НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.8:004.93

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2020 р.

Магістерська дисертація

на здобуття ступеня магістра

освітньо-науковою програмою
«Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Алгоритмічно-програмний метод колоризації зображень»

Виконав:
студент ІІ курсу, групи КП-81мн
Топчієв Борис Сергійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент
Сулема Євгенія Станіславівна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри СПСКС, к.т.н., доцент,
Боярінова Юлія Євгенівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ Іван ДИЧКА

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Топчієву Борису Сергійовичу

1. Тема дисертації «Алгоритмічно-програмний метод колоризації зображень», науковий керівник дисертації Сулема Євгенія Станіславівна, к.т.н., доцент, затверджена наказом по університету від «07» квітня 2020 р. №964-С

2. Термін подання студентом дисертації «15» травня 2020 р.

3. Об'єкт дослідження: процес обробки зображень за допомогою математичних алгоритмів та алгоритмів штучного інтелекту.

4. Предмет дослідження: методи та алгоритми колоризації зображень автоматичним та напівавтоматичним шляхом.

5. Перелік завдань, які потрібно вирішити:
   - провести аналіз існуючих методів роботи з зображеннями в контексті задачі колоризації;
   - дослідити архітектури нейронних мереж для навчання;
   - обрати необхідні дані та розробити методи їх попередньої обробки;
   - розробити програму для навчання моделей;
   - провести тестування натренованих моделей;
   - розробити користувацький інтерфейс для експлуатації технології користувачем;
   - провести аналіз отриманих результатів.

6. Орієнтовний перелік публікацій:
   - Тези доповіді "Покращення якості зображень за допомогою генеративних змагальних мереж"
   - Стаття " Інтелектуальна колоризація зображень за допомогою генеративних змагальних мереж "

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., к.т.н., доцент		

8. Дата видачі завдання «11» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.12.2018	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.03.2019	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	16.05.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення; підготовка матеріалів доповіді на конференції ПМК-2019	14.10.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	15.12.2019	
6.	Проведення наукового дослідження; робота над третім та четвертим розділами магістерської дисертації	20.02.2020	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу	16.04.2020	
8.	Оформлення текстової і графічної частини магістерської дисертації	13.05.2020	

Студент Борис ТОПЧІЄВ

Науковий керівник Євгенія СУЛЕМА

РЕФЕРАТ

Актуальність теми. Редагування зображень за допомогою засобів машинного навчання та штучного інтелекту є актуальним напрямком у галузі обробки зображень. Через надзвичайну популярність соціальних мереж, у яких основним контентом є зображення (Instagram, PicsArt, DevianArt, Flickr), велика кількість людей щоденно вдаються до редагування власних фото за допомогою програмного забезпечення. Традиційні фото-редактори засновані на стабільних математичних алгоритмах та повертають однозначні результати, проте є обмежені власними алгоритмами, які можуть бути використані лише для вирішення дуже конкретних локальних задач, таких як зміна віддтінків на усьому зображенні, маніпуляції з насиченістю, накладання декількох зображень, посилення та послаблення тіней, шумів, тощо.

Для даної магістерської дисертації було вирішено розробити та протестувати технологію обробки зображень для розв'язку задачі інтелектуальної колоризації, так як дана задача досі є актуальною, а існуючі методи можна значно покращити.

Об'єктом дослідження є спрощення процесу обробки зображень за допомогою математичних алгоритмів та алгоритмів штучного інтелекту.

Предметом дослідження є методи та алгоритми колоризації зображень автоматичним та напівавтоматичним (з внесенням додаткової інформації користувачем) шляхом.

Мета роботи полягає у розробці ефективного методу колоризації зображень у напівавтоматичному і автоматичному режимі на основі згорткових та генеративних змагальних нейронних мереж.

Методи дослідження. В роботі використовуються методи теоретичного дослідження: аналіз та синтез. Також застосовувалися емпіричні методи: експеримент, вимірювання та порівняння.

Наукова новизна роботи полягає у розробленні модифікованого методу для колоризації зображень, який дозволяє виконувати користувачу контроль над процесом і повертає результати високої якості і працює зі швидкістю 1-1.2 секунди на зображення.

Практична цінність отриманих результатів роботи полягає в тому, що запропонований метод дає змогу підвищити якість колоризації та здійснювати контроль над процесом з боку користувача.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019 та опубліковані у збірнику тез доповідей.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень.

У першому розділі описано існуючі проблеми, пов'язані з предметною областю (обробка зображень) та розглянуто способи усунення цих проблем, наведено різні приклади алгоритмів та приклади дефектів зображень і причини їх виникнення.

У другому розділі розглянуто принцип роботи математичних алгоритмів для контрольованої колоризації зображень, розглянуто алгоритми, що базуються на згорткових нейронних мережах і виконують задачу автоматичної колоризації та реставрації зображень, розглянуто алгоритми, засновані на генеративних змагальних мережах.

У третьому розділі сформовані основні вимоги до розробленого методу; розглянуто різні варіанти архітектур нейронних мереж та розкрито основні принципи їх роботи; обґрунтовано вибір архітектур для

розробленого методу; описано загальний вигляд системи нейронних мереж, що реалізує напівавтоматичну колоризацію зображень.

У четвертому розділі наведена інформація про дані, що використовувались для тренування та методи їх попередньої обробки; обґрунтовано вибір гіпер параметрів для тренування моделей; розкрито сутність процесу тренування та тестування системи нейронних мереж; наведено кінцеві результати роботи.

У висновках проаналізовано отримані результати роботи.

У додатках наведено фрагменти вихідного коду програмної реалізації способу, копії графічних матеріалів та слайдів презентації.

Робота виконана на 140 аркушах, містить 2 додатки та посилання на список використаних літературних джерел з 38 найменувань. У роботі наведено 32 рисунків.

Ключові слова: колоризація зображень,нейронні мережі, згорткові нейронні мережі, генеративні змагальні мережі, сегментація зображень, генерація зображень.

ABSTRACT

Actuality. Image editing with the help of machine learning and artificial intelligence is an important area in the field of image processing. Due to the huge popularity of social networks, in which the main content is images (Instagram, PicsArt, DevianArt, Flickr), a large number of people daily resort to editing their own photos using software. Traditional photo editors are based on stable mathematical algorithms and return unambiguous results, but are limited by proprietary algorithms that can only be used to solve very specific local problems, such as changing shades throughout the image, manipulating saturation, overlaying multiple images, enhancing and attenuation of shadows, noises, etc.

For this master's thesis, it was decided to develop and test image processing technology to solve the problem of intellectual colorization, as this problem is still relevant, and existing methods can be significantly improved.

Object of research is to simplify the process of image processing using mathematical algorithms and artificial intelligence algorithms.

Subjects of research are methods and algorithms for coloring images automatically and semi-automatically (with additional information by the user).

Goal of the work is to develop an effective method of coloring images in a semi-automatic and automatic mode based on convolutional and generative competitive neural networks.

Methods of research include methods of theoretical research: analysis and synthesis. Also there were used empirical methods: experiment, measurement and comparison.

Scientific novelty of the work is to develop a modified method for coloring images, which allows the user to control the process and returns high quality results and works with speed of 1-1.2 seconds per image.

Practical value of the obtained results is that the proposed method allows to improve the quality of colorization and control the process by the user.

Approbation. The main provisions and results of the work were presented and discussed at the XII scientific conference of masters and postgraduates "Applied Mathematics and Computer" PMK-2019 and published in the proceedings.

Structure and content of the thesis. Master's thesis consists of an introduction, four chapters, conclusions and appendices.

The introduction provides the general characteristic of work, the estimation of a modern condition of a problem, the urgency of a direction of researches is proved, the purpose and tasks of researches are formulated.

The first chapter describes the existing problems related to the subject area (image processing) and discusses ways to solve these problems, gives various examples of algorithms and examples of image defects and their causes.

The second chapter considers the principle of operation of mathematical algorithms for controlled colorization of images, algorithms based on convolutional neural networks and performing the task of automatic colorization and restoration of images, algorithms based on generative competing networks.

In the third section, the basic requirements to the developed method are formed; different variants of neural network architectures are considered and the basic principles of their work are revealed; the choice of architectures for the developed method is substantiated; describes a general view of a system of neural networks that implements semi-automatic colorization of images.

The fourth chapter provides information on the data used for training and methods of their pre-processing; the choice of hyper parameters for model training is substantiated; the essence of the process of training and testing of the neural network system is revealed; the final results of the work are given.

The conclusion contains brief overview of the results obtained in the work.

The appendices contain fragments of the source code of the software implementation of the method, copies of graphic materials and slides of the presentation.

The work is performed on 140 pages, contains 2 appendixes and a link to the list of used literature sources from 38 titles. The paper contains 32 figures.

Keywords: image colorization, neural networks, convolutional neural networks, generative competing networks, image segmentation, image generation.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Інтелектуальна колоризація – процес обробки зображень, з метою зміни його кольорів, який виконується за допомогою використання певних математичних алгоритмів та алгоритмів штучного інтелекту, для забезпечення інтерактивності та гнучкості процесу.

Згорткова нейронна мережа – це клас глибинних штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень, аудіо, відео та базується на операціях згортки та об'єднання.

Генеративна змагальна нейронна мережа – це клас алгоритмів штучного інтелекту, що використовуються в навчанні без учителя, реалізовані системою двох штучних нейронних мереж, які змагаються одна з одною в рамках гри з нульовою сумою.

State-of-art – певний метод чи алгоритм (в більшості випадків такий, що відноситься до сфери штучного інтелекту), який визнаний багатьма науковцями як еталонний за якістю та часом.

Super Resolution – технологія штучного інтелекту, яка базується на згорткових нейронних мережах і виконує операцію підвищення якості зображень (як у контексті розподільної якості, так і розширення).

Хроматична аберація – оптичний дефект, зумовлений дисперсією світла, який проявляється у вигляді кольорової облямівки контрастних елементів зображення.

Дисторсія – аберація оптичних систем, при якій лінійне збільшення змінюється по полю зору, порошуючи подібність між об'єктом та його зображенням.

Віньєтування – затемнення зображення по краях кадру в фотографії і оптиці в наслідок ослаблення частини світлового потоку, що проходить під більшим кутом по відношенню до оптичної осі в оптичній системі.

Муаровий візерунок – інтерференційний візерунок, створений, наприклад, при накладенні двох періодичних сітчастих малюнків. Явище зумовлене тим, що елементи двох малюнків, що повторюються, слідують з трохи різною частотою і то накладаються один на одного, то утворюють проміжки.

Color cast – явище, при якому усе зображення або його частина набуває певного небажаного кольорового відтінку.

Depth Of Field – це відстань між найближчими та найвіддаленішими об’єктами на фотографії, яка здається прийнятною різкою.

ВСТУП

Редагування зображень за допомогою засобів машинного навчання та штучного інтелекту є актуальним напрямком у галузі обробки зображень. Через надзвичайну популярність соціальних мереж, у яких основним контентом є зображення (Instagram, PicsArt, DevianArt, Flickr), велика кількість людей щоденно вдаються до редагування власних фото за допомогою програмного забезпечення. Також фото-редагування активно використовується у великому бізнесі (інтернет-магазини з каталогами товарів, веб-сайти, кіноіндустрія, Youtube, фото-магазини та багато інших).

Традиційні фото-редактори засновані на стабільних математичних алгоритмах та повертають однозначні результати, проте є обмежені власними алгоритмами, які можуть бути використані лише для вирішення дуже конкретних локальних задач, таких як зміна віддтінків на усьому зображенні, маніпуляції з насиченістю, накладання декількох зображень, посилення та послаблення тіней, шумів, тощо.

Якщо ж користувач бажає застосувати більш складні зміни до зображення, наприклад усунути наявні на зображенні недоліки, то він повинен якісно орієнтуватися у професійних програмних засобах, виконати велику кількість простих математичних операцій для отримання бажаного результату, що є досить складною задачею для звичайного користувача. Більш складні задачі у галузі обробки фото не можуть бути розв’язані за допомогою стандартних алгоритмів, адже такі алгоритми не враховують зміст зображення, а лише застосовують матричні перетворення, задані за замовчуванням. Тому зараз для вирішення подібних задач активно використовують алгоритми штучного інтелекту, так як вони є більш

гнучкими та чутливими до змісту і мають потенціал бути адаптованими до майже будь якої задачі шляхом налаштування та навчання.

На сьогодні існує кілька рішень по редагуванню фото, які базуються на нейронних мережах та алгоритмах машинного навчання, які виконують задачі стилізації чи редагування фото(такі рішення як Prizma, Vinci, FaceApp, Reflect). Також нейронні мережі починають з'являтися у професійних програмах, для полегшення життя користувачів, проте таке ПЗ і досі залишається досить складним для опанування, тому звичайні користувачі надають перевагу простим та зрозумілим мобільним додаткам.

На основі проведеного дослідження було виявлено, що на ринку програм для обробки зображень не існує продуктів, що дозволяють користувачу одночасно автоматично і високоякісно редагувати власні фотографії. Аналогічні програмні засоби є або складними у використанні, або не є досить гнучкими, щоб надати користувачеві якісні результати роботи.

Дана робота присвячена дослідженню цифрових зображень, дефектів та способів їх уникнення шляхом застосування алгоритмів штучного інтелекту та машинного навчання, а саме технології генеративних змагальних мереж, у контексті їх використання з метою автоматизувати, покращити та прискорити оброблення зображень. Дуже велика кількість досліджень на ці теми вже була виконана і велика кількість таких проблем вже була вирішена за допомогою якісно навчених нейронних мереж, такі дослідження також будуть описані у магістерській дисертації. Для даної магістерської дисертації було вирішено розробити та протестувати технологію обробки зображень для розв'язку задачі інтелектуальної колоризації, так як дана задача досі є актуальною, а існуючі методи можна значно покращити.

1. АНАЛІЗ МЕТОДІВ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

1.1. Види цифрових зображенньь

Метою цієї магістерської дисертації є дослідити існуючі на сьогодні методи обробки зображень, їх якість та швидкість. Однією з основних характеристик цифрових зображень є колір, отже, для початку необхідно розкрити математичний зміст кольорів та кольорових схем.

Колір у зображеннях подається у вигляді числа або декількох чисел, в залежності від обраної колорової схеми, і вже на засобі відображення інформації, числа інтерпретуються як колори. За допомогою одного байта (8 біт) можна закодувати 256 кольорів. При цьому числу 0 відповідає абсолютно чорний колір, а 255 – абсолютно білий, а усі числа поміж ними інтерпретуються як відтінки сірого кольору (якщо взяти за основу монохромну кольорову схему).

Існує багато варіантів кодування кольорів, ці варіанти називаються кольоровими схемами. Кожна окремо взята кольорова схема відповідає за виведення зображення, з опором на певні характеристики (яскравість, різниця між кольорами, контраст, тощо). Найбільш поширеною кольоровою схемою є RGB, що описує спосіб синтезу кольору, за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Колір пікселя задається як поєднання трьох каналів кольорів (червоного – R, зеленого – G, синього – B) у різних співвідношеннях, завдяки чому дана кольорова схема може відобразити 16,7 мільйонів різних кольорів та їх відтінків [1].

Растрові зображення використовуються у всіх випадках, коли необхідно відтворити аналоговий оригінал: фотографію, малюнок чи складний елемент оформлення, який нераціонально конвертувати у

векторний вигляд. Загалом растрові зображення – найбільш поширений вид зображень, що активно використовується на телебачення та на веб-ресурсах. Відеозапис – це массив растрових зображень, які відтворюються послідовно один за одним з непомітною для людського ока затримкою (за це відповідає параметр FPS – frames per second) [1].

При аналого-цифровому перетворенні завжди відбувається втрата деякої кількості інформації, оскільки дискретизація завжди проводиться шляхом усереднення та узагальнення потоку вхідної аналогової інформації. Звідси витікає основний недолік растрових цифрових зображень – неможливість їх масштабування без втрати якості. Приклад цього недоліку наведено на рис. 1.1. Даний недолік на сьогодні можна усунути багатьма методами, основним з яких є інтерполяція зображення до зображення більшого розміру. Також активно набирає обертів збільшення розміру і якості зображення за допомогою нейронних мереж, state-of-art з яких на сьогодні є технологія Super Resolution GAN [2]. Усунення такого дефекту є дуже важливим, тому вже існують технологічні стартапи, які виконують покращення якості зображень у комерційному масштабі, одним з яких є Київська компанія Let’s Enhance [3].

Іншим видом цифрових зображень є векторні зображення. Найменшими одиницями векторного зображення є вектори і криві Безьє. Вектор у комп'ютерній графіці – це відрізок, що з'єднує дві точки з заданими координатами. Основними керувальними елементами кривої Безьє є вузол (node), контрольна точка (CP, control point) або контрольна вершина (CV, control vertex). Ступінь кривизни лінії визначається координатами вузла і двох керувальних точок [4].

Рис. 1.1. Втрата якості при збільшенні розмірності
оригінального зображення [1]

Контур елемента векторного зображення в цифровому вигляді представляє собою масив даних, що містить координати контрольних та керувальних точок, а також характеристики кривої в цілому – її товщину, колір, напрямок, а якщо крива замкнута – то й колір і тип заливки.

Примітиви представляють собою прості геометричні форми, які в масиві даних кодуються цілком, без поділу на криві Безьє і вектори, умовним кодом тієї чи іншої геометричної фігури, а також кодами розміру фігури, її координатами, кодами типу і кольору заливки фігури, товщини і кольору контуру та інших характеристик. Такими примітивами є трикутник, коло, прямокутник, загалом – прості геометричні фігури [4].

Інформація про текст, що може бути на зображенні, як правило, буває представлена у вигляді кодів символів, що супроводжуються цифровою інформацією про візуальні характеристики текстового блоку – гарнітуру шрифту, накреслення, колір контуру і заливки, способи заливки, методи вирівнювання тексту в блоці тощо [4].

Векторні зображення отримують двома способами – шляхом ручного трасування оригіналу і шляхом автоматичного трасування. При ручному трасуванні художник або дизайнер фактично сам створює зображення, ніби обводячи уявні контури, за допомогою графічного редактора створюючи вектори, криві Безьє і графічні примітиви. Автоматичне трасування оригіналу

проводиться за допомогою програмного забезпечення, яке за допомогою інтелектуальних алгоритмів розпізнає контури оригіналу, поданого растровим зображенням, і на основі отриманої інформації створює лінії, заливки тощо таким чином, щоб з них склалося векторне зображення, максимально близьке до оригіналу. Такі технології наявні у професійних фото-редакторах як Adobe Photoshop.

Основна перевага векторного зображення – це можливість масштабування без втрати якості. Ще однією перевагою векторних зображень є порівняно невеликий розмір файлів.

Головний недолік векторних зображень – це те, що вони майже завжди відтворюють оригінал у спрощеному вигляді. Деякі деталі оригіналу буває неможливо відтворити у векторному зображенні.

Основна сфера застосування векторних зображень – це друковані ЗМІ, та уся поліграфія загалом (плакати, постери, друк на одязі, тощо) [4].

Цифрові зображення змішаного типу являють собою масиви даних, що містять інформацію як у вигляді матриці пікселів, так і у вигляді опису векторів, кривих Безьє, примітивів і текстових блоків [1].

В основі вертикальної структури векторно-растрових зображень лежить поняття шару (layer). Шар – це область даних, що містить інформацію про окремий елемент вертикальної структури зображення. Також поняття шарів лежить у основі фото-редакторів – там кожні окремі аспекти зображення можна виділити у окремий шар, та здійснювати певні операції окремо від загального зображення, що редагується. Підхід редагування зображення з використанням шарів вже є активно використовуваним в усіх фото-редакторах, також він використовується у цій дисертації при поданні даних у нейронні мережі для навчання (окремо подаються різні частини зображення,

сегментаційні мапи зображень, кольорові підказки для нейронної мережі, тощо).

Векторно-растрові зображення отримують з вихідних векторних і растрових елементів шляхом зведення за допомогою графічних редакторів. Також умовно до зображень змішаного типу слід віднести результати роботи програм комп'ютерної верстки, в яких основними векторними елементами виступають текстові блоки. Зображення змішаного типу поєднують в собі переваги й недоліки тих типів зображень, які присутні в них у вигляді елементів (шарів).

Основною перевагою зображень змішаного типу є можливість вільного редагування кожного шару окремо, а основним недоліком – великий обсяг масиву даних і, відповідно, кінцевого файлу [1].

Під час підготовки макетів векторні шари використовують для тексту і графічних символів (логотипів, товарних знаків, лого-груп, лінійок тощо), а растрові шари – для підкладок і фотозображень.

1.2. Види недоліків та дефектів на зображеннях

У даному підрозділі буде розглянуто існуючі проблеми, що стосуються якості зображень, та сформовано список недоліків та дефектів, наявних на зображеннях, а також існуючі алгоритми для їх усунення.

1.2.1. Оптичні дефекти зображень

Для початку розглянемо оптичні дефекти зображень, таких як хроматична аберація, дисторсія, віньєтування та передній та задній фокуси.

Хроматичні аберації – явище викликане дисперсією світла (розкладання променя світла на складові), що проходить через об'єктив. Справа в тому, що промені з різною довжиною хвилі (різних кольорів) заломлюються під

різними кутами (саме за цим принципом через трикутну призму з білого пучка світла ми отримуємо сім кольорів веселки). Хроматичні аберації призводять до зниження чіткості зображення і появи кольорових контурів (особливо на контрастних об'єктах) [5]. Приклад хроматичних аберацій наведено на рис. 1.2.

Рис. 1.2. Хроматичні аберації на фотографії [1]

Дисторсія – геометричне спотворення прямих ліній. Є два види дисторсии – подушкоподібна і бочкоподібна. Якщо прямі стали увігнутими – це подушкоподібна дисторсія, якщо опуклими – бочкоподібна. Дисторсія спостерігається у зум-об'єктивів при крайніх значеннях [5]. Приклад дисторсій наведено на рис. 1.3.

Рис. 1.3. Дисторсія на фотографіях з різних об’єктивів [5]

Віньєтування – це падіння яскравості від центру до країв зображення. Як правило, в центрі зображення немає затемнення, воно чітко видно лише на кутах. Віньєтування викликано конструктивними особливостями об'єктивів, через які обрізається світловий потік, який сильно відхиляється від осі об'єктива. Відповідно найсильніше віньєтування помітно на ширококутних об'єктивах. Так само віньєтування проявляється на об'єктивах з великою світлосилою, при максимальній діафрагмі. Віньєтування також використовується як художній елемент обробки зображень. Небажане віньєтування можливо доволі легко вручну усунути за допомогою балансування кольорів, висвітлення країв зображень [5].

Передній та задній фокуси виникають коли камера сфокусована на ближчий чи дальній об’єкт, через що на зображенні виникає розмиття деяких об’єктів. В ідеалі об'єктив посилає всі промені світла на одну площину, на якій знаходиться матриця. Але, якщо з якої-небудь причини об'єктив трохи зміщений – виходить така проблема [5]. Передній та задній фокуси наразі навпаки більше вважаються художнім прийомом, аніж недоліком, і наразі існують технології, влаштовані у сучасні смартфони, що здатні симулювати даний ефект чи навпаки прибирати його.

1.2.2. Дефекти растрових зображень

Дискретна структура растрового зображення приводить до певних спотворень зображень. Можна виділити два типи таких спотворень: ступінчастість ребер (“ефект сходів”) і некоректна візуалізація тонких деталей або фактури.

Основна причина появи “ефекту сходів” полягає в тому, що відрізки, ребра багатокутника, границі кольору і т. д. мають безперервну природу, тоді як растрове зображення є дискретним. Для подання відрізка, ребра багатокутника і т.д. у растровому вигляді, необхідно накреслити їх у дискретних координатах. В алгоритмах дискретизації відрізка і заповнення багатокутника, інтенсивність або колір пікселя визначаються інтенсивністю або кольором єдиною точки всередині області пікселя – його центру. Якщо центр знаходиться всередині пікселя, то активується весь піксель. Якщо центр знаходиться поза цих меж, то вся область пікселя ігнорується. В результаті виходить характерне ступінчасте, зазубрене ребро багатокутника або відрізок [6]. Приклад “ефекту сходів” зображено на рис. 4.

Другий тип дефектів растрових зображень це некоректна візуалізація тонких деталей або фактури – найбільш помітно проявляється при відображенні структур, що складаються з тонких ліній, розташованих паралельно або під деяким кутом. Якщо товщина ліній і відстань між ними менше або близькі до розміру пікселя, лінії зливаються, утворюючи деякі зафарбовані або незафарбовані зони складної форми, так звані муарові візерунки [7].

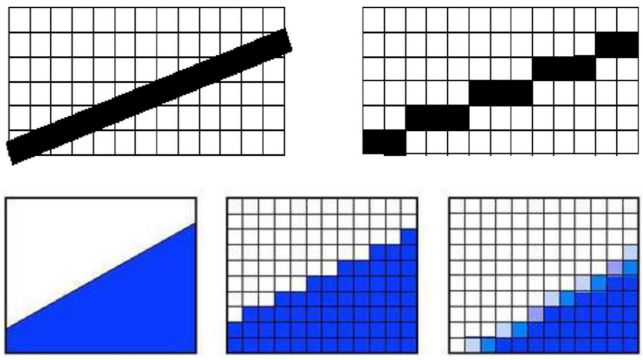


Рис. 1.4. “Ефект сходів” при растуванні відрізка
та країв заповненої фігури [6]

Ефективного методу боротьби з такими ефектами практично не існує, оскільки і метод усереднення, і збільшення дозволу растра можуть не дати необхідного ефекту. Тому при створенні растрових зображень слід уникати подібних регулярних структур. Приклад муарового візерунку наведено на рис. 1.5.

