МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

«Система покращення передачі кольору та освітлення зображень з використанням технологій штучного інтелекту»

Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп'ютерні науки»**
Освітня програма **«Комп'ютерні науки»**
Освітній рівень: бакалавр

Виконав: студент 4 курсу групи КН- 41
Козак Денис Юрійович

Керівник: Кіктєв Микола Олександрович
Кандидат технічних наук, доцент

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол №11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ – 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.
_____

"___"_____2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Козака Дениса Юрійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи): Система покращення передачі кольору та освітлення зображень з використання технологій штучного інтелекту

затверджена протоколом засідання кафедри від « 23 » грудня 2021 року. № 4

2. Термін здачі студентом закінченого проекту (роботи) «29» травня 2022 року
3. Вихідні дані до проекту (роботи): Програмний модуль покращення передачі кольору та освітлення
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

   Аналіз предметної області, вибір архітектури, розробка нейронної мережі, розробка клієнтської та серверної частини програмного забезпечення.

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

   Презентація PowerPoint, яка містить короткі відомості про задачу, опис нейронної мережі, архітектуру системи, програмну реалізацію, демонстрацію роботи програми.

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Кіктєв М.О.	01.03.2022	01.03.2022
2	Кіктєв М.О.	05.04.2022	05.04.2022
3	Кіктєв М.О.	10.05.2022	10.05.2022

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ / Кіктєв М. О. /
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / Козак Д. Ю. /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Обговорення постановки завдання та змісту пояснювальної записки	25.01.2022 -22.02.2021	
2	Вибір та формування теми	23.02.2021 - 26.02.2021	
3	Аналіз предметної області	27.02.22 –10.03.22	
4	Вибір методів рішення задачі	11.03.22 – 20.04.22	
5	Створення програмного модулю	21.04.2022 - 10.05.2021	
6	Оформлення пояснювальної записки	11.05.2022 – 02.06.2022	

Студент-дипломник _____ / Козак Д. Ю. /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____/ Кіктєв М.О. /
(підпис) (ПІБ)

АНОТАЦІЯ

Козак Денис Юрійович виконав випускну кваліфікаційну роботу на тему «Система покращення передачі кольору та освітлення зображень з використанням технологій штучного інтелекту».

Дана випускна кваліфікаційна робота фокусується на дослідженні сучасних технологій в сфері обробки зображень з використанням технологій штучного інтелекту, аналізу існуючих рішень, та розробці системи, яка проводить покращення передачі кольору та освітлення зображень.

Ключові слова: штучний інтелект, нейронна мережа, покращення зображень, освітлення зображень.

ANNOTATION

The degree project «Artificial intelligence color reproduction and image lighting amelioration system» has been completed by Denys Kozak.

This paper focuses on current artificial intelligence image processing technologies research, analyzing existing solutions, color reproduction and image lightning amelioration system.

Key words: artificial intelligence, neural network, image amelioration, image lightning.

Зміст

Вступ

Останнім часом технології штучного інтелекту розвиваються швидкими темпами. Потреба в штучному інтелекті росте, оскільки з’являється більше можливих областей застосування через те, що продуктивність обчислювальних машин зростає та покращуються існуючі алгоритми. Розвиток технологій штучного інтелекту дає можливість заміняти рутинні процеси, які зазвичай виконувались людьми, на автоматизовані.

Одним з таких рутинних процесів є обробка зображень після зйомки професійними фотографами, на яку витрачається багато часу. Не дивлячись на те, що сучасні камери виконують досить непогану роботу з підбору правильних параметрів, для отримання найкращої якості знімку в будь-якому разі необхідно проводити додаткове налаштування параметрів освітлюваності.

Застосування нейронних мереж дозволить значно пришвидшити обробку зображень, а також значно спростити процес обробки фотографій для звичайних користувачів. Існуючі професійні рішення в сфері обробки фотографій зазвичай не використовують нейронні мережі,

Основна мета роботи полягає в розробці системи, яка проводить покращення передачі кольору та освітлення зображень за допомогою технологій штучного інтелекту.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналітичний огляд нейронних мереж

1.1.1 Історія розвитку нейронних мереж

В 1943 році, нейропсихолог Уоррен Маккалох та математик Уолтер Піттс випустили роботу, в якій описувалось, яким чином можуть працювати нейрони, та створили модель нейронної мережі за допомогою електронних схем.

В 1949 році, Дональд Хебб закінчив роботу над статтею «Організація поведінки» (англ. – «The Organization of Behavior»), в якій розповідалось про нейронні зв’язки, та способи, завдяки яким людина може навчатись. Згідно з роботою Хебба, якщо активувались 2 нейрони одночасно, то зв’язок між ними посилювався.

З часом, коли комп’ютерні технології розвивались, стало можливим запустити симуляцію нейронної мережі. В 1959 році, Бернард Уідроу та Маршіан Хофф розробили моделі, які отримали назву ADALINE та MADALINE, які є акронімами від «Multiple Adaptive Linear Elements» (в перекладі з англійської мови – «Множинні адаптивні лінійні елементи»). ADALINE розпізнавав двійкові патерни – якщо на вхід подавався потік бітів з телефонної лінії, система могла передбачити наступний біт. MADALINE був першою нейронною мережею, яку можна використовувати для вирішення реальних практичних задач, наприклад прибирання ехо з телефонних ліній.

Через 3 роки після цього, дослідники створили процедуру навчання за такою формулою:

$$W = P \cdot \frac{E}{N}, \qquad (1.1)$$

Де W – значення, на яке потрібно змінити ваги, P – значення попереднього рядка вагів, E – коефіцієнт помилки, N – кількість входів.

Дана формула заснована на ідеї, що коли один активний нейрон має більше значення помилки, він може зкорегувати значення інших ваг в мережі. Але навіть після застосування цієї формули може спровокувати помилку, якщо рядок перед вагами має значення 0, хоча зрештою він виправиться.

Перша мультишарова мережа була розроблена в 1975.

В 1982 році, з розвитком комп'ютерних технологій, інтерес до області штучного інтелекту виріс. Джон Хопфілд розробив систему, в якій використовувались двосторонні зв'язки між нейронами. Згодом того ж року, Рейлі та Купер використали гібридну мережу з декількома шарами, кожен з яких використовував різний підхід до вирішення задач.

В 1986 році 3 незалежні групи дослідників розробили системи з використанням зворотного поширення помилки. Гібридні мережі використовували 2 шари, тоді як системи з методом зворотного поширення помилки могли задіяти багато шарів, із-за чого вони навчались значно довше.

В теперішній час, розробка нових нейронних мереж упирається в ліміт обчислюваної продуктивності комп'ютерної техніки, тому що для навчання мереж може піти декілька місяців навіть на найновішому апаратному забезпеченні.

1.1.2 Штучний нейрон

В технологіях штучного інтелекту основна ідея – симуляція біологічного мозку. На даний час комп'ютерні технології не настільки розвинені, щоб можна було в повній мірі створити модель біологічного нейрону, але використання спрощеної моделі загалом достатньо для виконання поставлених задач.

Основні частини нейрону:

- Дендрити – деревоподібні гілки, які одержують дані від інших нейронів.

- Аксон – зв’язок, за допомогою якого нейрони посилають інформацію.
- Ядро – тіло клітини нейрона, обчислює дані, отримані від дендритів.
- Синапси – з’єднання між аксонами і дендритами інших нейронів.

Штучний нейрон отримує дані від інших нейронів, або з зовнішнього джерела. Кожен вхід має своє значення вагів w.

1.1.3 Типи навчання нейронних мереж

Типи навчання нейронних мереж впливають на те, як швидко та як вдало система зможе навитись. В залежності від розробленої мережі, можна виділити такі типи навчання:

- з учителем – нейронна мережа використовує учителя, який говорить їй, як правильно вчинити. Вчитель відмічає всі необхідні дані, для того щоб мережа вчилась на конкретних прикладах. З учителем навчання проходить швидше та якісніше. З використанням даного типу навчання стає можливим вирішити задачі регресії та класифікації;
- без учителя – в даному типі навчання мережа сама розбирає, до яких категорій належать елементи.;
- з підкріпленням – схоже на справжнє біологічне навчання – нейронну мережу штрафують за помилки, але нагороджують за правильні дії.

1.1.4 Архітектура нейромереж

Існує багато видів нейронних мереж, з яких можна виділити в такі категорії:

- нейронні мережі прямого поширення;
- згорткові мережі;
- автоенкодер;
- рекурентні мережі;
- мережі радіальних базисних функцій;
- модулярні нейронні мережі.

Нейронні мережі прямого поширення – тип нейронної мережі, чиї вузли не створюють петлю. Ці мережі називаються так тому, що інформація може йти лише прямо. Дані заходять у вхідні вузли, переходячи через приховані шари, і виходячи через вихідні вузли. Основна мета застосування нейронних мереж прямого поширення – апроксимація функцій.

Згорткові мережі – клас глибинних мереж прямого поширення, які приймають на вхід зображення, призначають важливість різним аспектам та об’єктам. Необхідність передпідготовки в згорткових мережах є набагато меншою, ніж в інших алгоритмах класифікації.

Автоенкодер – алгоритм глибинного навчання, який включає в себе три частини: енкодер – для стиснення вхідних даних, вузьке місце (боттлнек) – модуль який містить стиснену репрезентацію даних, та декодер – модуль, який допомагає мережі розпаковувати зображення. Після цього вихідні дані мережі порівнюються з перевірчим зображенням

Рекурентні мережі – тип нейронних мереж, що застосовує послідовні дані та дані часових рядів. Такі алгоритми зазвичай використовують у NLP – Natural Language Processing (в перекладі з англійської мови – «обробка природної мови»), у розпізнаванні мови.

Так, як і в прямих й згорткових нейронних мережах, для навчання використовуються дані для тренування. Але вони відрізняються пам’яттю, так як беруть дані з попередніх входів для впливу на нинішній вхід та вихід.

Зазвичай, в глибоких нейронних мережах виходи та входи не залежать один від одного, а попередні елементи рекурентних нейронних мереж

впливають на вихід. Не дивлячись на те, що майбутні події також представляють цінність при визначенні результату цієї послідовності, даний тип нейронних мереж не враховує ці дані.

Типи рекурентних мереж: один до одного, один до багатьох, багато до одного, багато до багатьох.

Функція активації – грає ключову роль в визначенні вихідного сигналу вузла, за рахунок неї вирішується, буде нейрон на виході видавати 0 або 1.

Поширені функції активації:

Сигмоїдна:

$$g(x) = \frac{1}{1+e^{-x}} \tag{1.2}$$

Гіперболічна дотична функція:

$$g(x) = \frac{e^{-x}-e^{-x}}{e^{-x}+e^{-x}} \tag{1.3}$$

Функція Relu:

$$g(x) = \max(0, x) \tag{1.4}$$

Варіанти рекурентних архітектур:

Двонаправлені рекурентні нейронні мережі – оскільки односпрямовані мережі можуть брати дані лише з попередніх входів, двонаправлені беруть майбутні дані, для того, щоб підвищити точність передбачень.

LSTM (від англ. – Long short-term memory) – популярна архітектура яку створили як рішення проблеми зникаючого градієнта. Якщо попередній стан, який впливає на поточний прогноз, не в недалекому минулому, рекурентна мережа скоріше за все не зможе точно передбачити поточний стан.

GRU (від англ. – Gated recurrent units) – цей варіант рекурентної мережі подібний до LSTM, він також є вирішенням проблеми короткочасної пам’яті рекурентної мережі. Замість використання інформації, яка регулює стан, вона використовує приховані стани, і має два шлюзи – скидання та оновлення.

Мережі радіальних базисних функцій – вони використовують архітектуру, яка сильно відрізняється від більшості інших нейронних архітектур. Більшість нейронних мереж включає в себе багато шарів і представляють нелінійність, багато разів застосовуючи нелінійні функції активації. В той час мережі радіальних базисних функцій складаються лише з вхідного шару, одного прихованого шару, и вихідного шару.

Вхідний шар не виконує обрахунків, а лише отримує дані і передає їх в спеціальний прихований шар мережі.

Вихідний шар використовує лінійну активаційну функцію для задач класифікації та регресії. На цьому шарі обчислення проходять так само, як і в звичайних нейронних мережах. Обчислення можна характеризувати такою формулою:

$$y = \sum_i^n w_i \phi_i, \qquad (1.5)$$

Де w_i – зв’язок вагів, ϕ_i – і-тий вихід нейрона з прихованого шару, y – очікуваний результат.

