображень до 299x299. Для досягнення цього використовуємо наступний скрипт:

```
python -m koalarization.dataset.resize 'data/original' 'data/resized'
```

Конвертація в TFRecords.

Спочатку завантажуємо попередньо навчену модель Inception для вилучення ознак, а потім використовуємо скрипт `lab_batch` для обробки всіх зображень з папки зі зміненим розміром:

```
wget -O -
'http://download.tensorflow.org/models/inception_resnet_v2_2016_08_30.tar.gz'
| tar -xzv -C 'data'
python -m koalarization.dataset.lab_batch -c
'data/inception_resnet_v2_2016_08_30.ckpt' 'data/resized' 'data/tfrecords'
```

Якщо -c пропущено, скрипт самостійно завантажить контрольну точку, проте наполегливо рекомендується завантажувати її окремо.

Набір валідації.

Деякі записи Tfrecords вибрано для використання як набір валідації. Це робиться простим перейменуванням, наприклад:

```
mv 'data/tfrecord/lab_images_0.tfrecord' 'data/tfrecord/val_lab_images_0.tfrecord'
```

Щодо місця на диску, то з перших 200 посилань ми отримуємо 131 дійсне зображення, які у своєму початковому розмірі займають загалом 17 МБ, а потім 2,5 МБ після зміни розміру до 299x299.

Записи TFRecords.

Спочатку зображення зберігаються з використанням стиснення jpeg, що робить їхній розмір досить малим. З іншого боку, після зберігання в TFRecord

вони будуть просто у форматі необроблених байтів і займатимуть набагато більше місця.

Треба пам’ятати, що один приклад складається з:

1 зображення L (299x299x1 float32)

1 зображення a*b* (299x299x2 float32)

1 вбудовування (1001 float32)

1 байтовий рядок

Для економії місця можна використовувати один із варіантів стиснення TFRecord або стиснути файли після створення за допомогою команди типу:

```
RECORD='data/tfrecord/lab_images_0.tfrecord'
7z a -t7z -m0=lzma -mx=9 -mfb=64 -md=32m -ms=on "$RECORD.7z" "$RECORD"
```

3.3 Програмна реалізація модуля розфарбовування зображень

Основними класами програми є:

- network_definition.py (реалізує архітектуру згорткової нейронної мережі по рис. 2.9),
- fusion_layer.py (реалізує шар злиття згорткової нейронної мережі по рис. 2.9),
- train.py (реалізує навчання згорткової нейронної мережі по рис. 2.9),
- training_utils.py (обслуговує процес навчання згорткової нейронної мережі по рис. 2.9: організовує навчальний конвеєр, підраховує метрики, втрати, оцінює процес навчання, зберігає контрольні точки),
- evaluate.py (оцінює процес навчання).

Лістинги деяких класів наведено у додатку Б.

Проєкт базується на Python 3.6, для керування залежностями, що містяться в requirements.txt, рекомендується віртуальне середовище

```
python3.6 -m venv venv
source venv/bin/activate
pip install -e
```

Ще краще використовувати середовище Conda

```
conda create -y -n koalarization python=3.6
conda activate koalarization
pip install -e
```

Перед навчанням зображення з ImageNet потрібно завантажити, змінити їх розмір та обробити.

Вилучення вбудовування в режимі реального часу під час навчання може потенційно уповільнити процес, тому початкові вбудовування обчислюються заздалегідь, зберігаються разом із зображеннями та передаються безпосередньо до шару злиття.

Під час навчання для кожного зображення нам потрібні версії у градаціях сірого та кольорові, а також початкові вбудовування. Зберігання зображень та вбудовування як окремих файлів на диску (наприклад, jpeg/png та csv) вплине на продуктивність під час навчання, тому всі пари зображення-вбудовування зберігаються у двійковому форматі у великих безперервних TFRecords.

Навчання та оцінювання.

Перед навчанням переконайтеся, що дані папки містять:

- записи навчання як `tfrecords/lab_images_*.tfrecord`,
- записи валідації як `tfrecords/val_lab_images_*.tfrecord` (просто перейменуйте деякі записи навчання на валідаційні, але зробіть це перед будь-яким навчанням).

Скрипт навчання навчатиметься на всіх навчальних зображеннях, регулярно перевірятиме ваги та зберігатиме на диску деякі кольорові зображення з набору валідації.

Усі журнали навчання, метрики та контрольні точки зберігаються в `runs/run_id`.

```
python -m koalarization.train \
  --run-id 'run1' \
  --train-steps 100 \
  --val-every 20 \
  'data/tfrecords' 'runs/'
```

Скрипт оцінювання завантажить останню контрольну точку, розфарбує зображення із записів перевірки та збереже їх на диск. Наразі неможливо працювати зі звичайними файлами зображень (наприклад, jpeg або png), але зображення спочатку потрібно обробити як TFRecords.

```
python -m koalarization.evaluate \
  --run-id 'run1' \
  'data/tfrecords' 'runs/'
```

Як зазначалось, для розробки програмного коду модуля використовувались бібліотеки Keras і Tensorflow. Фрагмент коду для реалізації енкодеру наведено нижче:

```
def _build_encoder():
    model = Sequential(name="encoder")
    model.add(InputLayer(input_shape=(None, None, 1)))
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same", strides=2))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same", strides=2))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same", strides=2))
    model.add(Conv2D(512, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(512, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same"))
    return model
```

Із фрагменту коду видно, що він містить 8 згорткових шарів згідно табл. 2.1, активаційна функція – ReLU. Фрагмент коду для реалізації декодеру наведено нижче:

```
def _build_decoder(encoding_depth):
    model = Sequential(name="decoder")
    model.add(InputLayer(input_shape=(None, None, encoding_depth)))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same"))
    model.add(UpSampling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(UpSampling2D((2, 2)))
    model.add(Conv2D(32, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(2, (3, 3), activation="tanh", padding="same"))
    model.add(UpSampling2D((2, 2)))
    return model
```

Із цього фрагменту коду видно, що декодер містить крім вхідного шару 5 згорткових шарів та 3 шари апсемплінгу згідно табл. 2.1, активаційна функція – ReLU.

3.4 Висновок до розділу 3

У розділі було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек Keras та TensorFlow для програмної реалізації модуля розфарбовування зображень на основі згорткової нейронної мережі. Проведено аналіз набору даних зображень на основі відомої бази даних ImageNet, в який було відібрано 60000 зображень, з яких 54000 було взято як навчальні та 6000 як тестові. Описано основні етапи програмної реалізація та функціонування модуля розфарбовування зображень на основі згорткової нейронної мережі, що складається з завантаження набору даних, реалізації архітектури згорткової нейронної мережі, навчання згорткової нейронної мережі, оцінювання процесу навчання.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО МОДУЛЯ РОЗФАРБОВУВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

4.1 Тестування програмного модуля розфарбовування зображень на основі згорткової нейронної мережі

Результатом роботи є програмний застосунок, що виконує розфарбовування напівтонових (сірих) зображень. Для того щоб впевнитися, що програма працює правильно, було проведено її тестування.