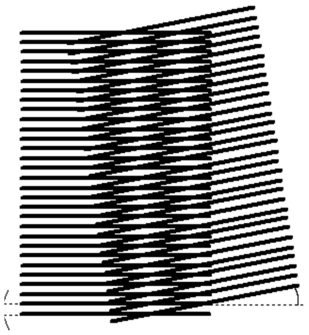


Рис. 1.5. Муаровий візерунок, утворений перетином ліній [7]

1.2.3. Інші недоліки та дефекти зображень

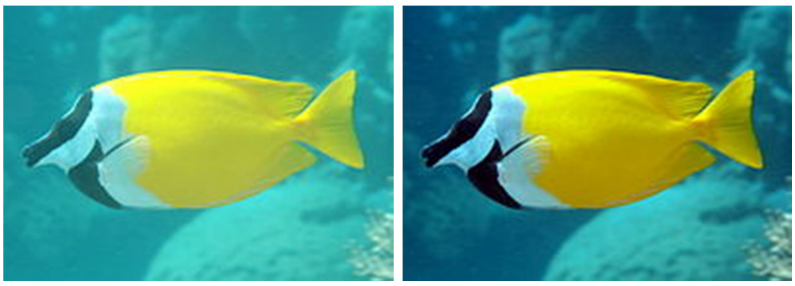
Також існує ряд інших недоліків, як оптичного походження, так і таких, що виникають вже під час обробки зображень комп’ютером. Одним з таких дефектів є накладання одного кольорового відтінку (color cast) на усе зображення.

Кольоровим відтінком є відтінок певного кольору, зазвичай небажаного, який впливає на все або на частину зображення рівномірно. Деякі типи світла можуть призвести до кольорового відтінку плівки та цифрових камер [8]. Підсвічування об'єкта джерелами світла різної колірної температури, як правило, призводить до виникнення проблем з кольором у тіні. Загалом, людське око не помічає неприродного кольору, тому що наші очі і мозок коригують і компенсують різні види світла так, як камери цього робити не можуть.

Поява кольорового відтінку може також відбуватися на старих фотографіях внаслідок вицвітання барвників, особливо під дією ультрафіолетового світла. Приклад наведено на рис. 1.6. Частково дана проблема буде вирішуватися у даній дисертації, адже загалом розроблена система виконує інтелектуальне коригування кольорів, тому відтінки на фото можуть бути змінені за бажанням користувача системи [8].

Дуже розповсюдженими також є такі недоліки як розмиття, неправильна глибина різкості, відблиски об’єктиву, природні та цифрові шуми.

Розмиття зображення викликано тремтінням камери чи переднім та заднім фокусами. Під час зйомки з меншою витримкою зображення часто розмивається [9].



а) б)

Рис. 1.6. Зображення з синім кольоровим відтінком (а), та виправлене вручну зображення з реалістичними кольорами (б) [8]

Щоб робити чіткі зображення та уникати розмиття, необхідно під час зйомки використовувати штатив, або займати максимально стабільне положення. Проте на сьогодні існує дуже багато технологій для усуненні розмиття вже на етапі цифрової обробки, починаючи від Unsharpening фільтрів, що підвищують різкість об’єктів на зображенні за допомогою операції згортки з заздалегідь визначеним фільтром, і закінчуючи потужними

згортковими нейронними мережами, що навчені “відновлювати” оригінальне зображення на основі його розмитого зразку. Приклад роботи наразі найбільш актуального на сьогодні такого алгоритму (DeblurGAN [10]) наведено на рис. 1.7.

Багато зображень мають дуже хороший предмет і композицію, але вони втрачають виразність образів та естетичний вигляд через невідповідну глибину поля. Глибина різкості (Depth Of Field) – це частина сцени, яка виглядає різкою на зображенні. Хороші зображення мають правильну глибину поля. Глибину поля можна контролювати на фотоапараті за допомогою параметрів діафрагми. Використовується велике значення діафрагми під час зйомки портретів і менше значення діафрагми під час зйомки пейзажів [11]. На сьогодні алгоритми для коригування глибини різкості влаштовано у більшість сучасних смартфонів та у вигляді окремих додатків для більш застарілих смартфонів, у Adobe Photoshop.

Рис. 1.7. Результат роботи технології DeblurGAN з усунення розмиття на фото [10]

Правильна глибина різкості є обов'язковою умовою для отримання найкращих зображень. Приклад правильно підібраної глибини різкості наведено на рис. 1.8.

Відблиски об’єктива створюються, коли світло, що не формує зображення, потрапляє у об'єктив та у зону чутливості камери. Зазвичай вони виглядають як полігональні світлі кулі, і дуже часто зустрічаються під час зйомки на сонці.

Рис. 1.8. Правильно підібрана глибина різкості для фотографії [5]

Відблиски об’єктива не тільки створюють небажані кулі на зображенні, але й зменшують контрастність зображення, спричиняють ефект, подібний до віньєтування. Приклад таких фото наведено на рис. 1.9.

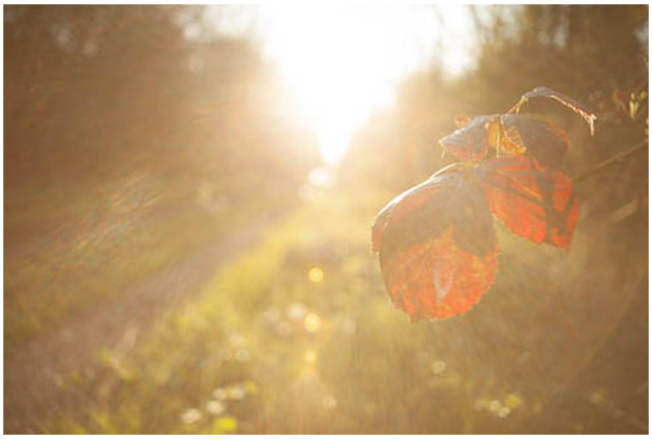


Рис. 1.9. Фото зі значним “засвітом” та відблисками об’єктива [11]

Поширеним явищем є поява шумів на зображеннях, збільшення та зменшення експозиції, та так званий “ефект червоних очей”.

Шум на зображенні – це випадкова, зазвичай небажана зміна яскравості або кольорової інформації на зображенні, природного чи цифрового походження (цифровий шум може виникати при компресії зображень та здійсненні певних операцій з редагування).

Деякий рівень шуму необхідний у зображенні, але більш високорівневі шуми можуть дати небажані результати. Шум на зображенні прямо пропорційний до швидкості ISO (світлочутливість камери), що обернено пропорційно освітленості. Висока швидкість ISO при низькій освітленості означає більше шуму на зображенні.

Збільшення експозиції виникає, коли надлишок світла падає на датчик чи плівку довше, ніж потрібно. Зображення зі збільшеною експозицією має білі ділянки, на такому зображенні втрачаються деталі підсвічування.

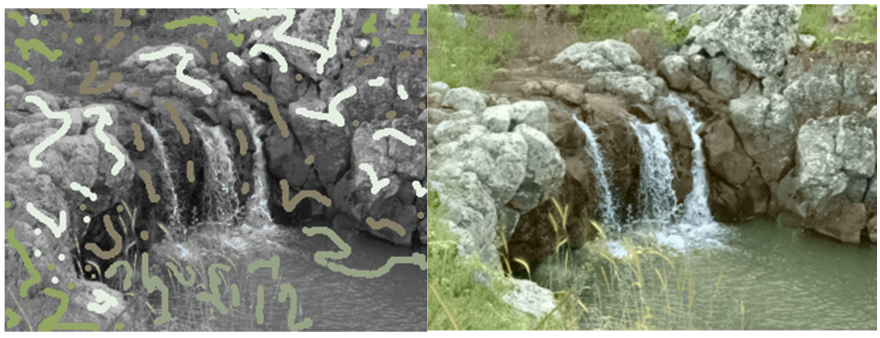
Зниження експозиції відбувається, коли світло падає на датчик чи плівку менше ніж потрібно. У зображенні зі зниженою експозицією темні частини не відрізняються від абсолютно чорних, тобто втрачають тіньові деталі.

“Ефект червоних очей” є найпоширенішою проблемою, з якою стикаються фотографи, використовуючи спалах. Червоні очі є результатом відображення світла з відкритої сітківки ока у людей та тварин. На сьогодні усунення даного дефекту також представлено у більшості фото-редакторів [12].

1.3. Задача інтелектуальної колоризації зображень

Загалом, як можна зрозуміти з попередніх підрозділів, існують різні види дефектів і майже щодня створюються нові технології для боротьби з

ними. Одна з цілей магістерської дисертації це розробити таку технологію, яка була б корисною для користувачів, економила їх час та енергію, тому було вирішено не намагатись оптимізувати вже й так добре працюючі алгоритми, а обрати головною метою дисертації вирішення певної більш складної та нішової проблеми. Однією з таких проблем є коригування кольорів на зображенні окремих об’єктів, реставрація старих монохромних фото та певні інші маніпуляції з кольорами. Коригувати кольори окремих об’єктів на фото є дуже кропіткою та монотонною роботою, адже потрібно виділяти вручну чи за допомогою інструментів фото-редакторів (таких як “Чарівна паличка” та “Авто-виділення”) кожен окремий об’єкт, потім вибрати колір у який хочете перефарбувати виділену ділянку фото, маніпулювати яскравістю та прозорістю новоствореного шару та враховувати темні та світлі ділянки об’єкта, щоб зображення мало реалістичних вигляд після корекції кольорів. На сьогодні є дуже мало способів вирішення цієї задачі, одним з них є алгоритм “Colorization using optimization” [13], детальніше принцип роботи якого буде описано у наступному розділі. Приклад задачі інтелектуальної колоризації та результат роботи цього алгоритму наведено на рис. 1.10.



а) б)

Рис. 1.10. а) Монохромне зображення з нанесеними користувачем помітками; б) результат дії алгоритму “Colorization using optimization” [13]

Задача реставрації старих фото взагалі вирішується тільки інтуїтивним способом, адже немає правдивої інформації про кольори об’єктів на таких фото. Проте задача реставрації старих фото вже доволі якісно вирішує нейронна мережа DeOldify [14], яка навчена на великій кількості різних високоякісних фото, щоб по чорно-білому варіанту зображення розуміти, який зміст зображення та співставляти цей зміст з найбільш вірогідними та реалістичними кольорами для об’єктів. Детальніше як працює дана технологія буде описано у наступних розділах, а приклад роботи цієї нейронної мережі наведено на рис. 1.11.

Рис. 1.11. "Terrasse de café, Paris" (1925), відреставрована за допомогою технології DeOldify [14]

Отже, можна зробити висновок, що задача колоризації окремих об’єктів є доволі актуальною, та потребує більш інтелектуального, не стандартного рішення. Якщо декомпозувати дану задачу на такі процеси як сегментація по об’єктам на зображенні, нанесення кольору та його відтінків на окремі об’єкти, врахування тіней, темних та світлих ділянок об’єктів, змішування кольорів, то є доволі очевидним, що стандартні математичні алгоритми, такі як нанесення фільтрів, алгоритм математичної хвилі для виділення об’єктів, то стане зрозуміло, що навіть потужна комбінація таких методів не покриє усі

можливі варіанти цієї задачі. А тому, у даній магістерській дисертації висувається гіпотеза, що реалізувати технологію, що буде успішно вирішувати проблему колоризації зображень можливо за допомогою найбільш сучасних методів штучного інтелекту для роботи з зображеннями, а саме – згортковими та генеративними змагальними мережами.

1.4. Наукова постановка задачі

В рамках визначеної наукової проблеми, для проведенняуспішного дослідження, необхідно проаналізувати існуючі методи для покращення якості зображень, як в цілому, так і у контексті інтелектуальної колоризації зображень, дослідити стандартні математичні методи та методи штучного інтелекту для вирішення задачі та розробити власні методи на основі згорткових та генеративних змагальних мереж, такі, які дозволятимуть виконувати інтелектуальну колоризацію зображень напівавтоматичним шляхом на основі простих підказок з боку користувача за оптимальний час та повертатимуть результати високої якості.

На основі наукової проблеми було зроблено більш детальний план виконання магістерської дисертації, за яким необхідно вирішити поставлену наукову задачу.

Для успішного виконання магістерської дисертації, необхідно:

1. Детально, з урахуванням усіх внутрішніх нюансів, дослідити існуючі математичні методи, таких як накладання фільтрів та ітеративні алгоритми підрахунку освітленості об’єктів, визначитись з тим, яка кольорова схема є математично найбільш придатною для виконання обробки нейронною мережею, які методи попередньої обробки є найбільш корисними для тренування нейронної мережі.

2. Знайти актуальні для навчання набори даних з великою кількістю різних зображень, що містять велику вибірку різних образів різних кольорів різної якості та створених різними пристроями (смартфони та камери різних виробників з різними оптичними параметрами та параметрами зйомки). Сформувати з цих наборів даних навчальну та тестувальну вибірку даних великого обсягу.
3. Дослідити як працюють існуючі архітектури нейронних мереж для задачі обробки зображень в цілому, так і для задач інтелектуального нанесення кольору на окремі об’єкти (задача сегментації). Дослідити сучасні методи градієнтного спуску для проведення максимально ефективного тренування мережі. Визначити емпіричним шляхом найбільш оптимальні параметри для шарів нейронних мереж (stride, padding, kernel size, activation functions) та їх оптимізаторів градієнтного спуску (learning rate, beta, momentum).
4. На основі проведених досліджень розробити власну архітектуру мережі генератора (згорткова нейронна мережа, що відповідає за обробку зображень) та мережу дискримінатора (згорткова нейронна мережа, що виконує інтелектуальну оцінку роботи мережі генератора та контролює якість згенерованих зображень). Альтернативним варіантом є взяти за основу одну з існуючих архітектур та накласти власні модифікації для підлаштування існуючої архітектури під конкретну наукову задачу.
5. Натренувати створені моделі на обраних даних та провести детальну оцінку роботи шляхом тестування на нових зображеннях, тестування користувачами та надати математичну оцінку роботи моделей і обрати найбільш якісну модель для подальшого використання та майбутніх досліджень.

6. Виконати оптимізацію обраної моделі для стабільної та швидкої роботи у реальних випробуваннях.

1.5. Висновки

Проаналізувавши існуючі види зображень, було вирішено працювати з зображеннями растрового типу, адже вони є більш зручними для декодування та виконання над ними математичних операцій, також вони є найбільш поширеним типом зображень, що означає, що буде доволі легко знайти необхідні дані для навчання мереж, а також, що розроблені для них технології будуть актуальні та вживані. Також при навчанні системи нейронних мереж будуть використовуватися техніки різних шарів, для подачі різної важливою інформації до мережі більш відокремлено (зміст зображення планується відділяти від інформації про кольори).

Також було досліджено недоліки та дефекти різних зображень та розгляно існуючі методи для боротьби з ними. Було сформульовано мету дослідження, конкретизовано технологію, яку необхідно імплементувати для досягнення поставленої мети.

З багатьох розглянутих проблем, наявних у растрових зображеннях, деякі проблеми легко та якісно розв’язуються за допомогою вже існуючих програмних засобів. Наприклад такі дефекти як “ефект червоних очей”, “ефект сходів”, зайві шуми, розмиття на зображенні усуваються простими фото-редакторами досить якісно та без складних маніпуляцій з боку користувача.

Розглянувши існуючі дефекти зображень та запропоновані загало на ринку програмні рішення для тої чи іншої проблеми, було помічено, що майже немає якісних та зручних рішень для роботи з кольорами на зображенні. Більшість таких методів дозволяють просто накладати кольори

на усе зображення без урахування контексту (color overlay), чи коригувати кольори (баланс білого, тіні, віньєтування), проте задача з перефарбування окремих об’єктів без псування загального вигляду фото наразі залишається досить важким та кропітким процесом, який потребує використання різних масок, пензлів, декомпозицію зображення та інших маніпуляцій, що є досить важким для опанування та затратним по часу процесом. Також на сьогодні існує ціла окрема професія – колорист, суть якої складається саме у органічній зміні кольорів на зображеннях та відео.

Тому було вирішено взяти за основу дослідження саме задачу “інтелектуальної колоризації” та розробити власну технологію на основі нейронних мереж, що дозволятиме виконувати напівавтоматичне розфарбування зображень за допомогою дуже простих операцій нанесення кольорових мазків на об’єкти, які користувач бажає перефарбувати, після чого нейронні мережі будуть інтерпретувати цю інформацію і виконувати зміну кольорів тільки на бажаних частинах зображення.

З технологічної точки зору, результатом дослідження буде програмне забезпечення з мінімальним користувацьким інтерфейсом, що дозволяє виконувати колоризацію зображень, шляхом простих маніпуляцій з боку користувача.

Програма буде розбита на модулі, кожен з яких буде створений на основі різних технологій та алгоритмів (нейронні мережі, стандартні математичні алгоритми, та їх поєднання), і кожен з яких буде виконувати свою роль у обробці зображення.

Приблизний опис системи виглядає наступним чином:

1. Модуль взаємодії з користувачем – користувач може завантажити власне зображення та нанести кольорові помітки на ті частини зображення, які він бажає перефарбувати.

2. Модуль попередньої обробки зображення – завантажує зображення та мапу кольорів, що була створена користувачем та форматує цю інформацію у потрібний вигляд для обробки нейронною мережею (зменшує розмір зображення, змінює кольорову схему, тощо).
3. Модуль нейронної мережі – інформація, подана до нейронної мережі обробляється нейронною мережею та повертається нова інтерпретація цієї інформації, зі зміненими згідно з кольоровою мапою кольорами.
4. Модуль фінальної обробки – відбувається кінцеве перетворення згенерованого нейронною мережею зображення.

2. АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КОЛОРИЗАЦІЇ ЗОБРАЖЕНЬ

Колоризація – це комп'ютерний процес додавання кольору до монохромного зображення або фільму. Зазвичай процес включає сегментацію зображень на регіони та відстеження цих регіонів через послідовності зображень [13]. Жодне з цих завдань не може бути якісно та швидко виконано на практиці звичайними алгоритмами; колоризація вимагає значного втручання користувача та залишається кропітким, трудомістким та дорогом по часу процесом. Більшість алгоритмів для обробки зображень базуються на технологіях комп'ютерного зору та базових математичних поняттях.

2.1. Алгоритм "Colorization using optimization"

Задачу інтелектуальної колоризації можна вирішити без застосування складних у розробці та тренуванні алгоритмів штучного інтелекту. У основі алгоритму лежить поняття "світлоти" об'єктів одного кольору. Саме за цим показником, можна зрозуміти яким саме чином проводити колоризацію, де є межі різних об'єктів та їх частин різного кольору. Основною гіпотезою, висунутою та частково підтвердженою у даному дослідженні є те, що сусідні пікселі, що мають близькі один до одного значення інтенсивності ("рівень сірого"), належать до одного сегменту зображення (отже такі пікселі у більшості випадків належать до одного об'єкта на зображенні).

Для знаходження та підрахунку нових кольорів для пікселів одного об'єкта, даний алгоритм використовує квадратичну функцію втрат, що записана у формулі (2.1) [13]. Для успішної роботи алгоритму, від користувача потребується нанести на чорно-біле зображення кольорові

штрихи та подати їх на вхід для виконання алгоритму, також при використанні необхідно встановити кількість ітерацій алгоритму, яку для кожного зображення доводиться підбирати емпіричним шляхом, адже зображення бувають різного розміру, сегменти які необхідно розфарбувати також відрізняються тим, яка саме кількість ітерацій буде достатньою для їх повного розфарбування.

У рамках цього алгоритму розробники працювали у кольоровому просторі YUV, ця кольорова схема зазвичай використовується у відео. Кожен канал такого зображення демонструє власні окремі характеристики: де Y – це монохроматичний канал яскравості (світлота), який надалі фігурує у дисертації як інтенсивність, тоді як U і V – канали хромування, які зберігають оригінальні значення кольорів. Алгоритм отримує у якості вхідних даних массив інтенсивностей зображення $Y(x, y, t)$ і повертає на вихід два масиви кольорових характеристик $U(x, y, t)$ і $V(x, y, t)$. Для зручності триплети параметрів (x, y, t) ми надалі позначатимемо літерами r та s [13].

Таким чином, $Y(r)$ – це інтенсивність конкретного пікселя. Метою алгоритму є накласти такі обмеження, щоб два сусідніх пікселя r та s мали близькі одне до одного кольори, при умові, що їх інтенсивності подібні. Таким чином випливає, що нам необхідно мінімізувати різницю у кольорі між значенням кольору $U(r)$ у пікселі r та середньозваженим значенням кольорів сусідніх пікселів.

$$J(U) = \sum_{r} (U(r) - \sum_{s \in N(r)} w_{rs} U(s))^2 , \qquad (2.1)$$

де w_{rs} – це функція зважування, сумма якої дорівнює одиниці, або більше одиниці, коли $Y(r)$ подібна до $Y(s)$ і менше одиниці, якщо значення цих двох інтенсивностей є різними. Подібні функції зважування широко використовуються в алгоритмах сегментації зображень, де їх зазвичай називають функціями спорідненості. У ході дослідження було проаналізовано

дві різні вагові функції. Найпростіша з них зазвичай використовується алгоритмами сегментації зображень та заснована на різниці у квадраті між двома інтенсивностями [13]:

$$w_{rs} \propto e^{-(Y(r)-Y(s))^2/2\sigma_r^2}. \tag{2.2}$$

Друга функція зважування заснована на нормованій кореляції між двома інтенсивностями:

$$w_{rs} \propto 1+\frac{1}{\sigma_r^2}(Y(r)-\mu_r)(Y(s)-\mu_r), \tag{2.3}$$

де μ_r і σ_r^2 – середнє значення та дисперсія інтенсивності у вікні навколо пікселя *r*. В одному кадрі ми визначаємо два пікселі як сусіди, якщо на зображенні вони розташовані поблизу [13].

Спочатку ми визначаємо заданий користувачем набір штрихів r_i, де вказані кольори користувачем $u(r_i) = u_i$, $v(r_i) = v_i$ та намагаємось мінімізувати функції $J(U)$ та $J(V)$ до цих обмежень. Оскільки функції втрат є квадратичними а обмеження лінійними, ця проблема оптимізації дає велику, розріджену систему лінійних рівнянь, яку можна вирішити за допомогою стандартних методів [13].

Загалом даний алгоритм показує високу якість роботи на зображеннях, проте є занадто повільним для того, щоб використовуватись у повсякденному житті (на фото 1024 на 1024 пікселі час виконання становитиме приблизно 5-7 хвилин, залежно від встановленої кількості ітерацій). Проте даний метод містить дуже перспективні та потенційно корисні ідеї для використання під час тренування нейронних мереж. З проведеного аналізу методу було виявлено, що для роботи з кольорами зображення, необхідно використовувати більш релевантну до задачі кольорову схему, аніж стандартні нотації, такі як RGB чи BGR, а саме формати YUV та LAB, як було продемонстровано у цьому дослідженні.

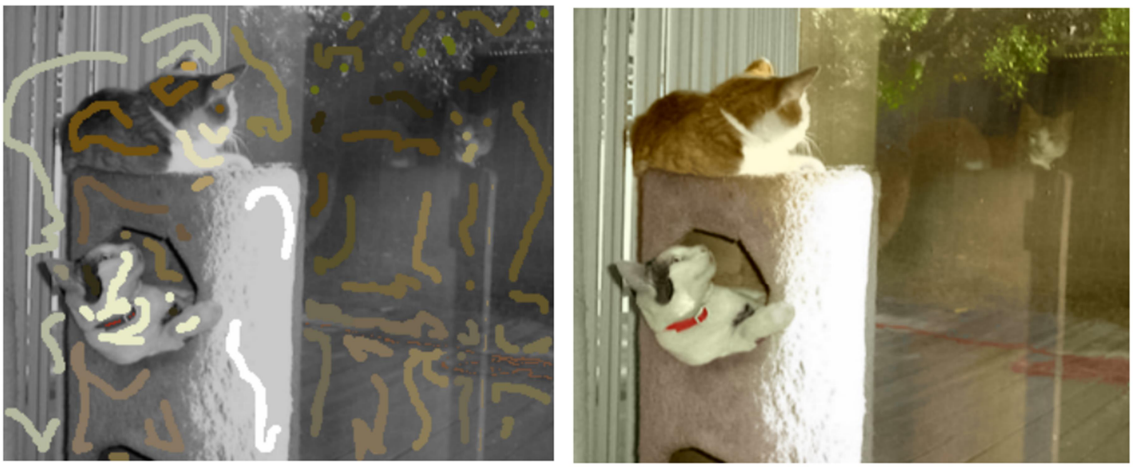


Рис. 2.1. Результат роботи алгоритму “Колоризація шляхом оптимізації” [14]

Головною перевагою цього алгоритму є те, що він не потребує від користувача виконання операції сегментації бажаних об’єктів для розмалювання вручну, що є його перевагою та «ноу-хау» для алгоритмів без використання більш складних технологій, таких як штучний інтелект.