Модулярні нейронні мережі – складаються з декількох нейронних мереж, які зв’язані з собою за допомогою проміжного зв’язку. Модульні нейронні мережі дають змогу керувати більш складними нейронними мережами.

У даному випадку, численні нейронні мережі діють як модулі, кожен з яких вирішує частину поставленої задачі. Інтегратор відповідає за поділ проблеми на кілька модулів, а також за інтеграцію відповідей для надання результатів роботи мережі.

1.2 Аналітичний огляд технологій передачі кольору

Колір на зображеннях складається з трьох основних частин – Відтінок, освітлюваність та насичення.

Відтінок (від англ. Hue) – домінуюча довжина хвилі, саме ця характеристика визначає назву кольору (червоний, синій і т.д.).

Освітлюваність (від англ. Value) – визначає наскільки колір буде світлим чи темним. Показує кількість відбитого світла.

Насичення (від англ. Saturation) – характеризує інтенсивність кольору. Відображає наскільки яскравий відтінок буде використано.

Графічне відображення значення наведених вище характеристик зображено на рисунку 1.1.

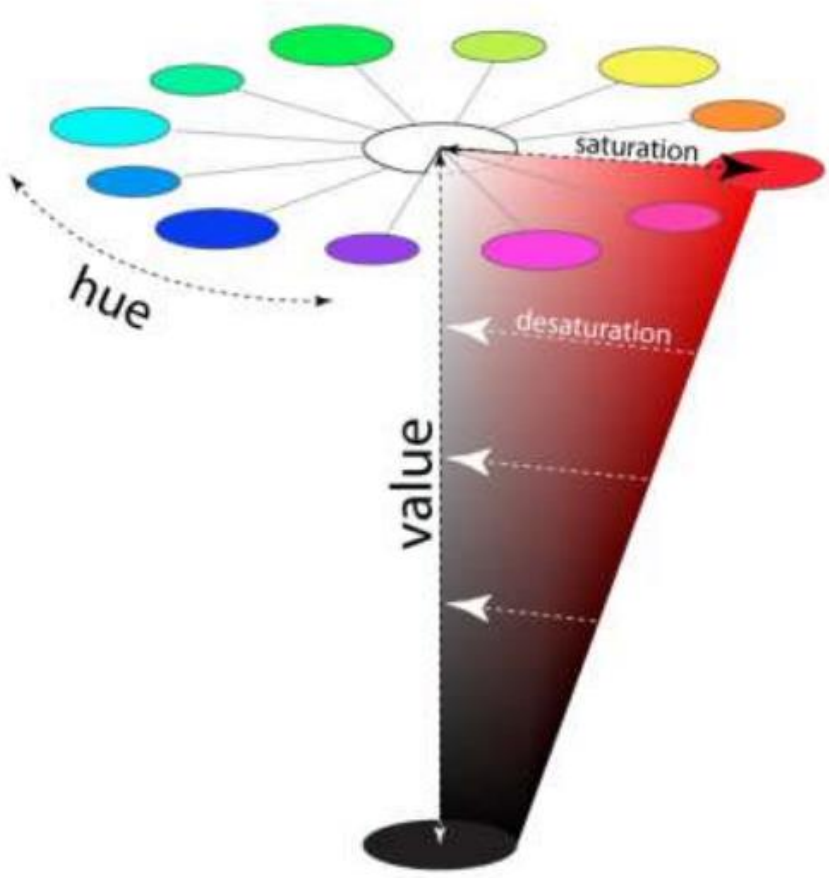

Рисунок 1.1 Графічне відображення відтінку, освітлюваності та насичення

Найбільш важливою характеристикою, яка впливає на сприйняття людиною зображень, є баланс білого кольору. Він показує, з якою кольоровою температурою має відображатись фотографія. Кольорова температура, в свою чергу, відображає, який повинен бути відтінок у джерела світла.

Баланс білого перевіряє, чи всі об’єкти на зображенні відображаються потрібним відтінком кольору. Дана характеристика є першою, яка застосовується необробленого (RAW) зображення

1.3 Область застосування програмного рішення, яке розробляється

Налаштування параметрів освітлюваності вручну займає багато часу та потребує відповідних навичок в роботі з професіональними додатками для редагування фотографій.

Програмне рішення, яке буде розроблено в даній роботі, може бути використане для спрощення, прискорення обробки зображень, як професійними фотографами, так і звичайними користувачами, які хочуть отримати більш якісне, красиве та реалістичне зображення.

1.4 Постановка задачі

Об’єктом дослідження даної дипломної роботи є засоби покращення передачі кольору та освітлюваності зображень.

Предметом дослідження є технології нейронних мереж, які дозволяють виконувати покращення параметрів зображення.

Система, яка буде розроблена, надає користувачу можливість вибирати та завантажувати на сервер одне, або багато зображень, отримуючи у відповідь декілька варіантів з різним рівнем освітлюваності, з яких можна вибрати найкраще, після чого завантажити архів з готовими фотографіями.

Актуальність роботи пояснюється тим, що технології покращення освітлюваності зображень є достатньо новими, і наразі не існує програмного рішення, яке б дозволило вирішувати описані вище задачі, при цьому виконуючи відповідний рівень коректування зображень, та зберігаючи швидкодію та зручність роботи програмного застосунку, які очікує користувач.

Рисунок 1.2 Зображення різних варіантів налаштувань балансу білого кольору

Варіанти балансу білого кольору зображені на рисунку 1.2. Опис варіантів:

- Auto White Balance – автоматична конфігурація, яка підлаштовується під задані параметри освітлюваності, тому сцена виглядить майже так, як в реальному житті.
- Shade White Balance –використовується для вуличних знімків, коли немає яскравого джерела світла. Температура кольору – 7000К.
- Cloudy White Balance – використовується для вуличних знімків під час похмурої погоди. Температура – 6000К.
- Daylight White Balance – використовується для вуличних змімків під час яскравої погоди. Температура – 4200К.

- Fluorescent White Balance – використовується для зйомки в приміщенні з флуоресцентними лампами. Температура – 4000К.
- Tungsten White Balance – використовується для зйомки в приміщені з лампами розжарювання. Температура – 3200К.

1.4.1 Опис зацікавлених сторін

- користувачі, які хочуть покращити якість та реалістичність передачі кольору своїх зображень;
- користувачі, які мають потребу в швидкому оброблені великої кількості зображень;
- професійні фотографи, які потребують відредактованого зображення одразу після зйомки.
- компанії, які займаються комерційною діяльністю з корегування та покращення зображень;

1.4.2 Розробка функціональних вимог

- зручний інтерфейс;
- сучасний дизайн;
- вибір та завантаження необхідних фотографій на сервер;
- вибір необхідного варіанту налаштування рівня балансу білого кольору;
- можливість завантаження архіву з фотографіями з серверу.

1.4.3 Розробка нефункціональних вимог

- швидка робота програми;
- кросплатформеність;
- стабільність;
- конфіденційність;

- низькі апаратні вимоги;
- надійність.

1.5 Огляд існуючих рішень

Adobe Lightroom – програма для ручного редагування параметрів освітлюваності зображень, випущена компанією Adobe в 2017 році. Порівняння оригіналу зображення, та такого, яке вдалось отримати з використанням даного програмного рішення, зображено на рисунку 1.3.

Рисунок 1.3 Результат роботи в Adobe Lightroom

Станом на 2022 рік, дане рішення коштує 9,99 доларів на місяць. Основна аудиторія користувачів – професійні фотографи, яким необхідно отримати ультимативне рішення для обробки зроблених знімків.

Рисунок 1.4 Користувацький інтерфейс програмного рішення Adobe Lightroom Classic CC 8.1

Тим не менш, інтерфейс даного програмного рішення, який зображено на рисунку 1.4, складно назвати інтуїтивно зрозумілим, тому що існує велику кількість налаштувань, які можуть бути неочевидними звичайним користувачам. Разом з тим, ручна обробка фотографій займає багато часу, тому можна зробити висновок, що Adobe Lightroom орієнтується на дуже вузькоспеціалізований зріз користувачів.

Google Photos – початково розроблялось як безкоштовна галерея фотографій, на даний момент дане програмне рішення включає в себе функціонал, який дозволяє виконувати невелике налаштування параметрів фотографій. Інтерфейс рішення зображено на рисунку 1.5.

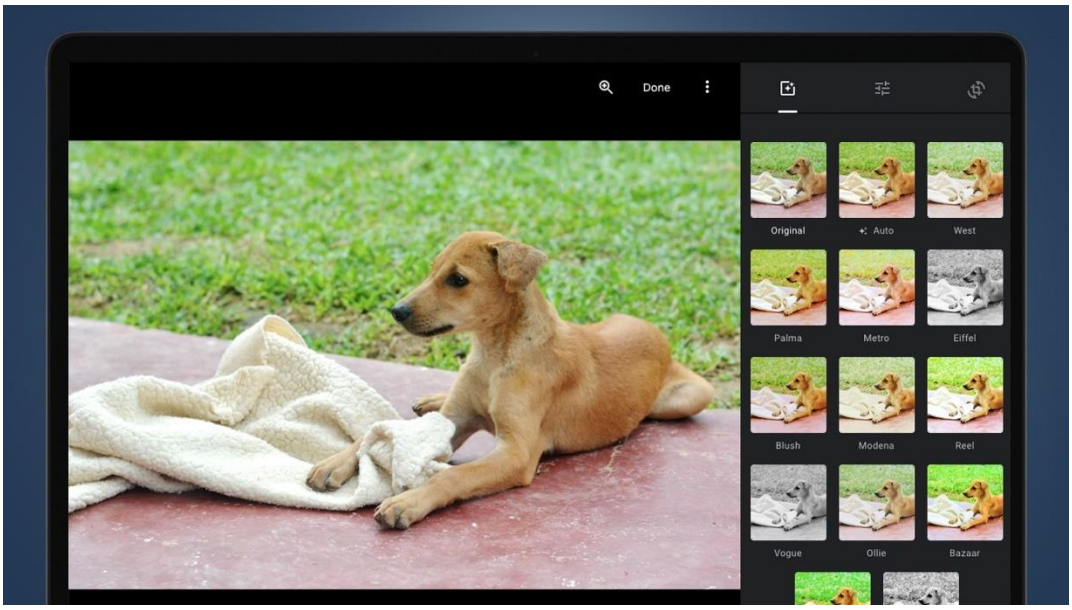

Рисунок 1.5 Інтерфейс програмного застосунку Google Photos

Однак, для більш детального редагування, а саме використання таких функцій, як портретне розмиття, портретне світло, фокус кольору, налаштування кольору неба, необхідно мати підписку google one, найдешевша з яких коштує 1,99 доларів за місяць, станом на 2022 рік.

Topaz Adjust AI – новітнє програмне рішення, яке використовує нейронні мережі для покращення параметрів фото. Повна версія програми коштує 79,99 доларів станом на 2022 рік. Результат роботи програмного рішення наведено на рисунку 1.6.