Після навчання ми подали в нашу мережу кілька зображень. Результати виявилися досить хорошими для деяких зображень, створюючи майже фотореалістичні картинки. Однак через невеликий розмір нашого навчального набору наша мережа краще працює, коли присутні певні ознаки зображення. Наприклад, природні елементи, такі як море чи рослинність, здаються добре розпізнаними. Проте, специфічні об’єкти не завжди правильно забарвлюються. На рис. 4.1 показано результати для кількох прикладів, де наша мережа створює альтернативні кольорові оцінки.

Чорно-біле **З нейромережі** **Реальне**

Рисунок 4.1 – Результати розфарбовування для кількох прикладів.

На рис. 4.1 у першому рядку запропонований модуль здатний розпізнати зелену рослинність. Однак метелик не був забарвлений. Крім того, у прикладі в другому рядку ми бачимо, що мережа змінює колір одягу веслярів із зеленувато-жовтого на червоно-синій. Останній рядок показує приклад пейзажу, де наша модель створила фотореалістичне зображення.

На рис. 4.2 представлено згенеровані кольорові зображення за допомогою розробленого модуля та програми-аналога [8].

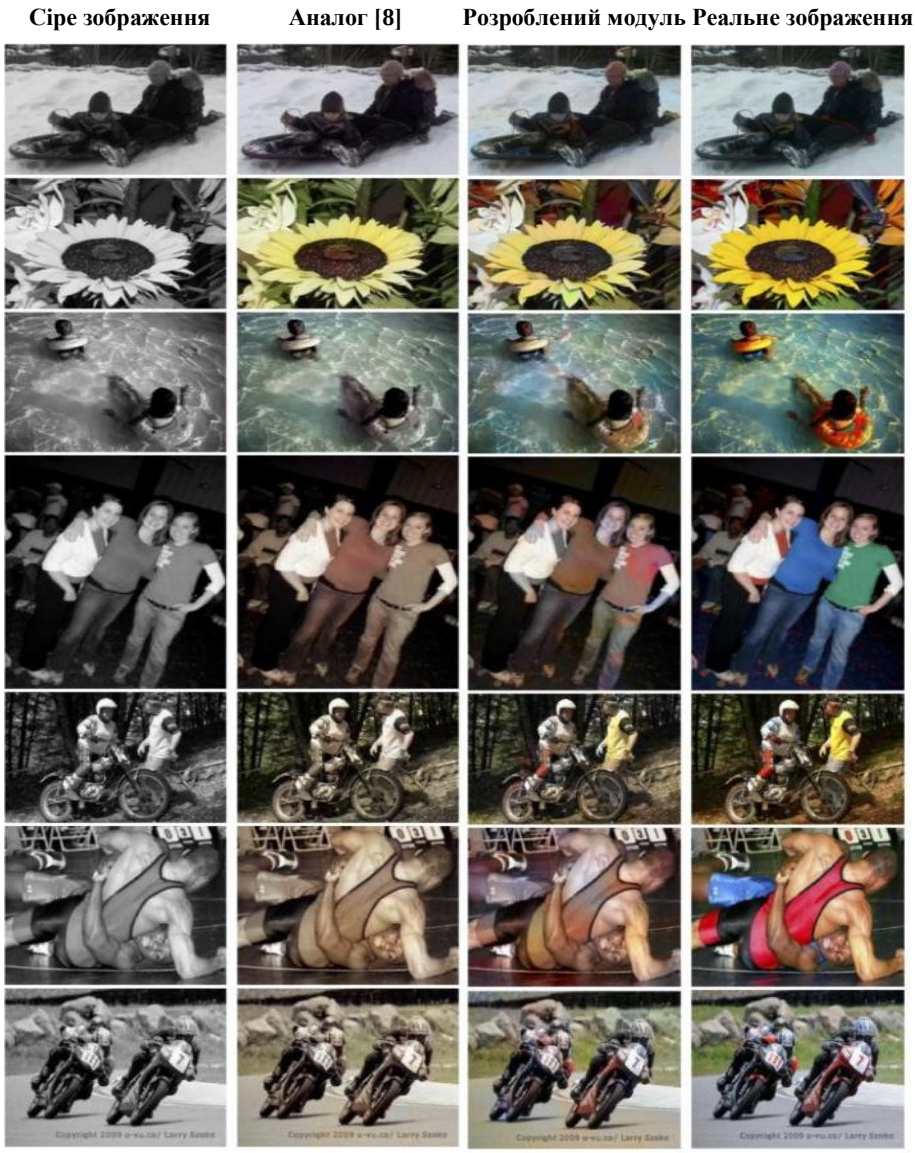


Рисунок 4.2 – Порівняння результатів, отриманих запропонованим модулем розфарбування, з програмою-аналогом

На рис. 4.2 перший стовпець показує вхідне сіре зображення. Стовпець 2 показує результати програми-аналога від Їзука та ін. [8]. Стовпець 3 показує результати розробленого модуля, і, нарешті, останній стовпець містить відповідні справжні зображення. У представлених прикладах у рядках 5 і 7 розроблений модуль перевершує програму-аналог, створюючи більш фотореалістичні зображення. В інших зображеннях (рядках) деякі області згенерованих розробленим модулем зображень мають недостатню насиченість. Зображення взяті з набору даних ImageNet.

З результатів було помічено, що хоча деякі розфарбовані зображення були досить хорошими, деякі згенеровані зображення мають тенденцію до низької насиченості, при цьому мережа створює сіруваті кольори там, де оригінал був яскравішим (наприклад, із зображеннями фруктів, квітів чи одягу). Наше тлумачення полягає в тому, що мережа, намагаючись мінімізувати втрати між зображеннями, де, наприклад, квіти червоні, і тими, де квіти сині, робить дуже обережні прогнози, а саме призначає нейтральний сірий колір.

4.2 Аналіз результатів роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі

4.2.1. Якісна оцінка результатів роботи програмного модуля розфарбовування зображень.

Оцінити продуктивність розробленого модуля можна емпірично за допомогою (2.2), але також цікаво, наскільки переконливо виглядають кольори для людського спостерігача, що важко оцінити, використовуючи лише математичні інструменти. При оцінці результатів програми-аналогу [8] було вирішено оцінити вигляд деяких штучно розфарбованих зображень за допомогою дослідження з користувачами. Для цього було обрано дванадцять розфарбованих зображень, дев’ять з яких показані на рис. 4.3, і для кожного з них ставили питання: «Фальшиве чи справжнє?». Обирались результати, які,

імовірно, могли б «обдурити» людське око, відкидаючи всі зображення, які були погано розфарбовані. Опитування проводили серед 41 різного користувача.