Одночасно головним недоліком цього алгоритму є його швидкість – час виконання залежить напряму від кількості ітерацій перерахунку сусідніх пікселів, а отже алгоритм має дуже високу складність ($O(n^2)$) і навіть для невеликого зображення розміром 256 на 256 пікселів, потребує близько 5000 ітерацій та 3-4 хвилини на їх здійснення. Також недоліком алгоритму для вирішення поставленої у даній магістерській дисертації задачі є те, що алгоритм не виконує color preserving, або збереження кольорів тих об’єктів оригінального зображення, які користувач не має наміру перефарбовувати, для цього необхідно спочатку вручну виділити ті частини зображення, які не повинні бути пофарбовані, виконати розділення зображення на такі шари, та застосувати алгоритм тільки на обраних сегментах, а потім з'єднати шари назад для отримання фінального результату. Це підводить нас до висновку, що для вирішення поставленої задачі необхідне більш швидке та інтелектуальне рішення.

2.2. Згорткові мережі для обробки зображень

На сьогодні стають дуже актуальними технології по обробці зображень, які базуються на згорткових нейронних мережах, для розгляду таких технологій та їх подальшого аналізу, необхідно визначити, що таке згорткова нейронна мережа, чим вона відрізняється від інших типів нейронних мереж та з яких шарів може бути утворена. Також необхідно розглянути переваги та недоліки цих мереж, у порівнянні з іншими алгоритмами обробки зображень.

2.2.1. Згорткові нейронні мережі

Згорткові нейронні мережі – це окремий вид глибоких нейронних мереж (глибокою нейронною мережею вважається мережа, що має більше одного прихованого шару), призначений для роботи з мультимедійними даними (аудіо,відео, зображення) і активно використовується у сфері комп'ютерного зору (задача, поставлена у даній дисертації також відноситься до задач комп'ютерного зору). Такі алгоритми активно використовуються для задачі класифікації (приклад: відрізнити кота від собаки на зображенні), регресії (визначити об'єкт та його межі на зображенні – подібний підхід може використовуватися у системах безпеки з камерами спостереження, також – для реалізації систем штучного інтелекту для автоматичного керування автомобілем та іншими транспортними засобами), задача розпізнавання (смартфони розпізнають голос власника, камери спостереження ідентифікують людей коли ті опиняються у кадрі), задача сегментації (активно використовується у медицині для аналізу знімків). Також можна до таких задач віднести задачу редагування та генерації даних (сучасні методи обробки фото, нейронні мережі для автоматичної генерації витворів мистецтва, тощо). Усе це стало можливим саме завдяки розвитку ідей, закладених ще у 80-х роках минулого сторіччя (саме тоді і утворилася

концепція згорткових мереж), та значному технологічному розвитку, що дозволяє виконувати величезну кількість обчислень на процесорах GPU і обробляти фото високої якості [15].

Загалом, усі базові концепції нейронних мереж присутні у згорткових, проте ці мережі було адаптовано для виконання обробки та аналізу зображень. Окрім стандартно-наявних механік для зворотного поширення помилки (backpropagation) методами градієнтного спуску, шарів подачі та виведення даних та функцій активації, у згорткових нейронних мережах є власні, властиві тільки такому типу мереж види прихованих шарів (hidden layers). Варто розглянути деякі з цих типів шарів більш детально [15].

Шар згортки (Convolutional Layer). Аудіо, зображення та відео є неструктурованими даними, які містять у собі різноманітні шуми та велику кількість математично неоднозначних сигналів (людина може відрізнити кота від собаки на фото дуже просто, адже головний мозок одразу містить всю необхідну для цього інформацію і сам процес мислення має не математичну природу, проте орієнтуючись тільки на значення пікселів зображення неможливо визначити усі фактори та нюанси за допомогою простих математичних операцій чи многорівневого персептрона) [16]. Отже, для аналізу такої інформації, необхідно визначати певні математичні залежності та найбільш важливі особливості даних. Для чого і використовуються шари згортки. По суті шари згортки отримують на вхід інформацію у матричному вигляді, застосовують фільтри, що змінюються протягом навчанні, які підсилюють сигнали, що мають більший вплив, та послаблюють (відрізають) сигнали, які не впливають на дані. Таким чином, проходячи шари згортки, вхідні дані поступово зменшуються у розмірі та формі після кожного шару, і залишається тільки важлива інформація, яка поступає на наступне “відсіювання”. Математично, операція згортки це поелементне перемноження

матриці, та формування за певними правилами нової матриці, у яку записується отримана від перемноження сума. Перша матриця – це вхідна інформація, друга матриця – це фільтр, що “накладається” на інформацію. У кожному шарі нейронної мережі може бути виконано перемноження вхідної інформації на велику кількість різних фільтрів. Усі результуючі матриці після перемноження проходять конкатенацію.

Накладання фільтрів виконується наступним чином:

- заданий фільтр починає рухатись по матриці вхідних даних з лівого верхнього кута;
- виконується перемноження частини вхідної матриці, що співпадає за розміром з фільтром, на фільтр;
- отримані значення записуються у нову матрицю;
- фільтр пересувається на значення stride (відступ), що встановлене у параметрах шару у праву сторону, якщо рухатись далі нікуди, то фільтр пересувається на значення відступу вниз і починає цикл заново з лівої сторони.

Також у шарах згортки часто задають параметр padding, який по суті відповідає за штучне створення навколо вхідної матриці додаткових рядків та колонок з нулями.

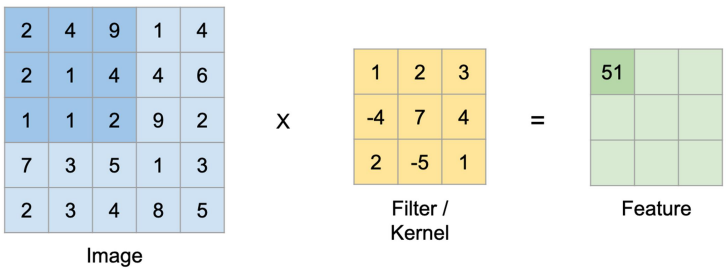


Рис. 2.2. Приклад роботи операції згортки [16]

Робиться це для того, щоб числа, що розташовані по краях вхідної матриці більше впливали при операції згортки, адже інакше математично, ця інформація на крайніх рядках та стовпцях ризикує бути проігнорованою [16].

Шар об'єднання (Pooling Layer). Інший вид шарів, що наявні майже у будь-якій архітектурі згорткових нейронних мереж, це шари об'єднання. Сутність цих шарів полягає в тому, щоб узагальнити отриману після шарів згортки інформацію та зменшити її розмір (знову ж таки, залишивши лише найвиразніші особливості інформації) [16].

Загалом, операція об'єднання відбувається наступним чином:

- визначається розмір "вікна", що рухається по вхідній матриці;
- зліва на право, та згори вниз "вікно" рухається з заданим відступом, та в залежності від типу об'єднання записує дані у результуючу матрицю.

На сьогодні актуальними видами об'єднальних шарів є max pooling та average pooling. Max pooling з обраного алгоритмом "вікна" вхідної матриці знаходить максимальний елемент і записує у результуючу матрицю. Average pooling рахує середнє арифметичне елементів, та записує у результуючу матрицю. Науково було доведено, що для більшості задач оптимальним є саме max pooling, average pooling та інші види об'єднальних шарів використовуються на практиці дуже рідко. Також математично max pooling підкреслює тільки найбільш високорівневі особливості інформації, коли при використанні average pooling є ризик провести окрім важливої інформації ще й зайві низькорівневі особливості та шуми. Приклад роботи обох варіантів наведено на рис. 2.3.

Повністю з'єднаний шар (Fully Connected Layer). Даний вид шарів, використовується у згорткових мережах після блоків шарів згортки та об'єднання. Для обробки цим шаром, наша інформація компресується до

вигляду одномірного вектора [16]. Отриманий одномірний вектор по суті є набором даних для одно- чи багатошарового персептрона.

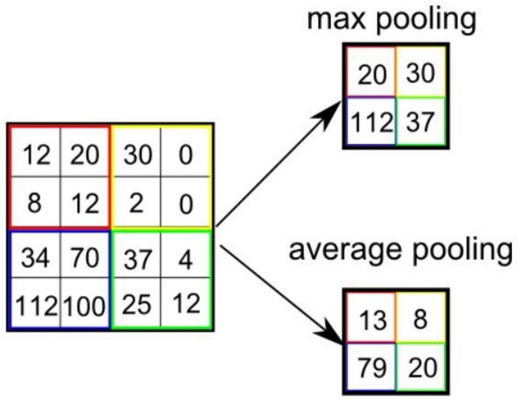


Рис. 2.3. Приклад роботи операцій об’єднання max pooling та average pooling [16]

По суті, отриманий вектор представляє собою певні особливості зображення різних рівнів і їх комбінація подається на вхід до повністю з’єднаного шару, який після перемноження матриць та виконання функції активації повертає вектор вірогідностей певних подій (для задачі класифікації це буде вектор з числами від 0 до 1, що демонструють вірогідність того, що той чи інший об’єкт наявний на зображенні). Тобто, основна ідея використання цих шарів у згорткових мережах, це знаходження взаємозв’язку між отриманими після згорток та об’єднань набору особливостей, та певною величиною, що репрезентує самі умови задачі (для класифікації, необхідно навчити мережу встановлювати відповідності між зображенням, та об’єктом, що на ньому наявний, для задачі регресії, окрім класифікації об’єкту, необхідно ще апроксимувати його координати на зображенні, і т.д.). Загалом, класичний вигляд згорткової нейронної мережі наведено на рис. 2.4.

Проте, якщо перед нами стоїть задача сегментації, обробки чи генерації даних, то наведений вище приклад не є достатнім. Для того, щоб окрім визначення особливостей різного рівня, застосовувати перетворення даних,

нам необхідно під’єднати розгорткову мережу (deconvolutional Neural Network). Така мережа складається з шарів роз’єднання (unpooling layer) та розгортки (deconvolution layer). Операції у цих шарах інверсні за дією до операцій об’єднання та згортки.

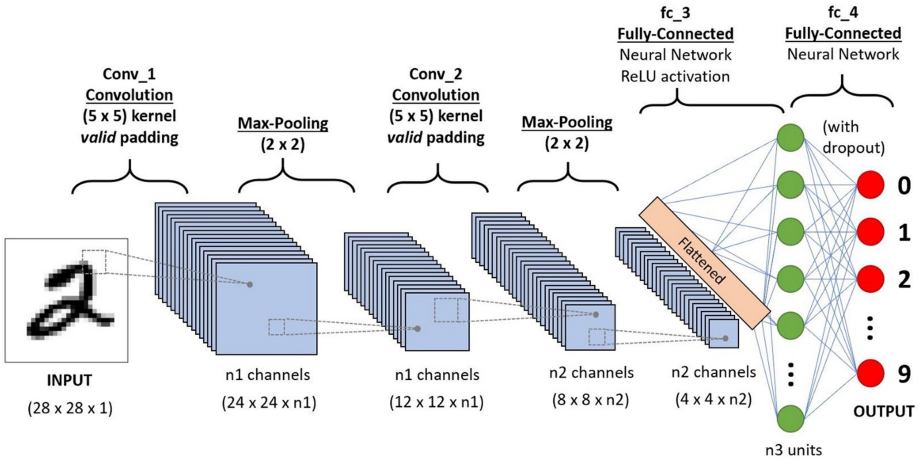


Рис. 2.4. Класичний вигляд архітектури згорткової нейронної мережі, для розпізнавання рукописних цифр [15]

Шар розгортки (deconvolution layer). Цей шар, працює аналогічно до шару згортки, проте для отримання у результаті матриці більшої розмірності, вхідну матрицю розширюють до певного необхідного розміру і заповнюють невідому інформацію одиницями. Внаслідок дії фільтрів, що навчаються протягом тренування, стає можливим якісно реконструювати скомпресовану інформацію [17].

Шар роз’єднання (unpooling layer). Даний шар працює обернено до шару об’єднання, отримуючи на вхід скопресовану інформацію, даний шар копіює її на мапу особливостей більшого розміру. Суть полягає у тому, що при виконання операції об’єднання, у нейронній мережі зберігається мапа локацій максимальних елементів (при використанні max pooling), яка

передається до шарів роз'єднання [17]. Завдяки цьому може бути забезпечена якісна реконструкція інформації. Приклад операції роз'єднання наведено на рис. 2.5.

Отже, при проєктування повної згорткової мережі (Full Convolution Network [18]), яка виконує як згортання так і розгортання вхідних даних, стає можливим змінювати ці дані з середини. Одним з таких прикладів є image2image translation алгоритми, суть яких полягає у тому, що при навчанні з учителем (наявні як вхідні дані, так і відповідні до них, очікувані результати), ставиться задача перетворити зображення з однієї категорії на зображення з іншої, зі збереженням контексту [17].

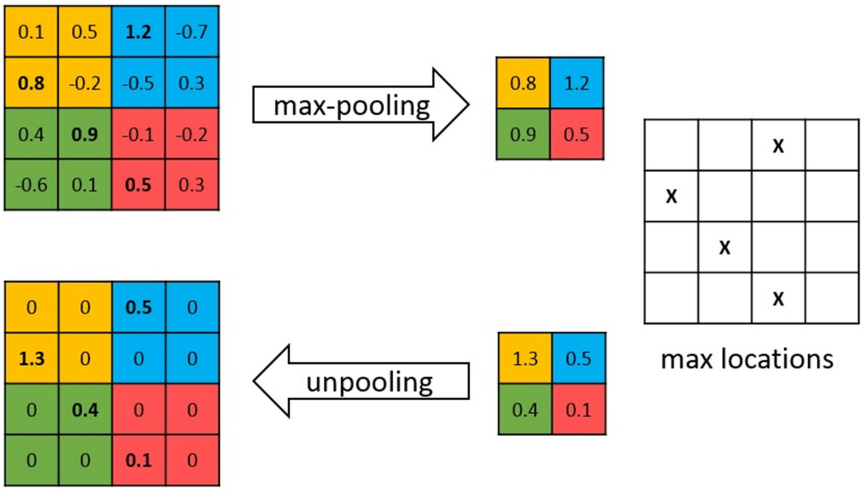


Рис. 2.5. Приклад операції роз'єднання [16]

Як приклад: обробка літніх фотографій та перетворення їх на зимні фото зі снігом; конвертація аерознімків на мапи будівель; перетворення знімків з відеореєстратора на мапу сегментів, які подаються на подальший аналіз. Результати роботи таких алгоритмів наведено на рис. 2.6.

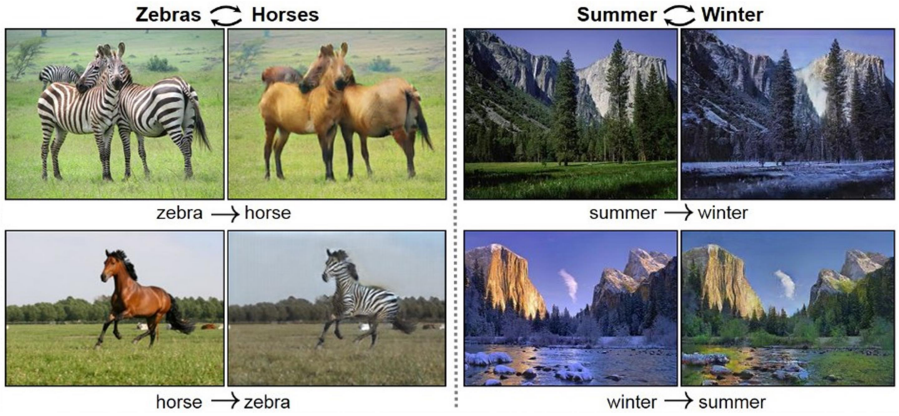


Рис. 2.6. Приклад роботи технології image2image translation для перетворення зебри на коня на зображення та зміни пір року [17]

Також, яскравим прикладом можливостей згорткових мереж у обробці зображень є так звані технології перенесення стилю (style transferring [19]), які за допомогою згорткових мереж виділяють певні абстрактні особливості стилю цільового зображення (кольори, текстури, штрихи) та за допомогою повної згорткової мережі виконують трансляцію цих особливостей на оригінальне зображення (із збереженням контексту – об’єктів та їх розташування, пропроцій та форм). Результат таких алгоритмів наведено на рис. 2.7.

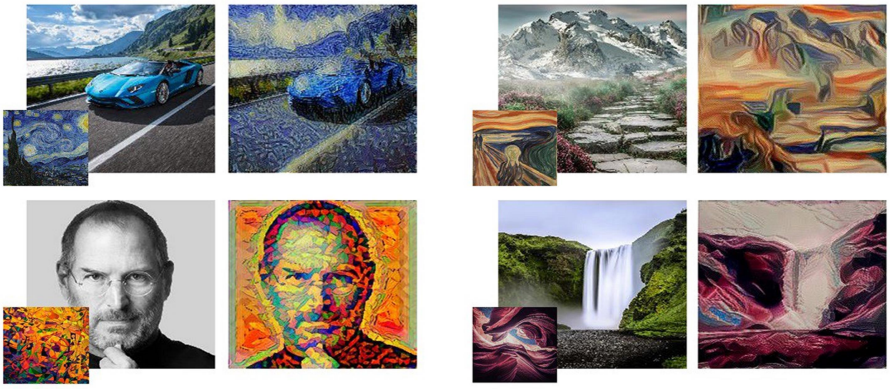


Рис. 2.7. Приклад роботи технології трансляції стилю (зліва на право: цільове зображення; оригінал; результат обробки) [17]

Загалом, головною особливістю згорткових нейронних мереж є здатність успішно фіксувати просторові та часові залежності в зображеннях, відео чи аудіо, та за допомогою цього більш якісно та успішно виконувати задачі класифікації, регресії, детекції та розпізнавання об’єктів. А повні згорткові нейронні мережі здатні також виконувати обробку та генерацію даних. Саме повні згорткові мережі і стануть основою для подальшої реалізації поставленої у дисертації наукової задачі.

2.2.2. Технологія Colorful Image Colorization

Дане рішення задачі колоризації було запропоновано у 2016-му році, основною ідеєю цього дослідження було навчити згорткову нейронну мережу автоматично перетворювати чорно-білі фотографії на кольорові, з реалістичними та насиченими кольорами без втручання користувача. Попередні дослідження нейронних мереж для вирішення цієї задачі були менш якісними і повертали розмиті, пересвічені чи дуже “димчасті” результати (фото виглядали “вицвілими”). Проте результати даного дослідження були значно кращими за попередні спроби, тому варто детально оглянути запропоновані техніки з навчання мереж та роботи з даними і можливо, використати їх у власній реалізації.

Дана технологія працює на основі однієї згорткової нейронної мережі для виконання колоризації. Мережу натреновано на більше ніж мільйона кольорових фотографій [20]. В рамках тестування якості роботи мережі було проведено тест Тьюринга (людям на вибір дають два зображення, одне – оригінал, інше – колоризоване нейронною мережею і люди намагаються вгадати де яке зображення) в рамках якого 32% людей помилилися у власних здогадках, що значно вище за результати попередніх досліджень.

Дані у мережу подаються переведеними у кольоровий простір LAB (*L* – світлота, *a*, *b* – інформація про кольори). Детальніше про цю кольорову схему буде розказано у четвертому розділі цієї дисертації. На вхід подається лише *L*-канал зображення, а канали *a* та *b* використовуються для порівняння наприкінці (мережа навчається за схемою “навчання з учителем” або supervised-learning) [20]. У попередніх дослідження результати були “вицвілими” через те, що у якості функції втрат для мережі використовувався класичний підрахунок Евклідової відстані (аналізує попіксельне відхилення результату від Ground Truth значень пікселів), тому можна було спостерігати певне усереднення кольорів для оптимізації заданої функції. Автори даної технології пропонують інший метод оцінки під час навчання, а саме функції втрат, вживану спеціально для задачі колоризації. Задовільні кольори для певних об’єктів можуть сильно варіюватися (яблуко на фото може бути червоним, жовтим чи зеленим), проте існують певні значення кольорів, які є неприродніми (те саме яблуко наприклад не може бути синім чи рожевим), тому для цього у мережі вивчається розподіл різних можливих кольорів, що відповідають різним образам, що зустрічаються у даних. Для цього loss-функція “перезважується” (відбувається певна зміна коефіцієнтів) під час навчання, а кінцевий результат формується шляхом підрахунку середніх значень у результуючому розподілі.

За основу для проєктування мережі була взята архітектура VGG [21] з більш поглибленими блоками згортки, яка повертає як результат прогнозований розподіл кольорів, який потім поєднується з вхідним каналом освітленості. Архітектуру мережі зображено на рис. 2.8.

Особливістю архітектури є те, що тут немає pooling-шарів, тільки блоки шарів згортки з застосуванням BatchNormalisation-операцій та функції активації ReLu.

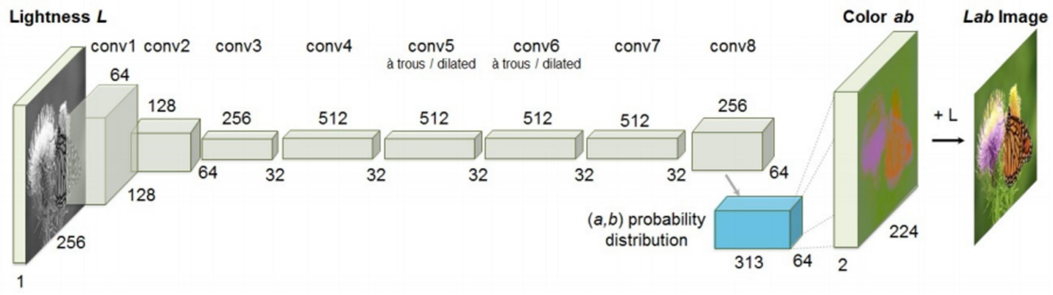


Рис. 2.8. Архітектура мережі для Colorful Image Colorization [18]

У якості функції помилок використовується наступна формула:

$$L_{cl}(\hat{Z}, Z) = -\sum_{h,w} v(Z_{h,w}) \sum_{q} Z_{h,w,q} \log(\hat{Z}_{h,w,q}), \tag{2.4}$$

де Z – розподіл кольорів оригіналу (попередньо закодованих до необхідного вигляду з каналів a та b зображення), $\hat{Z}$ – отриманий розподіл кольорів з нейронної мережі, $v(Z_{h,w})$ – терм для перезважування з урахуванням кожного окремого розподілу. Перерахунок цього коефіцієнту відбувається відносно кожного пікселя за формулою:

$$v(Z_{h,w}) = w_{q^*}, \tag{2.5}$$

де q^*:

$$q^* = \arg\max_{q} Z_{h,w,q}. \tag{2.6}$$

Після отримання розподілу кольорів, для переведення багатомірного розподілу до вигляду каналів a та b вихідного зображення та коригування параметрів насиченості кольорів, отриманий розподіл інтерполюють з використанням метод віджигу за формулами:

$$H(Z_{h,w}) = \mathrm{E}[f_T(Z_{h,w})], \tag{2.7}$$

де H – функція перетворення розподілу Z, а $f_T(Z_{h,w})$ визначають як:

$$f_T(z) = \frac{\exp(\log(z)/T)}{\sum_q \exp(\log(z_q)/T)}, \tag{2.8}$$

при цьому чим ближче параметр T до нуля, тим більш насиченими виходять зображення. Порівняння впливу значення T на результат наведено на рис. 2.9.

Рис. 2.9. Вигляд результатів при різних значеннях параметру T [18]

2.3. Генеративні змагальні мережі для обробки зображень

2.3.1. Генеративні змагальні мережі

Концепція генеративних змагальних мереж базується на тих самих принципах, що і робота згорткових нейронних мереж, проте при використанні генеративних моделей, ми маємо справу вже з декількома нейронними мережами, що пов’язані одна з одною під час навчання, та можливо й під час експлуатації натренованих мереж. Надалі буде викладено базовий приклад генеративних моделей, що заснований на 2-х нейронних мережах – генераторі та дискримінаторі. Проте з розвитком даного напрямку штучного інтелекту, кількість мереж для вирішення більш складних задач може збільшуватись (використовуються додатково мережі-енкодери та

декодери, додаються декілька дискримінаторів для оцінювання різних особливостей згенерованих даних, додається генератор для реконструкції даних і т.д.) [22]. Частину цих ідей по розширенню стандартного pipeline-у буде розкрито детальніше у наступних розділах та реалізовано для виконання поставленої у дисертації задачі.

Основна ідея генеративних змагальних мереж, полягає у тому, щоб зробити навчання моделей більш адаптивним, швидким та якісним. Такі моделі призначені для виконання задачі генерації даних, редагування, видозміни вхідних даних чи вивчення просторових та часових особливостей і виділення їх у вигляді багатомірних розподілів (distributions), які згодом можуть використовуватися у інших задачах (так, наприклад для технологій DeepFake [23] спочатку мережа вивчає закономірності обличчя та міміки на зображеннях однієї людини, формує з цієї інформації певний розподіл, а потім інша мережа виконує підміну обличчя іншої людини, за розподілом, отриманим до цього).

Загалом, так звані розподіли є базовою концепцією усього поглибленого навчання – вони відображають приховані закономірності, особливості даних у вигляді багатовимірних векторів, природу та суть яких людина знати не може (такі приховані частини у нейронних мережах називають black box, або “чорна скринька”, адже ми не знаємо за яку саме характеристику відповідає те чи інше число у розподілі, так само у більшості задач неможливо встановити, за виділення та активацію яких саме особливостей відповідають нейрони у шарах мережі).