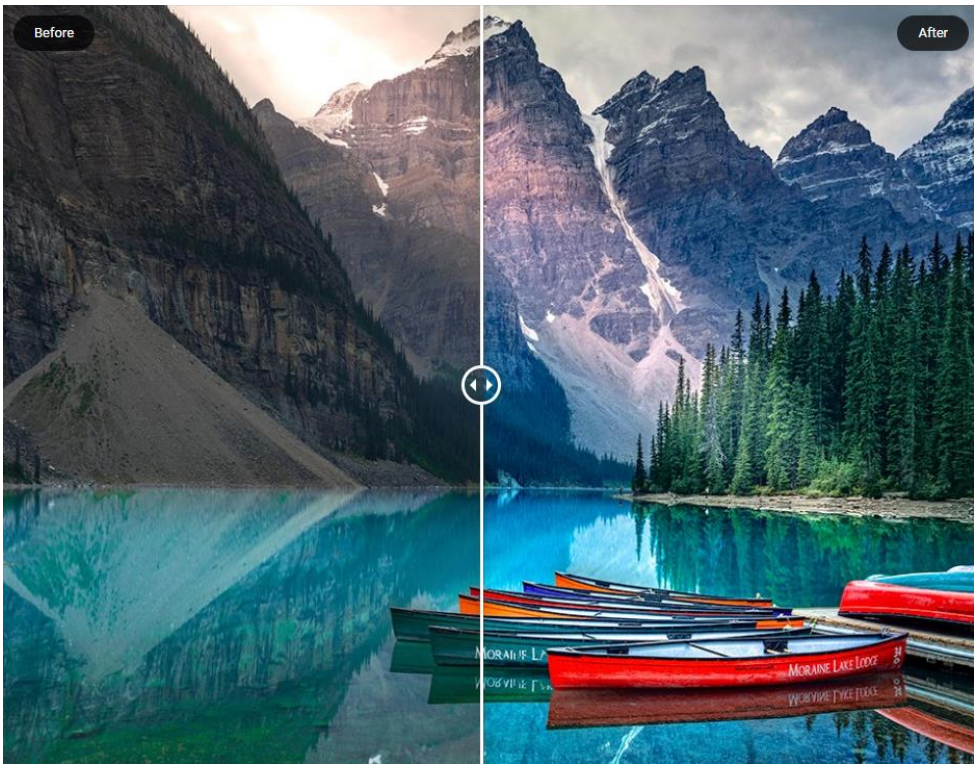

Рисунок 1.6 Результат роботи в програмному рішенні Topaz Adjust AI

Користувацький інтерфейс наведено на рисунку 1.7.

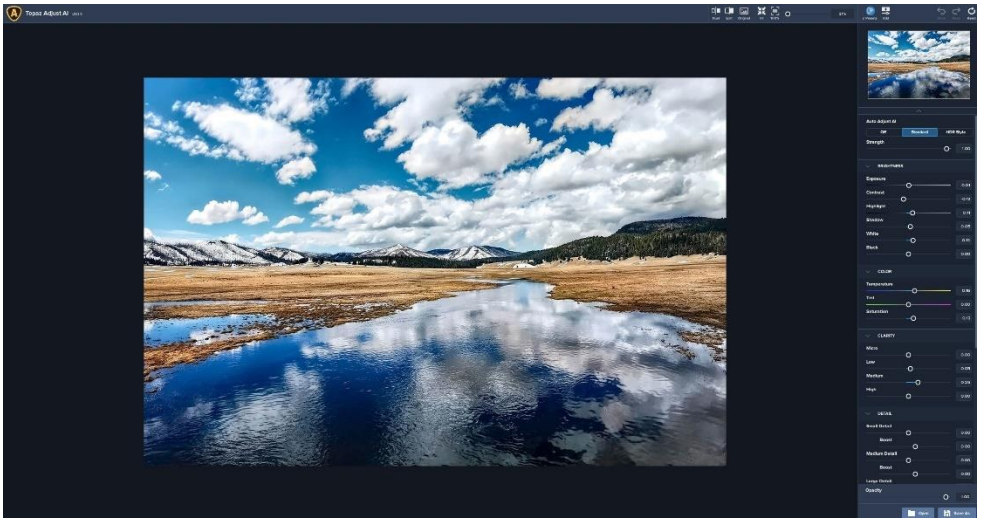

Рисунок 1.7 Користувацький інтерфейс програмного рішення Topaz Adjust AI

Як видно з рисунку 1.7, дане програмне рішення дозволяє виконувати базові налаштування фотографій для кращого вихідного результату. Тим не менш, даний застосунок дозволяє працювати лише з одним зображенням за раз,

також користувачу потрібен час для ручного налаштування параметрів знімку. Враховуючи вищесказане, можна зробити висновок, що дане програмне рішення не є ідеальним варіантом для широкої аудиторії.

Висновки до першого розділу

У процесі написання першого розділу було розглянуто історію та основні види нейронних мереж, досліджено технології, які використовуються для передачі кольору зображення, визначено область застосування, постановку задачі.

Після дослідження описаних вище особливостей нейронних мереж, параметрів зображення та вимог до програмного модулю, було виявлено, що найкраще для поставленої задачі підходить нейронна мережа типу encoder, яка буде корегувати баланс білого кольору в заданих зображеннях.

РОЗДІЛ 2 ПРОЄКТНІ РІШЕННЯ

2.1 Архітектура програмного рішення

Клієнтська частина:

- завантаження вхідних зображень на сервер;
- відображення отриманих оброблених зображень;
- можливість вибору необхідного варіанту зображення та завантаження всіх оброблених зображень у виді архіву.

Серверна частина:

- завантаження зображень;
- запуск нейронної мережі;
- видача оброблених зображень клієнтській частині.

Нейронна мережа:

- навчання моделі;
- обробка заданих зображень.

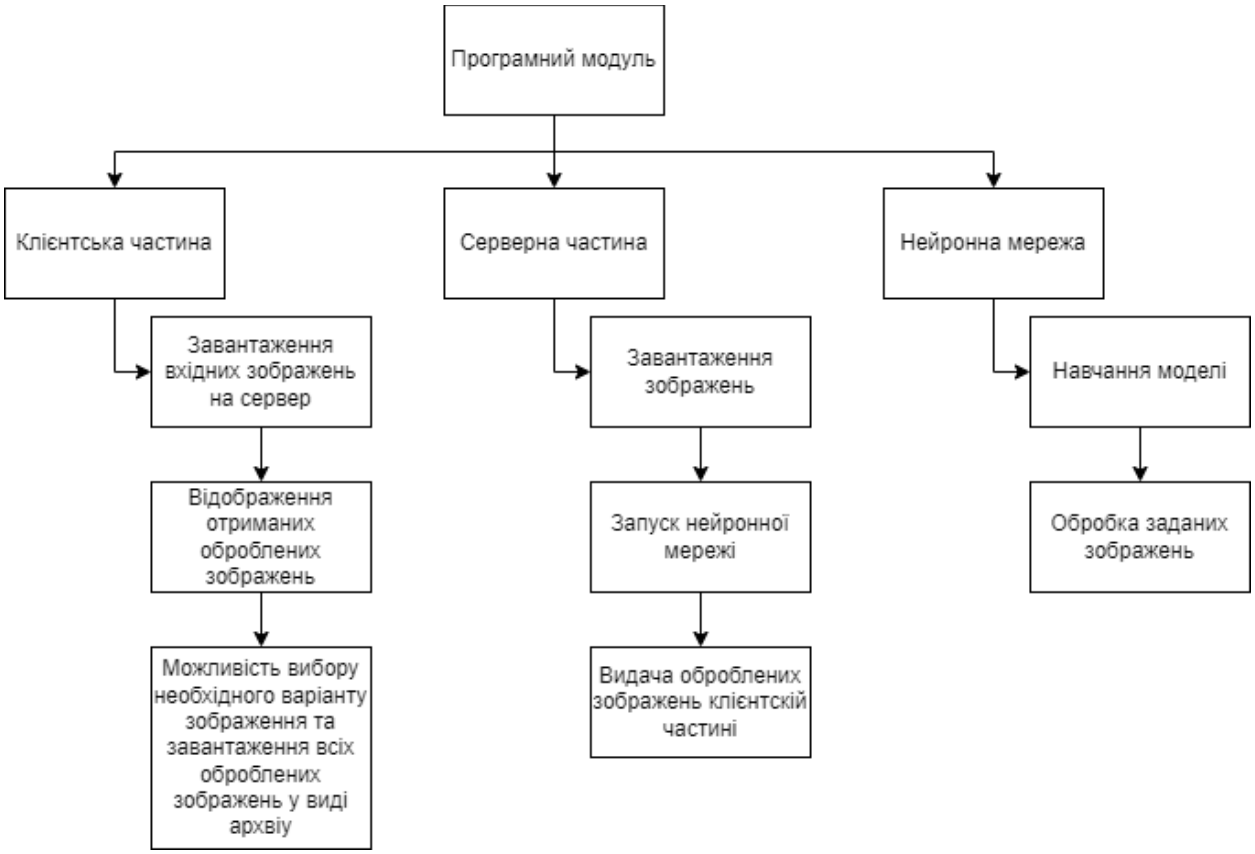

Рисунок 2.1 Архітектура програмного модулю

Користувач повинен мати змогу завантажити фотографії, які необхідно покращити, на сервер, та отримати у відповідь оброблені зображення. Повинен бути реалізований функціонал вибору необхідного зображення зі списку оброблених, після чого додаток повинен архівувати вибрані зображення і відправляти їх користувачу.

Серверна частина завантажує фотографії користувача, обробляє їх за допомогою нейронної мережі та відправляє необхідні зображення користувачу.

Модуль нейронної мережі повинен мати функціонал навчання моделі на необхідному датасеті за вибраними параметрами, після чого з’явиться можливість обробки зображень за допомогою навченої моделі

2.2 Розробка нейронної мережі

2.2.1 Дослідження нейронної мережі

Для виконання поставлених задач доцільно використовувати архітектуру нейронної мережі Autoencoder, яка дозволяє виконувати навчання без вчителя за допомогою методу зворотного поширення помилки.

Розроблена нейрона мережа буде налаштовувати баланс білого кольору використовуючи 3 конфігурації:

- Auto White Balance – коректне налаштування балансу білого кольору для сцени.
- Shade White Balance – репрезентує баланс білого для вуличних знімків.
- Tungsten White Balance – відображає баланс білого для знімків у приміщенні.

Використовуючи вихідні дані з моделей Shade WB та Tungsten WB можна інтерполювати результати для таких типів балансу білого:

- Cloudy, Daylight WB – тепліші варіації балансу білого для вуличних знімків.
- Fluorescent WB– більш холодна варіація налаштувань балансу білого для знімків у приміщенні.

В мережі будуть використані шари ReLu, max-pooling, depth concatenation та згорткові, та транспоновані згорткові.

Згорткові шари застосовують до вхідних даних операцію згортки, яка є імітацією реакції на зоровий стимул.

Транспоновані згорткові шари – на відміну від згорткових шарів, транспоновані виконують функцію апсемплінгу, тобто вхідні дані будуть мати менший розмір, ніж вихідні.

ReLu (від англ. Rectified linear unit – випрямлений лінійний вузол) – функція активації, є позитивною частиною від аргументу:

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

Переваги ReLu – рідше виникає проблема зникання градієнту, швидкий в обчисленні, інваріантний відносно масштабування

Max-pooling (з англ. – максимізаційне агрегування) – операція агрегування, яка обчислює максимум, тобто найбільше значення в кожній частині кожної feature map.

Depth concatenation – на вхід приймає вхідні дані однакової висоти та ширини, й об'єднує їх уздовж третього виміру.

Функція втрат – нейронна мережа повинна мінімізувати L_1 функцію втрат між реконструйованою та перевірчою частиною зображення.

$$\sum_i \sum_{p=1}^{3hw} \left| P_{WB^{(i)}}(p) - C_{WB^{(i)}}(p) \right|, \tag{2.2}$$

Де h та w – ширина та висота зображення, p проходиться по всім пікселям частини зображення, P_{WB} – частина зображення для тренування, C_{WB} – перевірче зображення.

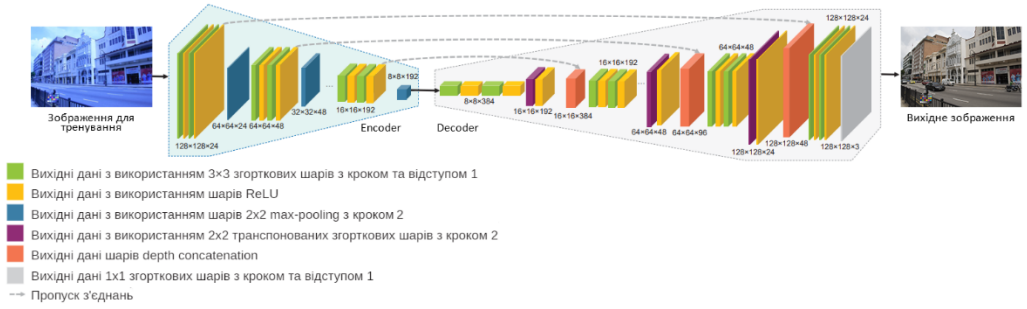

Рисунок 2.2 Архітектура нейронної мережі

На рисунку 2.2 зображено архітектуру нейронної мережі, яку потрібно розробити. Як видно зі схеми, дані будуть спочатку енкодуватись за допомогою згорткових, ReLu та max-pooling шарів, після чого декодуватись з використанням шарів ReLu, згорткових, depth concatenation та траспонованих згорткових, разом з тим пропускаючи деякі зв'язки для уникнення проблеми деградації через утрату даних.

