

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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інженерія програмного
забезпечення мультимедійних та інформаційно-пошукових систем»**

спеціальності 121 Інженерія програмного забезпечення

**на тему: «Вебсервіс для реставрування та колоризації фотографій
на основі нейронних мереж»**

Виконала:

студентка IV курсу, групи КП-91

Тарелкіна Катерина Олексіївна _____

Керівник:

Зав. кафедри ПЗКС, д.т.н., доцент,

Сулема Євгенія Станіславівна _____

Консультант з нормоконтролю:

Доцент кафедри ПЗКС, к.т.н., доцент,

Онай Микола Володимирович _____

Рецензент:

Старший викладач кафедри ПМ,

Темнікова Олена Леонідівна _____

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студентка _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення
мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Євгенія СУЛЕМА

«__» _____ 2022 р.

ЗАВДАННЯ

на дипломний проєкт студентці

Тарелкіній Катерині Олексіївні

1. Тема проєкту «Вебсервіс для реставрування та колоризації фотографій на основі нейронних мереж», керівник проєкту Сулема Євгенія Станіславівна, д.т.н., доцент, затверджені наказом по університету від «31» травня 2023 р. №2107-с.
2. Термін подання студентом проєкту «16» червня 2023 р.
3. Вихідні дані до проєкту: див. Технічне завдання.
4. Зміст пояснювальної записки:
 - аналіз мов програмування та технологій розроблення нейронних мереж і вебсервісів, огляд архітектур нейронних мереж;
 - розроблення вебсервісу для реставрування та колоризації фотографій;
 - опис розроблених алгоритмів та підпрограм;
 - аналіз розробленого програмного продукту.
5. Перелік обов'язкового графічного матеріалу:
 - UML-діаграма класів нейронної мережі (креслення);

- UML-діаграма компонентів системи (креслення);
- блок-схема алгоритму тренування мережі (плакат);
- архітектура вебсервісу (плакат).

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «31» жовтня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Огляд існуючих рішень. Аналіз літературних джерел.	15.10.2022	
2.	Підготовка технічного завдання.	31.10.2022	
3.	Розробка нейронних мереж.	01.02.2023	
4.	Тренування та тестування нейронних мереж.	01.03.2023	
5.	Розробка серверної частини системи.	15.03.2023	
6.	Розробка клієнтської частини системи.	01.04.2023	
7.	Тестування системи.	07.04.2023	
8.	Підготовка першого розділу пояснювальної записки.	01.05.2023	
9.	Підготовка другого розділу пояснювальної записки.	07.05.2023	
10.	Підготовка третього розділу пояснювальної записки.	14.05.2023	
11.	Підготовка четвертого розділу пояснювальної записки.	19.05.2023	
12.	Підготовка інших видів документації проєкту. Підготовка графічної частини документації.	07.06.2023	

Студентка

Катерина ТАРЕЛКІНА

Керівник проєкту

Євгенія СУЛЕМА

АНОТАЦІЯ

Застосування штучного інтелекту для роботи із зображеннями сьогодні надзвичайно широке: від генерування творів мистецтва до визначення дефектів у продукції та аналізу медичних знімків. В останні роки для задач обробки зображень набула популярності архітектура GAN. Серед переваг такої мережі – ефективність та висока якість роботи. Модель GAN складається з двох підмереж – генератора та дискримінатора – які в процесі тренування «змагаються між собою». GAN лежить в основі багатьох сучасних мереж, призначених для роботи із зображеннями.

У роботі розглянуто три різні задачі обробки зображень: колоризація (надання чорно-білому фото кольору), відновлення зображень від пошкоджень (шум, наявність «подряпини», занижкі або зависокі контраст і яскравість), підвищення роздільної здатності зображень.

У цілому, перші дві задачі можна віднести до однієї категорії «переклад зображення до зображення» (англ. «image-to-image translation»), оскільки при обробці розмірність вхідного фото зберігаються. Свою ефективність для такої категорії задач показала умовна генеративна змагальна мережа Pix2pix, запропонована в роботі «Image-to-Image Translation with Conditional Adversarial Networks» (P. Isola, J. Zhu, T. Zhou та ін.). Модель Pix2pix є універсальною, тобто її архітектура не змінюється від тематики задачі.

Для збільшення роздільної здатності було реалізовано модель SRGAN із роботи «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network» (C. Ledig, L. Theis, F. Huszar та ін.). Мережа дозволяє виконувати збільшення у довільну кількість разів, проте найчастіше виконують дво-, чотири- або восьмикратне збільшення. У роботі було досліджено роботу мережі на чотирикратному збільшенні.

На основі розроблених нейронних мереж було створено завершений програмний продукт із веб-інтерфейсом користувача.

ABSTRACT

Today, the application of artificial intelligence to image processing is extremely broad: from generating works of art to identifying defects in products and analyzing medical images. In recent years, it is the GAN architecture that has gained popularity for image processing tasks. One of the advantages of such a network is efficiency and high quality of results. The GAN model consists of two subnets – a generator and a discriminator – which “compete with each other” during the training process. GANs lie at the heart of many modern image-processing networks.

In this work, three different image processing tasks are analyzed: colorizing (giving color to a black-and-white photo), restoring images from damage (noise, “scratches”, low or high contrast and brightness), and increasing the resolution of images.

Generally speaking, the first two tasks belong to the same category “image-to-image translation”, since the dimensions of the input photo are preserved during processing. The Pix2pix model has shown great effectiveness for this type of tasks. Pix2pix is a conditional generative adversarial network, proposed in the paper “Image-to-Image Translation with Conditional Adversarial Networks” (P. Isola, J. Zhu, T. Zhou and others). Pix2pix is universal – that is, its architecture does not depend on the task details.

For solving a super-resolution problem, a SRGAN model from the paper “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network” (C. Ledig, L. Theis, F. Huszar and others) has been implemented. The network allows to increase the resolution by any factor, though the most common values are two, four, and eight-factor upsampling. In this work, the network performs a four-factor upsampling.

Based on the neural networks, a complete software product with a web-interface has been developed.

ДП.045480-01-90 Вебсервіс для реставрування та колоризації фотографій
на основі нейронних мереж. Відомість проекту

[illegible]

[illegible]

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2022 р.

ВЕБСЕРВІС ДЛЯ РЕСТАВРУВАННЯ ТА КОЛОРИЗАЦІЇ
ФОТОГРАФІЙ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Технічне завдання

ДП.045480-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Євгенія СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Катерина ТАРЕЛКІНА

ЗМІСТ

1. Найменування та галузь застосування	3
2. Підстава для розроблення	3
3. Призначення розробки	3
4. Вимоги до програмного продукту	3
5. Вимоги до проєктної документації	4
6. Етапи проєктування	5
7. Порядок тестування розробки	5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебсервіс для реставрування та колоризації фотографій на основі нейронних мереж.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматичної обробки фотографій, а саме колоризації та реставрування. Програма призначена для використання у персональних цілях.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Основними функціями програми є:

- 1) колоризація фотографій (надання чорно-білому зображенню кольору);
- 2) відновлення фотографій від пошкоджень (низький контраст, зависока або занижка яскравість, шум, «тріщина» або «розрив»);
- 3) покращення якості фотографій (збільшення роздільної здатності зображення у чотири рази).

Основні характеристики програми:

- 1) програма повинна працювати із фотографіями людей, природи, тварин, архітектури, інтер'єру, побутових предметів;
- 2) програма повинна бути доступна у вигляді вебсервісу;

- 3) для користування програмою не вимагається реєстрація та авторизація;
- 4) обробка зображень здійснюється на стороні сервера;
- 5) система має надавати веб-інтерфейс користувача;
- 6) зображення, надані користувачем, повинні перебувати на сервері тільки під час обробки; зображення не повинні бути збережені на стороні сервера.

Додаткові вимоги:

- 1) вебсайт проєкту повинен бути доступний кількома мовами (англійська, українська, французька);
- 2) час обробки одного зображення системою не повинен перевищувати однієї хвилини.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

Під час виконання проєкту повинна бути розроблена наступна документація:

- 1) пояснювальна записка;
- 2) програма та методика тестування;
- 3) керівництво користувача;
- 4) креслення:
 - «Організація програмного коду нейронної мережі. UML-діаграма класів»;
 - «Архітектура програми. UML-діаграма компонентів».