Головним ноу-хау генеративних моделей є їх принцип роботи, який можна сформулювати наступним чином: “Існує дві згорткові нейронні мережі, а саме – генератор, яка відповідає за генерацію даних, які надалі називатимуться результатом, та дискримінатор, згорткова мережа яка по

своїй суті є бінарним класифікатором, що отримує на вхід результат роботи генератора, та приклад реальних даних. Головна задача дискримінатора – виявити підробку, задача генератора – генерувати підробки настільки якісно, щоб дискримінатор не міг правильно ідентифікувати підробку. Навчання розділене на ітерації та епохи, після заданої кількості епох, виконуються перевірки на дискримінаторі, і та мережа, яка програє перевірку, додатково навчається. Головна ідея такого навчання, щоб обидві мережі постійно були на одному рівні навчання та змушували один одну працювати краще. Тоді, після якогось часу стає можливим отримати на виході чудовий класифікатор та генератор підробок.” [22].

Дані алгоритми дозволяють генерувати як аудіо, зображення, відео, так і текст, починають використовуватися у сфері криптографії (технології encoder-decoder), можуть використовуватися для генерації реалістичних статистичних даних (наприклад, коли неможливо взяти дані з реальних фінансових бірж чи інших закритих джерел, для експериментів можна розробити синтетичні дані).

Для більш глибокого розуміння та експлуатації даної технології, необхідно розглянути її з математичної точки зору. Задачею генератора G є вивчити розподіл p_g з даних x, та за цим розподілом генерувати нові дані, подібні до x. Задачею дискримінатора $D(x,\theta_d)$, що повертає скалярне значення, є правильно визначати вірогідність того, що вхідна інформація походить з розподілу реальних даних x, а не з синтетичного розподілу генератора p_g. Отже, ми тренуємо дискримінатор D з метою максимізувати вірогідність правильної класифікації даних, що виражається формулою:

$$L(D) = \log(D(x)) + \log(1 - D(G(z))), \tag{2.9}$$

де у правій частині перший логарифм показує чи правильно працює дискримінатор на даних з реального розподілу, а другий – на даних, синтезованих генератором.

Генератор G навчається з метою мінімізувати функцію помилок, що визначається формулою:

$$L(G) = \log(1 - D(G(Z))). \tag{2.10}$$

Як було зазначено раніше, мережі можуть вигравати чи програвати раунди, і навчатися почергово. Проте, як показали дослідження, при кінцевій кількості даних, дискримінатор внаслідок такого варіанту навчання може перевчитися на даних і починає під них підлаштовуватись, що призводить до поганої якості навчання генератора та незадовільних результатів. Тому автори технології запропонували систематизувати процес навчання для мереж наступним чином: на генераторі застосовують зворотне поширення кожну ітерацію навчання, а дискримінатор – k разів у ітерацію (значення підбирається емпірично, в залежності від розміру набору даних). Це допомагає тренувати мережі більш збалансовано, знижуючи ризик значного відставання однієї мережі від іншої та уникнути перенавчання дискримінатора.

При задачі генерації нових даних з нуля, генератору на вхід не подається реальна інформація, замість цього, за допомогою певних алгоритмів генерується деякий шум з розподілу p_g (зазвичай це розподіл Гауса, Пуассона або інший). Цей шум проходить крізь усю шари мережі (згортки та об'єднання, розгортки та роз'єднання) і потім за допомогою функції помилок отриманий результат порівнюється з реальними даними (ground truth). На початку навчання, генератор не виконує задачу якісно, і візуально згенеровані дані зовсім не є схожими на реальні приклади. Проте при достатньому обсязі даних та добре підібраний функції помилок, за

допомогою зворотного поширення, шари мережі навчаються перетворювати вхідний шум на інформацію, подібну до реальних даних.

Загалом, алгоритм навчання можна сформулювати наступним чином: протягом n ітерацій, для заданих k кроків навчання дискримінатора виконуються наступні дії:

- генеруються нові дані з розподілу p_g;
- обираються випадково частина даних з p_{data};
- перераховуються градієнти для дискримінатора;
- генеруються нові дані з розподілу p_g;
- перераховуються градієнти для генератора.

Глобальною метою навчання таких мереж є прирівняти розподіл p_g, утворений генератором, до розподілу реальних даних p_{data}.

Саме за такими принципами навчаються генеративні змагальні мережі. Згодом, технологію значно розвинули за допомогою різноманітних оптимізацій та додаванням нових видів функцій помилок, додаткових шарів у архітектури, створенням цілком нових архітектур, та комбінування багатьох генеративних моделей для сумісного тренування.

Головною перевагою такого підходу є його гнучкість (є можливість підлаштовувати багато параметрів, та більше фокусувати мережу на усуненні конкретних недоліків), висока якість навчання (звичайні згорткові мережі застосовують більш прості критерії оцінки, тому без внесення власних модифікацій можуть зациклитись на обраному наборі даних, не вивчаючи при цьому увесь можливий внутрішній розподіл даних, також дискримінатор є доведено кращим та більш глибоким засобом для оцінки якості роботи мережі, ніж прості функції помилок, наявні у згорткових мережах). Тому, якщо взяти достатньо містку архітектуру для генератора та дискримінатора (щоб мати можливість охопити велику кількість різноманітних даних),

правильно налаштувати параметри та внести необхідні модифікації, то стає можливим виконати поставлену наукову задачу.

Головним недоліком генеративних змагальних мереж є велика нестабільність цих мереж при тренуванні та експлуатації (якщо мережа натренована не досить якісно і надійно). Для уникнення перенавчання чи низької якості роботи мереж, необхідно застосувати певний ряд дій та модифікацій та провести детальний аналіз набору даних для навчання, про ці та інші нюанси реалізації йтиме мова у наступних розділах магістерської дисертації. А у наступному пункті буде розглянуто існуючі рішення для колоризації зображень на основі технології генеративних моделей.

2.3.2. Технологія DeOldify

Починаючи з появи технології трансферу стилю (Style Transferring, 2015), згорткові нейронні мережі, такі як VGG-16, VGG-19, U-Net, FCN почали активно використовуватися у задачах сегментації, стилізації та обробки зображень. Для задачі колоризації найбільш якісною та універсальною розробкою на сьогоднішній день є технологія DeOldify, яка використовується для реставрації старих чорно-білих фото та відео [14].

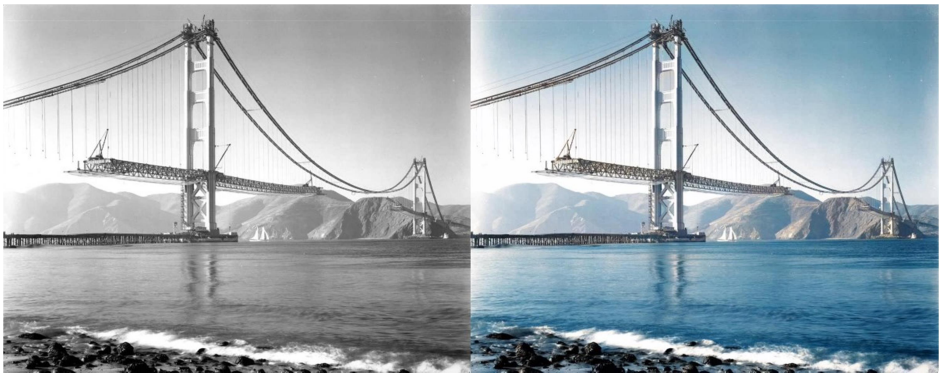


Рис. 2.10. Результат роботи алгоритму DeOldify [14]

Алгоритм базується на згортковій нейронній мережі U-Net з певними модифікаціями, про яку буде більше розповідатися у наступному розділі, натреновану за певними параметрами на великому наборі даних з поставленою задачею розуміти контекст зображень у чорно-білому форматі, і шляхом розпізнавання образів та побудові кольорових асоціацій виконувати задачу колоризації зображень. Останні версії цієї технології також почали використовувати генеративні змагальні мережі, що значно покращило якість результатів, що також є підтвердженням висунутої у даній магістерській дисертації гіпотези, що на сьогодні, генеративні змагальні мережі є найбільш оптимальним та якісним рішенням задачі з обробки зображень, в тому числі і проведення інтелектуальної колоризації.

2.4. Висновки

На основі проведеного дослідження існуючих методів рішення задачі напів-автоматичної колоризації зображень, можна сказати, що на сьогодні існують досить непогані технології на основі як і математичних алгоритмів, так і на основі глибоких нейронних мереж. Проте усі досліджені рішення мали певного роду недоліки, або у рамках рішення не було запропоновано вирішення саме поставленою у даній магістерській дисертації завдання. Алгоритм “Colorization using optimization” за якістю результатів в цілому задовольняє поставленій задачі у деяких випадках, проте він є дуже повільним методом, та не може здійснювати адекватне редагування на кольорових зображеннях (кольори оригіналу не зберігаються, навіть якщо користувач зазначив, що дана область зображення не потребує перефарбування). Запропоновані рішення на основі нейронних мереж є більш швидкими та гнучкими, проте задача автоматичної та напівавтоматичної колоризації в нашому випадку суттєво відрізняються, тому є необхідним, як і

було вказано раніше, розробити власну технологію на основі існуючих рішень, архітектур та математичних “ноу-хау”.

У результаті аналізу було виділено певні “best practices” по роботі з задачею колоризації, основні з яких наведено нижче:

- за основу потрібно взяти кольорову схему, у якій поняття кольору більш відокремлене (YUV, LAB) ніж у стандартних RGB чи BGR;
- технологія генеративних змагальних мереж на сьогодні є найбільш оптимальним рішенням для виконання дослідження, через ряд їх суттєвих переваг перед звичайними згортковими мережами, серед яких є швидкість та якість навчання, більш висока гнучкість мереж та можливість додавати до основного процесу модифікації, делегувати певні частини роботи між декількома нейронними мережами;
- за основу мережі-генератора у майбутній системі генеративних змагальних мереж варто взяти архітектуру U-Net [24] як найбільш стабільну та високоефективну архітектуру, придатну до розв’язку різноманітних задач сегментації та генерації даних.

3. АРХІТЕКТУРА ГЕНЕРАТИВНОЇ МЕРЕЖІ ДЛЯ КОЛОРИЗАЦІЇ ЗОБРАЖЕНЬ

3.1. Обгрунтування вибору програмних засобів та бібліотек

Перед тим, як приступати до програмної реалізації поставленої задачі, необхідно провести аналіз програмних засобів, мов програмування та бібліотек для роботи з зображеннями та для імплементації процесу глибокого навчання, та визначити з них найбільш оптимальні і надійні засоби для подальшої реалізації.

3.1.1. Мова програмування

Вибір мови програмування для задач поглибленого навчання на сьогодні є досить широкий, проте найбільш оптимальним вибором вже як 5 останніх років є мова Python. Вона має певні недоліки, наприклад швидкість навчання на мові C++ з застосуванням графічного чіпу буде значно вище, проте мова C++ має свої недоліки, такі як відсутність деяких необхідних бібліотек та підвищена важкість розробки подібних систем та їх аналізу, обмежені можливості з аналізу коду та детекції помилок.

Мова R, є на сьогодні досить потужним засобом для роботи з великими даними, здійсненням статистичних обчислень та аналізу. Також до мови надаються потужні засоби візуалізації інформації, що є дуже необхідним для контролю процесу навчання (виведення проміжних результатів, графіків функцій помилок та інших параметрів). R показує себе як надійна і популярна мова у сфері статистичного аналізу, розв’язку задач регресії, класифікації та формування дерев рішень. Ця мова вже активно використовується для аналізу даних у провідних світових компаніях, таких як Google та Facebook. Проте у даній магістерській дисертації поставлена задача генерації/редагування

зображень, для якої мова R не є достатньо адаптованою та вживаною, з чого випливає відсутність деяких бібліотек [26].

Отже, у рамках цього наукового дослідження використовуватиметься мова Python, адже вона має ряд переваг, у порівнянні з іншими мовами, коли йдеться про задачі поглибленого навчання, особливо у академічних, а не комерційних цілях. Першою перевагою є просто мови, її синтаксису та принципів роботи. Навіть не маючи великого досвіду у програмуванні, починаючий науковець може розробляти та тренувати власні алгоритми за допомогою мови та її споміжних бібліотек. Другою перевагою є швидкість. Не дивлячись на те, що Python вважається досить повільною мовою, в першу чергу через те, що код на цій мові не компілюється, а інтерпретується, проте для машинного навчання на сього вже існує дуже велика кількість бібліотек та фреймворків, що значно прискорюють процес навчання нейронних мереж, оптимізуючи розподілення даних поміж процесорами GPU, та працюючи з числами у більш швидкі способи, ніж це можливо у мові за замовчанням. Про це докладніше буде розповідатися у наступному пункті.

Ну і найважливішою перевагою мови Python є величезна кількість бібліотек з реалізацією більшості існуючих задач та алгоритмів, через, що мова Python просто усуває необхідність “створювати велосипед” беручись за будь-яку тривіальну задачу [25]. Також для мови Python представлена найбільша кількість бібліотек для реалізації будь-якої задачі штучного інтелекту, про що також вказано у наступних пунктах.

3.1.2. Бібліотеки для поглибленого навчання

Для реалізації власних чи запозичених архітектур нейронних мереж, проведення тренування та тестування і інших необхідних маніпуляцій, будь

яке подібне ПЗ повинне базуватися на певній бібліотеці машинного навчання, яка називається “фреймворк”.

Існує декілька фреймворків машинного навчання, загалом усі з них мають необхідні для розробки абстракції (класи нейронних мереж, реалізації базових операцій, таких як згортка чи об’єднання, оптимізацію мереж при навчанні та інше), проте все ж між собою ці фреймворки мають певні відмінності.

Бібліотека Tensorflow від компанії Google – на сьогодні найбільш вживане та популярне рішення для розробки, окрім усього базового функціоналу для поглибленого навчання, тут також наявні засоби для створення нейронних мереж під мобільні платформи (iOS, Android), проте наявні і певні недоліки. По-перше: бібліотека потребує великий об’єм коду, функції мають досить складний синтаксис та не є інтуїтивно зрозумілими у використанні. По-друге: при тренування на TensorFlow, утворюється статичний граф підрахунків (архітектура створеної мережі є сталою і не має можливості її змінювати протягом навчання) [27]. Також, це означає, що навіть при внесенні незначних змін до архітектури, увесь експеримент необхідно повторювати з нуля (процес тренування).

Фреймворк Keras – це обгортка до бібліотеки TensorFlow, яка значно спрощує процес розробки, має більш простий та зрозумілий синтаксис, та у якості бек-енду (для побудови архітектури та підрахунків) може використовувати як бібліотеку TensorFlow, так і деякі інші бібліотеки [29]. Даний фреймворк є чудовим вибором для новачків та вирішення простих задач, проте йому присущі недоліки бібліотеки TensorFlow, і створювати іноваційні та великі за розміром рішення краще на більш технологічній бібліотеці.

Як основний засіб для побудови нейронних мереж було обрано бібліотецку PyTorch – ця бібліотека є на сьогодні найбільш гнучкою та зручною для здійснення прототипування, наукових експериментів та використання у академічних цілях. Граф підрахунків є динамічним, тому немає необхідності проводити тренування заново після внесення архітектурних змін. Також бібліотека підтримує розподілене (тренування на різних комп'ютерах) та паралельне (на багатьох графічних процесорах одночасно) тренування [28]. Також для бібліотеки наявна велика кількість раніше натренованих (pre-trained) моделей для виконання різних задач.

Як відомо, до нейронних мереж дані поступають у нормалізованому вигляді (зазвичай приведені до чисел від 0 до 1) та запаковані до спеціального типу даних – тензору (багатомірний масив даних). Для приведення даних до цього вигляду, а також для проведення різних операцій над даними, необхідно використовувати спеціальну бібліотеку для роботи з багатомірними даними. Еталонним рішенням для цього є бібліотека numpy для мови Python. Дана бібліотека дає користувачу доступ до величезної кількості різних математичних алгоритмів (таких як генерація випадкових величин за розподілом, генерація випадкових чисел, операції додавання, віднімання, множення багатомірних масивів, та дуже багато іншої функціональності). Саме за допомогою цієї бібліотеки, вхідні дані будуть попередньо оброблятися у вигляді numpy-масивів, нормалізуватися та конвертуватися у тензори для подальшого навчання. Так само отримані результати будуть конвертуватися у вигляд, необхідний для візуалізації.

CUDA це програмно-апаратна архітектура паралельних обчислень, що дозволяє суттєво прискорити та оптимізувати обчислювальну продуктивність за допомогою графічних процесорів компанії Nvidia (AMD-відеокарти не підтримуються) [30]. CUDA SDK дозволяє користувачам, використовуючи

команди на високорівневих мовах, таких як Python та C++ для створення алгоритмів, що виконуватимуться на графічних та тензорних процесорах Nvidia. Це значно прискорює довготривалі процеси обчислень при тренуванні алгоритмів поглибленого навчання, розрахунку статистичних моделей, симуляцій реального світу (дослідження клімату, екології, медицини та багато іншого). З основних переваг, окрім швидкості, відзначають, що дана архітектура має доволі зручний інтерфейс програмування (CUDA API), що має спростити процес вивчення та застосування архітектури у реальних дослідженнях. Також архітектура дозволяє виконувати більш ефективні транзакції між пам'яттю центрального процесора і відео пам'яттю, надає повну апаратну підтримку цілочисельних та побітових операцій [30]. Побудова навчання на даній архітектурі може прискорити увесь процес тренування у 2-4 рази у порівнянні з відеокартою без використання CUDA, та у 10-15 разів у порівнянні з проведенням обчислень на CPU. Також за допомогою CUDA можна оптимальніше використовувати ресурси комп’ютера. З самою архітектурою постачається велика кількість корисних бібліотек, таких як cuDNN (для роботи з глибокими нейронними мережами), cuBLAS (для підтримки стандартних методів лінійної алгебри), cuSPARSE (для обчислень багатовимірних тензорів) та інші.

3.1.3. Додаткові бібліотеки

Для реалізації поставленої задачі та проведення якісного дослідження, окрім бібліотек для проєктування та тренування нейронних мереж, також знадобляться інші споміжні бібліотеки для роботи з ОС, файлами, засоби логування, візуалізації та обробки зображень. Тож нижче буде розглянуто основні бібліотеки, які будуть використовуватися у процесі розробки.

Matplotlib – бібліотека з дуже широкими можливостями з візуалізації інформації, а саме побудова графіків будь-якого вигляду (стовпчикові

діаграми, графіки функцій, 3D-діаграми, pieslice-діаграми, тощо), грідів з результатами (поєднання декількох графіків, вивід порівняльних зображень) які вкрай необхідні для спрощення процесу досліджень та демонстрації продуктивності роботи мереж під час тренування та тестування [32]. У даній магістерській дисертації надалі усі наведені графіки створені саме за допомогою цієї бібліотеки (в першу чергу графіки функцій помилок та їх порівняння на різних стадіях навчання).

OpenCV – дуже потужна бібліотека, у якій реалізовано найбільша за обсягом колекція алгоритмів комп'ютерного зору та редагування зображень (накладання фільтрів, розмиття, підвищення різкості, конвертації зображень у іншу кольорову схему). Дана бібліотека буде використовуватися у розробці для проведення попередньої обробки даних, таких як читання, приведення до потрібної кольорової схеми, застосування аугментацій [31]. Також бібліотека використовуватиметься для пост-обробки зображення після їх обробки у нейронних мережах та збереження результатів у коректному вигляді.

PIL (Pillow) – інша бібліотека для роботи з зображеннями, вона не містить алгоритми комп'ютерного зору, проте надає досить глибоку функціональність з точки зору обробки зображень і для навчання мереж на обраному раніше фреймворку PyTorch зручніше використовувати саме цю бібліотеку для подання даних на вхід для подальшого навчання [33].

3.2. Вибір архітектури мережі-генератора

Мережа-генератор виконує задачу генерації нових даних, які надалі будуть оцінюватися мережею-дискримінатором. Задача такої мережі полягає у тому, щоб згенерувати максимально подібні дані до оригінальних даних, що використовуються під час навчання.

Загалом, генерувати можливо будь які типи даних – звук, зображення, відео, текст, 3D-моделі і багато чого іншого. Звичайно усе це можливо лише при правильній підготовці архітектур та попередній обробці даних (потрібно переводити зображення у необхідну кольорову схему, реалізувати коректні методи для векторизації даних, адже будь які дані у мережі подаються як набір чисел та векторів зі значеннями у проміжку [0, 1]).

Конкретно для поставленої задачі, даними будуть зображення, як на вхід мережі, так і на вихід (адже задача полягає саме у перетворенні вхідного зображення у аналогічне за контекстом, яким у даній задачі виступають об’єкти на зображенні та їх текстури, проте зі зміненими кольорами цих об’єктів). Загалом, яку б архітектуру мережі ми не обрали, нам необхідно, щоб дані були досить однозначними та коректними. Ми маємо справу з таким типом машинного навчання, як навчання з учителем, тому важливо, щоб кожен екземпляр даних був коректний (під екземпляром мається на увазі пара зображень – звичайне зображення та зображення з виправленими кольорами з одним і тим самим контентом).

У рамках нашої задачі навчальною парою буде одне і те саме зображення, у різних поданнях (чорно-білий варіант для входу, та його канали, що відповідають за кольори – для порівняння на виході). Також, маючи справу з перетворенням даних одного класу (звичайне зображення) у дані іншого класу (зображення з виправленими кольорами), нам важливо, щоб нейронна мережа мала усю необхідну інформацію для перетворення. Тому, чим більш контрольоване навчання і роботу мережі ми хочемо отримати, тим більш місткими та однозначними мають бути наші дані. Саме для цього окрім переведення зображення у чорно-білу нотацію, було впроваджено додатковий вхід на мережу, через який подається маска з кольоровими сегментами, які позначають відповідні сегменти на зображенні,

які треба відредагувати, та колір, яким це рекомендується зробити (також передаються сегменти з тими кольорами, які коригувати непотрібно). На порівняння до мережі-дискримінатора, та внутрішніх підрахунків у генераторі подається також оригінальне кольорове зображення, воно визначає які саме результати повинна повертати мережа-генератор.

Задача з перетворення зображення – це задача сегментації. Для розв'язку такого типу задач існує декілька видів архітектур згорткових нейронних мереж, з них виділяють дві основні – U-Net та FPN. Було проведено дослідження та порівняння цих двох архітектур, після якого для побудови архітектури генератора було обрано архітектуру U-Net, так як вона є більш зручною і якісніше виконує задачу перетворення зображень, судячи з багатьох проведений досліджень (в тому числі колоризацію зображень, усування шумів та розмиття та різних інших можливих дефектів на зображеннях), в той час як FPN активно використовується для інших задач (медицина, аналіз зображень для систем керування автомобілем, системи безпеки).

U-Net вважається "state-of-art" (загальноприйнятий еталон) серед архітектур згорткових нейронних мереж для задачі сегментації зображень, коли потрібно не тільки визначити класи наявні на зображення, але і сегментувати об'єкти за їх класами, тобто створити маски [24].

Архітектура складається з розподіленого шляху для захоплення контексту і симетрично розширюваного шляху, який дозволяє здійснити точну локалізацію об'єктів. Мережа навчається наскрізним способом на невеликій кількості зображень. Крім того, ця мережа працює швидко. Сегментація зображення 512 × 512 займає менше секунди на сучасному графічному процесорі.

Для U-Net характерно:

- досягнення високих результатів в різних реальних задачах, особливо для біомедичних застосувань;
- можливе використання невеликої кількості даних для досягнення хороших результатів, завдяки активному використанню аугментацій до вхідних даних.

Нижче, на рис. 3.1, наведено стандартну архітектуру мережі U-Net, яку ми надалі будемо модифікувати [24].

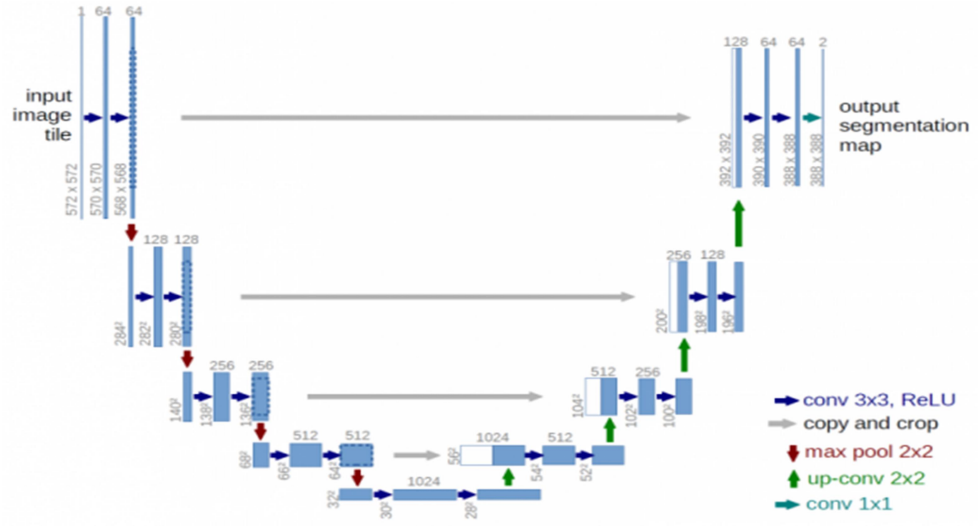


Рис. 3.1. Стандартна архітектура мережі U-Net [24]

Інноваційною цю архітектуру у порівнянні з попередніми розробками робить те, що вона відходить від концепції sliding-window для подання даних на обробку (ця методика “розрізає” зображення на менші фрагменти, кожен з яких обробляється у мережі окремо), і це значно прискорило процес сегментації. Щоб забезпечити при цьому високу якість результатів, у мережі було реалізовано механізми для передавання контекстної інформації різних рівнів (для врахування як великих так і маленьких особливостей) між шарами згортки та розгортки. Таким чином мережа на різних етапах навчається сегментувати об’єкти на основі особливостей різних рівнів, що призводить до

підвищення якості результатів. Шари роз'єднання було замінено на шари збільшення розмірності (upsampling layer).