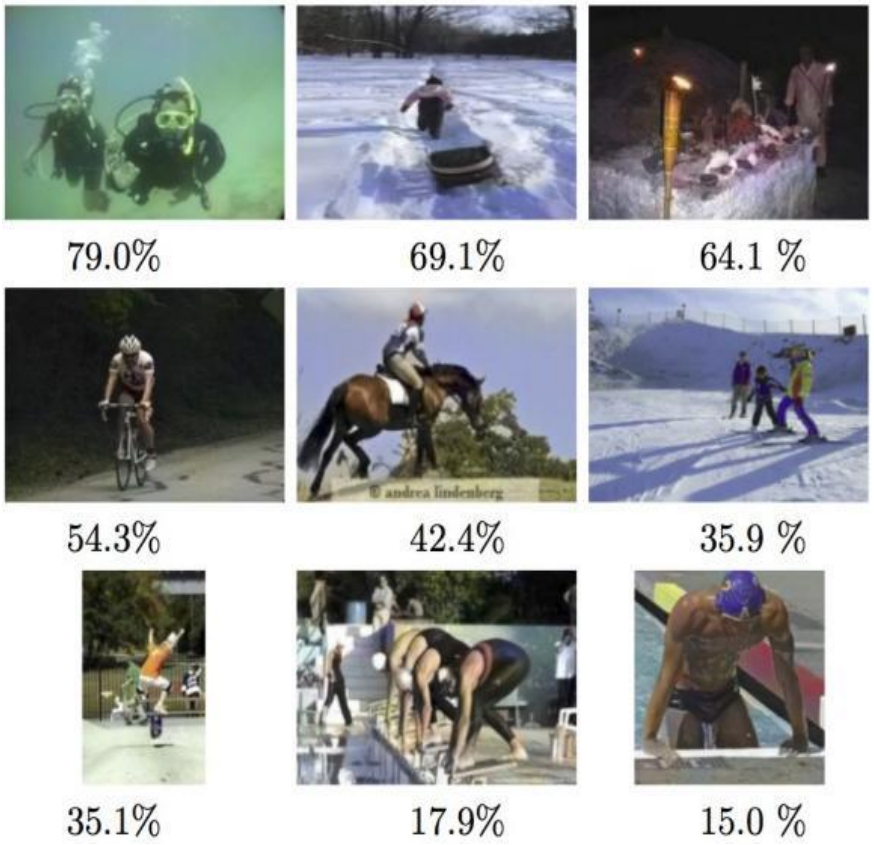


Рис. 4.3 - Розфарбовані зображення з відсотком користувачів, які вважають їх справжніми

На рис. 4.3 для кожного розфарбованого зображення вказано відсоток користувачів, які відповіли «справжнє» на питання «Фальшиве чи справжнє?». Зображення відсортовані за їхньою схожістю на справжнє. Зокрема, в аналогічній програмі [8] спостерігали, що в деяких випадках сприйняття як справжнього досягало майже 80,0%. Однак ці результати також вказують на те, що правдоподібність зображення сильно залежить від того, що саме зображено. Імовірно, збільшення розміру та різноманітності навчального набору могло б частково вирішити цю проблему. Загалом було підраховано,

що 45,87% користувачів неправильно класифікували розфарбовані аналогом зображення як оригінальні.

Аналогічні дослідження з 41 користувачем були проведені і для розробленого програмного модуля. Для оцінки користувачам надавались ті ж самі дванадцять розфарбованих зображень, що і в [8]. Середній відсоток сприйняття розфарбованих зображень як справжніх для розробленого модуля склало 57,24%, що майже на 11,6 % краще, ніж в аналога. Ці результати наведено у табл. 4.1.

4.2.2. Кількісна оцінка результатів роботи програмного модуля розфарбовування зображень.

Хоча продуктивність моделі найкраще оцінюється за візуальною якістю результатів, кількісні показники можуть слугувати корисним показником, який допоможе нам оцінити або змоделювати та спостерігати макротенденції в її поведінці.

Як кількісний показник точності розфарбовування, тобто метрикою оцінки якості розфарбовування зображення можна обрати середньоквадратичну помилку прогнозування кольору, усереднену по всіх зображеннях із тестової вибірки (УСКП – усереднена середньоквадратична помилка). Розглянемо як вона знаходиться.

У розробленій програмі вхідними даними нейромережі є матриця чисел, що представляє чорно-біле зображення. На виході нейромережі отримуємо дві матриці зі значеннями кольорів. Між вхідними та вихідними значеннями ми створюємо фільтри, щоб зв’язати їх разом. Це згорткова нейронна мережа.

Коли ми навчаємо мережу (а також коли тестуємо), ми використовуємо пари відповідних один одному зображень (чорно-біле зображення)-(кольорове зображення). Ми перетворюємо кольори RGB у колірний простір Lab*. Чорно-білий шар є входом нейромережі, а два кольорових шари є виходом нейромережі.

На рис. 4.4 ліворуч ми маємо чорно-білий вхід, наші фільтри та передбачення нашої нейронної мережі. Також на рис. 4.4 відображено прогнозовані (mapped prediction) та реальні (mapped truth) значення кольорового зображення в одному інтервалі. Таким чином ми можемо порівняти ці значення (знайти похибку - error). Інтервал значень кольорових зображень коливається від -1 до 1, оскільки для відображення прогнозованих значень використовується функція активації tanh . Для будь-якого значення, яке надається функції tanh, вона поверне від -1 до 1.

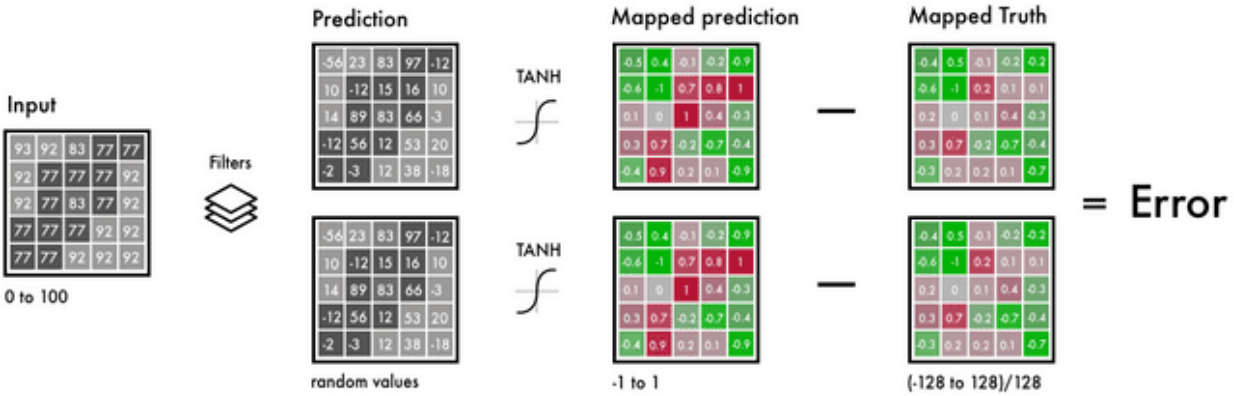


Рис. 4.4 – Пояснення знаходження похибки прогнозування кольору

Значення кольору справжніх зображень коливаються від -128 до 128. Це стандартний інтервал у просторі кольорів Lab*. Якщо розділити їх на 128, вони також потраплять в інтервал від -1 до 1. Ця «нормалізація» дозволяє нам обчислити помилку прогнозу і реального зображення. Тільки ми використовуємо не абсолютне значення помилки, а середньоквадратичне по (2.2).