6. ЕТАПИ ПРОЄКТУВАННЯ

Огляд існуючих рішень. Аналіз літературних джерел	15.10.2022
Підготовка технічного завдання	31.10.2022
Розробка нейронних мереж	01.02.2023
Тренування та тестування нейронних мереж	01.03.2023
Розробка серверної частини системи	15.03.2023
Розробка клієнтської частини системи	01.04.2023
Тестування системи	07.04.2023
Підготовка першого розділу пояснювальної записки	01.05.2023
Підготовка другого розділу пояснювальної записки	07.05.2023
Підготовка третього розділу пояснювальної записки	14.05.2023
Підготовка четвертого розділу пояснювальної записки	19.05.2023
Підготовка інших видів документації проєкту	06.06.2023
Підготовка графічної частини документації	07.06.2023

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2023 р.

**ВЕБСЕРВІС ДЛЯ РЕСТАВРУВАННЯ ТА КОЛОРИЗАЦІЇ
ФОТОГРАФІЙ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ**

Пояснювальна записка

ДП.045480-03-81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Євгенія СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Катерина ТАРЕЛКІНА

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ	3
ВСТУП.....	8
1. АНАЛІЗ МОВ ПРОГРАМУВАННЯ ТА ТЕХНОЛОГІЙ ДЛЯ РОБОТИ ІЗ ЗОБРАЖЕННЯМИ	11
1.1. Мови програмування.....	11
1.2. Огляд нейронних мереж для роботи із зображеннями.....	13
1.3. Огляд продуктів-аналогів	22
2. РОЗРОБЛЕННЯ ВЕБСЕРВІСУ ДЛЯ КОЛОРИЗАЦІЇ ТА ВІДНОВЛЕННЯ ФОТОГРАФІЙ ВІД ПОШКОДЖЕНЬ.....	24
2.1. Аналіз вимог до функціональності вебсервісу	24
2.2. Архітектура програми	30
3. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ.....	35
3.1. Нейронні мережі для роботи з зображеннями	35
3.2. Вебклієнт	43
3.3. Вебсервер.....	45
4. АНАЛІЗ РОЗРОБЛЕНОГО ВЕБСЕРВІСУ ДЛЯ КОЛОРИЗАЦІЇ ТА ВІДНОВЛЕННЯ ФОТОГРАФІЙ ВІД ПОШКОДЖЕНЬ.....	48
4.1. Особливості реалізації сервісу	48
4.2. Тренування нейронних мереж та тестування сервісу	53
4.3. Порівняння розробки з існуючими аналогами	56
4.4. Рекомендації щодо подальшого вдосконалення.....	60
ВИСНОВКИ	63
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ.....	66
ДОДАТКИ	72

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

cGAN (англ. *conditional generative adversarial network*) – умовна генеративна змагальна мережа. Підтип GAN, головною особливістю є те, що генератор та дискримінатор мають додатковий ввід (наприклад, для задачі колоризації таким вводом є чорно-біле зображення).

CNN (англ. *convolutional neural network*) – згорткова нейронна мережа. Клас нейронних мереж для обробки зображень.

GAN (англ. *generative adversarial network*) – генеративна змагальна мережа, складається з двох підмереж: генератора та дискримінатора. Процес навчання полягає в «змаганні» двох підмереж.

Ground truth – фундаментальна істина, в алгоритмах для нейронних мереж це оригінальні дані, з якими порівнюється результат роботи мережі.

Pix2pix – умовна генеративна змагальна мережа, запропонована в роботі «Image-to-Image Translation with Conditional Adversarial Networks» [1]. Застосовується до задач типу «image-to-image translation».

Single-image super-resolution, SISR – підтип SR, що відтворює одне високоякісне зображення на основі одного зображення з низькою роздільною здатністю. На противагу SISR існує *multi-image super-resolution*, що відтворює високоякісне зображення на основі кількох фотографій одного й того ж об'єкта. *Multi-image super-resolution* здатний надавати більш точні результати, але SISR є більш практичним, а тому частіше вживаним [2].

SRGAN – генеративна змагальна мережа, запропонована в роботі «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network» [3], призначена для підвищення роздільної якості зображення.

Super resolution, SR – збільшення роздільної здатності зображення за допомогою штучного інтелекту.

U-Net – тип згорткової нейронної мережі, оригінально розроблений для сегментації біомедичних зображень в роботі «U-Net: Convolutional Networks for Biomedical Image Segmentation» [4]. Складається з етапів кодування (збільшення кількості каналів зображення з одночасним зменшенням розмірності) та декодування (зменшення кількості каналів та збільшення розмірності).

Агрегування (англ. *pooling*) – алгоритм зменшення роздільної здатності зображення за допомогою матричного фільтру для виділення високорівневих характеристик зображення.

Бінарна перехресна ентропія (англ. *binary cross-entropy*) – функція втрат, формула (1) якої наведена нижче.

$$H(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})], \quad (1)$$

де y – істинне значення, $\hat{y}$ – значення, отримане з мережі.

Функція застосовується для задач бінарної класифікації.

Випрямлений лінійний вузол (англ. *ReLU, rectified linear unit*) – одна з поширених функцій активації, формула (2) якої наведена нижче.

$$\begin{aligned} f(x) &= 0, x < 0, \\ f(x) &= x, x \geq 0. \end{aligned} \quad (2)$$

Згортка «вгору» (англ. *Up convolution*) – матричний фільтр; операція над зображенням, результатом якої є зображення більшої розмірності та з меншою кількістю каналів.

Згортка (англ. *Convolution*) – матричний фільтр; операція над зображенням, результатом якої є зображення меншої розмірності та з більшою кількістю каналів. Допомагає виділити високорівневі характеристики зображення.

Індекс структурної подібності (англ. *SSIM, structural similarity index measure*) – індекс для визначення подібності між двома зображеннями. На відміну від попиксельного порівняння, надає оцінку структурній схожості зображень. Значення знаходяться в межах $[-1, 1]$.

Максимізаційне агрегування (англ. *max pooling*) – алгоритм агрегування, у якому за вихідне значення беруть максимальне серед значень, що входять до області, покритої фільтром.

Механізм зворотного поширення помилки (англ. *backpropagation*) – алгоритм оновлення вагів мережі для мінімізації функції втрат.

Нещільний ReLU (англ. *ReLU, leaky rectified linear unit, leaky ReLU*) – одна з поширених функцій активації, формула (3) якої наведена нижче.

$$\begin{aligned}f(x) &= \alpha x, x < 0, \\f(x) &= x, x \geq 0, \\ \alpha &\in R, 0 < \alpha \ll 1.\end{aligned}\tag{3}$$

Значення α стає впродовж тренування.

Операція змішування пікселів (англ. *Pixel Shuffle*) – операція, що перетворює зображення розмірності $H \times W \times C \cdot scale^2$ у зображення розмірності $scale \cdot H \times scale \cdot W \times C$, де $scale$ – заданий параметр (масштаб збільшення). Оригінально запропонована у роботі «Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network» [5].

Пакетна нормалізація (англ. *batch normalization*) – нормалізація вихідного сигналу нейрона з використанням середнього значення та дисперсії (обчислених для одного пакету даних) [6].

Параметричний ReLU (англ. *ReLU, parametric rectified linear unit, parametric ReLU*) – одна з поширених функцій активації, формула (4) якої наведена нижче.

$$\begin{aligned}f(x) &= \alpha x, x < 0, \alpha \in R, \\f(x) &= x, x \geq 0.\end{aligned}\tag{4}$$

Мережа змінює значення α у процесі навчання для оптимізації тренування.

Переклад зображення до зображення (англ. *image-to-image translation*) – клас задач машинного навчання, у яких потрібно здійснити

перехід від одного зображення до іншого (наприклад, денний пейзаж – нічний пейзаж, чорно-біле зображення – кольорове зображення, рисунок – фотографія тощо). Розмірність вхідного та вихідного зображень однакова.