2.2.2 Вибір інструментів для розробки програмного модулю нейронної мережі

- Python – високорівнева мова програмування, в основі логіки розробки якої на першому місці стоїть читабельність коду, яка досягається обов'язковим використанням відступів. Дана мова програмування надає динамічну типізацію та збір сміття, що дозволяє менше часу приділяти не критичним, для роботи програми, речам. Для Python безліч пакетів та фреймворків, які значно спрощують розробку нейронних мереж.
- Pytorch – фреймворк для машинного навчання з відкритим кодом. Дане рішення використовується в таких продуктах, як Tesla Autopilot, Hugging Face's Transformers, Uber Pyro, та інших успішних проєктах.
- TorchVision – складається з популярних датасетів, архітектури моделі, поширених трансформацій зображень.
- CudaToolKit – забезпечує середовище розробки для створення програм з використанням GPU (від англ. – graphics processing unit, графічний процесор) прискорення. Набір інструментів включає бібліотеки з GPU прискоренням, інструменти для дебагу й оптимізації.
- TensorBoard – надає інструменти для візуалізації, які необхідні для експериментів з машинним навчанням.
- Numpy – надає генератори випадкових чисел, комплексні математичні функції, перетворення Фур'є, підпрограми лінійної алгебри, тощо.,

- Pillow – дозволяє проводити обробку зображень в інтерпретаторі Python.
- Matplotlib – надає інструменти для створення статичних, анімованих, інтерактивних візуалізацій на Python.
- Scipy – надає алгоритми для інтегрування, інтерполяції, оптимізації, алгебраїчних рівнянь, диференціальних рівнянь, статистики та багатьох інших класів задач.
- Scikit-learn – надає прості та ефективні інструменти для прогнозованого аналізу даних

В якості набору даних для тренування було обрано датасет Rendered WB [3]. В даному датасеті використовується 62535 згенерованих зображень з різними випадковими параметрами для тренування та відповідний набір зображень з правильним балансом білого кольору. Приклад зображень наведено на рисунку 2.3.

Рисунок 2.3 Приклад зображень з датасету Rendered WB.

2.3 Аналіз та вибір інструментів для розробки клієнтської частини

Для прискорення та більшого комфорту при розробці веб-додатків мовою програмування JavaScript використовують фреймворки. Найпопулярніші з них:

Reactjs –бібліотека JavaScript для створення користувацьких інтерфейсів, який належить компанії Meta. Основні переваги фреймворку React – використання компонентів для інкапсуляції коду, використання стану, який

дозволяє зберігати дані і оновлювати компонент лише при зміні цих даних. Так як React використовує велика кількість розробників, то існує багато бібліотек саме для цього фреймворку, що робить React швидким, оптимізованим та зручним для розробки.

Angular – фреймворк, розроблений компанією Google. Він дозволяє швидко розробляти проєкти, в яких бере участь велика команда, використовуючи зрозумілу логіку роботи та типізацію з використанням Typescript.

Vue.js – відкритий open source проєкт, створений незалежним розробником. Vue використовує модель веб компонентів, проводячи ефективний рендеринг DOM (Document Object Model), реактивний менеджмент станів, та не включає пакети, які не використовуються в додатку, що дозволяє оптимізувати кількість даних, які завантажує користувач.

Проаналізувавши основні фреймворки JavaScript з’ясовуємо, що найкращим вибором для системи, яка розробляється в даній роботі є фреймворк React. Так як даний фреймворк є найпопулярнішим, існує велика кількість документації та допоміжних матеріалів, що дозволяє пришвидшити розробку та зменшити кількість коду, яку потрібну писати розробникам.

Логіка React передбачає використання компонентів, кожен з яких має свої можливості для реалізації інтерактивності та дизайну. Важливий плюс компонентів полягає в тому, що їх можна перевикористовувати, таким чином полегшивши розробку проекту та зробивши його набагато зручнішим та зрозумілішим.

Ще одним плюсом React є використання Virual DOM (Document Object Model – в перекладі з англійської – «Об’єктна модель документа») замість класичного HTML DOM. Це дозволяє додатку працювати набагато швидше, оскільки Virtual DOM, на відміну від звичайного HTML DOM, при зміні одного

елемента оновлює не повністю все дерево, а лише ту частину, яка змінилась, а справжній DOM оновлюється лише тоді, коли це дійсно потрібно.

Разом з тим, React, на відміну від деяких інших фреймворків, має чудові інструменти для оптимізації пошукових запитів (SEO), що дозволяє отримати більший потік клієнтів на проєкт.

Бібліотеки, які будуть використовуватись у проєкті:

- antd – UI бібліотека для React, яка була розроблена згідно зі специфікацією Ant Design, написана з передбачуваними статичними типами використовуючи TypeScript. Основні переваги – великий набір високоякісних компонентів, кастомізація елементів. Дана бібліотека дозволяє швидко та без зайвих зусиль створити інтерфейс, який буде сучасним та зручним для користувача. Приклад веб-застосунку, розробленого за допомогою antd, наведено на рисунку 2.4.

Рисунок 2.4 Приклад веб-застосунку з використанням antd

- Uuid – дозволяє генерувати унікальний id для створюваних об’єктів. Дана бібліотека має 5 версій генерації, серед яких найвикористовуваніші це v1, v3 та v5. V1 генерує унікальний код ідентифікації ґрунтуючись на MAC-адресі мережевої карти та поточному часі. Дана версія не гарантує

безпеку, тому її бажано використовувати лише тоді, коли необхідно отримати просто унікальний id. V3 використовує MD5 хеш з простору імен. V5 в свою чергу, використовує хеш SHA-1, тому це найбезпечніша версія.

- Eslint – знаходить та виправляє проблеми в коді, допомагає дотримуватись заданого стилю впродовж всього проєкту. Оскільки JavaScript є гнучкою мовою програмування, це дозволяє писати код різними способами, що може призвести до значної кількості помилок, особливо на великих проєктах. Eslint дозволяє створити власний стиль коду, якого необхідно притримуватись. Це значно спрощує розробку та робить код більш читабельним і передбачуваним.

2.4 Вибір інструментів для розробки серверної частини

Nodejs – представляє асинхронне середовище виконання JavaScript. Після кожного підключення запускається callback (в перекладі з англійської мови – зворотній виклик), якщо задач немає, nodejs переходить у режим сну, на відміну від більш поширеної моделі паралельності, коли використовуються потоки операційної системи.

Express – мінімалістичний фреймворк для Nodejs, який дозволяє гнучко та швидко налаштувати сервер.

Flask – мікрофрейморк для Python, який зберігає основний функціонал простим, але розширюваним. Переваги Flask – висока швидкодія, низькі витрати пам’яті, велика кількість допоміжних пакетів, широка підтримка ком’юніті.

Так як нейронна мережа написана за допомогою мови Python, доцільніше використовувати Flask для написання серверної частини додатку.

2.5 Макет інтерфейсу застосунку

Веб-сайт повинен складатись з двох сторінок – на першій користувач може завантажити необхідні фотографії на сервер, після чого відбувається перехід до другої сторінки додатка, де відображаються оброблені зображення, отриманні з серверу.

При натисканні на оброблене зображення, повинно відкриватись модальне вікно з різними варіантами обробленого зображення та оригіналом для порівняння.

Після обирання найкращого зображення, даний вибір повинен зберегтись для того, щоб користувач мав змогу завантажити архів з усіма обробленими зображеннями.

Для створення даного макету застосунку найкращим вибором стане програмне забезпечення Figma, яке є популярним графічним редактором для веб-розробки. Figma дозволяє швидко розробити зрозумілий макет застосунку, подивитись, як рішення буде виглядати на різних пристроях з різною шириною браузерної області перегляду

На рисунку 2.4 зображено макет першої сторінки. Для прикладу було зображено сірі прямокутники з підписами User’s photo 1 та User’s photo 2, які позначають фотографії, що користувач буде завантажувати на сервер. Кнопка Add повинна давати змогу вибрати зображення для завантаження, кнопка Upload – забезпечити початок обробки завантажених зображень та перехід на другу сторінку застосунку при виконанні даної задачі серверною частиною веб додатку.

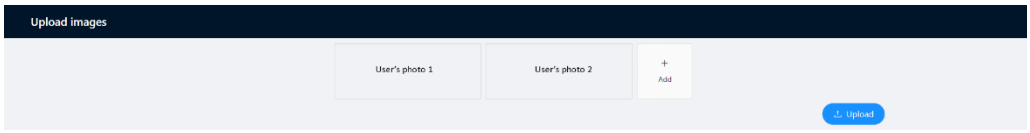

Рисунок 2.4 Макет першої сторінки

На рисунку 2.5 зображено макет другої сторінки веб-застосунку, яку користувач бачить при натисненні кнопки Upload на першій сторінці. На даному макеті бачимо сірі прямокутники Processed photo 1 та Processed photo 2, які означають оброблені сервером зображення користувача. У верхньому правому кутку є кнопка Download all, яка дозволяє завантажити архів з усіма поточними вибраними фотографіями.

Рисунок 2.5 Макет другої сторінки

При натисканні на оброблене зображення повинно відкриватись модальне вікно, зображене на рисунку 2.6.

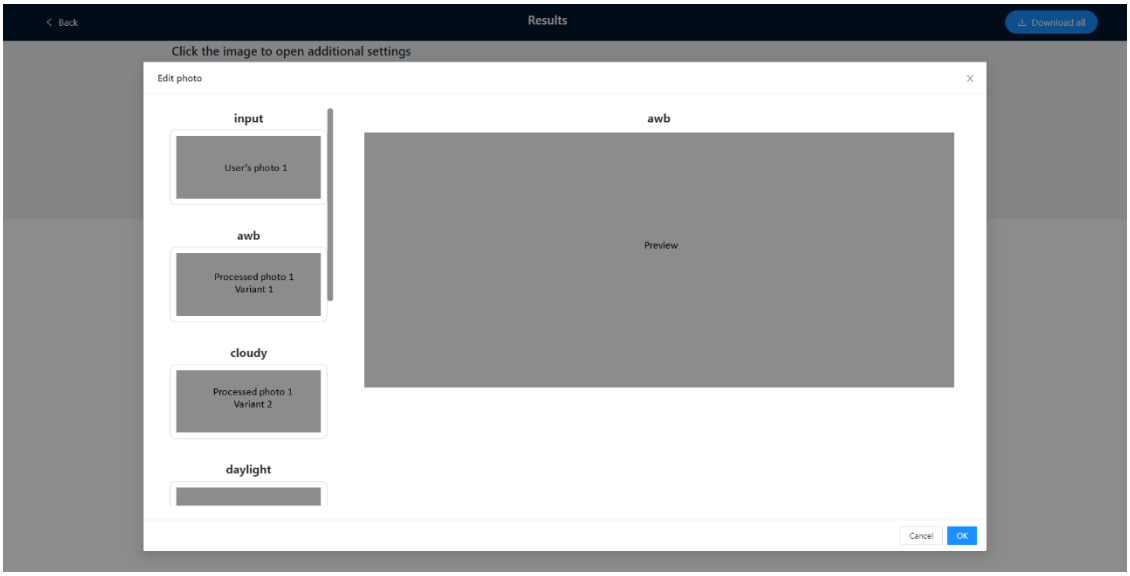

Рисунок 2.6 Модальне вікно на другій сторінці

Сірий прямокутник User’s photo 1 позначає зображене користувачем фото, Processed photo 1 Variant 1 та 2 – різні варіанти обробленого зображення, з яких користувач може вибрати найкраще. Прямокутник Preview відображає поточне вибране зображення. При натисканні кнопки Ok поточний вибраний варіант повинен зберегтись та замінити попередній варіант цієї фотографії, та закрити модальне вікно. При натисканні Cancel варіант повинен скинутись та повернути попередній, закривши модальне вікно.

Висновки до другого розділу

У процесі написання другого розділу було проведено необхідні дослідження можливих варіантів розробки програмного застосунку, побудовано архітектуру програмного рішення.

Був проведений аналіз існуючих рішень для розробки програмного забезпечення та обрано ті, які є оптимальними для даної роботи, побудовано макет застосунку та описано необхідну логіку роботи інтерфейсу.