В режимі навчання після обчислення остаточної помилки мережа оновлює фільтри, щоб зменшити загальну помилку. Мережа продовжує працювати в цьому циклі, доки помилка не стане якомога меншою.

А в режимі тестування програма просто обчислює середню помилку для всіх зображень із тестової вибірки (6 000 зображень). Середньоквадратична

помилка прогнозування кольору, усереднена по всіх тестових зображеннях, наведена у табл. 4.1.

Таблиця 4.1 – Основні усереднені показники якості процесу розфарбовування зображень

	Програма – аналог [8]	Розроблений програмний модуль
Середній відсоток сприйняття розфарбованих зображень як справжніх	45,87%	57,24%
Усереднена середньоквадратична помилка прогнозування кольору	0,00307	0,00248

Із табл. 4.1 видно, що розроблений програмний модуль має кращі усереднені показники якості процесу розфарбовування зображень, ніж програма-аналог. Так, розроблений програмний модуль має середній відсоток сприйняття розфарбованих зображень як справжніх 57,24%, а програма-аналог 45,87%, тобто по цьому показнику якість розфарбовування підвищена майже на 11,6%.

Також із табл. 4.1 видно, що розроблений програмний модуль має усереднену середньоквадратичну помилку прогнозування кольору 0,00307, а програма-аналог 0,00248, тобто по цьому показнику якість розфарбовування підвищена в абсолютній величині на 0,00059, а у відносній на (0,00307-0,00248)/0,00307=19,2%.

Таким чином, можна зробити висновок, що розроблений програмний модуль розфарбовування зображень на основі згорткової нейронної мережі має порівняно з програмою-аналогом збільшену на 11÷19% якість розфарбовування чорно-білих зображень. Тобто мета роботи досягнута – якість розфарбовування чорно-білих зображень підвищена.

4.3 Висновок до розділу 4

У розділі в результаті тестування програми було доведено її працездатність та відповідність поставленому завданню. Продуктивність процесу розфарбовування зображень була оцінена якісно та кількісно. Для якісної оцінки було використано таку метрику як середній відсоток сприйняття розфарбованих зображень як справжніх групою користувачів (експертів). Для кількісної оцінки було використано таку метрику як усереднена середньоквадратична помилка прогнозування кольору. Результати аналізу якості процесу розфарбовування зображень показали, що розроблений програмний модуль розфарбовування зображень на основі згорткової нейронної мережі має порівняно з програмою-аналогом збільшену на 11÷19% якість розфарбовування чорно-білих зображень. Тобто мета роботи досягнута – якість розфарбовування чорно-білих зображень підвищена.

ВИСНОВКИ

У роботі було розглянуто задачу розфарбовування чорно-білих зображень на основі згорткової нейронної мережі. В ході аналізу предметної області було описано детальну постановку задачі розфарбовування чорно-білих зображень. Також розглянуто різні методи розв'язання задачі розфарбовування зображень і оцінено їх переваги та недоліки. Розроблюваний метод відноситься до автоматичних методів мультимодального розфарбовування зображень у градаціях сірого. Було обрано напрямок досліджень, що базується на згорткових нейронних мережах типу енкодер-декодер, однак включає попередньо навчену додаткову модель згорткової нейромережі. Крім цього, було обгрунтовано вибір аналогу до розроблюваного модуля розфарбовування зображень.

В ході проектування було розроблено метод розфарбовування зображень на основі згорткової нейронної мережі з використанням колірного простору Lab*. Було розроблено архітектуру згорткової нейронної мережі, яка лежить в основі розробленого методу. Архітектура нейромережі побудована по схемі енкодер-декодер та включає гілку вилучення ознак на основі попередньо навченої нейронної мережі Inception-ResNet-v2. Розроблено послідовність та параметри процесу навчання згорткової нейронної мережі для розфарбовування зображень. Розроблено алгоритм роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі.

Було обґрунтовано вибір мови програмування Python та спеціалізованих бібліотек Keras та TenzorFlow для програмної реалізації модуля розфарбовування зображень на основі згорткової нейронної мережі. Проведено аналіз набору даних зображень на основі відомої бази даних ImageNet, в який було відібрано 60000 зображень, з яких 54000 було взято як навчальні та 6000 як тестові. Описано основні етапи програмної реалізація та функціонування модуля розфарбовування зображень на основі згорткової нейронної мережі, що складається з завантаження набору даних, реалізації

архітектури згорткової нейронної мережі, навчання згорткової нейронної мережі, оцінювання процесу навчання.

В результаті тестування програми було доведено її працездатність та відповідність поставленому завданню. Продуктивність процесу розфарбовування зображень була оцінена якісно та кількісно. Для якісної оцінки було використано таку метрику як середній відсоток сприйняття розфарбованих зображень як справжніх групою користувачів (експертів). Для кількісної оцінки було використано таку метрику як усереднена середньоквадратична помилка прогнозування кольору. Результати аналізу якості процесу розфарбовування зображень показали, що розроблений програмний модуль розфарбовування зображень на основі згорткової нейронної мережі має порівняно з програмою-аналогом збільшену на 11÷19% якість розфарбовування чорно-білих зображень. Тобто мета роботи досягнута – якість розфарбовування чорно-білих зображень підвищена

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Гречкосій М.П., Короленко О.О., Колесницький О.К. ПРОГРАМНИЙ МОДУЛЬ РОЗФАРБОВУВАННЯ ЗОБРАЖЕНЬ НА ОСНОВІ ЗГОРТКОВОЇ НЕЙРОМЕРЕЖІ / Міжнародна науково-практична Інтернет-конференція студентів, аспірантів та молодих науковців «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ (МН-2025)», ВНТУ, Вінниця, 15.10.2024 - 15.06.2025. https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/24949/20690

2 Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR abs/1512.00567 (2015)

3 He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)

4 Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)

5 Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR abs/1602.07261 (2016)

6 He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. CoRR abs/1603.05027 (2016)

7 Cheng, Z., Yang, Q., Sheng, B.: Deep colorization. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 415–423

8 Iizuka, S., Simo-Serra, E., Ishikawa, H.: Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. ACM Transactions on Graphics (TOG) 35(4) (2016) 110

9 Messaoud, S., Forsyth, D., Schwing, A.G., 2018. Structural consistency and controllability for diverse colorization. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 596–612.

10 Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. In: ACM Transactions on Graphics (ToG). Volume 23., ACM (2004) 689–694

11 Larsson, G., Maire, M., Shakhnarovich, G.: Learning representations for automatic colorization. In: European Conference on Computer Vision, Springer (2016) 577– 593

12 Su, J.-W., Chu, H.-K., Huang, J.-B., 2020. Instance-aware image colorization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7968–7977.

13 Yoo, S., Bahng, H., Chung, S., Lee, J., Chang, J., Choo, J., 2019. Coloring with limited data: Few-shot colorization via memory augmented networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

14 Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision, Springer (2016) 649–666

15 Deshpande, A., Rock, J., Forsyth, D.: Learning large-scale automatic image colorization. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 567–575

16 Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser Ł., Polosukhin I. Attention is all you need Advances in Neural Information Processing Systems (2017), pp. 5998-6008

17 Kumar, M., Weissenborn, D., Kalchbrenner, N., 2020. Colorization transformer. In: International Conference on Learning Representations.