Як можна побачити на рис. 3.1, мережа складається з шляху звуження (зліва) та шляху розширення (праворуч). Шлях звуження – типова архітектура згорткової нейронної мережі. Блоки цієї під-мережі складаються з двох згорток з розміром ядра 3×3, за кожним з яких слідують функція активації ReLU і операція max-pooling (2×2 з відступом 2) для зниження розмірів тензорів. Також наприкінці кожного такого звужуючого блоку відбувається копіювання та відтинання мапи особливостей і її передача на вхід до відповідного блоку розгортки у блоку розширення. Блок розширення складається з трьох шарів згортки з використанням функції активації ReLu, наприкінці блоку згорток виконується операція підвищення розмірності з ядром розміром 2×2. Також варто зазначити, що на вхід кожного такого блока, перед виконанням послідовних операцій згортки виконують конкатенацію мап особливостей отриманих з симетричного блоку згортки та поточної мапи особливостей, отриманої з попереднього блоку. Останнім шаром архітектури є згортка з розміром ядра 1×1, яка виконує приведення отпиманої мапи особливостей до розміру результуючого зображення. Всього мережа містить 23 згортальних шару.

Мережа навчається методом стохастичного градієнтного спуску з високим параметром моментуму (0.99 у початковій реалізації), на основі вхідних зображень і відповідних їм карт сегментації. Через згортку без параметра padding (відповідає за створення додаткових рамок навколо тензорів, заповнених нулями), результуюче зображення виходить менше вхідного за розміром. Для процесу навчання попіксельно застосовують функцію soft-max:

$$p_k(x) = \exp(a_k(x)) / (\sum_{k'=1}^{K} \exp(a_{k'}(x))), \tag{3.1}$$

де $a_k(x)$ – значення функції активації каналу k для тензору x, K – це кількість відомих класів для мережі, а $p_k(x)$ – тензор активацій різних класів на зображень ($p_k(x) = 1$ означає, що для класу k активація є максимальною, $p_k(x) = 0$ означає, що для класу k активації немає, а отже певний клас об'єктів присутній/відсутній на сегментованому зображенні). Далі цю функцію комбінують з функцією крос-ентропії для підрахунку функції помилок.

Перехресна ентропія між двома розподілами ймовірності p та q над спільним простором подій вимірює середню кількість біт, необхідних для впізнання події з простору подій. Перехрестна ентропія обчислюється в кожній точці розподілу та визначається наступною формулою (формула для дискретного випадку):

$$E = \sum_{x \in \Omega} w(x) \log(p_{l(x)}(x)), \tag{3.2}$$

і має на меті накладати штраф у кожній точні, орієнтуючись на відхилення кожної активації від реальних значень $p_{l(x)}$. $w(x)$ – мапа вагів оригінальної сегментації, що заздалегідь підраховується за формулою:

$$w(x) = w_c(x) + w_0 \cdot \exp(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}), \tag{3.3}$$

де $w_c(x)$ – карта ваг для балансування частот класів, $d_1(x)$ – відстань до границі з найближчим іншим об'єктом, а $d_2(x)$ – відстань до границі другого найближчого об'єкту.

Архітектура U-net досягає видатної продуктивності і точності в самих різних задачах сегментації. Метод вимагає лише кількох помічених зображень для тренування і має прийнятний час навчання: всього лише 10 годині на графічному процесорі Nvidia Titan (6 ГБ). Саме тому було обрано цю мережу, як основу для нашої мережі-генератора. До стандартної

архітектури мережі та даних було застосовано значні зміни для адаптації архітектури для виконання задачі колоризації зображень, про що буде детальніше розказано у четвертому розділі даної магістерської дисертації.

3.3. Вибір архітектури мережі-дискримінатора

Мережа-дискримінатор, як було зазначено раніше, це згорткова нейронна мережа, яка виконує задачу бінарної класифікації вхідних даних, та повертає у якості результата одномірний вектор ймовірностей з двома елементами $[p_{fake}, p_{true}]$, де p_{fake} – ймовірність того, що вхідне зображення було створено генератором та p_{true} – ймовірність того, що зображення було справжнім зображенням з набору даних (ground truth).

Для побудови мережі-дискримінатора зазвичай беруть не дуже глибоку архітектуру звичайної згорткової мережі, що містить послідовні шари згортки, об’єднання та повністю з’єднані шари наприкінці для утворення вектору передбачень. Проте під нашу задачу необхідно буде виконати певні модифікації такої мережі, однією з яких є подача даних на вхід. В нашому випадку, мережа генератор отримуватиме на вхід одномірне подання зображення (канал освітленості початкового зображення) та два кольорових канала з користувацькими правками по колоризації. На виході метою генератора є синтезувати нову кольорову мапу (двовимірний тензор, який потім об’єднується з початковим каналом освітленості) яка є реалістичним розподілом кольорів на зображенні. Саме релевантність цього розподілу і необхідно оцінювати за допомогою дискримінатора.

Під час тренування після певної кількості ітерацій, виконуватиметься валідація генератора за допомогою дискримінатора. На вхід до дискримінатора буде подаватися реальне зображення з набору даних (точніше його кольоровий розподіл), та потім синтезований розподіл за допомогою генератора. У результаті дискримінатор повертатиме по вектору передбачень

для кожного зображення, з яких потім буде вираховано загальну функцію втрат дискримінатора:

$$L_{disc} = 0.5 * \log D(x) + 0.5 * \log(1 - D(G(z))), \quad (3.4)$$

де $D(x)$ – передбачення для справжнього зображення (розподілу), а $D(G(z))$ – для синтезованого. Метою дискримінатора є мінімізувати помилку при аналізі синтезованих зображень і максимізувати класифікацію для справжніх.

Проте, у випадку використання звичайної згорткової мережі у якості дискримінатора, аналіз здійснюється на усьому згенерованому зображенні, що може призводити до усереднення значень сусідніх пікселів генератором для успішного “обману” дискримінатора. Внаслідок цього генератор буде генерувати не дуже якісні зображення, а дискримінатор не зможе їх розпізнати як підробку і навчання зупиниться. Для усування цієї проблеми необхідно використовувати додаткові функції втрат (щоб визначати підробку не тільки за попіксельним порівнянням) та аналізувати частини поданого на вхід дискримінатора зображення окремо. Саме для цього було розроблено архітектуру PatchGAN [34], яку ми і будемо використовувати за основу у даному дослідженні. Основна ідея цієї архітектури полягає у тому, що під час проходження мережі, зображення розбивається на N x N окремих сегментів невеликого розміру, для яких виконується бінарна класифікація (чи є той чи інший фрагмент зображення підробкою чи ні), потім, результати класифікації поєднують для повернення загального висновку стосовно зображення. На рис. 3.2. наведено приклад цієї архітектури. Як можна побачити, мережа повертає тензор розмірністю 32 x 32, який містить ймовірності для кожного з 1024 частин вхідного зображення. Загалом архітектура складається з послідовно з’єднаних шарів згортки, об’єднання, після згортки

застосовується операція нормалізації даних (instance normalization), а між шарами виконується функція активації Leaky ReLu.

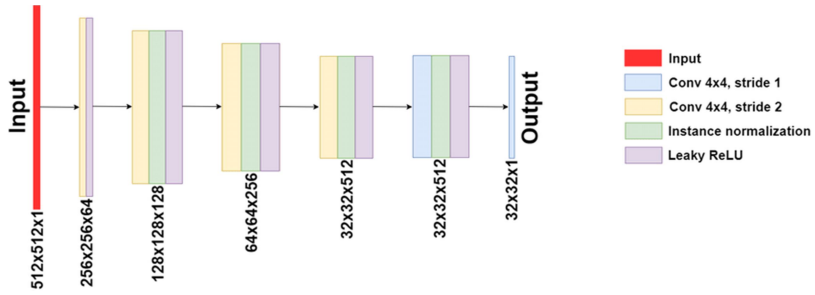


Рис. 3.2. Приклад архітектури PatchGAN [34]

3.4. Обгрунтування вибору даних та способів їх попередньої обробки

Важливість вибору правильної архітектури та внесення модифікацій для її підлаштування до задачі та поліпшення якості результатів, так само як і налаштування параметрів тренування мережі є дуже важливою задачею, про яку мова йтиме у наступному розділі цієї магістерської дисертації. Але варто не забувати, що такой дуже впливовими факторами успішного навчання є правильно сформований набір даних та якісні методи його обробки. Саме про вибір набору даних та ці методи попередньої обробки і йтиме мова у даному пункті.

3.4.1. Обгрунтування вибору набору даних

Вибирати набір даних необхідно, виходячи з поставленої задачі. За поставленою задачею, ми будемо здійснювати тренування системи нейронних мереж за допомогою навчання з учителем.

Загалом є три способи навчання нейронних мереж, які використовуються у різного роду задачах. Перший з них називається навчання з учителем (supervised learning), і у даному випадку для успішного навчання моделей, ми подаємо дані у вигляді навчальної пари, яка містить

вхідні дані та очікуваний результат. Таким чином, нейронні мережі під час навчання можуть чітко вирахувати та локалізувати власні помилки і коригувати їх. Недоліком такого навчання є ризик так званого “перенавчання” (overfitting), яке виникає якщо нейронна мережа починає підлаштовувати власні ваги саме під тренувальну вибірку і не набуває властивостей генералізувати інформацію. Це призводить до того, що мережа з дуже високою чіткість відпрацьовує на тренувальних та навіть тестувальних даних (якщо вони дуже подібні між собою), проте абсолютно не здатна виконувати поставлену задачу в умовах реального світу (на якихось нових даних). Для уникнення цієї проблеми використовуються різні техніки, такі як регуляризація вагів та аугментація даних. Також у випадку перенавчання варто розширити існуючу навчальну вибірку і зробити її більш різноманітною.

Другий вид тренування називається навчання без учителя (unsupervised learning), у даному виді тренування не формується навчальна пара. Умовно, на вхід подаються певні дані, а на виході з певного набору даних, що подібний до очікуваного (пряма відповідність вхідних і вихідних даних не потрібна), тобто задача мережі зводиться до вивчення розподілу очікуваних даних, і інтерпретація розподілу вхідних даних до вихідних. Ця методика є дуже ефективна при вивченні невідомих навіть людині розподілів (дослідження у сфері хімії та фізики, біології), адже тут немає чітко заданих обмежень для мережі у навчанні, тому мережа може більш якісно генералізувати певні особливості даних та навіть розширити розподіл можливих результатів. Недоліком такого навчання є його нестабільність та неоднозначність, адже даний вид навчання не передбачений для задач, які потребують чітко визначених результатів, а більше спрямований для дослідження властивостей даних та їх незвичайні інтерпретації.

Третій вид навчання це так зване “навчання з підкріпленням” (reinforcement learning). На сьогодні це найменш досліджена тема у сфері штучного інтелекту, яка по суті пропонує виконувати навчання мереж за допомогою так званих еволюційних алгоритмів та певних інших технік. Штучна нейронна мережа створюється у певному середовищі, має певний набір можливостей та поставлену задачу і нагороду за наближення до її досягнення. За ітерацію навчання створюється покоління таких мереж, які мають досягти певних результатів. Потім мережі, що досягли якихось успіхів стають основою для створення нового покоління. Так триває до того часу, коли нейронна мережа зможе виконувати поставлену задачу при різних умовах середовища. Найкращим прикладом на сьогодні є симуляція таких нейронних мереж усередині комп’ютерних ігор. Наприклад, поставлена задача мережі звучить як “керуючи автомобілем, доїхати до вказаної точки на карті, не порушуючи правил автодорожнього руху та уникаючи аварій”. Мережа за вхідні дані має лише методи керування машиною (вперед, назад, вліво, вправо) і значення нагороди (чим ближче машина приїде до вказаної точки, тим воно вище). Також, якщо мережа намагається “махлювати”, вона отримує додатковий штраф (наприклад, за проїзд по тротуару чи збиття пішохода). І такі симуляції можуть повторюватись більше мільйона разів, поки мережа не зможе безпечно та якісно користуватись автомобілем у межах комп’ютерної гри. Це дуже перспективна та цікава галузь досліджень, проте для рішення поставленої задачі такий тип навчання не є можливим.

Отже, для реалізації навчання з учителем, нам необхідно сформувати поняття навчальної пари і обрати та завантажити набір даних. Так як ми маємо справу з задачею колоризації, то у якості навчальної пари ми можемо використовувати різні канали одного і того самого зображення. На вхід подаємо контекстну інформацію, на вихід – інформацію про очікуваний

розподіл кольорів, як у методі “Colorful Image Colorization” розглянутому у попередньому розділі цієї дисертації.

Так як перед нами стоїть задача розробити універсальний засіб для колоризації будь яких фото, то дані не повинні бути сфокусовані на конкретній предметній області, а навпаки – наша тренувальна та тестувальна вибірка повинна містити максимально різноманітну інформацію як про образи так і про кольори. Очевидно, що для якісного вивчення зображень з різними об’єктами та кольорами, цих самих зображень необхідно дуже багато. Загалом для навчання підходить багато відкритих наборів даних з фотографіями різних об’єктів, проте було зупинено вибір на наборі даних IMAGENET 2012-го року [35], що містить зображення більш ніж тисячі різних класів та сумарний об’єм якого дотягує до 1.3 мільйона зображень. Судячи з аналізу попередніх досліджень, такої кількості даних повинно вистачити для успішного проведення навчання.

У наборі даних наявні фотографії різних розмірів та якості, з різних джерел, що містять різні образи та велике різноманіття кольорових розподілів, що і буде корисно у нашій задачі, адже розроблені нейронні мережі повинні мати дуже широке уявлення про умовно правильні розподіли кольорів для різних зображень. Приклади даних наведено на рис. 3.3.

Також, у подальших дослідженнях планується виконати тренування на інших наборах даних, адже дані з IMAGENET підлягають під дію закону про авторське право та є доступними для використання у різних сферах тільки у рамках академічних та некомерційних досліджень. Отже, якщо на меті є варіант комерційного застосування розроблюваної технології, необхідно будет виконати повторне тренування порожньої моделі (початкова модель за заданою архітектурою, усі ваги якої дорівнюють нулю) на новому наборі даних, для якого наявний дозвіл використання даних у комерційних проєктах.

Рис. 3.3. Приклади зображень набору даних IMAGENET 2012 [15]

Наступним важливим кроком у розробці є правильний вибір методів формування навчальної пари для навчання з учителем та попередня обробка даних, що є необхідною для подальшого подання навчальної пари у мережу.

3.4.2. Обгрунтування вибору кольорової схеми

У результаті дослідження існуючих методів колоризації та проведення певних власних експериментів було виявлено, що стандартна RGB-нотація для зображень не є вичерпною з точки зору інформації про кольори окремих об’єктів, дану нотацію актуально використовувати у задачах детекції образів, коли необхідно аналізувати окремо узяті пікселі і знаходити їх взаємну залежність. Для задачі колоризації нам необхідно оцінювати об’єкти за допомогою каналу світлості, адже як було доведено у методі “Colorization using optimization” один окремий об’єкт на зображенні має наближені значення “світлості” пікселів. Тому для тренування було протестовано використання двох кольорових схем, які відокремлюють світлоту у окремий

канал та визначають колір за допомогою двох інших каналів – YUV та LAB.

YUV – це система подання кольорів яка кодує кольорове зображення або відео з урахуванням людського сприйняття, дозволяючи зменшити пропускну здатність для компонентів кольору, тим самим, що дозволяє ефективніше замаскувати помилки передачі або артефакти стиснення для людського сприйняття. Модель YUV визначає кольоровий простір з точки зору одного компонента освітленості (Y) та двох компонентів кольору, що називаються U (синя проекція) та V (червона проекція) відповідно.

Переведення зображення з простору RGB у простір YUV відбувається наступним чином:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \qquad (3.5)$$

LAB – це кольорова схема, що виражає колір як три значення: L для освітленості від чорного (0) до білого (100), A – від зеленого (–128) до червоного (+127), та B від синього (–128) до жовтого (+127). LAB був розроблений таким чином, що однакова кількість числових змін цих значень відповідає приблизно однаковій кількості візуально сприйнятих змін. Оскільки вимірюються три параметри, сам простір – це тривимірний простір реальних чисел, у якому можливо отримати нескінченно багато можливих кольорів. На практиці простір зазвичай відображається на тривимірний цілий простір для цифрового представлення, і таким чином значення L, a і b зазвичай є абсолютними, заздалегідь визначеним діапазоном. Значення світлості, L, являє собою найтемніший чорний при $L = 0$, та найяскравіший білий при $L = 100$. Кольорові канали a і b, представляють нейтрально сірі значення при $a = 0$ і $b = 0$. Значення та межі осей a і b залежать від конкретної реалізації, як описано нижче, але частіш за все вони перебувають у діапазоні ±100 або від –128 до +127.

Загалом обрані формати видають приблизно однакові результати при навчанні і у цілому мають ряд спільних рис, тому який саме формат є найкращим ще не було вирішено.

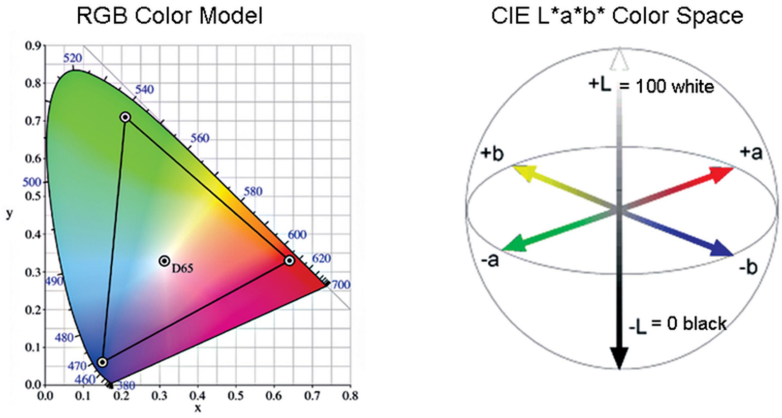


Рис. 3.4. Порівняння кольорових схем RGB та LAB [14]

Отже, як стає зрозумілим з цього підрозділу, перед поданням зображень на навчання, ми виконуємо переведення зображення з RGB-нотації до LAB (YUV), і розділяємо канали L (Y) та AB (UV) для застосування на різних етапах тренування, про що докладніше буде розказано у наступному розділі.

4. РЕАЛІЗАЦІЯ ТА АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБРОБЛЕННЯ ЗОБРАЖЕНЬ

У попередніх розділах магістерської дисертації було проаналізовано існуючі рішення задачі, математичні алгоритми та алгоритми штучного інтелекту для роботи з зображеннями, зазначено основні властивості візуальної інформації та її представлення у цифровому вигляді. На основі проведених досліджень вже є можливим сформулювати, як і технічні вимоги та рекомендації до системи, так і вимоги до програмного продукту, який буде надавати користувачу доступ до технології. В рамках цієї магістерської дисертації метою є створити технологію для здійснення інтелектуальної колоризації зображень та якісний демонстраційний прототип до неї (у вигляді програмного додатку з веб-інтерфейсом).

Отже, вимоги та рекомендації до технології:

1. Необхідно розробити архітектуру до генеративної нейронної мережі, на основі вказаних у третьому розділі мереж генератора та дискримінатора.
2. Необхідно провести додаткові дослідження стосовно рекомендованих початкових параметрів для тренування.
3. Також необхідно розробити модуль обробки навчальної вибірки (набір даних з зображеннями). Даний модуль містить функції з обробки зображень, необхідні для підготовки тренувальної вибірки, а саме:
    i. Зниження розмірності та обрізка зображень.
    ii. Переведення зображень у монохромний вигляд.
    iii. Конвертація зображень до інших колірних схем, а саме YUV, LAB, GRAYSCALE, RGBA.

iv. Алгоритм для експорту кольорових міток об’єктів та їх конвертація у окремий шар зображення.
v. Функція поєднання декількох шарів з зображеннями.

4.1. Обгрунтування вибору початкових параметрів для навчання

Навчання нейронних мереж це складний та кропіткий процес, успіх якого напряму залежить від дуже багатьох факторів, таких як правильний вибір та підготовка даних, коректна архітектура обраних нейронних мереж (вони повинні мати вхідні та вихідні шари правильної розмірності, бути достатньо глибокими, проте не бути перевантаженими занадто батьма операціями, адже тоді це може сказатися не тільки на швидкості роботи мережі, але і на результатах, мати правильну логічну структуру – коректні послідовності операцій згортки, об’єднання, правильно підібрані функції активації). Також надзвичайно важливими є гіпер параметри мереж (внутрішні параметри, що використовуються під час процесу тренування та тестування нейронних мереж), такі як розмір батчу (batch size), алгоритм оптимізації градієнтного спуску та його внутрішні параметри, кількість епох навчання, learning rate, learning rate decay, розмірність вхідних даних, внутрішні параметри функцій помилок, що використовуються.

Розмірність вхідних даних – це заданий розмір зображення, який повинен відповідати вхідному шару мережі, до якої зображення подається. Даний параметр слід обирати емпіричним шляхом, проте як у задачах класифікації та детекції об’єктів, так і у задачах обробки зображень, чим вища розмірність вхідних та вихідних даних, тим більш задовільний результат роботи ми отримуємо у результаті навчання. Проте тренувати мережі на великих зображеннях не завжди є можливим, адже чим більший розмір зображення, тим більше математичних операція здійснюватиметься, та

більше відеопам’яті буде зайнято. Для виконання даної магістерської дисертації було обрано розмір зображень 256 на 256, адже він вміщується у обмеження 4 гігабайт відеопам’яті які були мені доступні для навчання (тренування мережі здійснювалось на відеокарті Nvidia GeForce 1050).

Розмір батчу – ще один параметр, який показує, скільки вхідних зображень мережа оброблюватиме за одну ітерацію навчання. Від цього параметру залежить швидкість навчання, проте на цей параметр накладається обмеження стосовно кількості наявної відеопам’яті. Тому під час проведення тренування, розмір батчу становив 2, отже за одну ітерацію, мережа оброблювала 2 зображення розміром 256 на 256 пікселів одночасно.

Learning rate – це один з найголовніших параметрів навчання, адже він показує швидкість (міру) навчання мережі за ітерацію, а саме впливає на те, наскільки сильно у сторону шуканого глобального мінімуму будуть змінюватися ваги мережі. Якщо обрати цей параметр занадто малим, то навчання буде просуватися дуже повільно та не матиме можливість виходити з локальних мінімумів, а якщо завеликим – тоді існує великий ризик потрапляння навчання у якийсь локальний мінімум (в такому випадку навчання необхідно перезапускати зі зміненими параметрами). Даний параметр обирається емпірично, при навчанні моделей, розглянутих у цій магістерській дисертації було обрано значення learning rate рівним 0.001, що є досить невеликим, що допомагає уникати попадання навчання у локальний мінімум. Дане значення було узято на основі розглянутих схожих рішень для обробки зображень.

Також для того, щоб стабілізувати навчання, learning rate було зроблено динамічним, а саме застосовано learning rate decay – поступове зменшення

learning rate з часом (під час навчання learning rate несуттєво зменшувався кожні 100 ітерацій).

У якості алгоритма оптимізації було обрано алгоритм Adam. У наступному пункті детально описується цей алгоритм та його альтернативи, та обґрунтовується вибір для даної магістерської дисертації.

4.1.1. Алгоритми оптимізації градієнтного спуску

При навчанні нейронних мереж, найголовнішою операцією звичайно є градієнтний спуск – метод пошуку оптимального розв'язку поставленої задачі у багатомірному просторі. Градієнтний спуск застосовується для поступового приведення значень вагів різних шарів нейронної мережі у відповідність до даних, на яких навчається мережа. Таким чином, шари нейронної мережі та їх фільтри навчаються виявляти та розподіляти між собою певні особливості вхідних даних та інтерпретувати їх необхідним чином, для повернення якісного результату.

З цим процесом існує велика кількість труднощів та проблем, адже сам по собі метод градієнтного спуску має ряд недоліків. Саме для вирішення цих недоліків застосовуються алгоритми оптимізації – по суті це покращені версії стандартного алгоритму градієнтного спуску з певними модифікаціями, які допомагають стабілізувати навчання та уникати притаманих градієнтному спуску недоліків, таких як потрапляння у локальний мінімум функції, необхідність обчислювати середнє значення градієнта на усьому наборі даних та інших.

Перший з таких алгоритмів це стохастичний градієнтний спуск – модифікація градієнтного спуску, у якій пропонується модифікувати ваги, обчислюючі середнє значення результатів на одному батчі вхідних даних [36]. Це суттєво прискорює процес навчання у порівнянні з класичним

варіантом, у якому ваги модифікуються на основі підрахунку результатів усього набору даних. Також цей метод дозволив виконувати навчання будь якого обсягу даних на відеокартах (без цієї оптимізації могло просто не вистачити пам'яті).

Також при оновленні значень вагів враховується магнітуда отриманих результатів, формулу оновлення результатів наведено у формулі:

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}, \tag{4.1}$$

де α – learning rate, $\frac{\partial L}{\partial w_t}$ – поточне значення градієнту.