Перехресна ентропія (англ. *cross-entropy*) – функція втрат, формула (5) якої наведена нижче. Формуло взято з джерела [7].

$$H(p, q) = -\sum_{i=1}^n p(x_i) \log q(x_i), \quad (5)$$

де x – вектор значень з кількістю елементів n , p , q – ймовірнісні розподіли (де p – істинний розподіл, q – передбачений мережею).

Поля (англ. *padding*) – кількість рядків та стовпців, що додаються з боків зображення для того, щоби збільшити його розмірність й тим самим уникнути втрати крайніх пікселів при застосуванні фільтру [8]. Найчастіше «доповнені» пікселі мають значення 0 [8].

Середнє агрегування (англ. *average pooling*) – алгоритм агрегування, у якому за вихідне значення беруть середнє арифметичне від значень, що входять до області, покритої фільтром.

Сигмоїда – одна з поширених функцій активації, формула (6) якої наведена нижче.

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (6)$$

Фільтр (матричний фільтр) – квадратна матриця, що застосовується до матриці зображення. Програма проходить фільтром по матриці зображення й за певними правилами вираховує значення вихідної матриці. *Stride* – крок, з яким «рухається» фільтр. Найбільш поширені значення кроку – 1, 2.

Функція активації – функція, що застосовується до вихідного значення нейрону. Підвищує ефективність мережі завдяки додаванню нелінійності. У випадку лінійних функцій активації (наприклад, $f(x) = x$), комбінація цих функцій є також лінійною функцією, тому додавання рівнів у таку мережу не підвищує її складність і мережа лишається погано

пристосованою для вирішення складних задач [9]. Відтак, найчастіше обирають нелінійні функції активації.

Функція втрат (англ. loss, loss function) – функція для оцінки роботи мережі. Обчислює різницю між справжнім зображенням та зображенням, отриманим з мережі. Задачею тренування мережі є налаштувати ваги таким чином, щоби мінімізувати функцію втрат.

Щільний рівень (англ. dense layer, fully-connected layer) – рівень мережі, кожен нейрон якого пов'язаний з кожним нейроном попереднього рівня. Найбільш поширений тип рівня у нейронних мережах [10].

ВСТУП

Якість старих фотографій не порівняти із сучасними знімками: перші часто мають низьку роздільну здатність й безліч недоліків: шум, фізичні пошкодження, невдалі яскравість та контраст тощо. Крім того, значна кількість «архівних» світлин чорно-біла. Одним із рішень є віддати такі фото фахівцям з реставрування: їхня робота, однак, є трудомісткою та потребує багато часу. Сьогодні альтернативним рішенням є використання нейронних мереж, що працюють із графічними даними.

Наразі застосування мереж для роботи з зображеннями надзвичайно широке: від автоматичної обробки зображень і генерування витворів мистецтва на основі тексту до систем розпізнавання обличчя, контролю якості продукції та програм обробки медичних знімків.

Очевидно, деякі задачі можливо вирішити без допомоги штучного інтелекту. Наприклад, прибрати шум із зображення або підвищити різкість можна за допомогою звичайних матричних фільтрів. Про нейронні мережі – набагато більш потужний інструмент. Наприклад, розмите зображення неодмінно містить втрачені деталі, які матричний фільтр встановити не вможі. Нейронна ж мережа здатна «домальовувати» втрачені дані. Крім того, мережа не лише підвищує різкість, а й покращує якість фото в цілому.

Зауважмо також, що деякі завдання взагалі неможливо реалізувати через класичні алгоритми – як-от, наприклад, колоризація або анімування зображень. Наприклад, в роботі «Time-Travel Rephotography» [11] було запропоновано модель StyleGAN2, призначену для покращення старих світлин. Мережа не просто колоризує та прибирає недоліки з фотографії: відновлені знімки не відрізнити від фото, знятих на сучасний смартфон. Це досягнуто завдяки використанню спеціальних ефектів: наприклад, у минулому фотоплівка була слабо чутливою до червоного світла, тому на портреті людина часто мала більш виражені зморшки й затемнені ділянки рота і щок [11]. Мережа StyleGAN2 прибирає недоліки, спричинені

апаратними обмеженнями, а також вирівнює тон зображення, підвищує роздільну здатність та додає колір.

Результати роботи штучного інтелекту вражають. Наприклад, інструмент Deep Nostalgia від компанії MyHeritage дозволяє анімувати зображення людей (жести та вирази обличчя користувач обирає самостійно з запропонованих). Так, за перші 48 годин після запуску сервісу було анімовано більше 1 мільйона фотографій [12].

Усе це лише доводить актуальність та потужність нейронних мереж – сьогодні вони є невід’ємною частиною нашого життя і, в той же час, перспективною галуззю досліджень.

У цій роботі було досліджено три задачі автоматичної обробки зображень: колоризація, відновлення від пошкоджень та покращення роздільної здатності зображень.

Перші два завдання, по суті, можна віднести до однієї й тієї ж категорії задач «переклад зображення до зображення» (англ. *image-to-image translation*). Свою ефективність для таких задач показала модель cGAN (умовна генеративна змагальна мережа). Модель cGAN було запропоновано в роботі «Image-to-Image Translation with Conditional Adversarial Networks» [1] й названо Pix2pix. Модель Pix2pix є універсальною, тобто її можна застосовувати до різних задач типу *image-to-image translation* – архітектура моделі абсолютно не залежить від деталей задачі. Так, у цій роботі було досліджено роботу моделі для колоризації фотографій та відновлення їх від пошкоджень. Програмний код базовано на реалізації з офіційного репозиторію Tensorflow [13].

Для задачі підвищення якості зображень було використано модель SRGAN із роботи «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network» [3], програмний код базовано на реалізації зі статті «SRGAN: Super Resolution Generative Adversarial Networks» [14].

Для усіх трьох мереж було проведено вичерпне тренування й оцінено їхню ефективність. На основі моделей було створено завершений

вебзастосунок, що надає змогу протестувати роботу мереж в інтерактивному режимі.

Крім практичної, важливим компонентом роботи є теоретична частина. Було розглянуто принципи роботи нейронних мережах, основні алгоритми та поняття, що застосовуються при їхній розробці, а також деякі з поширених архітектур мереж для роботи з зображеннями.

1. АНАЛІЗ МОВ ПРОГРАМУВАННЯ ТА ТЕХНОЛОГІЙ ДЛЯ РОБОТИ ІЗ ЗОБРАЖЕННЯМИ

1.1. Мови програмування

Нижче наведено опис та обґрунтування обраних мов програмування та інструментів розробки.

1.1.1. Мова програмування *Python*

Python – динамічно типізована мова програмування широкого призначення. Найбільшу популярність Python здобув у галузі обробки даних і штучного інтелекту – передусім завдяки величезній кількості бібліотек, призначених для роботи з даними. Крім того, Python лежить в основі фреймворку машинного навчання Tensorflow, який застосовують для різних задач штучного інтелекту.

Для веб-серверу було використано Python-бібліотеку FastAPI. Головними її перевагами є швидкість, простота в освоєнні та підтримка REST архітектури [15].

Окрім цього, у роботі було використано бібліотеки PIL (Python Imaging Library) та scikit-image, що надають широкий набір функцій для роботи із зображеннями.

1.1.2. Фреймворк *Tensorflow*

Tensorflow – популярний фреймворк для машинного навчання. Tensorflow надає можливість працювати з моделями на рівні абстракцій, що значно підвищує ефективність та простоту розробки [16]. Крім цього, фреймворк надає датасети для різних типів задач, в тому числі для обробки зображень. Фреймворк має велику спільноту розробників та вичерпну документацію, що значно спрощує його вивчення та використання.

1.1.3. Мова програмування TypeScript та фреймворк Angular

Для створення клієнтської частини було використано фреймворк Angular та мову TypeScript.

TypeScript – строго типізована мова, побудована на основі JavaScript. У середньому, код, написаний мовою TypeScript, містить на 38% менше помилок, ніж код мовою JavaScript [17]. JavaScript – це прототипна однопоточна динамічна об’єктно-орієнтовна мова програмування, що підтримує парадигми імперативного та декларативного (функціонального) програмування [18]. JavaScript відома передусім як скриптова мова для використання на веб-сторінках.