18 Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

19 Tola, E., Lepetit, V., Fua, P.: A fast local descriptor for dense matching. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE (2008) 1–8

20 Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. ICLR (2015)

21 Zhang, R., Zhu, J.Y., Isola, P., Geng, X., Lin, A.S., Yu, T., Efros, A.A.: Realtime user-guided image colorization with learned deep priors. arXiv preprint arXiv:1705.02999 (2017)

22 Robertson, A.R.: The cie 1976 color-difference formulae. Color Research & Application 2(1) (1977) 7–11

23 Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) http: //www.deeplearningbook.org.

24 Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision, Springer (2014) 818–833

25 Bora, D.J., Kumar Gupta, A., Ahmad Fayaz, K.: Unsupervised diverse colorization via generative adversarial networks. International Journal of Emerging Technology and Advanced Engineering (2015)

26 Nixon, M.S., Aguado, A.S.: Feature Extraction & Image Processing for Computer Vision. Elsevier Ltd (2002)

27 Hoffman, G.: Cielab colorspace. Technical report, University of Applied Sciences, Emden (Germany), http://docs-hoffmann.de/cielab03022003.pdf (2003)

28 Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.A.: Striving for simplicity: The all convolutional net. CoRR abs/1412.6806 (2014)

29 Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014)

30 Python [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Python

31 TensorFlow [Електронний ресурс] – Режим доступу: https://www.tensorflow.org//

32 Keras [Електронний ресурс] – Режим доступу: https://keras.io/

33 Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для студентів денної та заочної форм навчання спеціальності 122 - «Комп’ютерні науки» / Укладачі А.А.Яровий, О.В.Сілагін, І.В.Богач. – Вінниця: ВНТУ, 2022. – 47 с.

Додаток А (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль розфарбовування зображення на основі нейромережі

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

Підрозділ кафедра комп'ютерних наук.
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 12,59 % .

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колесницький О.К.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Гречкосій М.П.
(підпис) (прізвище, ініціали)

Додаток Б (обов'язковий)

Лістинг програми

Клас network_definition.py

```
"""
Given the L channel of an Lab image (range [-1, +1]), output a prediction
over the a and b channels in the range [-1, 1].
In the neck of the conv-deconv network use the features from a feature
extractor (e.g. Inception) and fuse them with the conv output.
"""

from keras.engine import InputLayer
from keras.layers import Conv2D, UpSampling2D
from keras.models import Sequential

from .fusion_layer import FusionLayer


class Colorization:
    def __init__(self, depth_after_fusion):
        self.encoder = _build_encoder()
        self.fusion = FusionLayer()
        self.after_fusion = Conv2D(depth_after_fusion, (1, 1),
activation="relu")
        self.decoder = _build_decoder(depth_after_fusion)

    def build(self, img_l, img_emb):
        img_enc = self.encoder(img_l)

        fusion = self.fusion([img_enc, img_emb])
        fusion = self.after_fusion(fusion)

        return self.decoder(fusion)


def _build_encoder():
    model = Sequential(name="encoder")
    model.add(InputLayer(input_shape=(None, None, 1)))
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same",
strides=2))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same",
strides=2))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same",
strides=2))
    model.add(Conv2D(512, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(512, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(256, (3, 3), activation="relu", padding="same"))
    return model


def _build_decoder(encoding_depth):
    model = Sequential(name="decoder")
    model.add(InputLayer(input_shape=(None, None, encoding_depth)))
    model.add(Conv2D(128, (3, 3), activation="relu", padding="same"))
    model.add(UpSampling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same"))
```

```
    model.add(Conv2D(64, (3, 3), activation="relu", padding="same"))
    model.add(UpSampling2D((2, 2)))
    model.add(Conv2D(32, (3, 3), activation="relu", padding="same"))
    model.add(Conv2D(2, (3, 3), activation="tanh", padding="same"))
    model.add(UpSampling2D((2, 2)))
    return model
```

Клас fusion_layer.py

```
from keras import backend as K
from keras.engine import Layer


class FusionLayer(Layer):
    def call(self, inputs, mask=None):
        imgs, embs = inputs
        reshaped_shape = imgs.shape[:3].concatenate(embs.shape[1])
        embs = K.repeat(embs, imgs.shape[1] * imgs.shape[2])
        embs = K.reshape(embs, reshaped_shape)
        return K.concatenate([imgs, embs], axis=3)

    def compute_output_shape(self, input_shapes):
        # Must have 2 tensors as input
        assert input_shapes and len(input_shapes) == 2
        imgs_shape, embs_shape = input_shapes

        # The batch size of the two tensors must match
        assert imgs_shape[0] == embs_shape[0]

        # (batch_size, width, height, embedding_len + depth)
        return imgs_shape[:3] + (imgs_shape[3] + embs_shape[1],)
```

Клас train.py

```
import argparse
from pathlib import Path

import tensorflow as tf
from keras import backend as K

from .network_definition import Colorization
from .training_utils import (
    evaluation_pipeline,
    checkpointing_system,
    plot_evaluation,
    training_pipeline,
    metrics_system,
    Logger,
)

parser = argparse.ArgumentParser(description="Train")
parser.add_argument(
    "tfrecords",
    type=str,
    metavar="TFRECORDS_DIR",
    help="train using all tfrecords in TFRECORDS_DIR",
)
parser.add_argument(
    "output",
    type=str,
```

```
    metavar="OUR_DIR",
    help="save metrics and checkpoints in OUR_DIR",
)
parser.add_argument(
    "--run-id", type=str, required=True, metavar="RUN_ID", help="unique run
identifier"
)
parser.add_argument(
    "--train-steps",
    type=int,
    required=True,
    metavar="STEPS",
    help="train for STEPS",
)
parser.add_argument(
    "--val-every",
    type=int,
    required=True,
    metavar="STEPS",
    help="run validation and save checkpoint every STEPS",
)
args = parser.parse_args()
dir_tfrecords = Path(args.tfrecords).expanduser().resolve().as_posix()
dir_output =
Path(args.output).expanduser().resolve().joinpath(args.run_id).as_posix()

# PARAMETERS
run_id = args.run_id
val_number_of_images = 10
batch_size = 100
learning_rate = 0.001

# START
sess = tf.Session()
K.set_session(sess)

# Build the network and the various operations
col = Colorization(256)
opt_operations = training_pipeline(col, learning_rate, batch_size,
dir_tfrecords)
evaluations_ops = evaluation_pipeline(col, val_number_of_images,
dir_tfrecords)
summary_writer = metrics_system(sess, dir_output)
saver, checkpoint_paths, latest_checkpoint = checkpointing_system(dir_output)
logger = Logger(f"{dir_output}/output.txt")

with sess.as_default():
    # Initialize
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())

    # Coordinate the loading of image files.
    coord = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(coord=coord)

    # Restore
    if latest_checkpoint is not None:
        logger.write("Restoring from: {}".format(latest_checkpoint))
        saver.restore(sess, latest_checkpoint)
        logger.write(" done!")
    else:
        logger.write("No checkpoint found in: {}".format(checkpoint_paths))
```

```
    # Training loop
    for batch_idx in range(args.train_steps):
        logger.write(f"Batch: {batch_idx}/{args.train_steps}")
        res = sess.run(opt_operations)
        global_step = res["global_step"]
        logger.write(f'Cost: {res["cost"]} Global step: {global_step}')
        summary_writer.add_summary(res["summary"], global_step)

        if (batch_idx + 1) % args.val_every == 0:
            # Save the variables to disk
            save_path = saver.save(sess, checkpoint_paths, global_step)
            logger.write("Model saved in: %s" % save_path)

            # Evaluation step
            res = sess.run(evaluations_ops)
            summary_writer.add_summary(res["summary"], global_step)
            plot_evaluation(res, global_step, dir_output)

    # Finish off the filename queue coordinator.
    coord.request_stop()
    coord.join(threads)
```