RMSProp – інший метод оптимізації, основною ідеєю якого є те, що звичайне множення learning rate на значення градієнту не є досить гнучким та правильним підходом [37]. Замість цього, у алгоритмі пропонується додатково ділити learning rate на експоненціальну ковзаючу середню, яка за своїм визначенням враховує попередні зсуви градієнту в ту чи іншу сторону, тим самим задає навчання у напрямку більш адаптивно. Загальна формула обчислення нових значень вагів матиме вигляд:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \varepsilon}} \cdot \frac{\partial L}{\partial w_t}, \tag{4.2}$$

де $\varepsilon = 10^{-6}$ (для уникнення ділення на 0), α – learning rate, $\frac{\partial L}{\partial w_t}$ – поточне значення градієнту. v_t визначається як:

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2, \tag{4.3}$$

де β – константа, що показує вплив попереднього значення параметру на наступне.

Momentum – це інший алгоритм оптимізації градієнтного спуску, який також має адаптивний характер у роботі з градієнтами [36]. Основна ідея алгоритму полягає у аккумуляції попередніх значень градієнтів для врахування цих значень при наступному просуванні.

У цьому методі вводиться поняття моментуму, яке використовується при обчисленні вагів за наступною формулою:

$$w_{t+1} = w_t - \alpha m_t, \tag{4.4}$$

де α – learning rate, а значення mt визначається наступним чином:

$$m_t = \beta m_{t-1} + (1-\beta)\frac{\partial L}{\partial w_t} \tag{4.5}$$

Зазвичай параметр β обирають рівним 0.9, що як можна відзначити з формули, сприяє тому, що більшу вагу мають попередні значення моментуму та градієнтів, ніж поточне. Це допомагає змінювати значення вагів у правильно напрямку спадання заданої функції з правильною швидкістю, обходячи локальні мінімуми та зменшуючи кількість ітерацій для знаходження оптимального розв’язку задачі.

Adam це один з головніших на сьогодні алгоритмів оптимізації градієнтного спуску, який базується на ідеях градієнтного спуску з моментумом, та алгоритмом RMSProp [37]. Отже, для коригування вагів, тут враховується попередні зсуви градієнтів, які впливають як на значення learning rate, так і на поточні значення вагів. Загальна формула для вагів має вигляд:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \varepsilon}} \cdot m_t, \tag{4.6}$$

значення m_t та v_t визначаються за формулами:

$$m_t = \frac{m_t}{1-\beta_1^t},\ m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}; \tag{4.7}$$

$$v_t = \frac{v_t}{1-\beta_2^t},\ v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2. \tag{4.8}$$

Параметри β_1 та β_2 задаються при ініціалізації і частіше за все мають значення $\beta_1 = 0.9$ та $\beta_2 = 0.999$.

За результатами багатьох досліджень було доведено, що алгоритм Adam є дуже надійним та якісним оптимізатором для нейронних мереж, особливо часто він зустрічається при навчання згорткових нейронних мереж та при роботі з зображеннями. Тому для виконання тренування обраної архітектури було обрано саме цей метод оптимізації.

4.1.2. Обґрунтування вибору функцій помилок

При навчанні, для функцій помилок також є можливим задавати певні параметри щоб більш точно та якісно поставити задачу для нейронної мережі. Такі параметри доцільно використовувати для балансування мережі між декількома поставленими задачами (коли функція для оптимізації є досить складною та складається з кількох різних задач, якість рішення яких враховуються при підрахунку помилки).

Перша і найголовніша задача – базовий метод для колоризації зображення. Отже, нашою метою є, щоб мережа, отримуючи на вхід зображення без інформації про кольори, навчилася доповнювати чорно-біле зображення кольорами. Для цього наша мережа спроєктована таким чином, щоб на основі поданого на вхід каналу світлоти L генерувати канали A та B, що міститимуть кольорову мапу зображення. Для цього при обчисленні функції помилок, нам необхідно обчислити наскільки якісно мережа генерує кольорову мапу зображення шляхом порівняння результату роботи мережі з

оригінальною кольоровою мапу початкового зображення. Сама задача, яку виконує мережа називається реконструкцією зображення (коли ми маємо доповнити обмежену вхідну інформацію щоб отримати на виході правильне зображення, максимально подібне до оригіналу). При правильному навчанні, завдяки такій функції помилок, нейронна мережа зможе узагальнювати певні знання про кольорові розподіли різних зображень, різних об'єктів, і, якщо уникнути перенавчання, то за допомогою такої нейронної мережі буде можливим виконувати реконструкцію чорно-білих фото. Отже, початково, наша функція помилок має такий вигляд:

$$L_1(P) = \frac{1}{N} \sum_{p \in P} \left| x(p) - y(p) \right|, \tag{4.9}$$

де p – індекс пікселя на зображеннях x та y.

Сутність роботи цієї функції помилок заключається у по-піксельному порівнянні отриманого розподілу кольорів з оригінальним розподілом, для забезпечення стабільної роботи мережі у рамках задачі колоризації вхідного чорно-білого каналу зображення. Дана лосс-функція використовується при навчанні як основна функція для оцінки роботи мережі-генератора.

Окрім основної мережі-генератора, до навчання також було додано дві допоміжні мережі – мережу глобальних підказок (яка бере інформацію з оригіналу зображення) та мережу користувацьких підказок (вхідними даними для такої мережі буде кольорова мапа, намальована користувачем). Детальніше про ці мережі буде розказано у наступному пункті, наразі розглянемо їх функції помилок та початкові налаштування.

Мережа локальних підказок приймає на вхід зображення з точками чи лініями різного кольору, які було намальовано користувачем у спеціальному веб-інтерфейсі і переведено у кольорову схему LAB. Необхідно зауважити, що на вхід поступають лише A та B канали цього зображення. Також на вхід

подається бінарна маска, що відповідає заповненим кольором частинам користувацької мапи кольорів (щоб відрізняти яке походження мають різні кольорові підказки). Отже, вхідними даними до цієї мережі буде тензор з числами з плаваючою комою від 0 до 1, розміром H x W x 3, де H, W – розмір оброблюваного зображення (мережі, про які йдеться мова у даній магістерській дисертації навчалась на зображеннях з розміром 256 на 256 пікселів). Для тренування, такі кольорові підказки було згенеровано шляхом визначення числа точок (підказок) які будуть нанесені, та визначенням позиції на зображенні (усі ці числа визначались за допомогою стандартного алгоритму генерації випадкових чисел random.randint(a, b), який входить у базовий набір бібліотек мови Python). Кольори було узято з початкового кольорового зображення. Така операція робилась для кожного вхідного зображення під час навчання, і за результатами експериментів можна стверджувати, що такий спосіб покриває достатню кількість випадків, тож може забезпечити стабільну роботу мереж під час тестування та експлуатації. Вхідні дані проходять обробку паралельно до основної мережі-генератора і у результаті роботи мережі локальних підказок перетворюються на кольоровий розподіл (мережа виконує прогнозування каналів A та B), якій надалі передається у шари мережі-генератора. Отриманий у даній мережі розподіл попіксельно оцінюється з оригінальним розподілом за допомогою кросс-ентропії (по суті, ми вимірюємо відстань у просторі між двома розподілами).

Мережа глобальних підказок це альтернативний варіант подачі користувацької інформації, у даному випадку на вхід подається глобальна гістограма кольорів та усереднене значення насиченості зображення (у межах від 0 до 1) [38]. Під час навчання ці дані було узято з початкового зображення і з вірогідністю у 25% подавались до мережі-генератора, для коригування

процесу перетворення згідно з кольорами та яскравістю оригінального зображення.

У даному підрозділі було наведено певні початкові параметри, техніки та процеси, які надалі використовувалися під час навчання системи нейронних мереж. У наступних підрозділах буде описано деталі та особливості реалізації загальної техніки інтерактивної колоризації зображень.

4.2. Реалізація системи нейронних мереж та процесу навчання

Як було раніше зазначено, дана магістерська дисертація присвячена реалізації технології інтерактивної колоризації нейронних мереж на основі генеративних мереж, отже, головними компонентами створеної системи є мережа-генератор та мережа-дискримінатор, архітектури яких описано у попередньому розділі. У даному підрозділі будуть описані певні важливі деталі реалізації та модифікації, що було застосовано як на рівні архітектур, так і у процесі навчання.

4.2.1. Модуль генерації зображень

У даному модулі відбувається процес обробки вхідного зображення та генерація кольорового розподілу зображення, з урахуванням користувацьких змін. Загалом процес генерації можна розділити на три умовні гілки – головну гілку (перетворення вхідного чорно-білого зображення на кольорове, шляхом прогнозування реалістичного кольорового розподілу), гілку локальних підказок користувача (інтерпретація користувацьких підказок у необхідний вигляд для подання у головну мережу-генератор) та гілку глобальних підказок (експериментальна мережа, що виконує обчислення певних характеристик початкового зображення та подає їх у головну мережу). Почнемо з огляду особливостей головної мережі-генератора.

Головна мережа-генератор, побудована на архітектурі U-Net, є повною згортковою мережею з наявними блоками згортки вхідного зображення і розгортки його з урахуванням внесеною під час обробки допоміжної інформації з інших гілок модуля генерації. На вході у цю мережу під час навчання подається L-канал початкового зображення, на більш пізніх етапах згортки та розгортки також додається інформація про користувацькі підказки та глобальний розподіл початкового зображення. Архітектура мережі складається з 10 послідовних блоків згортки, у блоках 1-4 інформація зменшується вдвічі у просторовому вимірі та подвоюється у вимірі особливостей після кожної згортки. У блоках 7-10 відбувається зворотна зміна (по суті, застосовується розгортка). На блоках згортки 5-6 у тензор додається додаткова інформація. Також, як і у оригінальному варіанті мережі U-Net, у даній мережі присутні міжшарові поєднання (shortcut connections), які передають інформацію про особливості мережі не тільки у наступні шари, але й у симетричні шари розгортки (з 2-го шару згортки інформація передається в 9-й, з 3-го у 8-й із 4-го у 7-й). Таким чином ми забезпечуємо врахування особливостей різних рівнів на всіх етапах обробки і уникаємо забування контексту. Для запобігання інформаційних викидів (коли після згортки певні особливості мають аномально великі чи маленькі значення, у порівнянні з іншими особливостями у батчі) у мережі використовується техніка Batch Normalization, про яку детальніше розповідається у третьому розділі дисертації.

Мережа локальних підказок отримує на вхід зображення з користувацькою мапою кольорів, поєднане з чорно-білим каналом вхідного зображення (рис. 4.1.). Задачею цієї мережі є спрогнозувати можливий розподіл кольорів на зображенні при врахуванні початкового зображення та інформації про кольори з боку користувача. По суті, мережа локальних

підказок є механізмом для подачі користувацької інформації на різних шарах головної мережі.

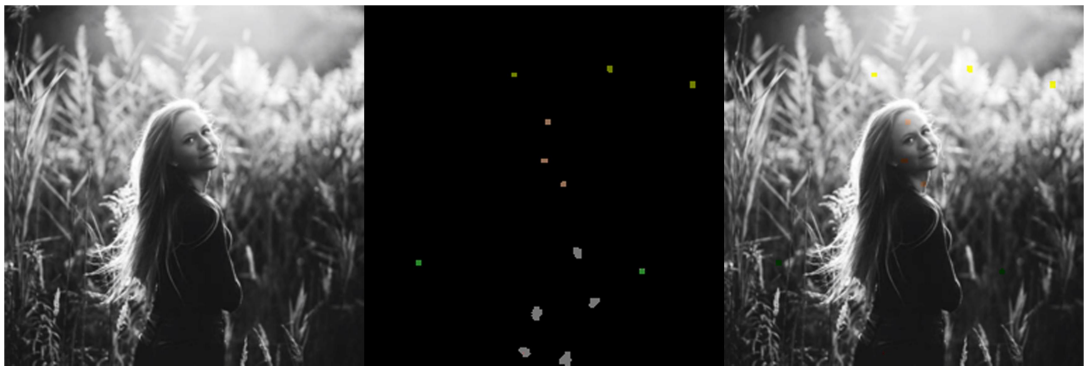


Рис. 4.1. Приклад вхідних даних для мережі-генератора та мережі локальних підказок.

Мережа глобальних підказок також є допоміжним механізмом, для коригування задачі колоризації, що здійснюється шляхом попередньої обробки початкового зображення для отримання значень його характеристик, а саме гістограми значень кольорів та коефіцієнт насиченості зображення. Для цього оригінальне кольорове зображення спочатку зменшують у чотири рази за допомогою алгоритму білінійної інтерполяції, потім кодують кожен окремий піксель таким чином, щоб знизити кількість різних кольорів (таке явище часто називають квантизацією кольорів), а потім вираховують середні значення. Таким чином ми отримуємо гістограму розподілу кольорів на початковому зображенні. Для отримання коефіцієнту насиченості, початкове зображення переводять у кольоровий простір HSV, після чого підраховують середнє значення насиченості, враховуючи лише канал S цього зображення. Далі ця вхідна інформація проходить 4 блоки згортки, аналогічних до головної мережі, після чого інформація набуває необхідної просторової

розмірності щоб бути поданою у 5-й шар головної мережі для врахування при подальшому перетворенні.

4.2.2. Модуль дискримінатора

Вище було описано основні принципи роботи модуля генерації зображення, який є незалежним і може у повному обсязі самостійно виконувати задачу колоризації зображень. Проте, шляхом різних попередніх досліджень було доведено, що згорткові нейронні мережі, призначені для обробки зображень, мають ряд недоліків, таких як виникнення зайвих артефактів, ефектів розмиття на зображеннях при реставрації зображень та усереднення кольорів. Особливо остання проблема має місце майже у всіх дослідженнях задачі колоризації. Причина цієї проблеми полягає у тому, що при навчання згорткової мережі нашею задачею є мінімізувати певну лосс-функцію, що так чи інакше є критерієм, за яким мережа має оцінювати власну роботу. Таким чином, при успішному навчанні, значення лосс-функції мінімізується і навчання сповільнюється. Така поведінка є нормальною, і у такому випадку є сенс призупиняти навчання і проводити тестування моделі. Проте проблеми обробки зображень є дуже чутливими, і єдиною реально об’єктивною метрикою оцінки є оцінка “на око”, саме тільки продивившись результати роботи мережі, можна робити релевантні висновки стосовно якості моделі. Отже, проводячи аналіз роботи моделей, що виконують обробку зображень було зроблено висновок, що на математичному рівні модель правильно слідує усім вказівкам і мінімізує задану функцію помилок, проте при цьому відбувається “підлаштовування” вагів моделі під тренувальну вибірку. Це явище називається перенавчанням (overfitting), міри для боротьби з ним вже якісно досліджені і використовувалися при навчанні описаних у цій дисертації нейронних мереж. Проте також, при колоризації

зображень мережа також може усереднювати кольорові розподіли з зображень тренувальної вибірки, і якщо, наприклад, більшість зображень, на яких навчалась мережа, мали зелений відтінок, то більшість результатів також матимуть цей відтінок, навіть якщо це суперечитиме постановці задачі і контексту даних. Це один з важливих недоліків згорткових мереж, для боротьби з яким використовують мережі-дискримінатори – окремі згорткові мережі, які вивчають згенеровані зображення, та порівнюють їх з оригінальними даними, не орієнтуючись на певні метрики. Така мережа є бінарним класифікатором, який по мапі особливостей зразка, може визначити чи є він підробкою, чи ні. Саме таку мережу і було використано при тренуванні, для коригування помилок навчання та стабілізації результатів. Детальніше про архітектуру мережі-дискримінатора розказано у попередньому розділі. На вхід така мережа приймає одне зображення, і за допомогою послідовних блоків згортки аналізує зображення в цілому, та його окремі частини (обрана архітектура для даної магістерської дисертації розкладає зображення на 32 частини) і робить окремі передбачення стосовно кожної окремої частини зображення, після цього повертає у якості результату масив з вірогідністю справжньості зображення, та вірогідністю того, що зображення виявиться підробкою. Під час тренування, на вхід дискримінатора по черзі подавались оригінальні зображення (очікуваний результат для яких є масив [1.0, 0.0]), та зображення, згенеровані мережею (очікуваний результат роботи для яких є масив [0.0, 1.0]). Надалі отримані передбачення враховувались при розрахунку значення лосс-функції дискримінатора, яка показує наскільки якісно він натренований відрізняти підробки від справжніх фото.

4.2.3. Процес навчання

Навчання відбувалося на усьому зібраному наборі даних з приблизно одного мільйона зображень протягом 6-7 днів. Загалом навчання було поділено на епохи, за одну епоху мережа навчалася на усьому наборі даних. Під час навчання фінальної версії моделі було проведено 25 епох, що не є вичерпною кількістю, тому у майбутньому, з метою покращення результатів планується збільшити кількість епох навчання.

Епоха поділена на ітерації, адже ми маємо певні обмеження кількості даних, які за один раз можуть пройти обробку у мережі. З обраним розміром батчу 2, кількість ітерацій у межах однієї епохи становила приблизно 500 000.

Одна ітерація навчання мала наступний вигляд:

- Підготовка вхідних даних для модуля генерації;
- Подання вхідних даних для розрахунку у модулі генерації;
- Подання згенерованого та справжнього зображення до модуля дискримінатора;
- Підрахунок лосс-функцій та здійснення зворотного поширення;

У якості ідей для покращення процесу навчання, було вирішено згодом збільшити кількість епох навчання, можливо внести зміни у лосс-функції генератора та змінити спосіб подання даних (щоб навчити мережу адекватно реагувати на кольорові помітки більші за площею, інтерпретувати вхідні помітки більш якісно). Також, можливо, покращити процес навчання може зміна оптимізатора зворотного поширення на більш гнучкий та адаптований під навчання генеративних моделей.

4.3. Аналіз процесу навчання та тестування системи мереж

Після того, як було задано усі початкові параметри для навчання, підготовлено дані для обробки та спроєктовано мережі, було розпочато

процес навчання. Загалом, цей процес має емпіричний характер, тобто під час проведенно деякої кількості навчання виявляються наявні помилки, виникають певного роду проблеми. Тому навчання можна назвати ітераційним процесом, адже після кожної знайденої несправності, необхідно усунути причину проблеми та запустити навчання заново. Поширеною практикою є використовувати заздалегідь натреновані мережі (pre-trained models), які були натреновані на задачі дещо іншого виду, проте співпадають за архітектурою. Ця практика є дуже корисною при обмеженій кількості власних даних, або при вирішенні задач, які частково вже було вирішено. По суті, замість початкових значень ваг рівних 0 чи згенерованих за допомою розподілу випадкових чисел, ми беремо вже натреновані значення вагів для подальшої оптимізації вже під нову задачу. Такий підхід зазвичай використовується разом з технікою “замороження” (freeze) вагів – процес, при якому той чи інший шар нейронної мережі не навчається протягом тренування. Техніка “замороження” використовується для початкових шарів згорткових мереж, на цих шарах мережа вивчає низькорівневі особливості зображень, такі як контури об’єктів, і як показує практика, будь яка згорткова мережа, що навчалась на зображеннях, на цих шарах має приблизно однакові значення та розпізнає одні і ті ж самі особливості. Тому сенсу витрачати час та потужності на навчання цих шарів немає.

Отже, коли усі налаштування виконано і помилки у значеннях параметрів та коді виправлено, можна розпочати повне тренування. Загалом під час тренування аналізувати стан речей можливо двома способами: переглядом та аналізом графіків лосс-функцій та проведенням валідації (аналіз роботи мережі на нових зображеннях). З метою досягнення якісних результатів навчання, було проведено аналіз роботи мережі обома цими

способами. На рис. 4.2. можна побачити графік лосс-функції мережі-генератора, а на рис. 4.3. графік лосс-функції мережі-дискримінатора.

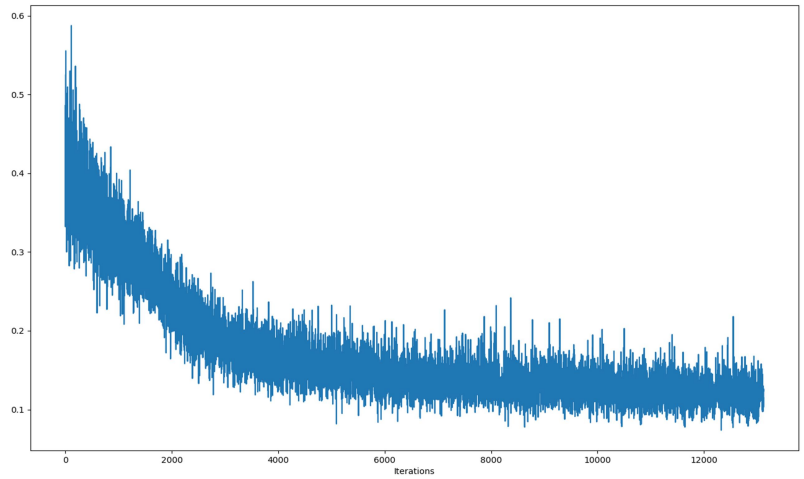


Рис. 4.2. Графік значень лосс-функції мережі-генератора протягом навчання

Як можна побачити на графіку мережі-генератора, навчання виконується правильним чином, про що свідчить монотонне спадання значень функції помилок, що каже нам про те, що мережа успішно навчається виконувати поставлену задачу.

За графіком мережі-дискримінатора, можна зробити висновок, що ця мережа також досить успішно навчається, хоча темп навчання демонструє більш повільний ніж мережа-генератор (це пов’язано з тим, що навчання дискримінатора відбувається не кожної ітерації, а лише кожну 5-ту ітерацію).

Загалом, за вказаними графіками, можна зробити висновок, що навчання проходить у правильному руслі, проте для отримання більш надійних та якісних мереж, необхідно продовжувати тренування, що і є у планах на подальший розвиток цього дослідження.

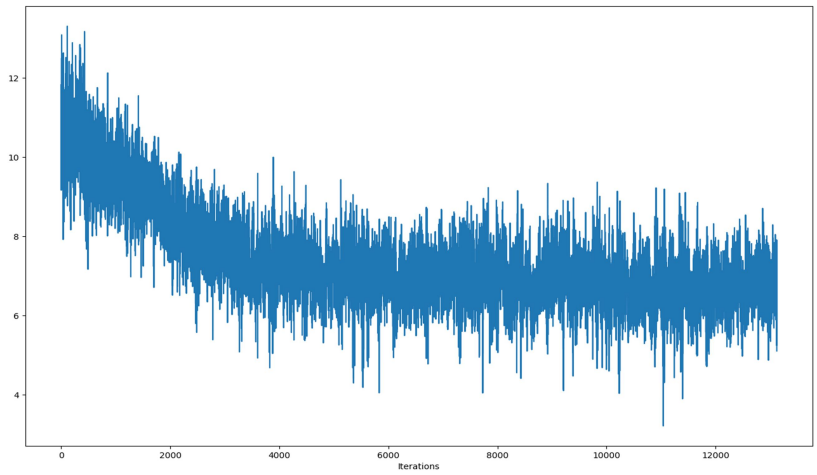


Рис. 4.3. Графік лосс-функції мережі-дискримінатора

Для тестування розробленої системи нейронних мереж, було розроблено простий веб-сервіс на мові Python з використанням фреймворку для веб-сервісів Django та декількох скриптів на мові JavaScript, який надає можливість користувачу завантажити будь-яке фото, нанести певні кольорові позначки на даний інтерфейс та виконати обробку фото за допомогою натренованих моделей. Результати проведеного тестування та висновки буде наведено у наступному підрозділі. Розроблене середовище для тестування зображено на рис. 4.4.

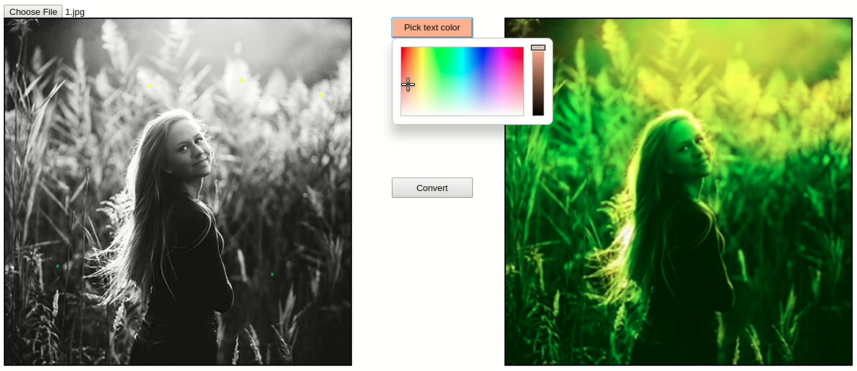


Рис. 4.4. Веб-інтерфейс для тестування моделі

4.4. Аналіз отриманих результатів та висновки

За результатами тестування натренованих моделей для інтерактивної колоризації зображень, було виявлено, що не дивлячись на неповний обсяг тренування, моделі вже досить якісно здатні працювати у автоматичному режимі (без користувацьких підказок), а саме підбирати приблизно схожі до оригіналу розподіли кольорів та якісно визначати межі різних об’єктів.