Фреймворк Angular надає комплексне рішення для побудови масштабованих веб-застосунків, а також для їхнього тестування [19]. Разом із Angular було використано бібліотеку RxJS – найбільш поширена серед фронтенд-розробників, бібліотека спрощує роботу з асинхронним та подійно-орієнтовним кодом завдяки використанню патерну Observable (Спостерігач) [20].

Гнучкість Angular дозволяє будувати як масштабні enterprise-застосунки, так і компактні вебсайти для персональних проєктів.

Нижче наведено порівняння Angular з іншими фреймворками та бібліотеками для фронтенд-застосунків.

Таблиця 1.1

Порівняльна характеристика фреймворків / бібліотек для фронтенд-частини застосунку

Фрейм-ворк або бібліотека	Переваги	Недоліки
Angular	Є досвід розробки. Використання TypeScript.	Наявність різноманітних сутностей окрім компонентів,

Фрейм-ворк або бібліотека	Переваги	Недоліки
Angular	Чітка структура проєкту. Повноцінний фреймворк, використання додаткових модулів здебільшого не потрібне. Вбудовані інструменти для написання та запуску unit-тестів	через що вважається складнішим в освоєнні. У деяких випадках нижча продуктивність порівняно з іншими фреймворками
React	Є досвід розробки. Невеликий розмір бібліотеки. Гнучкість у розробці. Швидкість розробки. Здебільшого інтуїтивно зрозуміла через використання JSX, принципів функціонального програмування. Швидкодія застосунку	Потрібні додаткові модулі, як-от для роутингу. Може знадобитися ручне налаштування збірки проєкту за допомогою Webpack, Babel. Немає чіткої структури проєкту. Шаблони змішано з логікою (JSX)
Vue	Невеликий розмір. Масштабованість. Простота в освоєнні	Немає досвіду розробки. Надає дуже високу гнучкість у розробці, через це краще підходить для використання досвідченими розробниками. Недостатньо велика спільнота порівняно з Angular і React, оскільки фреймворк відносно новий

1.2. Огляд нейронних мереж для роботи із зображеннями

Значну ефективність для роботи із зображеннями показали генеративні змагальні мережі (англ. GAN, *generative adversarial network*).

Принцип роботи таких мереж полягає в наступному. Мережа містить дві моделі-підмережі: генератор та дискримінатор.

Задача генератора полягає в тому, щоби згенерувати максимально правдоподібне (близьке до *ground truth*) зображення. Дискримінатор отримує згенеровані та справжні зображення й повинен визначити, які з зображень справжні, а які – результат роботи генератора. На основі рішення дискримінатора через механізм зворотного поширення помилки генератор оновлює свої ваги. Таким чином, тренування мережі проходить у вигляді змагання й у цілому не потребує втручання людини (зауважимо, що це – ідеальний випадок, на практиці ж мережі такого типу достатньо складно правильно відтрениувати й процес часто потребує людського втручання).

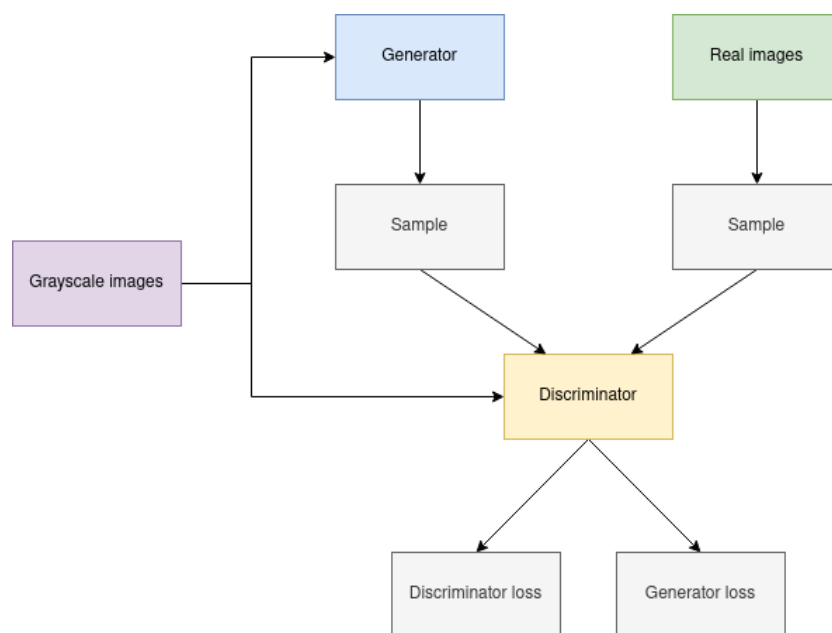


Рис. 1.1. Схема GAN (рисунок створено на основі схеми із [21])

Проілюструймо роботу мережі на основі задачі колоризації фотографій (рис. 1.1). У цьому випадку, генератор і дискримінатор приймають чорно-біле зображення як ввід. Генератор відтворює кольорове зображення. Дискримінатор, окрім чорно-білого, отримує як ввід кольорове зображення від генератора та оригінальне кольорове зображення. Його задачею є вирішити, яке з кольорових зображень – справжнє. За допомогою механізму зворотного поширення помилки мережа оновлює ваги генератора та дискримінатора, і з кожним кроком генератор створює

все більш якісні кольорові зображення, а дискримінатор усе краще розпізнає справжні та згенеровані фото.

1.2.1. Модель Pix2pix

Запропонована у 2018 р. в роботі «Image-to-Image Translation with Conditional Adversarial Networks» [1], модель Pix2pix показала високу ефективність у розв'язанні задач типу «переклад зображення до зображення» (*image-to-image translation*).

Pix2pix є генеративною змагальною мережею. Розглянемо деталі її архітектури.

Генератор побудований у вигляді U-Net. U-Net – згорткова нейронна мережа, оригінально розроблена для сегментації біомедичних зображень [4]. Перший етап обробки U-Net – кодування – полягає в тому, що мережа кодує («згортає») вихідне зображення. Зображення проходить через ряд таких операцій: згортка з фільтром $3 \times 3 \rightarrow$ випрямлений лінійний вузол (англ. ReLU) $\rightarrow$ максимізаційне агрегування із фільтром 2×2 . Після кожного такого кроку кількість каналів зображення збільшується вдвічі. Наприклад, якщо вхідне зображення має розмірність $572^2 \times 1$, то наприкінці етапу кодування його розмірність буде складати $30^2 \times 1024$. Таке перетворення дозволяє виділити високорівневі характеристики зображення [22].

Другий етап (декодування) складається з ряду таких кроків: згортка «вгору» (англ. up-convolution) із фільтром $2 \times 2 \rightarrow$ конкатенація з відповідним рівнем кодування $\rightarrow$ дві операції згортки з фільтром $3 \times 3 \rightarrow$ ReLU. У кінці до зображення застосовують операцію згортки з фільтром 1×1 для отримання потрібної кількості каналів (наприклад, три канали для RGB-зображення).

Особливістю U-Net є застосування конкатенації до рівнів кодування та декодування, а саме між кожним рівнем i та рівнем $n - i$, де n – загальне

число рівнів [1]. Такий підхід значно підвищує ефективність мережі [22].
Схема архітектури U-Net зображена на рис. 1.2 [4].