Клас training_utils.py

```
import pickle
import time
from os.path import join

import matplotlib
import numpy as np
import tensorflow as tf
from skimage import color

from .dataset.labels import inception_labels
from .dataset.shared import maybe_create_folder
from .dataset.tfrecords import LabImageRecordReader

matplotlib.use("Agg")
matplotlib.rcParams["figure.figsize"] = (10.0, 4.0)
import matplotlib.pyplot as plt


def loss_with_metrics(img_ab_out, img_ab_true, name=""):
    # Loss is mean square erros
    cost = tf.reduce_mean(tf.squared_difference(img_ab_out, img_ab_true),
name="mse")
    # Metrics for tensorboard
    summary = tf.summary.scalar("cost " + name, cost)
    return cost, summary


def training_pipeline(col, learning_rate, batch_size, dir_tfrecord):
    # Set up training (input queues, graph, optimizer)
    irr = LabImageRecordReader("lab_images_*.tfrecord", dir_tfrecord)
    read_batched_examples = irr.read_batch(batch_size, shuffle=True)
    # read_batched_examples = irr.read_one()
    imgs_l = read_batched_examples["image_l"]
    imgs_true_ab = read_batched_examples["image_ab"]
    imgs_emb = read_batched_examples["image_embedding"]
```

```
    imgs_ab = col.build(imgs_l, imgs_emb)
    cost, summary = loss_with_metrics(imgs_ab, imgs_true_ab, "training")
    global_step = tf.Variable(0, name="global_step", trainable=False)
    optimizer = tf.train.AdamOptimizer(learning_rate).minimize(
        cost, global_step=global_step
    )
    return {
        "global_step": global_step,
        "optimizer": optimizer,
        "cost": cost,
        "summary": summary,
    }  # , irr, read_batched_examples


def evaluation_pipeline(col, number_of_images, dir_tfrecord):
    # Set up validation (input queues, graph)
    irr = LabImageRecordReader("val_lab_images_*.tfrecord", dir_tfrecord)
    read_batched_examples = irr.read_batch(number_of_images, shuffle=False)
    imgs_l_val = read_batched_examples["image_l"]
    imgs_true_ab_val = read_batched_examples["image_ab"]
    imgs_emb_val = read_batched_examples["image_embedding"]
    imgs_ab_val = col.build(imgs_l_val, imgs_emb_val)
    cost, summary = loss_with_metrics(imgs_ab_val, imgs_true_ab_val,