Рис. 4.5. Результат автоматичної колоризації зображення

Також, доволі непогано мережа працює з користувацькими підказками, якісно визначає межі об’єктів, підбирає відтінки орієнтуючись на освітлення та наявні на початковому зображенні тіні. У деяких випадках мережа не замальовує увесь об’єкт, як того хотів би користувач, в такому випадку необхідно додати ще декілька кольорових підказок у ті сегменти зображення, які мережа пропустила. Найбільш якісно мережа працює з чорно-білими зображеннями, особливо у випадку, коли вказані користувачем кольори є реалістичними. Приклад такої роботи наведено на рис. 4.6.

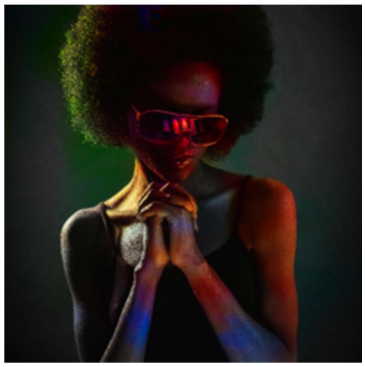


Рис. 4.6. Приклад роботи системи з використанням користувацьких підказок

Тоді результати колоризації виходять дуже високоякісними, а от з надто яскравими та неприродніми кольорами часто виникають певного роду “непорозуміння” (рис. 4.7.).

Загалом, судячи з результатів проведеного дослідження, можно з упевненістю заявити, що розроблений підхід до вирішення задачі інтерактивної колоризації демонструє досить непогані результати у порівнянні з попередніми дослідженнями, а також демонструє найкращу швидкість роботи (разом з попередньою обробкою зображень, час виконання дорівнює приблизно одній секунді).

Рис. 4.7. Приклад дефектів системи при використанні неприродніх кольорів

Також варто зазначити, що отриманий результат роботи не є кінцевим і існує декілька способів його покращити, такі як більш довге тренування, переналаштування параметрів та додавання до навчальної вибірки нових даних.

У рамках даної магістерської дисертації було проаналізовано існуючі методи колоризації зображень, досліджено способи автоматичної та напівавтоматичної колоризації, досліджено дефекти зображень та способи їх усунення. Було розглянуто математичні методи обробки зображень та сучасні алгоритми штучного інтелекту для вирішення поставленої задачі та розроблено власний підхід, на основі генеративних змагальних мереж, що дозволяє виконувати інтелектуальну колоризацію зображень за вказівками користувача за оптимальний час та з високою якістю отриманих результатів.

ВИСНОВКИ

Метою даної магістерської дисертації було розроблення методу для інтелектуальної колоризації зображень за участю користувача у процесі.

На початковому етапі роботи над дисертацією було виконано наступні роботи:

- аналіз цифрових зображень та їх недоліків;
- аналіз існуючих методів обробки зображень, алгоритмів для усунення недоліків на зображеннях;
- аналіз алгоритмів колоризації зображень;
- аналіз теоритичного підгрунтя та існуючих архітектур згорткових нейронних мереж та генеративних змагальних мереж;
- опанування підходів сегментації та обробки зображень за допомогою алгоритмів штучного інтелекту ;
- формулювання науково-інноваційної задачі.

На основі результатів виконання цих робіт було сформульовано метод інтелектуальної колоризації зображень за допомогою користувацьких підкахок та генеративних змагальних мереж, навчених на великій тренувальній вибірці. У ході дослідження було спроєктовано 3 взаємопов'язані нейронні мережі, це мережа-генератор, яка виконує обробку вхідного зображення, мережа-дискримінатор, яка аналізує згенероване зображення, та мережа локальних підказок, яка використовується для внесення користувачем власних побажань та для здійснення контролю над процесом обробки.

Сформульований метод було реалізовано у повній мірі, проведено якісне тренування моделей та отримано непогані результати, що випереджають за якістю та швидкістю попередні методи. Було проведено

детальне тестування моделей та розроблено веб-інтерфейс для подальшої експлуатації моделі. Виконано аналіз отриманих результатів дослідження та визначено подальші напрямки вдосконалення.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Wikipedia – Цифрове зображення [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Цифрове_зображення – Дата доступу: 04.02.2020 – Назва з екрану.
2. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1609.04802.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
3. Let’s Enhance [Електронний ресурс] – Режим доступу: https://letsenhance.io/ – Дата доступу: 04.02.2020 – Назва з екрану.
4. Wikipedia – Векторна графіка [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/ Векторна графіка – Дата доступу: 04.02.2020 – Назва з екрану.
5. Оптичні дефекти зображення [Електронний ресурс] – Режим доступу: https://fototips.ru/praktika/opticheskie-defekty-izobrazheniya/ – Дата доступу: 04.02.2020 – Назва з екрану.
6. Wikipedia - Зубці (комп'ютерна графіка) [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Зубці (комп'ютерна графіка) – Дата доступу: 04.02.2020 – Назва з екрану.
7. Муар (візерунок) [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D1%83%D0%B0%D1%80_(%D0%B2%D1%96%D0%B7%D0%B5%D1%80%D1%83%D0%BD%D0%BE%D0%BA) – Дата доступу: 04.02.2020 – Назва з екрану.
8. Colour_cast [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Colour_cast – Дата доступу: 04.02.2020 – Назва з екрану.

9. Image blurring [Електронний ресурс] – Режим доступу: https://patents.google.com/patent/US7469071B2/en – Дата доступу: 04.02.2020 – Назва з екрану.
10. DeblurGAN [Електронний ресурс] – Режим доступу: https://arxiv.org/abs/1711.07064 – Дата доступу: 04.02.2020 – Назва з екрану.
11. what-is-depth-of-field [Електронний ресурс] – Режим доступу: https://photographylife.com/what-is-depth-of-field – Дата доступу: 04.02.2020 – Назва з екрану.
12. Red-eye_effect [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Red-eye_effect – Дата доступу: 04.02.2020 – Назва з екрану.
13. Colorization using optimization [Електронний ресурс] – Режим доступу: https://webee.technion.ac.il/people/anat.levin/papers/colorization-siggraph04.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
14. DeOldify [Електронний ресурс] – Режим доступу: https://github.com/jantic/DeOldify – Дата доступу: 04.02.2020 – Назва з екрану.
15. ImageNet Classification [Електронний ресурс] – Режим доступу: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
16. Convolutional Neural Networks [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks – Дата доступу: 04.02.2020 – Назва з екрану.

17. Image to Image translation [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/image-to-image-translation-69c10c18f6ff – Дата доступу: 04.02.2020 – Назва з екрану.
18. FCN Tensorflow[Електронний ресурс] – Режим доступу: https://towardsdatascience.com/implementing-a-fully-convolutional-network-fcn-in-tensorflow-2-3c46fb61de3b – Дата доступу: 04.02.2020 – Назва з екрану.
19. A Neural Algorithm of Artistic Style [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1508.06576.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
20. Deep Learning Colorization [Електронний ресурс] – Режим доступу: https://richzhang.github.io/colorization/ – Дата доступу: 04.02.2020 – Назва з екрану.
21. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Generative Adversarial Nets [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1409.1556.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
22. Generative Adversarial Nets [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1406.2661.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
23. Deep Learning for Deepfakes Creation and Detection [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1909.11573.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
24. U-Net: Convolutional Networks for Biomedical Image Segmentation [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1505.04597.pdf – Дата доступу: 04.02.2020 – Назва з екрану.

25. python.org [Електронний ресурс] – Режим доступу: https://bugs.python.org/file47781/Tutorial_EDIT.pdf
26. R Language [Електронний ресурс] – Режим доступу: https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
27. tensorflow.org [Електронний ресурс] – Режим доступу: https://www.tensorflow.org/ – Дата доступу: 04.02.2020 – Назва з екрану.
28. pytorch.org [Електронний ресурс] – Режим доступу: https://pytorch.org/ – Дата доступу: 04.02.2020 – Назва з екрану.
29. keras.io [Електронний ресурс] – Режим доступу: https://keras.io/ – Дата доступу: 04.02.2020 – Назва з екрану.
30. docs.nvidia.com [Електронний ресурс] – Режим доступу: https://docs.nvidia.com/cuda/ – Дата доступу: 04.02.2020 – Назва з екрану.
31. docs.opencv.org [Електронний ресурс] – Режим доступу: https://docs.opencv.org/2.4/index.html – Дата доступу: 04.02.2020 – Назва з екрану.
32. matplotlib.org [Електронний ресурс] – Режим доступу: https://matplotlib.org/3.2.1/contents.html – Дата доступу: 04.02.2020 – Назва з екрану.
33. pillow.readthedocs.io [Електронний ресурс] – Режим доступу: https://pillow.readthedocs.io/en/3.1.x/reference/Image.html – Дата доступу: 04.02.2020 – Назва з екрану.
34. Image-to-Image Translation with Conditional Adversarial Networks [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1611.07004.pdf – Дата доступу: 04.02.2020 – Назва з екрану.

35. ImageNet Dataset Overview [Електронний ресурс] – Режим доступу: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks_2.pdf – Дата доступу: 04.02.2020 – Назва з екрану.
36. Gradient Descent Optimization algorithms [Електронний ресурс] – Режим доступу: https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9 – Дата доступу: 04.02.2020 – Назва з екрану.
37. Optimizing Gradient Descent [Електронний ресурс] – Режим доступу: https://ruder.io/optimizing-gradient-descent/index.html#batchgradientdescent – Дата доступу: 04.02.2020 – Назва з екрану.
38. Real-Time User-Guided Image Colorization [Електронний ресурс] – Режим доступу: https://arxiv.org/pdf/1705.02999.pdf – Дата доступу: 04.02.2020 – Назва з екрану.

ДОДАТКИ

Додаток 1

Фрагменти тексту програми

Лістинг 1. Модуль реалізації архітектури генеративних змагальних мереж

```
import os
import torch
from collections import OrderedDict
from . import networks

class BaseModel():
    @staticmethod
    def modify_commandline_options(parser, is_train):
        return parser

    def name(self):
        return 'BaseModel'

    def initialize(self, opt):
        self.opt = opt
        self.gpu_ids = opt.gpu_ids
        self.isTrain = opt.isTrain
        self.device = torch.device('cuda:{}'.format(self.gpu_ids[0])) if
self.gpu_ids else torch.device('cpu')
        self.save_dir = os.path.join(opt.checkpoints_dir, opt.name)
        if opt.resize_or_crop != 'scale_width':
            torch.backends.cudnn.benchmark = True
        self.loss_names = []
        self.model_names = []
        self.visual_names = []
        self.image_paths = []

    def set_input(self, input):
        self.input = input

    def forward(self):
        pass

    # load and print networks; create schedulers
    def setup(self, opt, parser=None):
        if self.isTrain:
            self.schedulers = [networks.get_scheduler(optimizer, opt) for
optimizer in self.optimizers]
        if not self.isTrain or opt.load_model:
            self.load_networks(opt.which_epoch)
        self.print_networks(opt.verbose)

    # make models eval mode during test time
    def eval(self):
        for name in self.model_names:
            if isinstance(name, str):
                net = getattr(self, 'net' + name)
                net.eval()

    # used in test time, wrapping `forward` in no_grad() so we don't save
intermediate steps for backprop
    def test(self, compute_losses=False):
        with torch.no_grad():
            self.forward()
            if(compute_losses):
                self.compute_losses_G()
```

```
  def get_image_paths(self):
        return self.image_paths

    def optimize_parameters(self):
        pass

    # update learning rate (called once every epoch)
    def update_learning_rate(self):
        for scheduler in self.schedulers:
            scheduler.step()
        lr = self.optimizers[0].param_groups[0]['lr']
        print('learning rate = %.7f' % lr)

    # return visualization images. train.py will display these images, and
save the images to a html
    def get_current_visuals(self):
        visual_ret = OrderedDict()
        for name in self.visual_names:
            if isinstance(name, str):
                visual_ret[name] = getattr(self, name)
        return visual_ret

    # return traning losses/errors. train.py will print out these errors as
debugging information
    def get_current_losses(self):
        errors_ret = OrderedDict()
        for name in self.loss_names:
            if isinstance(name, str):
                # float(...) works for both scalar tensor and float number
                errors_ret[name] = float(getattr(self, 'loss_' + name))
        return errors_ret

    # save models to the disk
    def save_networks(self, which_epoch):
        for name in self.model_names:
            if isinstance(name, str):
                save_filename = '%s_net_%s.pth' % (which_epoch, name)
                save_path = os.path.join(self.save_dir, save_filename)
                net = getattr(self, 'net' + name)

                if len(self.gpu_ids) > 0 and torch.cuda.is_available():
                    torch.save(net.module.cpu().state_dict(), save_path)
                    net.cuda(self.gpu_ids[0])
                else:
                    torch.save(net.cpu().state_dict(), save_path)

    def __patch_instance_norm_state_dict(self, state_dict, module, keys, i=0):
        key = keys[i]
        if i + 1 == len(keys):  # at the end, pointing to a parameter/buffer
            if module.__class__.__name__.startswith('InstanceNorm') and \
                    (key == 'running_mean' or key == 'running_var'):
                if getattr(module, key) is None:
                    state_dict.pop('.'.join(keys))
            if module.__class__.__name__.startswith('InstanceNorm') and \
               (key == 'num_batches_tracked'):
                state_dict.pop('.'.join(keys))
        else:
            self.__patch_instance_norm_state_dict(state_dict, getattr(module,
key), keys, i + 1)
```

```
    # load models from the disk
    def load_networks(self, which_epoch):
        for name in self.model_names:
            if isinstance(name, str):
                load_filename = '%s_net_%s.pth' % (which_epoch, name)
                load_path = os.path.join(self.save_dir, load_filename)
                net = getattr(self, 'net' + name)
                if isinstance(net, torch.nn.DataParallel):
                    net = net.module
                print('loading the model from %s' % load_path)
                # if you are using PyTorch newer than 0.4 (e.g., built from
                # GitHub source), you can remove str() on self.device
                state_dict = torch.load(load_path,
map_location=str(self.device))
                if hasattr(state_dict, '_metadata'):
                    del state_dict._metadata

                # patch InstanceNorm checkpoints prior to 0.4
                for key in list(state_dict.keys()):  # need to copy keys here
because we mutate in loop
                    self.__patch_instance_norm_state_dict(state_dict, net,
key.split('.'))
                net.load_state_dict(state_dict)

    # print network information
    def print_networks(self, verbose):
        print('---------- Networks initialized -------------')
        for name in self.model_names:
            if isinstance(name, str):
                net = getattr(self, 'net' + name)
                num_params = 0
                for param in net.parameters():
                    num_params += param.numel()
                if verbose:
                    print(net)
                print('[Network %s] Total number of parameters : %.3f M' %
(name, num_params / 1e6))
        print('-----------------------------------------------')

    # set requies_grad=Fasle to avoid computation
    def set_requires_grad(self, nets, requires_grad=False):
        if not isinstance(nets, list):
            nets = [nets]
        for net in nets:
            if net is not None:
                for param in net.parameters():
                    param.requires_grad = requires_grad

import torch
import torch.nn as nn
from torch.nn import init
import functools
from torch.optim import lr_scheduler

###############################################################################
#
# Helper Functions
```

```
##############################################################################
#


def get_norm_layer(norm_type='instance'):
    if norm_type == 'batch':
        norm_layer = functools.partial(nn.BatchNorm2d, affine=True)
    elif norm_type == 'instance':
        norm_layer = functools.partial(nn.InstanceNorm2d, affine=False)
    elif norm_type == 'none':
        norm_layer = None
    else:
        raise NotImplementedError('normalization layer [%s] is not found' %
norm_type)
    return norm_layer


def get_scheduler(optimizer, opt):
    if opt.lr_policy == 'lambda':
        def lambda_rule(epoch):
            lr_l = 1.0 - max(0, epoch + 1 + opt.epoch_count - opt.niter) /
float(opt.niter_decay + 1)
            return lr_l
        scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda_rule)
    elif opt.lr_policy == 'step':
        scheduler = lr_scheduler.StepLR(optimizer,
step_size=opt.lr_decay_iters, gamma=0.1)
    elif opt.lr_policy == 'plateau':
        scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min',
factor=0.2, threshold=0.01, patience=5)
    else:
        return NotImplementedError('learning rate policy [%s] is not
implemented', opt.lr_policy)
    return scheduler


def init_weights(net, init_type='xavier', gain=0.02):
    def init_func(m):
        classname = m.__class__.__name__
        if hasattr(m, 'weight') and (classname.find('Conv') != -1 or
classname.find('Linear') != -1):
            if init_type == 'normal':
                init.normal_(m.weight.data, 0.0, gain)
            elif init_type == 'xavier':
                init.xavier_normal_(m.weight.data, gain=gain)
            elif init_type == 'kaiming':
                init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')
            elif init_type == 'orthogonal':
                init.orthogonal_(m.weight.data, gain=gain)
            else:
                raise NotImplementedError('initialization method [%s] is not
implemented' % init_type)
            if hasattr(m, 'bias') and m.bias is not None:
                init.constant_(m.bias.data, 0.0)
        elif classname.find('BatchNorm2d') != -1:
            init.normal_(m.weight.data, 1.0, gain)
            init.constant_(m.bias.data, 0.0)

    print('initialize network with %s' % init_type)
```

```
    net.apply(init_func)


def init_net(net, init_type='xavier', gpu_ids=[]):
    if len(gpu_ids) > 0:
        assert(torch.cuda.is_available())
        net.to(gpu_ids[0])
        net = torch.nn.DataParallel(net, gpu_ids)
    init_weights(net, init_type)
    return net


def define_G(input_nc, output_nc, ngf, which_model_netG, norm='batch',
use_dropout=False, init_type='xavier', gpu_ids=[], use_tanh=True,
classification=True):
    netG = None
    norm_layer = get_norm_layer(norm_type=norm)

    if which_model_netG == 'unet_128':
        netG = UnetGenerator(input_nc, output_nc, 7, ngf,
norm_layer=norm_layer, use_dropout=use_dropout)
    elif which_model_netG == 'unet_256':
        netG = UnetGenerator(input_nc, output_nc, 8, ngf,
norm_layer=norm_layer, use_dropout=use_dropout)
    elif which_model_netG =='siggraph':
        netG = SIGGRAPHGenerator(input_nc, output_nc, norm_layer=norm_layer,
use_tanh=use_tanh, classification=classification)
    else:
        raise NotImplementedError('Generator model name [%s] is not
recognized' % which_model_netG)
    return init_net(netG, init_type, gpu_ids)


def define_D(input_nc, ndf, which_model_netD,
             n_layers_D=3, norm='batch', use_sigmoid=False,
init_type='xavier', gpu_ids=[]):
    netD = None
    norm_layer = get_norm_layer(norm_type=norm)

    if which_model_netD == 'basic':
        netD = NLayerDiscriminator(input_nc, ndf, n_layers=3,
norm_layer=norm_layer, use_sigmoid=use_sigmoid)
    elif which_model_netD == 'n_layers':
        netD = NLayerDiscriminator(input_nc, ndf, n_layers_D,
norm_layer=norm_layer, use_sigmoid=use_sigmoid)
    elif which_model_netD == 'pixel':
        netD = PixelDiscriminator(input_nc, ndf, norm_layer=norm_layer,
use_sigmoid=use_sigmoid)
    else:
        raise NotImplementedError('Discriminator model name [%s] is not
recognized' %
                                  which_model_netD)
    return init_net(netD, init_type, gpu_ids)


##############################################################################
# Classes
##############################################################################
```

```
class HuberLoss(nn.Module):
    def __init__(self, delta=.01):
        super(HuberLoss, self).__init__()
        self.delta=delta

    def __call__(self, in0, in1):
        mask = torch.zeros_like(in0)
        mann = torch.abs(in0-in1)
        eucl = .5 * (mann**2)
        mask[...] = mann < self.delta

        # loss = eucl*mask + self.delta*(mann-.5*self.delta)*(1-mask)
        loss = eucl*mask/self.delta + (mann-.5*self.delta)*(1-mask)
        return torch.sum(loss,dim=1,keepdim=True)

class L1Loss(nn.Module):
    def __init__(self):
        super(L1Loss, self).__init__()

    def __call__(self, in0, in1):
        return torch.sum(torch.abs(in0-in1),dim=1,keepdim=True)

class L2Loss(nn.Module):
    def __init__(self):
        super(L2Loss, self).__init__()

    def __call__(self, in0, in1):
        return torch.sum((in0-in1)**2,dim=1,keepdim=True)


# Defines the GAN loss which uses either LSGAN or the regular GAN.
# When LSGAN is used, it is basically same as MSELoss,
# but it abstracts away the need to create the target label tensor
# that has the same size as the input
class GANLoss(nn.Module):
    def __init__(self, use_lsgan=True, target_real_label=1.0,
target_fake_label=0.0):
        super(GANLoss, self).__init__()
        self.register_buffer('real_label', torch.tensor(target_real_label))
        self.register_buffer('fake_label', torch.tensor(target_fake_label))
        if use_lsgan:
            self.loss = nn.MSELoss()
        else:
            self.loss = nn.BCELoss()

    def get_target_tensor(self, input, target_is_real):
        if target_is_real:
            target_tensor = self.real_label
        else:
            target_tensor = self.fake_label
        return target_tensor.expand_as(input)

    def __call__(self, input, target_is_real):
        target_tensor = self.get_target_tensor(input, target_is_real)
        return self.loss(input, target_tensor)


class SIGGRAPHGenerator(nn.Module):
```

```
    def __init__(self, input_nc, output_nc, norm_layer=nn.BatchNorm2d,
use_tanh=True, classification=True):
        super(SIGGRAPHGenerator, self).__init__()
        self.input_nc = input_nc
        self.output_nc = output_nc
        self.classification = classification
        use_bias = True

        # Conv1
        # model1=[nn.ReflectionPad2d(1),]
        model1=[nn.Conv2d(input_nc, 64, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model1+=[norm_layer(64),]
        model1+=[nn.ReLU(True),]
        # model1+=[nn.ReflectionPad2d(1),]
        model1+=[nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model1+=[nn.ReLU(True),]
        model1+=[norm_layer(64),]
        # add a subsampling operation

        # Conv2
        # model2=[nn.ReflectionPad2d(1),]
        model2=[nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model2+=[norm_layer(128),]
        model2+=[nn.ReLU(True),]
        # model2+=[nn.ReflectionPad2d(1),]
        model2+=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model2+=[nn.ReLU(True),]
        model2+=[norm_layer(128),]
        # add a subsampling layer operation

        # Conv3
        # model3=[nn.ReflectionPad2d(1),]
        model3=[nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model3+=[norm_layer(256),]
        model3+=[nn.ReLU(True),]
        # model3+=[nn.ReflectionPad2d(1),]
        model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model3+=[norm_layer(256),]
        model3+=[nn.ReLU(True),]
        # model3+=[nn.ReflectionPad2d(1),]
        model3+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model3+=[nn.ReLU(True),]
        model3+=[norm_layer(256),]
        # add a subsampling layer operation

        # Conv4
        # model47=[nn.ReflectionPad2d(1),]
        model4=[nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model4+=[norm_layer(512),]
        model4+=[nn.ReLU(True),]
        # model4+=[nn.ReflectionPad2d(1),]
```

```
        model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model4+=[norm_layer(512),]
        model4+=[nn.ReLU(True),]
        # model4+=[nn.ReflectionPad2d(1),]
        model4+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model4+=[nn.ReLU(True),]
        model4+=[norm_layer(512),]

        # Conv5
        # model47+=[nn.ReflectionPad2d(2),]
        model5=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        # model5+=[norm_layer(512),]
        model5+=[nn.ReLU(True),]
        # model5+=[nn.ReflectionPad2d(2),]
        model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        # model5+=[norm_layer(512),]
        model5+=[nn.ReLU(True),]
        # model5+=[nn.ReflectionPad2d(2),]
        model5+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        model5+=[nn.ReLU(True),]
        model5+=[norm_layer(512),]

        # Conv6
        # model6+=[nn.ReflectionPad2d(2),]
        model6=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        # model6+=[norm_layer(512),]
        model6+=[nn.ReLU(True),]
        # model6+=[nn.ReflectionPad2d(2),]
        model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        # model6+=[norm_layer(512),]
        model6+=[nn.ReLU(True),]
        # model6+=[nn.ReflectionPad2d(2),]
        model6+=[nn.Conv2d(512, 512, kernel_size=3, dilation=2, stride=1,
padding=2, bias=use_bias),]
        model6+=[nn.ReLU(True),]
        model6+=[norm_layer(512),]

        # Conv7
        # model47+=[nn.ReflectionPad2d(1),]
        model7=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model7+=[norm_layer(512),]
        model7+=[nn.ReLU(True),]
        # model7+=[nn.ReflectionPad2d(1),]
        model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model7+=[norm_layer(512),]
        model7+=[nn.ReLU(True),]
        # model7+=[nn.ReflectionPad2d(1),]
        model7+=[nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model7+=[nn.ReLU(True),]
```

```
        model7+=[norm_layer(512),]

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

        # model3short8=[nn.ReflectionPad2d(1),]
        model3short8=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]

        # model47+=[norm_layer(256),]
        model8=[nn.ReLU(True),]
        # model8+=[nn.ReflectionPad2d(1),]
        model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # model8+=[norm_layer(256),]
        model8+=[nn.ReLU(True),]
        # model8+=[nn.ReflectionPad2d(1),]
        model8+=[nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model8+=[nn.ReLU(True),]
        model8+=[norm_layer(256),]

        # Conv9
        model9up=[nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2,
padding=1, bias=use_bias),]

        # model2short9=[nn.ReflectionPad2d(1),]
        model2short9=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # add the two feature maps above

        # model9=[norm_layer(128),]
        model9=[nn.ReLU(True),]
        # model9+=[nn.ReflectionPad2d(1),]
        model9+=[nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        model9+=[nn.ReLU(True),]
        model9+=[norm_layer(128),]

        # Conv10
        model10up=[nn.ConvTranspose2d(128, 128, kernel_size=4, stride=2,
padding=1, bias=use_bias),]

        # model1short10=[nn.ReflectionPad2d(1),]
        model1short10=[nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1,
bias=use_bias),]
        # add the two feature maps above

        # model10=[norm_layer(128),]
        model10=[nn.ReLU(True),]
        # model10+=[nn.ReflectionPad2d(1),]
        model10+=[nn.Conv2d(128, 128, kernel_size=3, dilation=1, stride=1,
padding=1, bias=use_bias),]
        model10+=[nn.LeakyReLU(negative_slope=.2),]

        # classification output
        model_class=[nn.Conv2d(256, 529, kernel_size=1, padding=0, dilation=1,
stride=1, bias=use_bias),]
```