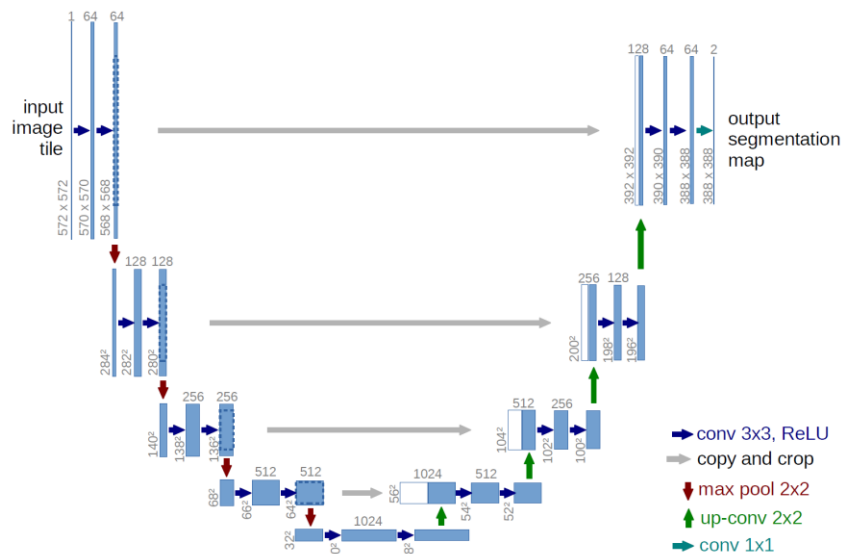


Рис. 1.2. Архітектура U-Net [4]

Дискримінатор у моделі Pix2pix є згортковою нейронною мережею, побудованою за архітектурою PatchGAN (запропонована в оригінальній роботі). Кожен блок мережі складається з таких операцій: згортка з фільтром $4 \times 4 \rightarrow$ пакетна нормалізація (англ. batch normalization) $\rightarrow$ нещільний ReLU (англ. leaky ReLU).

Відмінність PatchGAN від звичайної згорткової мережі полягає в тому, що PatchGAN працює з фрагментами («шматочками») зображення й намагається оцінити правдивість лише одного фрагменту, а не цілого зображення. Автори оригінальної роботи виявили, що найкраще використовувати фрагмент розмірністю 70×70 . У цьому випадку, якщо вхідне зображення має розмірність $256^2 \times 3$, то результатом роботи дискримінатора буде матриця 30×30 , кожен елемент якої міститиме булеве значення «правдоподібності» (1 або 0) для відповідного фрагменту вхідного зображення [22].

Розглянемо функції втрат генератора та дискримінатора [1]. Нижче наведено формулу (7) функції втрат дискримінатора:

$$L_D = L_{BCE}(J_{m,n}, y_D) + L_{BCE}(0_{m,n}, \hat{y}_D), \quad (7)$$

де y_D – вивід дискримінатора для оригінального зображення (матриця булевих значень розміром $m \times n$), $\hat{y}_D$ – вивід дискримінатора для згенерованого зображення, L_{BCE} – бінарна перехресна ентропія, $J_{m,n}$ – матриця одиниць розміром $m \times n$, $0_{m,n}$ – матриця нулів розміром $m \times n$.

Функцію втрат генератора записано у формулі (8):

$$L_G = L_{BCE}(J, \hat{y}_D) + L \cdot L_{L1}(y, \hat{y}), \quad (8)$$

де $\hat{y}_D$ – вивід дискримінатора для згенерованого зображення, J – матриця одиниць відповідної розмірності, L_{BCE} – бінарна перехресна ентропія, y – оригінальне зображення, $\hat{y}$ – згенероване зображення, L – раціональне число (авторами оригінальної роботи запропоновано встановлювати $L = 100$ [13]).

Функція L_{L1} має формулу (9), наведену нижче:

$$L_{L1}(y, \hat{y}) = \sum_{i=1}^n \sum_{j=1}^m |y_{i,j} - \hat{y}_{i,j}|, \quad (9)$$

де $m \times n$ – розмірність зображень.

Зауважмо на використання двох функцій для обчислення втрат генератора у формулі (8). Таке поєднання значно підвищує ефективність мережі: функція L_{BCE} націлена на генерування таких зображень, які не може розрізнити від оригінальних дискримінатор, а L_{L1} – на генерування в цілому правдоподібних зображень (тобто близьких до *ground truth*) [1].

Зауважимо, що Pix2pix належить до умовних генеративних змагальних мереж, оскільки генератор та дискримінатор мають додатковий ввід. Наприклад, для задачі колоризації таким вводом є чорно-біле зображення.

Як було зазначено вище, модель Pix2pix може бути успішно застосована до різних задач типу «переклад зображення до зображення». На рис. 1.3 з оригінальної роботи показано кілька прикладів застосування мережі.

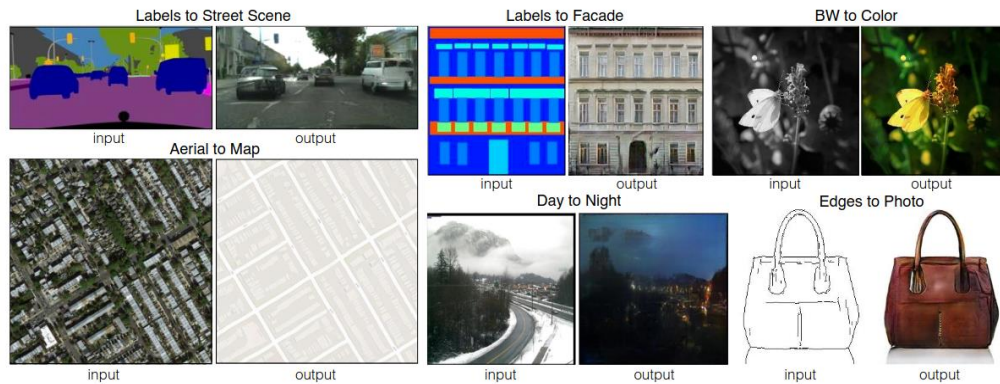


Рис. 1.3. Приклади застосування моделі Pix2pix [1]

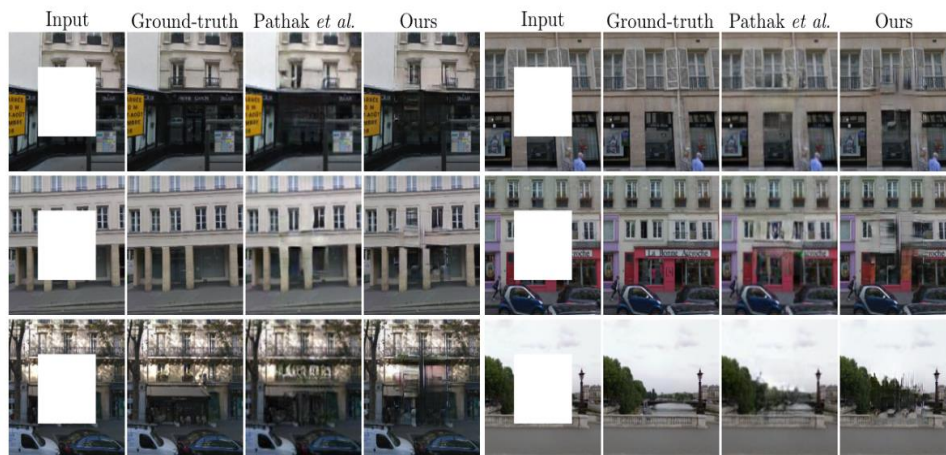


Рис. 1.4. Приклади застосування моделі Pix2pix для доповнення зображень [1]. Датасет, використаний для тренування – Paris StreetView [23]

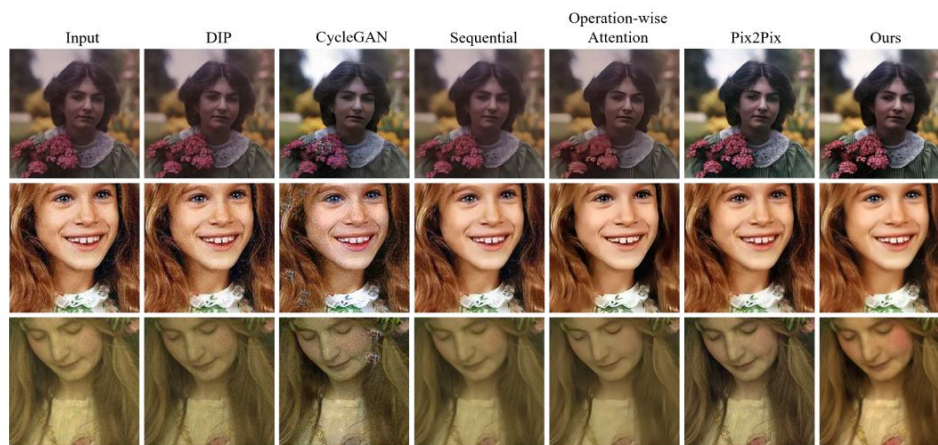


Рис. 1.5. Реставрація фотографій за допомогою різних нейронних мереж. Зліва – вхідні фотографії. Крайній стовпець справа – результати моделі, запропонованої в оригінальній роботі («Bringing Old Photos Back to Life»), другий стовпець справа – результати Pix2pix [24]

1.2.2. Модель SRGAN

Підвищення роздільної здатності зображень є одним із сучасних викликів серед задач машинного навчання. У минулому, до цієї проблеми широко застосовували згорткові нейронні мережі, проте вони були не завжди здатні «відтворити тонкі деталі та часто генерували розмиті зображення» [14]. На зміну згортковим мережам була запропонована архітектура SRGAN.