"validation")
    return {
        "imgs_l": imgs_l_val,
        "imgs_ab": imgs_ab_val,
        "imgs_true_ab": imgs_true_ab_val,
        "imgs_emb": imgs_emb_val,
        "cost": cost,
        "summary": summary,
    }


class Logger(object):
    def __init__(self, path):
        self.f = open(path, "a")

    def write(self, text):
        self.f.write(f'[{time.strftime("%c")}] {text}\n')
        self.f.flush()


def metrics_system(sess, dir_metrics):
    # Merge all the summaries and set up the writers
    train_writer = tf.summary.FileWriter(dir_metrics, sess.graph)
    return train_writer


def checkpointing_system(dir_checkpoints):
    # Add ops to save and restore all the variables.
    saver = tf.train.Saver()
    latest_checkpoint = tf.train.latest_checkpoint(dir_checkpoints)
    checkpoint_paths = join(dir_checkpoints, "weights")
    return saver, checkpoint_paths, latest_checkpoint


def plot_evaluation(res, global_step, dir_images):
    maybe_create_folder(dir_images)
    for k in range(len(res["imgs_l"])):
        img_gray = l_to_rgb(res["imgs_l"][k][:, :, 0])
        img_output = lab_to_rgb(res["imgs_l"][k][:, :, 0], res["imgs_ab"][k])
```

```
        img_true = lab_to_rgb(res["imgs_l"][k][:, :, 0],
res["imgs_true_ab"][k])
        top_5 = np.argsort(res["imgs_emb"][k])[-5:]
        try:
            top_5 = " / ".join(inception_labels[i] for i in top_5)
        except:
            ptop_5 = str(top_5)

        plt.subplot(1, 3, 1)
        plt.imshow(img_gray)
        plt.title("Input (grayscale)")
        plt.axis("off")
        plt.subplot(1, 3, 2)
        plt.imshow(img_output)
        plt.title("Network output")
        plt.axis("off")
        plt.subplot(1, 3, 3)
        plt.imshow(img_true)
        plt.title("Target (original)")
        plt.axis("off")
        plt.suptitle(top_5, fontsize=7)

        plt.savefig(join(dir_images, f"{global_step}_{k}.png"))
        plt.clf()
        plt.close()


def l_to_rgb(img_l):
    """
    Convert a numpy array (l channel) into an rgb image
    :param img_l:
    :return:
    """
    lab = np.squeeze(255 * (img_l + 1) / 2)
    return color.gray2rgb(lab) / 255


def lab_to_rgb(img_l, img_ab):