РОЗДІЛ 3 ТЕХНІЧНІ ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ

3.1 Встановлення та запуск розробленої системи

Для демонстрації роботи розробленої системи використовувався настільний персональний комп’ютер з встановленою операційною системою Microsoft Windows 10 Pro 20H2 таким апаратним забезпеченням:

- процесор - Intel Core i7-8700, має 6 фізичних ядер, які працюють на частоті 4.3ГГц, з технологією Hyper Threading, завдяки чому кількість потоків, які визначаються системою, зростає до 12, та 12 Мб кеш-пам’яті третього рівню.
- відеокарта – Nvidia GeForce GTX 1660 TI, яка має 1536 CUDA ядер та 6 Гб відеопам’яті з пропускною здатністю 288Гб/с.
- оперативна пам’ять – 16 Гб типу DDR4 на частоті 2.4ГГц в двоканальному режимі.

3.1.1 Запуск клієнтської частини

Для коректної роботи програми необхідно встановити NodeJs версії 16.4.2, завантажений з офіційного сайту.

Необхідно відкрити теку з клієнтською частиною, відкрити командний рядок та виконати команди:

```
npm install
npm start
```

Після цього відриється вікно браузера localhost:3000 з робочою клієнтською частиною.

3.1.2 Запуск та тренування нейронної мережі

Для роботи нейронної мережі необхідно встановити Python версії 3.10.2 та CUDA Toolkit 11.7, завантажені з офіційного сайту.

Після цього необхідно встановити наступні пакети:

- pytorch 1.11.0
- torchvision 0.12.0
- tensorboard 1.8.1
- numpy 1.22.4
- Pillow 9.1.1
- Future 0.18.2
- Tqdm 4.64.0
- matplotlib 3.5.2
- scipy 1.8.1
- scikit-learn 1.1.1

Для запуску необхідно відкрити командний рядок в теці з нейронною мережею та виконати команду

```
python train.py --training_dir ../dataset/ --fold 0 --epochs 500 --learning-rate-drop-period 50 --num_training_images 0
```

3.1.3 Запуск серверної частини

Для роботи серверної частини, крім Python 3.10.2, потрібно встановити такі пакети:

- flask
- flask_cors

Після цього необхідно відкрити командний рядок у теці з серверною частиною та виконати команду

```
python main.py
```

3.2. Опис структури розробленого рішення

3.2.1 Опис структури клієнтської частини

На рисунку 3.1 наведено знімок екрану структури клієнтської частини в засобі для розробки Webstrom.

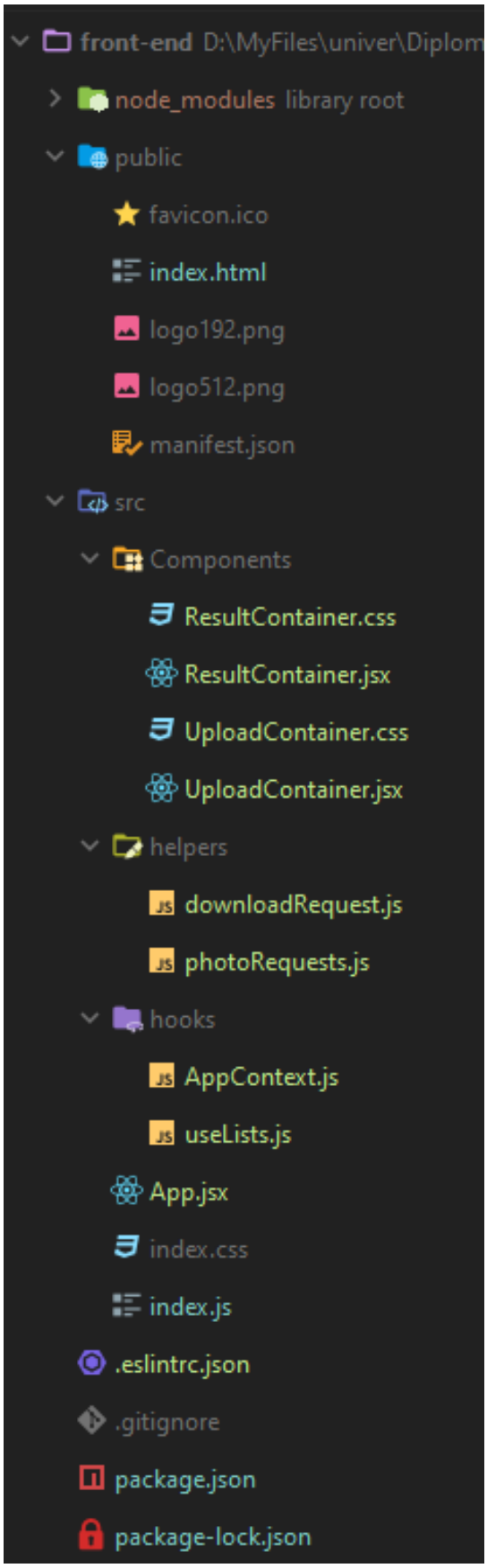

Рисунок 3.1 Структура клієнтської частини

Опис розроблених елементів додатку:

- package.json – використовується для конфігурації розробленого додатку, включає в себе доступні для виконання команди, список необхідних пакетів для коректної роботи програми та метадані;
- .eslintrc.json – включає конфігурацію пакету eslint, який дозволяє уніфікувати стиль розробленого коду, що потенційно може допомогти уникнути небажаних помилок та збільшити читабельність коду та зручність розробки;
- index.js – вхідна точка програми, використовується для збірки проекту;
- index.css –корневі стилі для веб-сторінки;
- App.jsx – файл, в якому підключено компоненти додатка, та розроблено коректний зв’язок між ними;
- useLists.js, AppContext.js – використовується для передачі даних між компонентами;
- photoRequests.js – відправляє фотографії на сервер;
- downloadRequest.js –обробляє дані з серверу;
- UploadContainer.jsx – компонент, який використовується для завантаження фотографій на сервер;
- ResultConatiner.jsx – компонент, що використовується для виведення на екран одержаних даних з серверу, надає можливість вибору необхідних фотографій;
- UploadContainer.css, ResualtContainer.css – стилі для відповідних компонент;
- index.html – шаблон розмітки веб сторінки;
- тека node_modules – використовується для збереження необхідних для роботи програми пакетів.

3.2.2 Опис структури серверної частини та нейронної мережі

На рисунку 3.2 наведено знімок екрану структури серверної частини, та нейронної мережі у засобі для розробки Pycharm.

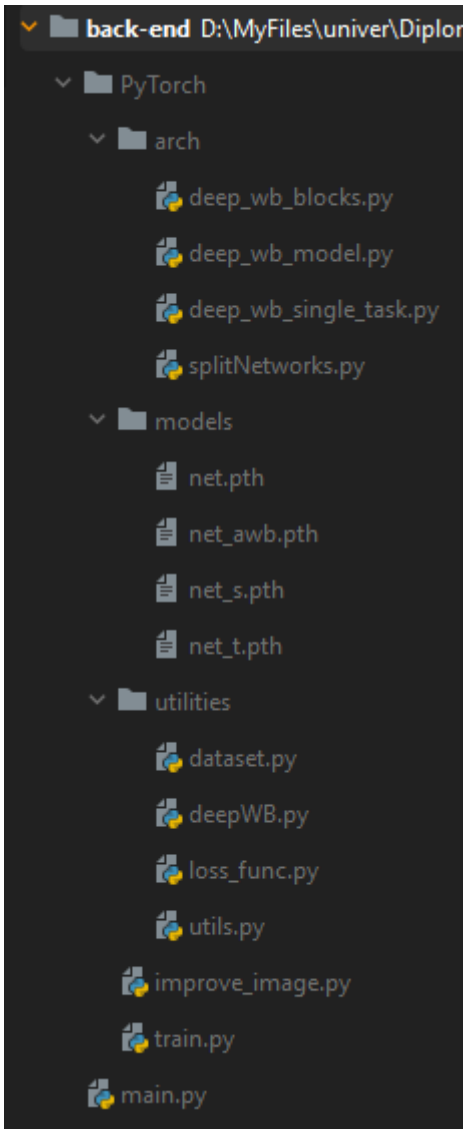

Рисунок 3.2 Структура серверної частини та нейронної мережі

Опис розроблених елементів додатку:

- main.py – вхідна точка програми, виконує запити з клієнтської частини, оброблює та видає необхідну інформацію;
- train.py – файл програми, яка використовується для тренування нейронної мережі;
- improve_image.py – запускає нейронну мережу, передає необхідні дані та видає результат;

- тека utilities – представляє набір допоміжних функцій для роботи нейронної мережі;
- тека models – зберігає натреновані моделі мережі;
- тека arch – допоміжні контролери, які дозволяють використовувати необхідні бібліотеки.

3.2.3 Опис структури інтерфейсу

При відкритті головної веб-сторінки бачимо меню вибору та завантаження фотографій на сервер. Знімок екрану з головною сторінкою представлений на рисунку 3.3.

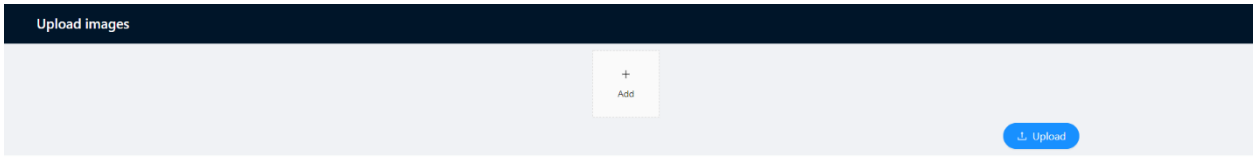

Рисунок 3.3 Знімок екрану головної сторінки

Натиснувши на кнопку «Add», відкривається вікно, в якому можемо вибрати зображення, які необхідно покращити (Рисунок 3.4).

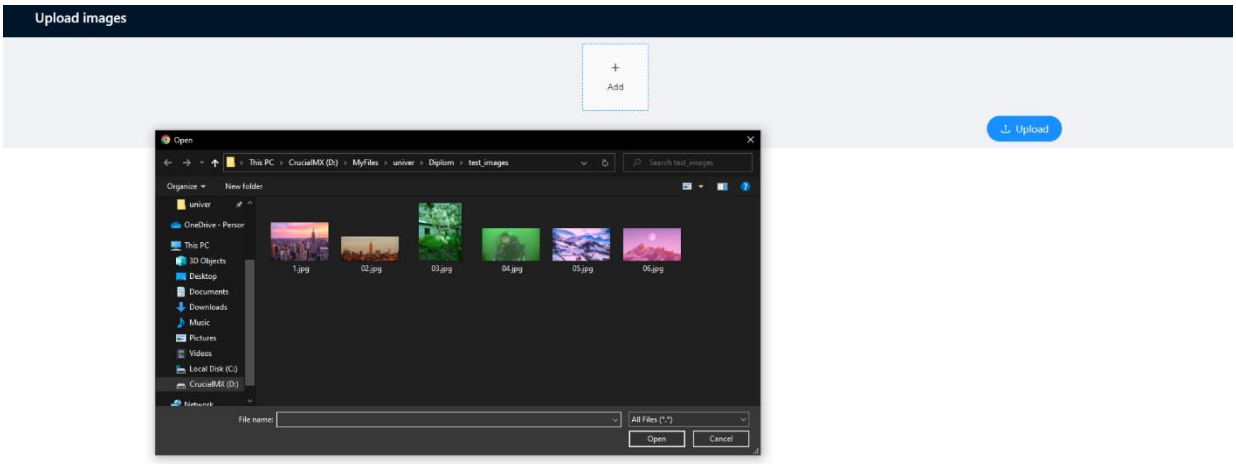

Рисунок 3.4 Знімок екрану з відкритим вікном завантаження зображень

Після вибору необхідних зображень та натискання кнопки «Open», бачимо, що вибрані зображення відображаються на головній сторінці (Рис. 3.5).