Розглянемо особливості такої мережі [3]. SRGAN, як і Pix2pix, є генеративною змагальною мережею, тобто складається з двох підмереж – генератора та дискримінатора – які в процесі тренування «змагаються» між собою.

З рис. 1.6 бачимо, що в основі генератора лежить B залишкових блоків, кожен з яких включає такі операції: згортка з фільтром $\rightarrow$ пакетна нормалізація $\rightarrow$ параметричний ReLU $\rightarrow$ згортка з фільтром $3 \times 3 \rightarrow$ пакетна нормалізація $\rightarrow$ поелементна сума із вхідним зображенням блоку.

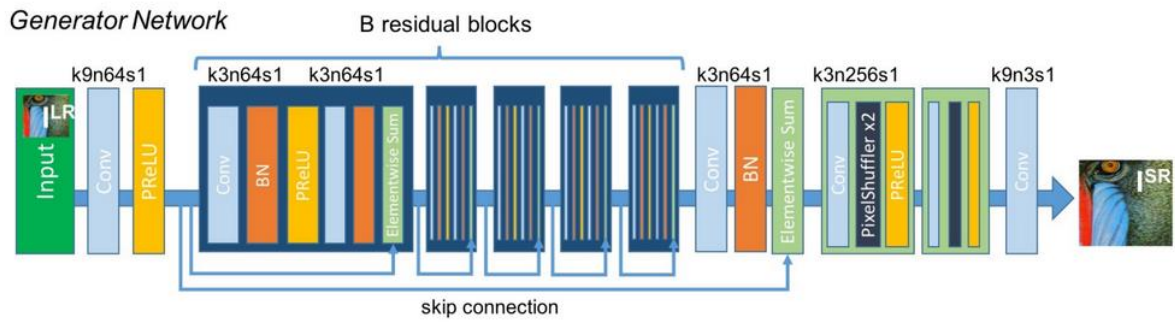


Рис. 1.6. Архітектура SRGAN. Генератор [3]

Очевидно, що результатом роботи генератора має бути зображення більшої розмірності, тому в цій мережі не використовують операцій максимізаційного агрегування, притаманного, наприклад, U-Net [14].

Зауважимо на використання параметричного ReLU: для задачі перетворення зображення з низькою роздільною здатністю у високу це – одна з найбільш ефективних функцій активації [14].

Для власне збільшення роздільної здатності зображення модель використовує операцію змішування пікселів (англ. Pixel Shuffle). Оригінально запропонована у роботі «Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network» [9], така операція перетворює зображення розмірності $H \times W \times C \cdot scale^2$ у зображення розмірності $scale \cdot H \times scale \cdot W \times C$, де $scale$ – заданий параметр (масштаб збільшення).

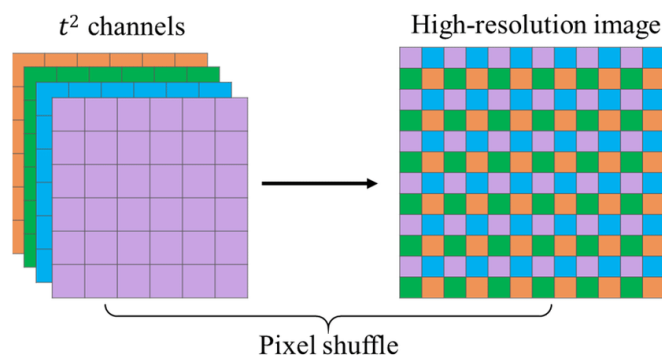


Рис. 1.7. Операція змішування пікселів [25]

Таким чином, змішування пікселів збільшує розмірність зображення, водночас скорочуючи кількість каналів. Приклад роботи цього перетворення наведено на рис. 1.7 [25].

Дискримінатор у моделі SRGAN складається з серії блоків із такими операціями: згортка з фільтром $3 \times 3 \rightarrow$ пакетна нормалізація $\rightarrow$ нещільний ReLU.

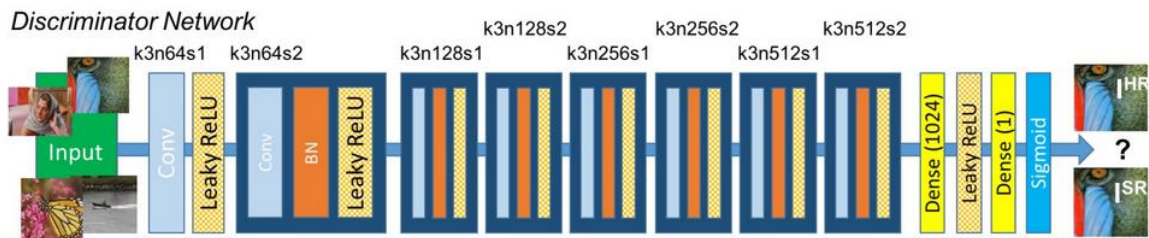


Рис. 1.8. Архітектура SRGAN. Дискримінатор [3]

Наприкінці додають два повноз'єднувальні шари з нещільним ReLU та функцію активації сигмоїду.

Задачею генератора є відтворити високоякісне зображення на основі вхідного, а задачею дискримінатора – визначити, яке з зображень оригінально мало високу роздільну здатність, а яке було покращене генератором.

Особливістю SRGAN є перцептивна функція втрат (англ. perceptual loss function), що обчислюється за формулою (10). Формуло взято із джерела [3].

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR}, \quad (10)$$

де l_X^{SR} – змістовна функція втрат (англ. content loss), l_{Gen}^{SR} – змагальна функція втрат (англ. adversarial loss).

Часто в ролі змістовної функції втрат беруть середню квадратичну похибку, яку формульно можна записати як (11). Формуло взято із джерела [3].

$$I_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} \left(I_{x,y}^{HR} - G_{\theta_G}(I_{x,y}^{LR}) \right)^2, \quad (11)$$

де $G_{\theta_G}(I^{LR})$ – результат роботи генератора (згенероване зображення з високою роздільною здатністю), W, H – ширина та висота зображення з низькою роздільною здатністю, r – фактор збільшення роздільної здатності, I^{HR} – оригінальне зображення з високою роздільною здатністю, $I_{x,y}$ – елемент зображення на перетині x -го рядка та y -го стовпця (піксель зображення).

Однак автори оригінальної роботи виявили, що ефективнішою є функція втрат, обчислена на основі нейронної мережі VGG19, запропонованої в роботі «Very Deep Convolutional Networks for Large-Scale Image Recognition» [26]. Наведемо формулу (12) такої функції втрат (узято з джерела [3]).

$$l_{VGG/i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j} - \phi_{i,j} \left(G_{\theta_G}(I^{LR}) \right)_{x,y} \right)^2, \quad (12)$$

де результат $\phi_{i,j}$ – зображення, отримане з VGG19 після j -го рівня згортки та функції активації перед i -м рівнем максимізаційного об'єднання, $W_{i,j}H_{i,j}$ – ширина та висота такого зображення. Мережа VGG19 вважається наданою.

У ролі змагальної функції втрат у формулі (10) використовують бінарну перехресну ентропію, де «істинним» зображенням є матриця одиниць такої ж розмірності, як і згенероване зображення [14].

1.3. Огляд продуктів-аналогів

Окрім дослідження роботи нейронних мереж, метою цієї праці є створити завершений програмний продукт, що надаватиме зручний та сучасний інтерфейс користувача та буде легким у використанні.