    """
    Convert a pair of numpy arrays (l channel and ab channels) into an rgb
image
    :param img_l:
    :return:
    """
    lab = np.empty([*img_l.shape[0:2], 3])
    lab[:, :, 0] = np.squeeze(((img_l + 1) * 50))
    lab[:, :, 1:] = img_ab * 127
    return color.lab2rgb(lab)
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ РОЗФАРБОВУВАННЯ ЗОБРАЖЕННЯ НА ОСНОВІ НЕЙРОМЕРЕЖІ»

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гречкосій М. П.
(прізвище та ініціали)

Керівник: к.т.н., проф. каф.КН
Колесницький О. К.
(прізвище та ініціали)
«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

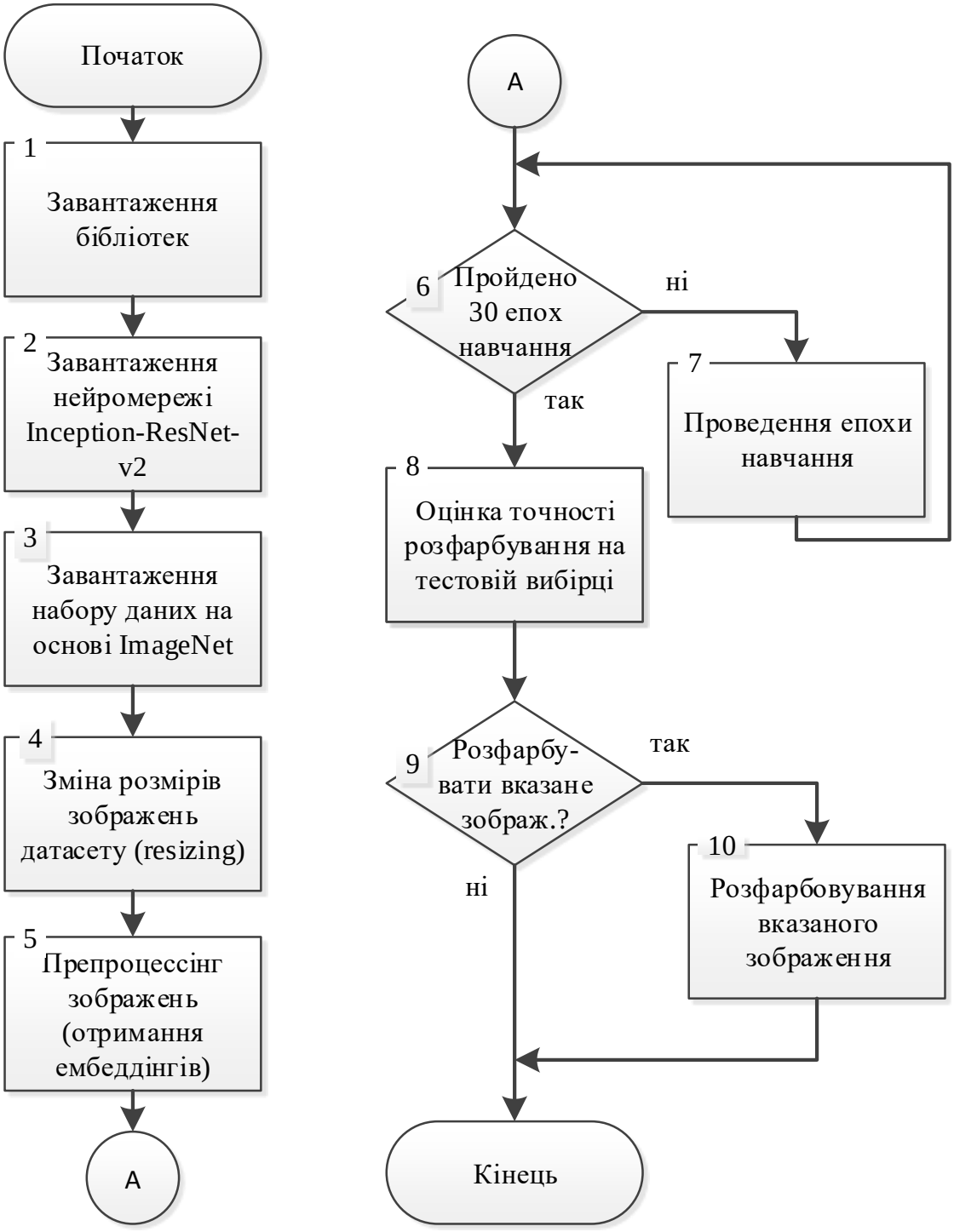


Рисунок В.1– Схема алгоритму роботи програмного модуля розфарбовування зображень на основі згорткової нейронної мережі

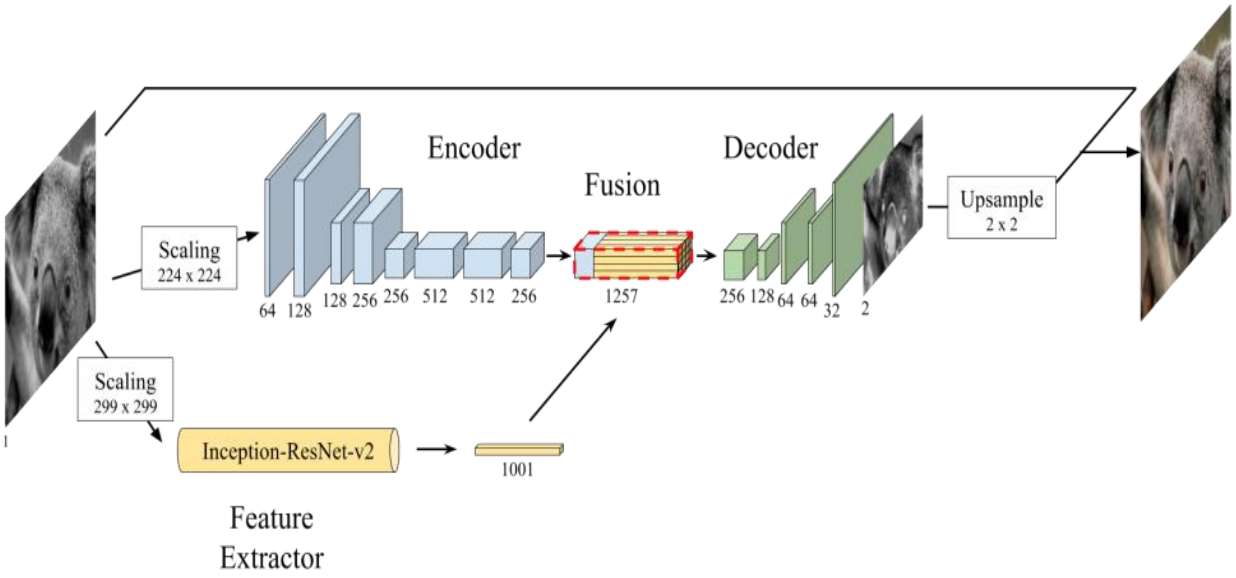


Рисунок В.2 – Архітектура використовуваної згорткової нейромережі

Таблиця В.1 - Структура шарів використовуваної згорткової нейромережі

Шар (Layer)	Ядро (Kernel)	Крок (Stride)	Шар (Layer)	Ядро (Kernel)	Крок (Stride)	Шар (Layer)	Ядро (Kernel)	Крок (Stride)
conv	64×(3×3)	2 × 2	fusion	-	-	conv	128×(3×3)	1 × 1
conv	128×(3×3)	1 × 1	conv	256×(1×1)	1 × 1	upsamp	-	-
conv	128×(3×3)	2 × 2				conv	64×(3×3)	1 × 1
conv	256×(3×3)	1 × 1				conv	64×(3×3)	1 × 1
conv	256×(3×3)	2 × 2				upsamp	-	-
conv	512×(3×3)	1 × 1				conv	32×(3×3)	1 × 1
conv	512×(3×3)	1 × 1				conv	2×(3×3)	1 × 1
conv	256×(3×3)	1 × 1				upsamp	-	-

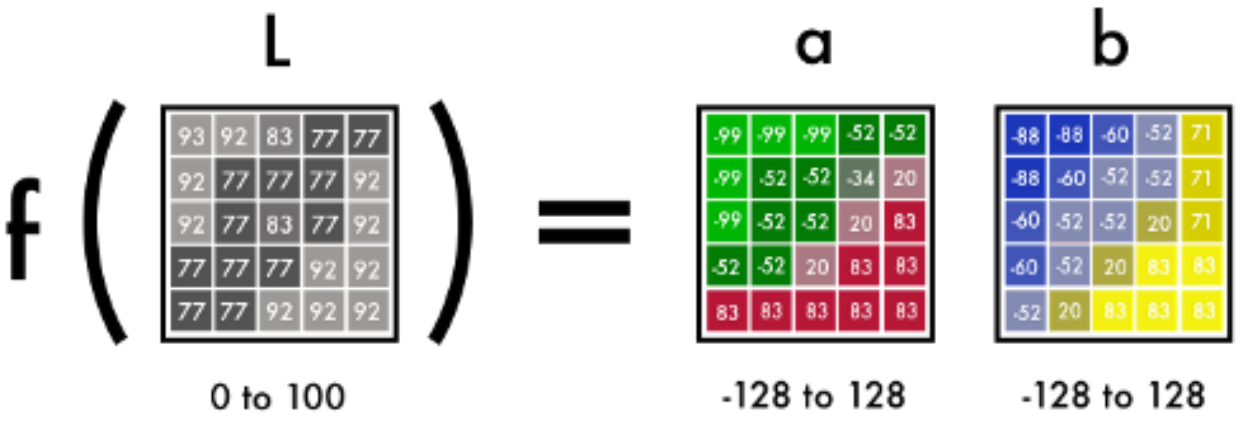


Рисунок В.3 – – Формалізація задачі розфарбовування сірого зображення у випадку застосування колірного простору Lab*

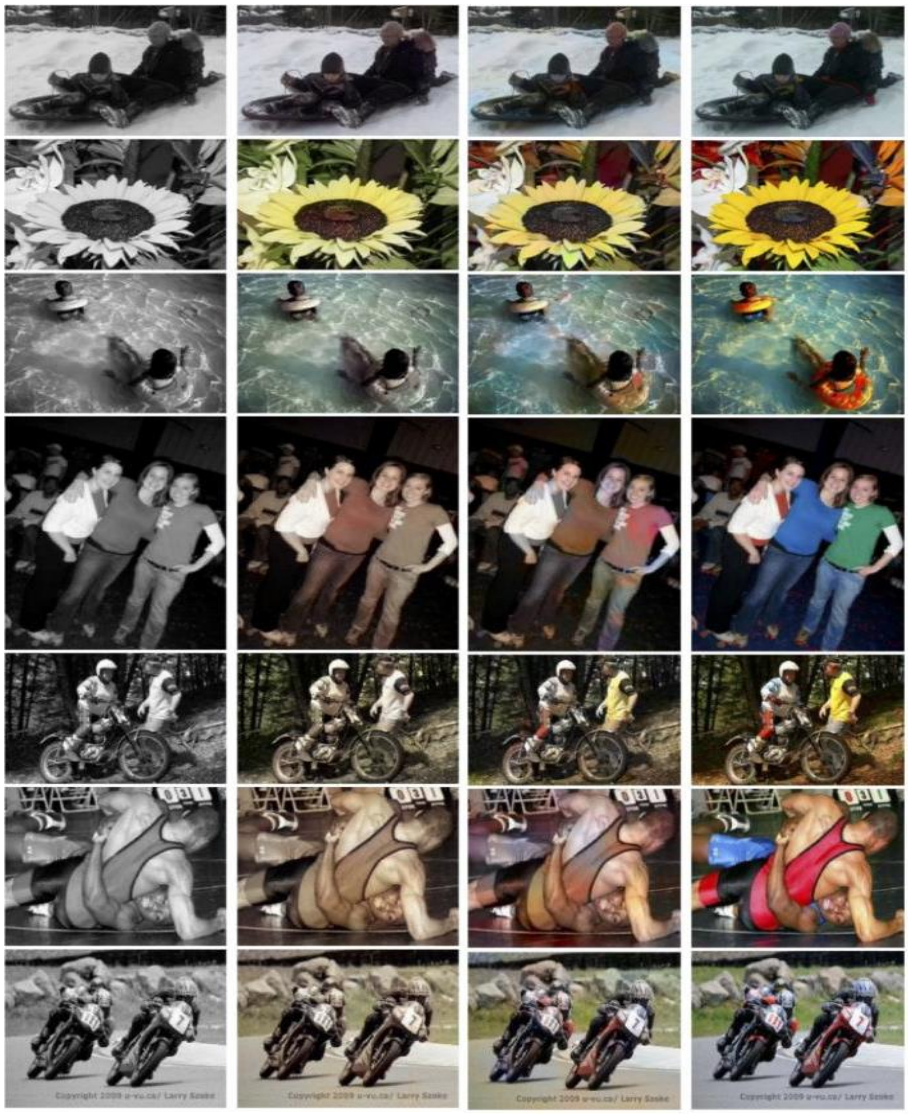


Рисунок В.4 – Порівняння результатів, отриманих запропонованим модулем розфарбування, з програмою-аналогом

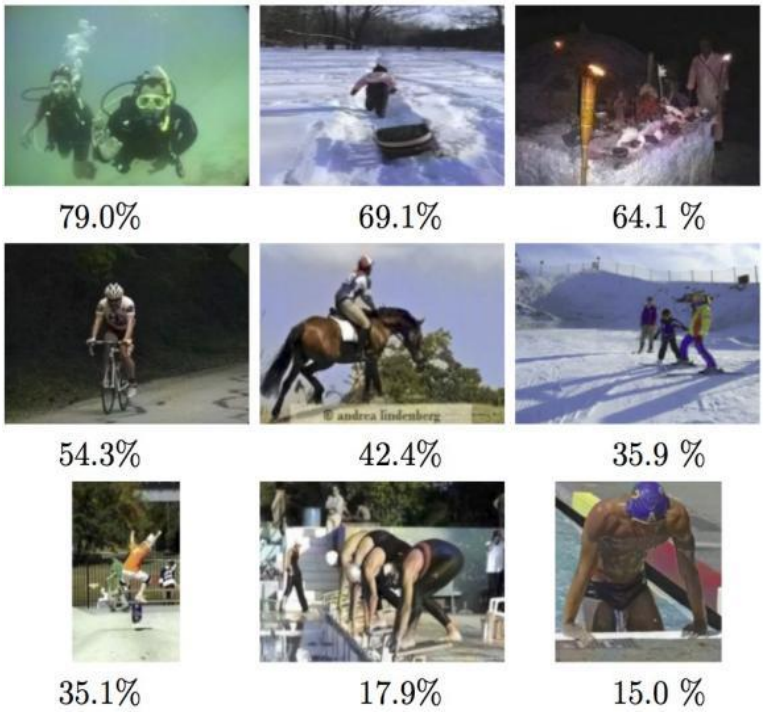


Рисунок В.5 – Розфарбовані зображення з відсотком користувачів, які вважають їх справжніми

Таблиця В.2 – Основні усереднені показники якості процесу розфарбовування зображень

	Програма – аналог [7]	Розроблений програмний модуль
Середній відсоток сприйняття розфарбованих зображень як справжніх	45,87%	57,24%
Усереднена середньоквадратична помилка прогнозування кольору	0,00307	0,00248