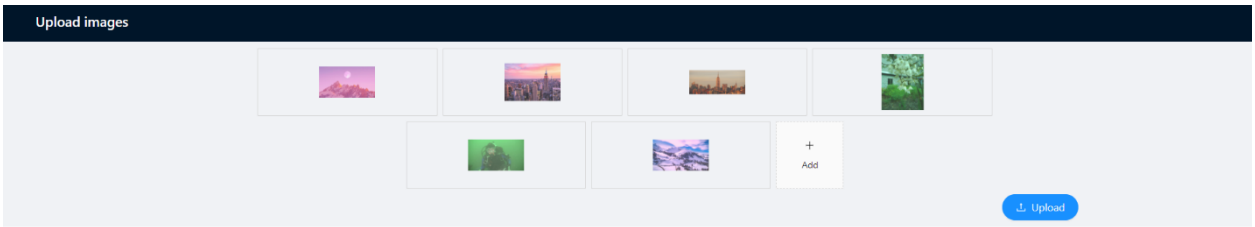

Рисунок 3.5 Знімок екрану головною сторінки після вибору фотографій

При наведені курсору на мініатюру зображення (рис. 3.6), з’являються кнопки для попереднього перегляду (рис 3.7), та видалення фотографії.

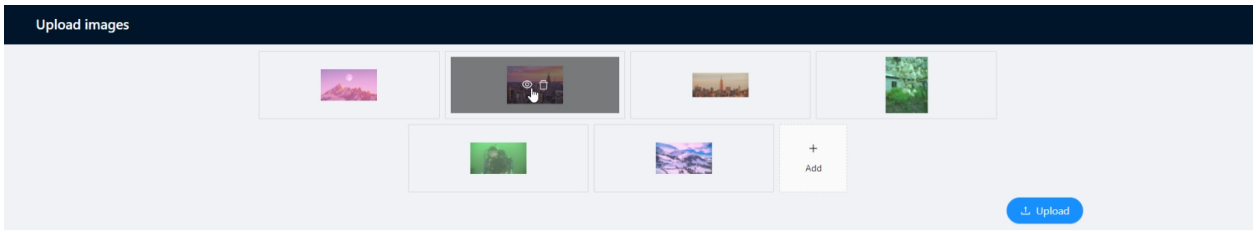

Рисунок 3.6 Знімок екрану головної сторінки при наведені курсору на мініатюру зображення

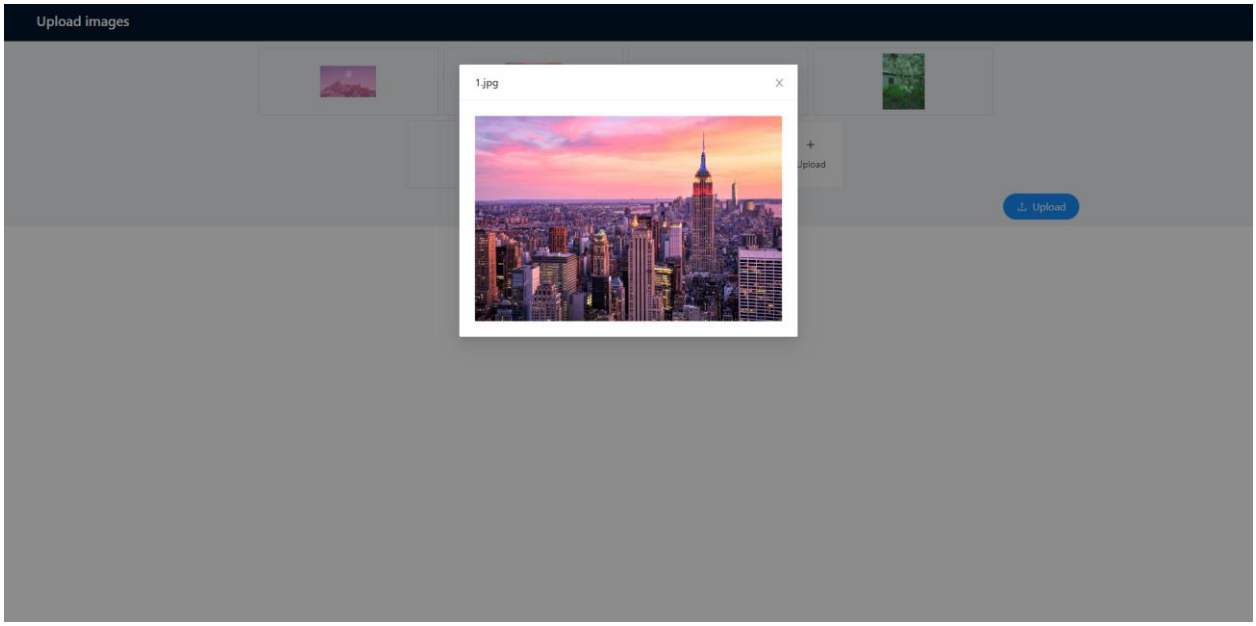

Рисунок 3.7 Знімок екрану головної сторінки при попередньому перегляді зображення

Після натискання на кнопку «Upload» зображення відправляються на сервер та оброблюються, результат обробки бачимо на рис. 3.8.

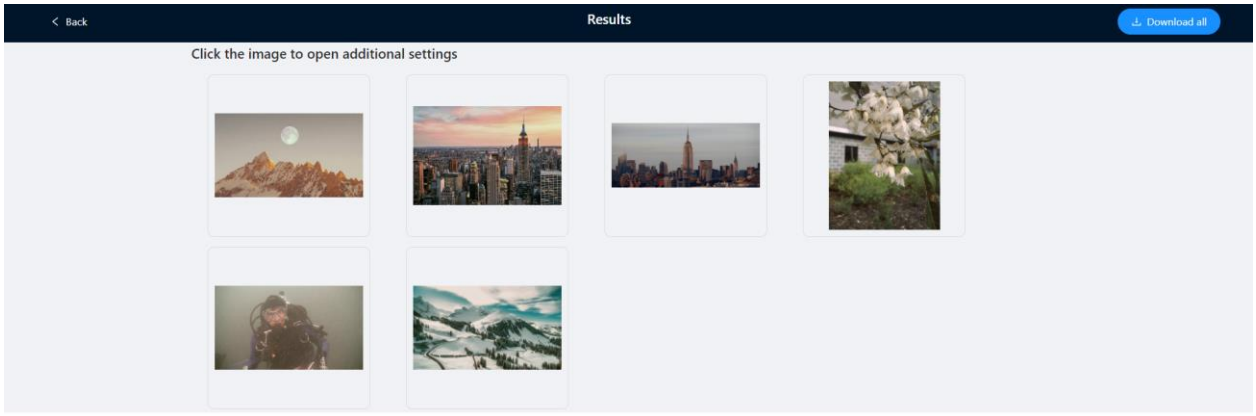

Рисунок 3.8 Знімок екрану з результатами обробки зображень

При натисканні на мініатюру зображення, воно відкривається в повному розмірі, де можна вибрати кращу варіацію зі згенерованих (Рис. 3.9), після чого вибір збережеться і відобразиться на сторінці з результатами (Рис. 3.10).

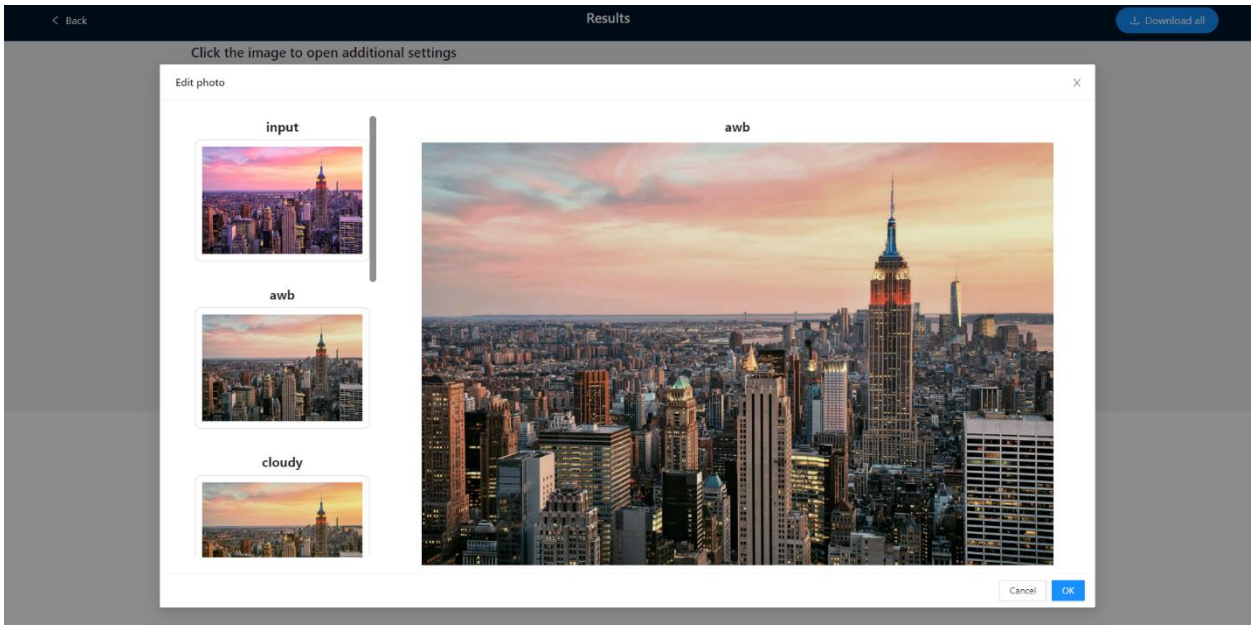

Рис. 3.9 Знімок екрану з вікном вибору варіанту зображення

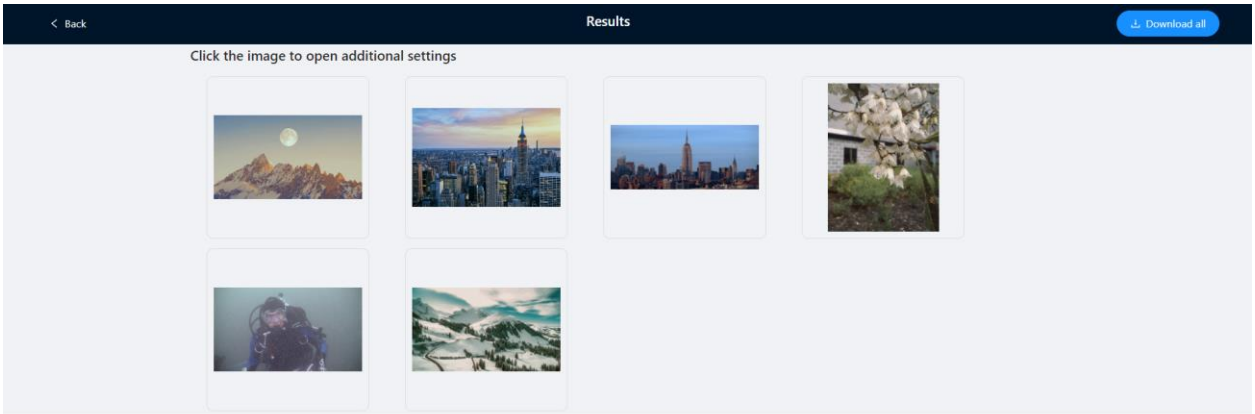

Рис. 3.10. Знімок екрану з результатами

Після вибору необхідних варіантів, можна натиснути на кнопку «Download all» для завантаження архіву з усіма вибраними зображенями.

3.3 Тестування розробленого програмного рішення

Виконавши попередньо описані кроки, стає можливим протестувати програму та порівняти вхідні зображення з обробленими нейронною мережею.

Оригінал зображення (зліва) та оброблене зображення за допомогою AWB – автоматичної корекції балансу білого (справа) представлені на тестових зображеннях нижче (рис. 3.11 – 3.14).

Рис. 3.11 Перше тестове зображення AWB

Рисунок 3.12 Друге тестове зображення AWB

Рисунок 3.13 Третє тестове зображення AWB

Рисунок 3.14 Четверте тестове зображення AWB

В якості зображення для тестування роботи інших параметрів корекції було прийнято рішення вибрати перше тестове зображення (Рис. 3.11). Результати роботи програми видно на рис. 3.15-3.21.

Рисунок 3.15 Вхідне зображення

Рисунок 3.16 Корекція Auto White Balance

Рисунок 3.17 Корекція Cloudy White Balance

Рисунок 3.18 Корекція Daylight White Balance

Рисунок 3.19 Корекція Fluorescent White Balance

Рисунок 3.20 Корекція Shade White Balance

Рисунок 3.21 Корекція Tungsten White Balance

Як можна побачити з результатів тестування, нейронна мережа чудово виконує поставлені завдання з покращення передачі кольору та освітлюваності зображення.