На сучасному ринку існує безліч подібних продуктів з найрізноманітнішими функціональними можливостями та користувацькими характеристиками та призначеними для різноманітних платформ: вебсервіси, мобільні та десктопні застосунки.

У таблиці нижче зведено основні риси онлайн-аналогів. З прикладами їхньої роботи можна ознайомитися в додатку 4.

Таблиця 1.2

Основні характеристики продуктів-аналогів

	Hotpot.ai [27]	img2go [28]	Photomyne [29]
Доля ринку	Реставрування, колоризація та інші сервіси AI-обробки фотографій	Колоризація та покращення якості фотографій (AI-сервіси). «Ручне» редагування зображень (застосування певних фільтрів тощо)	Реставрування, колоризація та інші сервіси AI-обробки фотографій
Конкурентні переваги	Швидкість роботи. Наявність додаткового налаштування для покращення роботи програми (наприклад, користувач може вказати, чи є на зображенні подряпина чи зазначити фактор колоризації)	Продукт надає дві різні моделі для колоризації (одна для зображень людей і природи, інша загальна). Продукт надає значну кількість інших послуг, як-от фоторедактор і додавання водяного знаку	Наявність мобільного застосунку. Можливість співпраці для бізнес-клієнтів
Схожі характеристики	Програма реставрує та розфарбовує чорно-білі фотографії та 'сепію'. Програма в основному працює з зображеннями людей	Програма розфарбовує чорно-білі фотографії та 'сепію'; дозволяє покращувати якість фотографій	Програма розфарбовує чорно-білі фотографії та 'сепію'; покращувати якість та реставрувати зображення.

2. РОЗРОБЛЕННЯ ВЕБСЕРВІСУ ДЛЯ КОЛОРИЗАЦІЇ ТА ВІДНОВЛЕННЯ ФОТОГРАФІЙ ВІД ПОШКОДЖЕНЬ

2.1. Аналіз вимог до функціональності вебсервісу

Нижче наведено детальний перелік вимог до програми.

2.1.1. Функції вироба

Основними функціями програми є:

- колоризація фотографій (надання чорно-білому зображенню кольору);
- відновлення фотографій від пошкоджень (низький контраст, зависока або занижка яскравість, шум, «тріщина» або «розрив»);
- покращення якості фотографій (збільшення роздільної здатності зображення).

Програма повинна працювати із фотографіями людей, природи, тварин, архітектури, інтер'єру, побутових предметів.

2.1.2. Характеристики користувачів

Програма повинна бути доступна у вигляді вебсервісу. Для користування програмою не вимагається реєстрація та авторизація.

2.1.3. Обмеження

Нижче наведено деякі обмеження, що мають бути враховані при розробці системи.

Обмеження апаратного забезпечення

Клієнтська частина програми не потребує спеціальних апаратних характеристик зі сторони користувача. Обробка зображень здійснюється на стороні сервера.

Інтерфейси користувача

Система має надавати веб-інтерфейс користувача.

Вимоги до безпеки

Зображення, надані користувачем, повинні перебувати на сервері тільки під час обробки. Зображення не повинні бути збережені на стороні сервера (наприклад, в базі даних або кеші).

2.1.4. Допущення і залежності

Клієнтська частина повинна коректно працювати в браузерях, список яких наведено в SI-2.

2.1.5. Специфічні вимоги

Далі перераховано детальні вимоги до інтерфейсу користувача та інтерфейсу зв'язку.

Інтерфейси користувача

UI-1. Програма має надавати веб-інтерфейс користувача (вебсайт).

UI-2. Інтерфейс користувача повинен коректно відображатися на екранах шириною від 720px включно. Усі елементи сторінки повинні бути повністю видимі за вказаної ширини екрану.

UI-3. Мінімальний розмір шрифту всіх веб-сторінок – 16px включно.

UI-4. Колір тексту сторінок повинен бути чорним, кольорових елементів – білим.

UI-5. Веб-сайт повинен бути доступним у таких мовах: англійська, українська, французька. При цьому всі назви наукових робіт повинні бути відображені мовою оригіналу. Допускається відображення термінів мовою оригіналу.

UI-6. Вебсторінка повинна мати структуру, наведену на рис. 2.1.

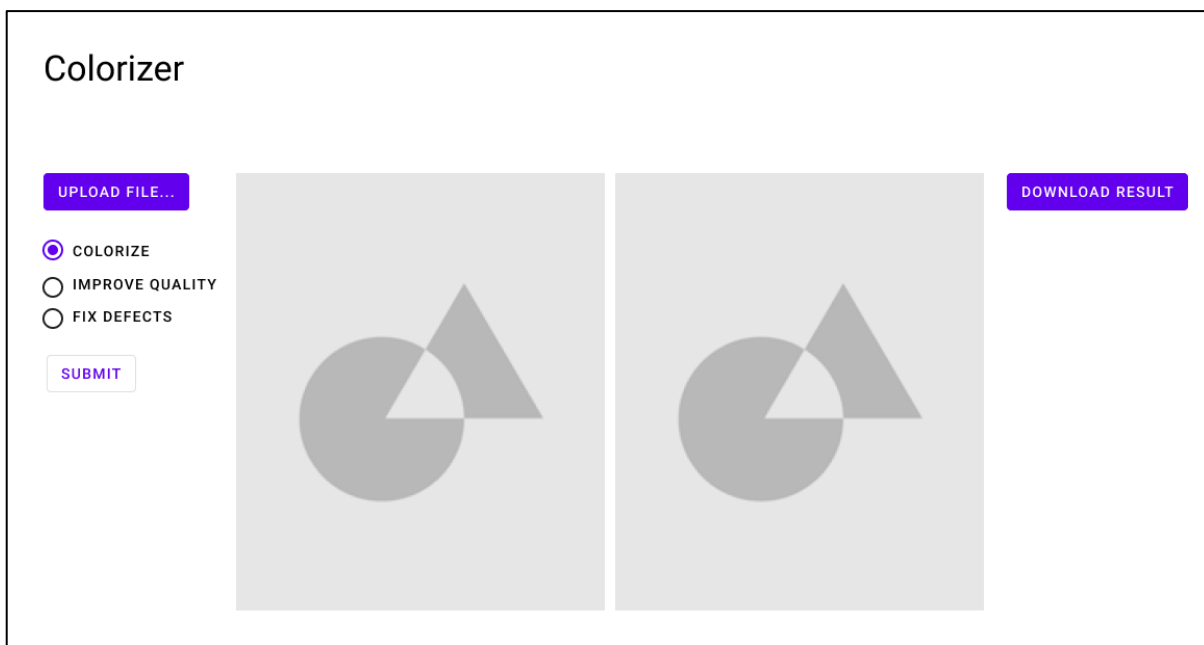


Рис. 2.1. Структура вебсторінки

Вебсторінка повинна містити елементи, описані нижче.

- Кнопка «Завантажити файл». При натисканні повинне з'явитися діалогове вікно менеджера файлів, де користувач може обрати файл. Повинні відображатися файли лише підтримуваних форматів (підтримувані формати наведено у FR-1).
- Меню зі списком сервісів. Користувач може обрати лише один сервіс за раз (сервіси наведено у FR-3).
- Кнопка «Підтвердити». Кнопка повинна бути недоступна, якщо користувач не обрав жодного сервісу. При натисканні повинен бути зроблений запит на сервер з фотографією, яку востаннє завантажив користувач, а також з обраним сервісом.
- Віконце «Поточна фотографія». Повинне містити фотографію, завантажену користувачем. У випадку, якщо користувач ще не завантажив жодної фотографії, віконце повинне містити напис: «No image yet».
- Віконце «Результат». Містить наступні елементи.
 - Завантажувач (елемент, який відображає те, що запит в обробці).

- Повідомлення про помилку (якщо під час обробки зображення сталася помилка. Повідомлення повинне містити текст помилки).
- Результуюче зображення (у випадку, якщо обробка пройшла успішно).
- У всіх інших випадках віконце повинне містити напис: «No image yet».
- Кнопка «Завантажити результат». Після натискання користувач повинен обрати папку, в яку завантажити результат. Після цього повинне початися завантаження в цю папку.