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

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

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

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

    def forward(self, input_A, input_B, mask_B):
        conv1_2 = self.model1(torch.cat((input_A,input_B,mask_B),dim=1))
        conv2_2 = self.model2(conv1_2[:,:,::2,::2])
        conv3_3 = self.model3(conv2_2[:,:,::2,::2])
        conv4_3 = self.model4(conv3_3[:,:,::2,::2])
        conv5_3 = self.model5(conv4_3)
        conv6_3 = self.model6(conv5_3)
        conv7_3 = self.model7(conv6_3)
        conv8_up = self.model8up(conv7_3) + self.model3short8(conv3_3)
        conv8_3 = self.model8(conv8_up)

        if(self.classification):
            out_class = self.model_class(conv8_3)

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

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

```
            out_reg = self.model_out(conv10_2)

        return (out_class, out_reg)

# Defines the Unet generator.
# |num_downs|: number of downsamplings in UNet. For example,
# if |num_downs| == 7, image of size 128x128 will become of size 1x1
# at the bottleneck
class UnetGenerator(nn.Module):
    def __init__(self, input_nc, output_nc, num_downs, ngf=64,
                 norm_layer=nn.BatchNorm2d, use_dropout=False):
        super(UnetGenerator, self).__init__()

        # construct unet structure
        unet_block = UnetSkipConnectionBlock(ngf * 8, ngf * 8, input_nc=None,
submodule=None, norm_layer=norm_layer, innermost=True)
        for i in range(num_downs - 5):
            unet_block = UnetSkipConnectionBlock(ngf * 8, ngf * 8,
input_nc=None, submodule=unet_block, norm_layer=norm_layer,
use_dropout=use_dropout)
        unet_block = UnetSkipConnectionBlock(ngf * 4, ngf * 8, input_nc=None,
submodule=unet_block, norm_layer=norm_layer)
        unet_block = UnetSkipConnectionBlock(ngf * 2, ngf * 4, input_nc=None,
submodule=unet_block, norm_layer=norm_layer)
        unet_block = UnetSkipConnectionBlock(ngf, ngf * 2, input_nc=None,
submodule=unet_block, norm_layer=norm_layer)
        unet_block = UnetSkipConnectionBlock(output_nc, ngf,
input_nc=input_nc, submodule=unet_block, outermost=True,
norm_layer=norm_layer)
        self.model = unet_block

    def forward(self, input_A, input_B, mask_B):
        # embed()
        return self.model(torch.cat((input_A,input_B,mask_B),dim=1))

# Defines the PatchGAN discriminator with the specified arguments.
class NLayerDiscriminator(nn.Module):
    def __init__(self, input_nc, ndf=64, n_layers=3,
norm_layer=nn.BatchNorm2d, use_sigmoid=False):
        super(NLayerDiscriminator, self).__init__()
        if type(norm_layer) == functools.partial:
            use_bias = norm_layer.func == nn.InstanceNorm2d
        else:
            use_bias = norm_layer == nn.InstanceNorm2d
        kw = 4
        padw = 1
        sequence = [
            nn.Conv2d(input_nc, ndf, kernel_size=kw, stride=2, padding=padw),
            nn.LeakyReLU(0.2, True)
        ]
        nf_mult = 1
        nf_mult_prev = 1
        for n in range(1, n_layers):
            nf_mult_prev = nf_mult
            nf_mult = min(2**n, 8)
            sequence += [
                nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
                          kernel_size=kw, stride=2, padding=padw,
bias=use_bias),
```

```
                norm_layer(ndf * nf_mult),
                nn.LeakyReLU(0.2, True)
            ]
        nf_mult_prev = nf_mult
        nf_mult = min(2**n_layers, 8)
        sequence += [
            nn.Conv2d(ndf * nf_mult_prev, ndf * nf_mult,
                      kernel_size=kw, stride=1, padding=padw, bias=use_bias),
            norm_layer(ndf * nf_mult),
            nn.LeakyReLU(0.2, True)
        ]
        sequence += [nn.Conv2d(ndf * nf_mult, 1, kernel_size=kw, stride=1,
padding=padw)]
        if use_sigmoid:
            sequence += [nn.Sigmoid()]
        self.model = nn.Sequential(*sequence)

    def forward(self, input):
        return self.model(input)


class PixelDiscriminator(nn.Module):
    def __init__(self, input_nc, ndf=64, norm_layer=nn.BatchNorm2d,
use_sigmoid=False):
        super(PixelDiscriminator, self).__init__()
        if type(norm_layer) == functools.partial:
            use_bias = norm_layer.func == nn.InstanceNorm2d
        else:
            use_bias = norm_layer == nn.InstanceNorm2d

        self.net = [
            nn.Conv2d(input_nc, ndf, kernel_size=1, stride=1, padding=0),
            nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf, ndf * 2, kernel_size=1, stride=1, padding=0,
bias=use_bias),
            norm_layer(ndf * 2),
            nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf * 2, 1, kernel_size=1, stride=1, padding=0,
bias=use_bias)]
        if use_sigmoid:
            self.net.append(nn.Sigmoid())
        self.net = nn.Sequential(*self.net)

    def forward(self, input):
        return self.net(input)
```

Лістинг 2. Модуль обробки вхідних даних та повернення результатів

```
import os
from art.colorize_ai.options.train_options import TrainOptions
from art.colorize_ai.models import create_model
from art.colorize_ai.util.visualizer import save_images
from art.colorize_ai.util import html
import string
import torch
import torchvision
import torchvision.transforms as transforms
import cv2
from PIL import Image
from art.colorize_ai.util import util
# from IPython import embed
import numpy as np

class ColorizeModel:

    def __init__(self):
        self.opt = TrainOptions()
        self.opt.load_model = True
        self.opt.nThreads = 1  # test code only supports nThreads = 1
        self.opt.batch_size = 1  # test code only supports batch_size = 1
        self.opt.display_id = -1  # no visdom display
        self.opt.phase = 'val'
        self.opt.dataroot = './dataset'
        self.opt.serial_batches = True
        self.opt.aspect_ratio = 1.

        self.model = create_model(self.opt)
        self.model.setup(self.opt)
        self.model.eval()
        self.transform =
transforms.Compose([transforms.Resize((self.opt.loadSize, self.opt.loadSize)),
transforms.ToTensor()])

        # create website
        self.web_dir = os.path.join(self.opt.results_dir, self.opt.name,
'%s_%s' % (self.opt.phase, self.opt.which_epoch))
        self.webpage = html.HTML(self.web_dir, 'Experiment = %s, Phase = %s,
Epoch = %s' % (self.opt.name, self.opt.phase, self.opt.which_epoch))
        self.to_visualize = ['gray', 'hint', 'hint_ab', 'fake_entr', 'real',
'fake_reg', 'real_ab', 'fake_ab_reg', ]


    def process_image(self, source_image_path, hints_image_path):
        source_image =
self.transform(Image.open(source_image_path)).unsqueeze(0)
        hints_map = cv2.imread(hints_image_path, cv2.IMREAD_UNCHANGED)[:, :,
3]
        hints_image =
self.transform(Image.open(hints_image_path)).unsqueeze(0)

        source_image = source_image.cuda()
        hints_image = hints_image.cuda()

        # source_image = util.crop_mult(source_image, mult=8)
```

```
        # with no points
        data = util.get_colorization_data(source_image, hints_image, self.opt)

        self.model.set_input(data)
        self.model.test(True)  # True means that losses will be computed
        visuals = util.get_subset_dict(self.model.get_current_visuals(),
self.to_visualize)

        save_images(self.webpage, visuals, str(0),
aspect_ratio=self.opt.aspect_ratio, width=self.opt.display_winsize)
```

Лістинг 3. Модуль взаємодії системи з користувачем

```
from django.http import HttpResponse, HttpResponseNotFound, JsonResponse
from django.shortcuts import render
import base64
import re
import base64
from django.views.decorators.csrf import csrf_exempt
import re
from PIL import Image
from art.colorize_ai.test_model import ColorizeModel
# from art.test_one_pic import ByModel
bm = ColorizeModel()

def main(request):
    return render(request, 'main_colorize.html')


@csrf_exempt
def render_post(request):
    if request.is_ajax():
        if request.method == 'POST':
            dataUrlPattern = re.compile('(.*)$')
            img = request.body.decode('utf-8')
            parsed_name = img.split(' ')[0]
            img = img.split(' ')[1]
            imgb64 = dataUrlPattern.match(img).group(1)
            with open('art/save_im/output.png', 'wb') as output:
                output.write(base64.b64decode(imgb64))
            print('Im here')
            bm.process_image('art/test/%s' %
(parsed_name),'art/save_im/output.png')
            # bm.run()
            im = base64.b64encode(open("art/static/0.png", "rb").read())
        return HttpResponse(im)
    else:
        return render(request, 'main_colorize.html', {'name': "0.jpg"})
```

Додаток 2

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Алгоритмічно-програмний метод колоризації зображень

Виконав: Топчієв Борис Сергійович

Науковий керівник: к.т.н, доцент, Сулема Євгенія Станіславівна

Київ – 2020

Актуальність теми

Редагування та обробка зображень є актуальним напрямком через високу популярність візуального контенту на великих платформах (Instagram, Facebook, Flickr, DevianArt) та також у сферах дизайну та кіноіндустрії.

Для даної магістерської дисертації було вирішено розробити технологію обробки зображень для розв’язку задачі інтелектуальної колоризації, так як дана задача є актуальною, а існуючі методи можна значно покращити.

Мета роботи

Розроблення методу колоризації зображень на основі згорткових та генеративних змагальних нейронних мереж для оброблення зображень у напівавтоматичному та автоматичному режимах.

Об’єкт та предмет дослідження

Об’єктом дослідження є процес оброблення зображень за допомогою математичних методів та алгоритмів штучного інтелекту.

Предметом дослідження є методи та алгоритми колоризації зображень автоматичним та напівавтоматичним (з внесенням додаткової інформації користувачем) шляхом.

Етапи виконання

1. Провести детальний аналіз існуючих методів колоризації зображень.
2. Дослідити існуючі архітектури нейронних мереж.
3. Обрати необхідні дані та розробити методи їх попередньої обробки.
4. Розробити програму для навчання моделей, налаштувати гіпер параметри моделі та умови навчання.
5. Провести тестування моделей та аналіз результатів.
6. Розробити користувацький інтерфейс для експлуатації технології користувачем.

Визначення

- Колоризація - програмний процес зміни кольорів на зображенні.
- Згорткова нейронна мережа - нейронна мережа, яка складається з блоків згортки та об’єднання і виконує ряд задач, пов’язаних з обробкою неструктурованих даних (аудіо, зображення, відео).
- Генеративна змагальна мережа - система нейронних мереж, які використовуються для обробки та генерації даних.

Colorization using optimisation

У даному алгоритмі дані подаються у кольоровій схемі YUV, та основною гіпотезою є те, що пікселі одного об’єкту на зображенні мають наближене один до одного значення каналу Y (monochromatic luminance value).

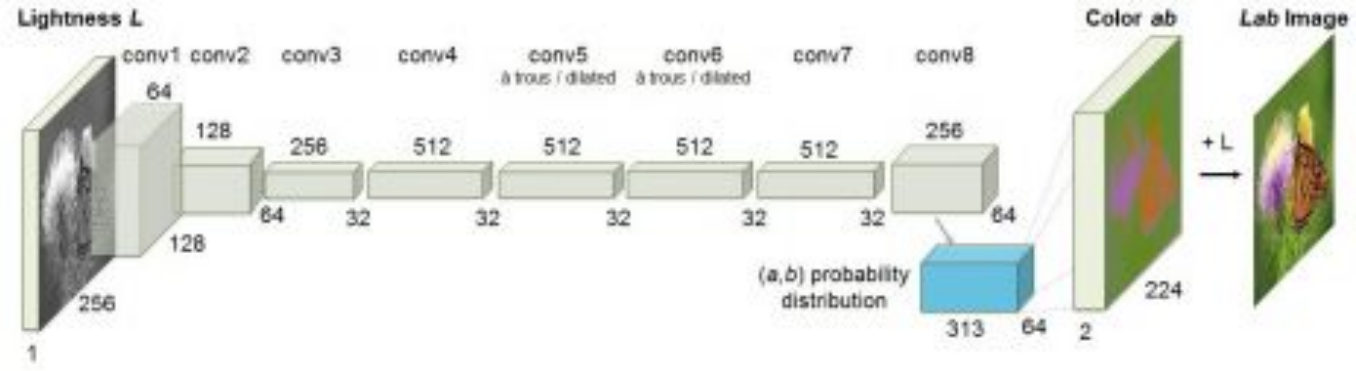
Алгоритм ітераційно аналізує сусідні пікселі на основі значення каналу світлоти і задає цим пікселям значення того кольору, який вказав користувач.

Marked B/W image

Result

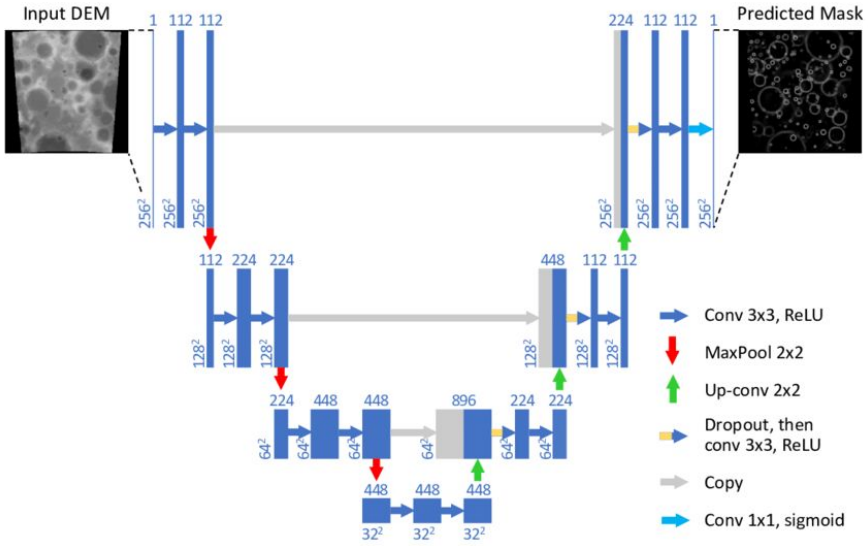
Colorful Image Colorization

Даний алгоритм використовує згорткову нейронні мережу U-Net для перетворення чорно-білих зображень у кольорові. Для цього зображення подається у мережу у кольоровій схемі LAB, використовується навчання з учителем. Архітектура мережі має наступний вигляд:



Архітектура U-Net

- Глибока нейронна мережа для сегментації зображень
- Складається з послідовних блоків згортки та об’єднання (pooling) та блоків розгортки та підвищення розмірності
- Використовує техніку shortcut connection для передачі найголовніших особливостей зображення між шарами



Генеративні змагальні мережі

- Система складається з двох мереж - мережі-генератора та мережі-дискримінатора
- Дискримінатор намагається виявити підробку, створену генератором
- Генератор намагається “обдурити” дискримінатор, генеруючи якісні підробки

Загалом, задачу генеративних мереж можна виразити формулою:

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

Запропонований метод

1. Користувач, за допомогою веб-інтерфейсу створює мапу кольорів яка слугує підказками для мережі.
2. Початкове зображення, та зображення з підказками переводять з RGB до LAB.
3. На вхід до мереж подається L-канал оригінального зображення, та A,B канали з зображення з користувацькими підказками.

Запропонований метод (продовження)

4. Мережа локальних підказок на кожному шарі передає інформацію до мережі-генератора, який виконує обробку.
5. Мережа-генератор повертає A та B канали обробленого зображення.
6. Канали A та B поєднуються з L-каналом та переводяться до кольорової схеми RGB та повертаються користувачу.

Умови навчання

- Для навчання мережі було обрано датасет IMAGENET-2012 з понад 1 мільйоном кольорових зображень
- У якості методі попередньої обробки вирішено використовувати кольорову схему LAB для спрощення та прискорення задачі сегментації зображення
- За основу архітектури генератора було узято мережу U-Net
- За основу архітектури дискримінатора було узято мережу PatchGan

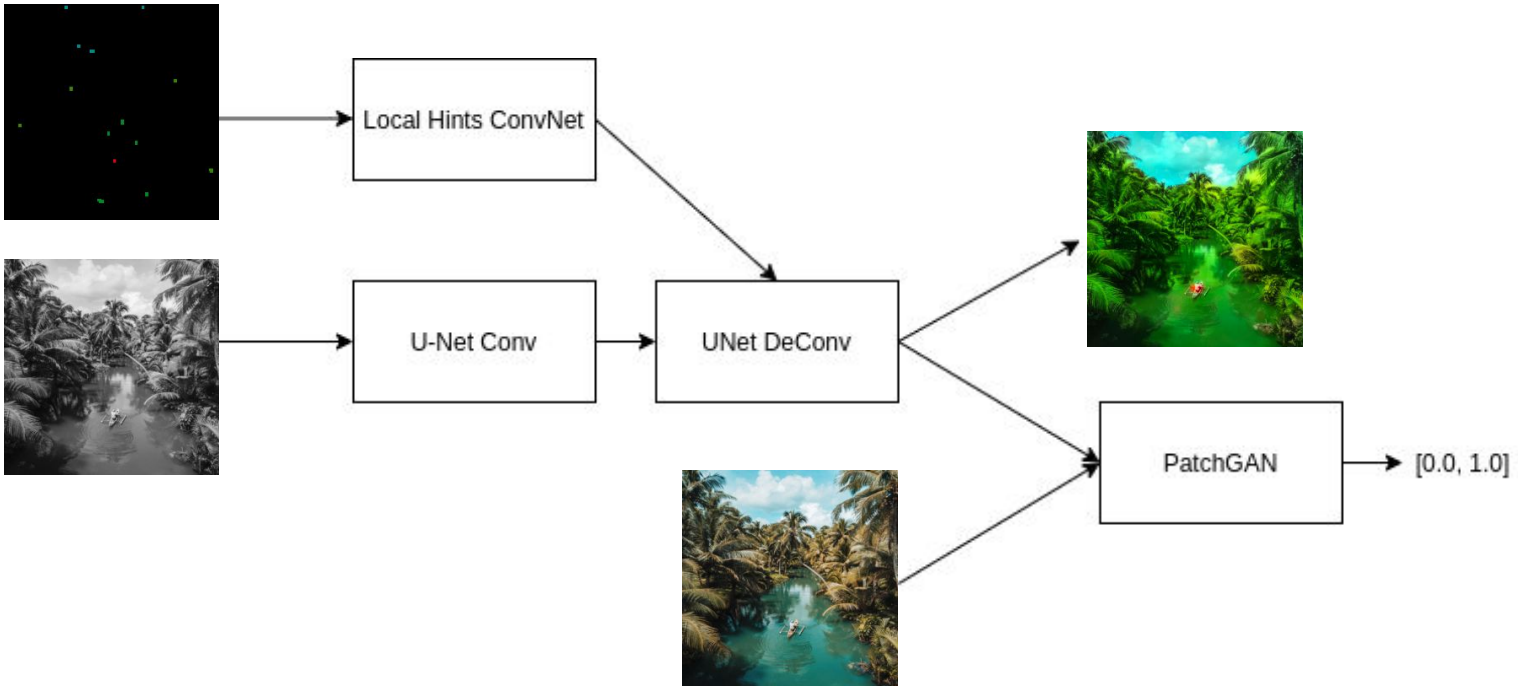
Налаштування параметрів мережі

- batch size = 2
- input size = 256 x 256
- learning rate = 0.001
- epochs = 25
- optimization method - Adam

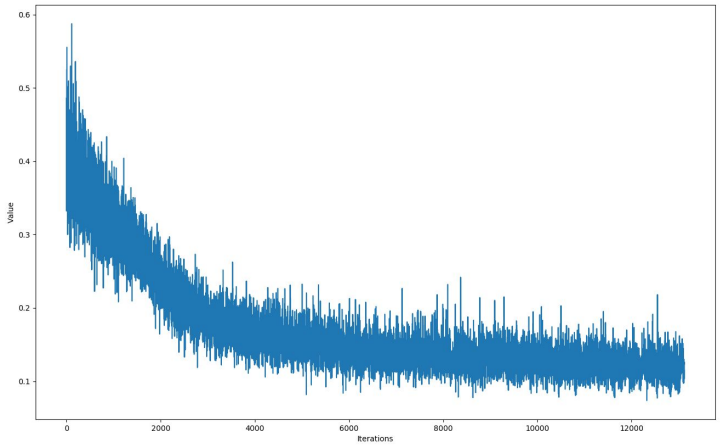
Local Hints Network

Для можливості колоризації зображень користувачем до основної мережі-генератора було додано допоміжну мережу підказок, яка враховує штрихи, введені користувачем та за допомогою операцій згорток виділяє особливості користувацьких підказок і подає їх до основної мережі у вигляді розподілу кольорів.

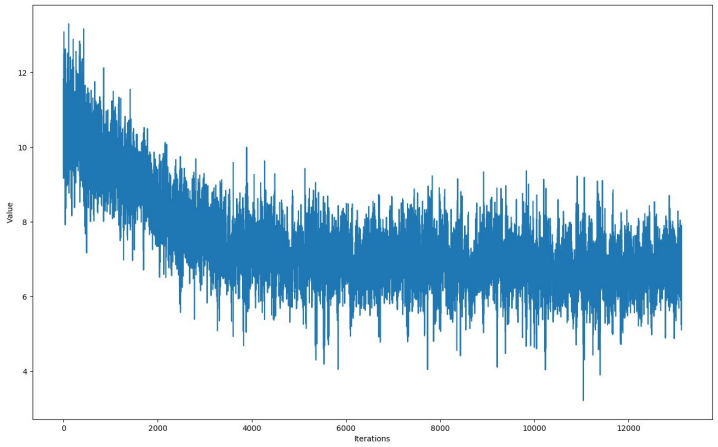
Загальний вигляд системи мереж



Графіки процесу навчання

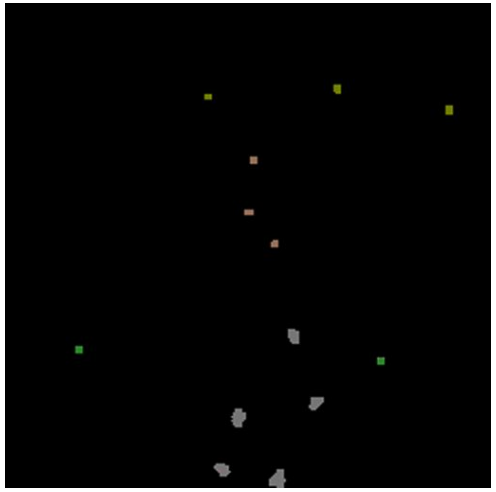


Лосс-функція мережі-генератора



Лосс-функція мережі-дискримінатора

Приклад роботи системи



Порівняння з існуючими методами

Назва методу	Автоматична колоризація	Можливість контролю з боку користувача	Швидкість обробки одного зображення
Colorization using optimisation	-	+	180-240 сек
Colorful Image Colorization	+	-	1.1-1.5 сек
Запропонований метод	+	+	1-1.2 сек

Шляхи подальшого дослідження

- додати до тренувальної вибірки нові дані;
- виконати тренування з більшою кількістю епох;
- додати додатковий вхід до архітектури нейронних мереж для врахування початкового розподілу кольорів з метою покращення роботи колоризації з користувацькими підказками;
- реалізація підтримки алгоритмом намальованих ліній різного розміру;
- реалізація підтримки колоризації відео.

Наукова новизна

Запропоновано метод колоризації зображень, який, на відміну від існуючих методів, дозволяє виконувати автоматичну та напівавтоматичну колоризацію та прискорити обчислення на 10-15%.

ВИСНОВКИ

1. Детально досліджено зображення та їх дефекти, методи усунення різного роду дефекти та способи обробки зображень
2. Проаналізовано існуючі методи колоризації зображень
3. Проаналізовано алгоритми штучного інтелекту для обробки зображень
4. Запропоновано власний метод інтелектуальної колоризації, який дозволяє користувачу вносити власні коригування
5. Розроблено та натреновано відповідні моделі на великій кількості даних
6. Проведено тестування та аналіз результатів моделей
7. Запропоновано шляхи подальшого дослідження

АПРОБУВАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

1. “ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ ЗА ДОПОМОГОЮ ГЕНЕРАТИВНИХ ЗМАГАЛЬНИХ МЕРЕЖ” на XII науковій конференції магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019)

Дякую за увагу!

Демонстрація роботи системи