Висновки до третього розділу

В результаті праці над третім розділом дипломної роботи було розроблено детальну інструкцію з процесу встановлення та запуску розробленого програмного рішення, навчання нейронної мережі, та використання готової системи для покращення зображень.

Було проведено тестування розробленої системи, в рамках якої отримано очікувані результати.

ВИСНОВКИ

В ході виконання випускної кваліфікаційної роботи, було розроблено систему покращення передачі кольору та освітлення зображень з використанням технологій штучного інтелекту. Відповідно до поставленої мети, програмне рішення приймає вхідне зображення та видає його покращені версії.

Під час виконання даної роботи було виконано аналіз існуючих видів нейронних мереж, досліджено технології, які використовуються для передачі кольору зображення, визначено область застосування, постановку задачі, побудовано програмне рішення, проведено тестування.

Розроблене рішення може бути впроваджено в комерційну діяльність в короткі терміни після незначних змін.

Список використаних джерел

1. Mahmoud Afifi and Michael S Brown. Deep White-Balance Editing. In CVPR [Електронний ресурс] – 2020. – режим доступу до ресурсу: https://openaccess.thecvf.com/content_CVPR_2020/papers/Afifi_Deep_White-Balance_Editing_CVPR_2020_paper.pdf
2. Mahmoud Afifi, Brian Price, Scott Cohen and Michael S. Brown. When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images [Електронний ресурс] – 2019. – режим доступу до ресурсу: https://openaccess.thecvf.com/content_CVPR_2019/papers/Afifi_When_Color_Constancy_Goes_Wrong_Correcting_Improperly_White-Balanced_Images_CVPR_2019_paper.pdf
3. Mahmoud Afifi, Brian Price, Scott Cohen and Michael S. Brown. When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images Dataset [Електронний ресурс] – 2019. – режим доступу до ресурсу: https://cvil.eecs.yorku.ca/projects/public_html/sRGB_WB_correction/dataset.html
4. Artificial Neuron Models [Електронний ресурс] – режим доступу до ресурсу: https://cnl.salk.edu/~schraudo/teach/NNcourse/ann-overview.html
5. Neural Networks History: The 1940's to the 1970's [Електронний ресурс]. – режим доступу до ресурсу: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html
6. Neural Networks History: The 1980's to the present [Електронний ресурс]. – режим доступу до ресурсу: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html
7. HUE, VALUE, SATURATION [Електронний ресурс]. – режим доступу до ресурсу:

https://www.abss.k12.nc.us/cms/lib02/NC01001905/Centricity/Domain/5234/2.01%20Hue%20Value%20SaturationNOTES.pdf

8. Todd Vorenkamp. Understanding White Balance and Color Temperature in Digital Images [Електронний ресурс]. – 2015. – режим доступу до ресурсу: https://www.bhphotovideo.com/explora/photography/tips-and-solutions/understanding-white-balance-and-color-temperature-digital-images
9. Mars Xiang. Convolutions: Transposed and Deconvolution [Електронний ресурс]. – 2020. – режим доступу до ресурсу: https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6
10. Jason Brownlee. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks [Електронний ресурс]. – 2019. – режим доступу до ресурсу: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/
11. What the ... White Balance? [Електронний ресурс]. – 2014. – режим доступу до ресурсу: http://www.boostyourphotography.com/2014/04/white-balance.html
12. IBM Cloud Education. Recurrent Neural Networks [Електронний ресурс]. – 2020. – режим доступу до ресурсу: https://www.ibm.com/cloud/learn/recurrent-neural-networks
13. Luthfi Ramadhan. Radial Basis Function Neural Network Simplified [Електронний ресурс]. – 2021. – режим доступу до ресурсу: https://towardsdatascience.com/radial-basis-function-neural-network-simplified-6f26e3d5e04d
14. Дмитро Чумаченко. Вступ до машинного навчання [Електронний ресурс]. – режим доступу до ресурсу: http://specials.kunsht.com.ua/machinelearning2
15. Prashant Sharma. Feedforward Neural Network: Its Layers, Functions, and Importance [Електронний ресурс]. – 2022. – режим доступу до ресурсу:

https://www.analyticsvidhya.com/blog/2022/01/feedforward-neural-network-its-layers-functions-and-importance/

16. Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [Електронний ресурс]. – 2018. – режим доступу до ресурсу: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
17. Hmirshav Bandyopadhyay. An Introduction to Autoencoders: Everything You Need to Know [Електронний ресурс]. – 2022. – режим доступу до ресурсу: https://www.v7labs.com/blog/autoencoders-guide

Додатки

Додаток А

Лістинг клієнтської частини

Лістинг файлу package.json

```
{
  "name": "front-end",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@ant-design/icons": "^4.7.0",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.2.0",
    "@testing-library/user-event": "^13.5.0",
    "antd": "^4.20.6",
    "antd-img-crop": "^4.2.3",
    "eslint": "^8.16.0",
    "eslint-config-airbnb": "^19.0.4",
    "react": "^18.1.0",
    "react-dom": "^18.1.0",
    "react-router-dom": "^6.3.0",
    "react-scripts": "5.0.1",
    "uuid": "^8.3.2",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
```

```
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
 }
}
```

Лістинг файлу index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
 <React.StrictMode>
  <App />
 </React.StrictMode>,
);
```

Лістинг файлу App.jsx

```
import React from 'react';
import { BrowserRouter, Routes, Route } from 'react-router-dom';

import UploadContainer from './Components/UploadContainer';
import ResultContainer from './Components/ResultContainer';

import AppContext from './hooks/AppContext';
import useLists from './hooks/useLists';

import 'antd/dist/antd.css';

function App() {
 const listHooks = useLists();
```

```
  return (
    <AppContext.Provider
      value={listHooks}
    >
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<UploadContainer />} />
          <Route path="/result" element={<ResultContainer />} />
        </Routes>
      </BrowserRouter>
    </AppContext.Provider>
  );
}

export default App;
```

Лістинг файлу useLists.js

```
import { useState } from 'react';

const useLists = () => {
  const [serverResponse, setServerResponse] = useState([]);

  return {
    serverResponse, setServerResponse,
  };
};
export default useLists;
```

Лістинг файлу AppContext.js

```
import { createContext } from 'react';

const AppContext = createContext(null);

export default AppContext;
```

Лістинг файлу photoRequest.js

```
const sendPhotos = async (list) => {
  const formData = new FormData();

  list.forEach((file) => {
```

```
    formData.append('files[]', file.originFileObj);
  });

  return fetch('http://localhost:5000/upload', {
    method: 'POST',
    body: formData,
  });
};

export default sendPhotos;
```

Лістинг файлу downloadRequest.js

```
const downloadArchive = async (list) => {
  return fetch('http://localhost:5000/archive', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(list),
  });
};

export default downloadArchive;
```

Лістинг файлу UploadContainer.jsx

```
import React, { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';

import {
  UploadOutlined, PlusOutlined,
} from '@ant-design/icons';

import {
  Button, Layout, message, Modal, Upload, Spin,
} from 'antd';

import './UploadContainer.css';

import AppContext from '../hooks/AppContext';
import sendPhotos from '../helpers/photoRequests';
```

```
const { Header, Content } = Layout;

const getBase = (file) => new Promise((resolve, reject) => {
  const reader = new FileReader();
  reader.readAsDataURL(file);

  reader.onload = () => resolve(reader.result);

  reader.onerror = (error) => reject(error);
});

function UploadContainer() {
  const { setServerResponse } = useContext(AppContext);
  const navigate = useNavigate();

  const [isLoading, setIsLoading] = useState(false);

  const [fileList, setFileList] = useState([]);
  const [previewVisible, setPreviewVisible] = useState(false);
  const [previewImage, setPreviewImage] = useState('');
  const [previewTitle, setPreviewTitle] = useState('');

  const handleUpload = async () => {
    setIsLoading(true);

    await sendPhotos(fileList)
      .then((res) => res.json())
      .then((json) => {
        setFileList([]);
        setServerResponse(json);

        setIsLoading(false);
        message.success('upload successful');
        navigate('/result');
      })
      .catch(() => {
        setIsLoading(false);
        message.error('upload failed');
      });
  };
```

```
const handleChange = ({ fileList: newFileList }) => {
  setFileList(newFileList);
};

const beforeUpload = (file) => {
  setFileList([...fileList, file]);
  return false;
};

const handleCancel = () => setPreviewVisible(false);

const handlePreview = async (paramFile) => {
  const file = paramFile;
  if (!file.url && !file.preview) {
    file.preview = await getBase(file.originFileObj);
  }

  setPreviewImage(file.url || file.preview);
  setPreviewVisible(true);
  setPreviewTitle(file.name || file.url.substring(file.url.lastIndexOf('/') + 1));
};

const uploadButton = (
  <div>
    <PlusOutlined />
    <div
      style={{
        marginTop: 8,
      }}
    >
      Add
    </div>
  </div>
);

return (
  <Layout>
    <Header>
      <h1 className="header">Upload images</h1>
      <Spin size="large" spinning={isLoading} />
    </Header>
    <Content>
```

```
        <Upload
          action="http://localhost:5000/upload"
          listType="picture-card"
          fileList={fileList}
          onPreview={handlePreview}
          onChange={handleChange}
          beforeUpload={beforeUpload}
          multiple
        >
          {fileList.length >= 200 ? null : uploadButton}
        </Upload>
        <Modal
          visible={previewVisible}
          title={previewTitle}
          footer={null}
          onCancel={handleCancel}
        >
          <img alt="example" style={{ width: '100%' }} src={previewImage} />
        </Modal>
        <Button
          className="upload-button"
          size="large"
          type="primary"
          shape="round"
          icon={<UploadOutlined />}
          onClick={handleUpload}
          disabled={isLoading}
        >
          Upload
        </Button>
      </Content>
    </Layout>
  );
}

export default UploadContainer;
```

Лістинг файлу ResultContainer.jsx

```
import React, { useContext, useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { v1 } from 'uuid';
```

```
import { DownloadOutlined, LeftOutlined } from '@ant-design/icons';
import {
  Button, Layout, message, Modal,
} from 'antd';

import AppContext from '../hooks/AppContext';
import './ResultContainer.css';
import downloadArchive from '../helpers/downloadRequest';

const { Header, Content } = Layout;

function UploadContainer() {
  const navigate = useNavigate();

  const [visible, setVisible] = useState(false);
  const [currentPhoto, setCurrentPhoto] = useState({});
  const [variants, setVariants] = useState([]);

  const { serverResponse } = useContext(AppContext);
  const [photosList, setPhotosList] = useState([]);

  const createPhotoList = () => {
    if (!Object.keys(serverResponse).length) {
      setPhotosList([]);
      return;
    }
    const list = serverResponse.data.map((file) => {
      return {
        host: serverResponse.host,
        type: 'awb',
        id: v1(),
        file,
      };
    });
    setPhotosList(list);
  };

  const openModal = (e) => {
    const { id } = e.target;
    const current = photosList.find((photo) => {
      return photo.id === id;
```

```
  });
  setCurrentPhoto(current);
  setVariants(['input', 'awb', 'cloudy', 'daylight', 'fluorescent', 'shade', 'tungsten'].map((type) => {
    return { ...current, type };
  }));
  setVisible(true);
};

const handleCancel = () => {
  setVisible(false);
};

const handleOk = () => {
  const changedList = photosList.map((e) => {
    if (e.id === currentPhoto.id) {
      e.type = currentPhoto.type;
    }
    return e;
  });

  setPhotosList(changedList);
  setVisible(false);
};

const changeVariant = (e) => {
  const key = e.target.id;
  const [type, id] = key.split('_');

  const current = photosList.find((photo) => {
    return photo.id === id;
  });

  const changedCurrent = { ...current };
  changedCurrent.type = type;

  setCurrentPhoto(changedCurrent);
};

const download = () => {
  const list = photosList.map((e) => {
    return ({
      type: e.type,
```

```
        file: e.file,
      });
    });

    downloadArchive(list)
      .then((res) => res.json())
      .then((json) => {
        window.open(json.url, '_blank');
        message.success('success');
      })
      .catch(() => {
        message.error('error');
      });
  };

  useEffect(() => {
    if (serverResponse) {
      createPhotoList();
    }
  }, [serverResponse]);

  return (
    <Layout>
      <Header>
        <div className="header-container">
          <Button
            type="text"
            className="back-button"
            shape="round"
            icon={<LeftOutlined />}
            size="large"
            onClick={() => navigate('/')}
          >
            Back
          </Button>
          <h1 className="header">Results</h1>
          <Button type="primary" shape="round" icon={<DownloadOutlined />} size="large"
onClick={download}>
            Download all
          </Button>
        </div>
      </Header>
```

```
<Content>
  <div className="container">
    <div className="wrap">
      <h1>Click the image to open additional settings</h1>
      <div className="preview-container">
        {
          photosList.map((item) => {
            const divKey = `c${item.id}`;
            return (
              <div className="item-container" key={divKey}>
                <img
                  className="preview-image"
                  alt="Preview"
                  src={`${item.host + item.type}/${item.file}`}
                  id={item.id}
                  key={item.key}
                  onClick={openModal}
                />
              </div>
            );
          })
        }
      </div>
    </div>
  </div>
</Content>
<Modal
  title="Edit photo"
  visible={visible}
  onOk={handleOk}
  onCancel={handleCancel}
  width="75%"
>
  <div className="modal-container">
    <div className="modal-variants-container">
      {variants.map((item) => {
        const key = `${item.type}_${item.id}`;
        return (
          <div className="variant-container" key={key}>
            <p className="variant-type">{item.type}</p>
            <img
              className="preview-variant"
```

```
                alt="Variant"
                src={`${item.host + item.type}/${item.file}`}
                id={key}
                onClick={changeVariant}
               />
              </div>
            );
           })}
         </div>
         <div className="variant-container">
          <p className="variant-type">{currentPhoto.type}</p>
          <img
           className="modal-image"
           alt="Preview"
           src={`${currentPhoto.host + currentPhoto.type}/${currentPhoto.file}`}
           id={currentPhoto.id}
          />
         </div>
        </div>
      </Modal>
    </Layout>
  );
}

export default UploadContainer;
```

Додаток Б

Лістинг серверної частини та нейронної мережі

Лістинг файлу main.py

```
import zipfile


app = Flask(__name__)
CORS(app)

appPath = 'http://localhost:5000/'


@app.route('/upload', methods=['POST'])
```

```
def upload_file():
    files_arr = []
    if not os.path.exists('./input'):
        os.mkdir('./input')

    for file in request.files.getlist('files[]'):
        file.save(os.path.join('./input', file.filename))
        improve(file.filename)
        files_arr.append(file.filename)
    return json.dumps({'host': appPath, 'data': files_arr}), 200, {'ContentType': 'application/json'}


@app.route('/input/<path:path>')
def send_input(path):
    return send_from_directory('./input', path)


@app.route('/awb/<path:path>')
def send_awb(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
    return send_from_directory('./improved/awb', filename)


@app.route('/shade/<path:path>')
def send_shade(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
    return send_from_directory('./improved/shade', filename)


@app.route('/tungsten/<path:path>')
def send_tungsten(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
    return send_from_directory('./improved/tungsten', filename)


@app.route('/fluorescent/<path:path>')
def send_flourescent(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
```

```
    return send_from_directory('./improved/fluorescent', filename)


@app.route('/daylight/<path:path>')
def send_daylight(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
    return send_from_directory('./improved/daylight', filename)


@app.route('/cloudy/<path:path>')
def send_cloudy(path):
    f_name, _ = os.path.splitext(path)
    filename = f_name + '.png'
    return send_from_directory('./improved/cloudy', filename)


@app.route('/archive', methods=['POST'])
def archive():
    photos_list = request.json

    if not os.path.exists('./zips'):
        os.mkdir('./zips')

    with zipfile.ZipFile('./zips/archive.zip', 'w') as my_zip:
        for photo in photos_list:
            if photo['type'] == 'input':
                my_zip.write('./input/' + photo['file'], photo['file'])
            else:
                f_name, _ = os.path.splitext(photo['file'])
                filename = f_name + '.png'
                my_zip.write('./improved/' + photo['type'] + '/' + filename, filename)

    my_zip.close()

    return json.dumps({'url': appPath + 'download'}), 200, {'ContentType': 'application/json'}


@app.route('/download')
def download():
    return send_from_directory('./zips', 'archive.zip', as_attachment=True)
```

```
if __name__ == "__main__":
    app.run()
```

Лістинг файлуimprove_image.py

```
import logging

import os
import torch

from PIL import Image

from PyTorch.arch.deep_wb_model import DeepWbNet
import PyTorch.utilities.utils as utls
from PyTorch.utilities.deepWB import deep_wb

import PyTorch.arch.splitNetworks as splitter
from PyTorch.arch import deep_wb_single_task



path_to_img = './input/'

model_dir = './Pytorch/models'
out_dir = './improved'

img_expansion = '.png'

S = 656


def improve(image):
    img_path = path_to_img + image
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    logging.info(f'Using device {device}')

    if not os.path.exists(out_dir):
        os.mkdir(out_dir)

    dir_list = ['/awb', '/shade', '/tungsten', '/fluorescent', '/daylight', '/cloudy']
```

```
for directory in dir_list:
    if not os.path.exists(out_dir + directory):
        os.mkdir(out_dir + directory)

if os.path.exists(os.path.join(model_dir, 'net_awb.pth')) and \
        os.path.exists(os.path.join(model_dir, 'net_t.pth')) and \
        os.path.exists(os.path.join(model_dir, 'net_s.pth')):

    net_awb = deep_wb_single_task.DeepWBnet()
    logging.info("Loading model {}".format(os.path.join(model_dir, 'net_awb.pth')))
    net_awb.to(device=device)
    net_awb.load_state_dict(torch.load(os.path.join(model_dir, 'net_awb.pth'),
                                       map_location=device))
    net_awb.eval()

    net_t = deep_wb_single_task.DeepWBnet()
    logging.info("Loading model {}".format(os.path.join(model_dir, 'net_t.pth')))
    net_t.to(device=device)
    net_t.load_state_dict(
        torch.load(os.path.join(model_dir, 'net_t.pth'), map_location=device))
    net_t.eval()

    net_s = deep_wb_single_task.DeepWBnet()
    logging.info("Loading model {}".format(os.path.join(model_dir, 'net_s.pth')))
    net_s.to(device=device)
    net_s.load_state_dict(
        torch.load(os.path.join(model_dir, 'net_s.pth'), map_location=device))
    net_s.eval()

    logging.info("Models loaded !")
elif os.path.exists(os.path.join(model_dir, 'net.pth')):
    net = DeepWbNet.deepWBNet()

    logging.info("Loading model {}".format(os.path.join(model_dir, 'net.pth')))
    net.load_state_dict(torch.load(os.path.join(model_dir, 'net.pth')))

    net_awb, net_t, net_s = splitter.split_networks(net)

    net_awb.to(device=device)
    net_awb.eval()

    net_t.to(device=device)
```

```
        net_t.eval()

        net_s.to(device=device)
        net_s.eval()
    else:
        raise Exception('Model not found!')

    logging.info("Processing image {} ...".format(img_path))
    img = Image.open(img_path)

    out_awb, out_t, out_s = deep_wb(img, net_awb=net_awb, net_s=net_s, net_t=net_t,
                                    device=device, s=S)
    out_f, out_d, out_c = utls.color_temp_interpolate(out_t, out_s)

    result_awb = utls.to_image(out_awb)
    result_t = utls.to_image(out_t)
    result_s = utls.to_image(out_s)
    result_f = utls.to_image(out_f)
    result_d = utls.to_image(out_d)
    result_c = utls.to_image(out_c)

    f_name, _ = os.path.splitext(image)

    result_awb.save(os.path.join(out_dir + '/awb', f_name + img_expansion))
    result_s.save(os.path.join(out_dir + '/shade', f_name + img_expansion))
    result_t.save(os.path.join(out_dir + '/tungsten', f_name + img_expansion))
    result_f.save(os.path.join(out_dir + '/fluorescent', f_name + img_expansion))
    result_d.save(os.path.join(out_dir + '/daylight', f_name + img_expansion))
    result_c.save(os.path.join(out_dir + '/cloudy', f_name + img_expansion))
```

Лістинг файлу utils.py

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression



def gamut_range_clipping(i):
    i[i > 1] = 1
    i[i < 0] = 0
    return i
```

```
def kernel_p(i):
    return (np.transpose((i[:, 0], i[:, 1], i[:, 2], i[:, 0] * i[:, 1], i[:, 0] * i[:, 2],
                          i[:, 1] * i[:, 2], i[:, 0] * i[:, 0], i[:, 1] * i[:, 1],
                          i[:, 2] * i[:, 2], i[:, 0] * i[:, 1] * i[:, 2],
                          np.repeat(1, np.shape(i)[0]))))


def get_mapping(image1, image2):
    image1 = np.reshape(image1, [-1, 3])
    image2 = np.reshape(image2, [-1, 3])
    m = LinearRegression().fit(kernel_p(image1), image2)
    return m


def apply_mapping(image, m):
    sz = image.shape
    image = np.reshape(image, [-1, 3])
    result = m.predict(kernel_p(image))
    result = np.reshape(result, [sz[0], sz[1], sz[2]])
    return result


def color_temp_interpolate(i_t, i_s):
    color_temperatures = {'T': 2850, 'F': 3800, 'D': 5500, 'C': 6500, 'S': 7500}
    cct1 = color_temperatures['T']
    cct2 = color_temperatures['S']

    cct1inv = 1 / cct1
    cct2inv = 1 / cct2

    temp_inv_f = 1 / color_temperatures['F']
    temp_inv_d = 1 / color_temperatures['D']
    temp_inv_c = 1 / color_temperatures['C']

    g_f = (temp_inv_f - cct2inv) / (cct1inv - cct2inv)
    g_d = (temp_inv_d - cct2inv) / (cct1inv - cct2inv)
    g_c = (temp_inv_c - cct2inv) / (cct1inv - cct2inv)

    i_f = g_f * i_t + (1 - g_f) * i_s
    i_d = g_d * i_t + (1 - g_d) * i_s
```

```
    i_c = g_c * i_t + (1 - g_c) * i_s

    return i_f, i_d, i_c


def color_temp_interpolate_w_target(i_t, i_s, target_temp):
    cct1 = 2850
    cct2 = 7500

    cct1_inv = 1 / cct1
    cct2_inv = 1 / cct2

    temp_inv_target = 1 / target_temp
    g = (temp_inv_target - cct2_inv) / (cct1_inv - cct2_inv)

    return g * i_t + (1 - g) * i_s


def to_image(image):
    return Image.fromarray((image * 255).astype(np.uint8))


def im_show(img, *arguments, color_temp=None):
    out_images_num = 0
    for _ in arguments:
        out_images_num += 1

    if out_images_num == 1 and not color_temp:
        titles = ["input", "awb"]
    elif out_images_num == 1 and color_temp:
        titles = ["input", "output (%dK)" % color_temp]
    elif out_images_num == 5:
        titles = ["input", "tungsten", "fluorescent", "daylight", "cloudy", "shade"]
    elif out_images_num == 6:
        titles = ["input", "awb", "tungsten", "fluorescent", "daylight", "cloudy", "shade"]
    else:
        raise Exception('Unexpected number of output images')

    if out_images_num < 3:
        fig, ax = plt.subplots(1, out_images_num + 1)

        ax[0].set_title(titles[0])
```

```
    ax[0].imshow(img)
    ax[0].axis('off')

    counter = 1
    for image in arguments:
        if out_images_num < 3:
            ax[counter].set_title(titles[counter])
            ax[counter].imshow(image)
            ax[counter].axis('off')
            counter = counter + 1
else:
    fig, ax = plt.subplots(2 + (out_images_num + 1) % 3, 3)
    ax[0][0].set_title(titles[0])
    ax[0][0].imshow(img)
    ax[0][0].axis('off')

    counter = 1
    for image in arguments:
        if counter == out_images_num and out_images_num == 6:
            ax[2][1].set_title(titles[counter])
            ax[2][1].imshow(image)
            ax[2][1].axis('off')
            ax[2][0].axis('off')
        else:
            ax[counter // 3][counter % 3].set_title(titles[counter])
            ax[counter // 3][counter % 3].imshow(image)
            ax[counter // 3][counter % 3].axis('off')
        counter = counter + 1

plt.xticks([]), plt.yticks([])
plt.axis('off')
plt.show()
```