Інтерфейси ПЗ

SI-1. Серверна частина повинна працювати на основі дистрибутиву Linux Ubuntu (від версії 22.04 включно).

SI-2. Список підтримуваних браузерів включає: Google Chrome (від версії 108.0.5359.71 включно), Mozilla Firefox (від версії 110.0 включно).

Інтерфейси зв'язку

Комунікація між клієнтом і сервером повинна здійснюватися за протоколом HTTP.

2.1.6. Функціональні вимоги

FR-1. Програма повинна працювати з фотографіями у наступних розширеннях: PNG, JPG, JPEG.

FR-2. Користувач повинен мати змогу завантажити результат у форматі PNG.

FR-3. Користувач повинен мати змогу обрати один із сервісів:

- колоризація фотографій (надання чорно-білому зображенню кольору);

- відновлення фотографій від пошкоджень (низький контраст, зависока або занижка яскравість, шум, «тріщина» або «розрив»);
- покращення якості фотографій (збільшення роздільної здатності зображення).

FR-4. Формат і роздільна здатність вихідного зображення повинні бути такими ж, як і у вхідного.

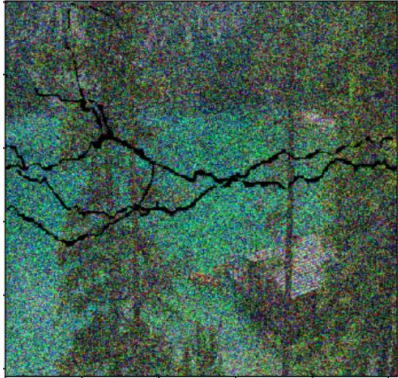
FR-4.1. Якщо користувач обрав сервіс «Покращення якості», роздільна здатність має бути вищою у чотири рази (фактор збільшення), але при цьому повинен бути збережений формат зображення (відношення ширини до висоти).

FR-5. Вимоги щодо роботи кожного із сервісів наведено у таблиці 2.1.
Джерела фотографій: [30], [31].

Таблиця 2.1

Вимоги та приклади роботи кожного із сервісів

Сервіс	Вхідні дані	Результат роботи
Колоризація	Чорно-біле зображення довільного розміру	Кольорове зображення такого ж формату, як і вхідне
		

Сервіс	Вхідні дані	Результат роботи
Відновлення фотографій від пошкоджень	Кольорове або чорно-біле зображення із дефектами	Кольорове або чорно-біле зображення такого ж формату, як і вхідне, із усуненими дефектами
		
Покращення якості фотографій	Кольорове або чорно-біле зображення з низькою роздільною здатністю	Кольорове або чорно-біле зображення з роздільною здатністю в чотири рази більшою за вхідну
		

2.1.7. Вимоги до робочих характеристик

PR-1. Максимальний час обробки одного зображення одним сервісом
– 1 хвилина.

2.2. Архітектура програми

Нижче наведено опис архітектури системи та обґрунтування прийнятих архітектурних рішень.

2.2.1. Огляд архітектури вебсервісу

Програма має багаторівневу архітектуру. Схема архітектури програми зображена на рис. 2.2.

Рівень представлення реалізовано у вигляді вебсайту. Кінцевий користувач має доступ лише до цього рівня. Сервісний рівень представляє RESTful вебсервер. Сервер приймає запити від клієнтської частини (рівень представлення) й передає дані на обробку до однієї з моделей (рівень даних).

Рівень даних представлено трьома моделями (нейронними мережами), що безпосередньо обробляють зображення:

- модель для колоризації фотографій;
- модель для реставрування фотографій;
- модель для збільшення роздільної здатності зображень.

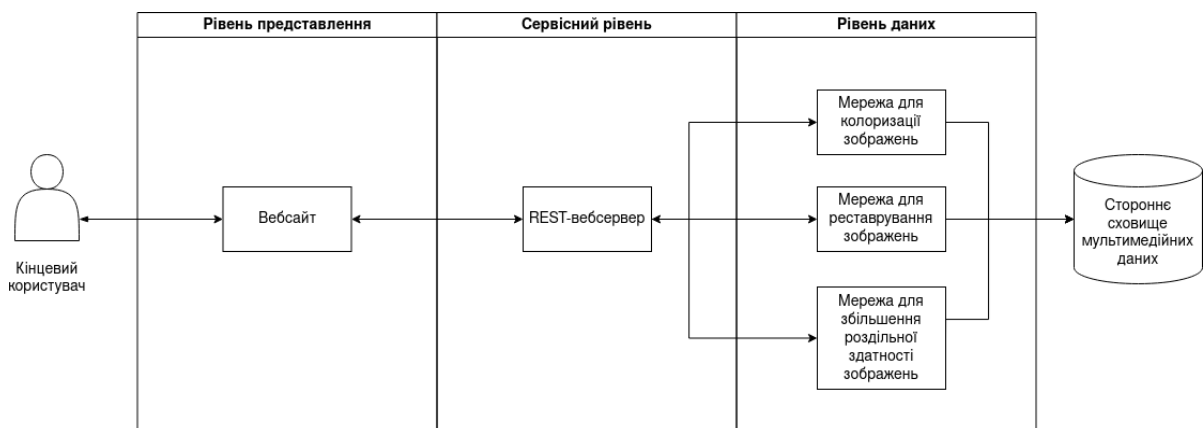


Рис. 2.2. Багаторівнева архітектура системи

Моделі працюють незалежно одна від одної та не мають спільного вихідного коду. Дані для тренування мереж беруться зі стороннього

джерела (наприклад, з пакету TensorFlow Datasets). На рис. 2.2 це відображено як зовнішнє сховище.

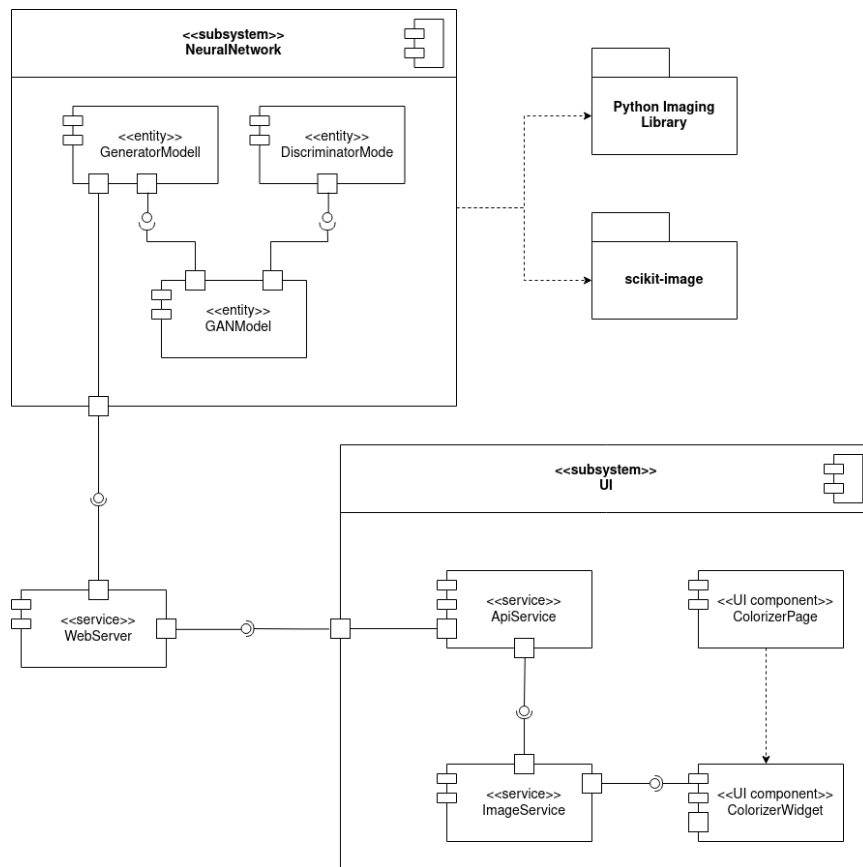


Рис. 2.3. UML-діаграма компонентів системи

На рис. 2.3 наведено діаграму компонентів системи. Підсистема NeuralNetwork (тут назва умовна) представляє модуль однієї з мереж. Підсистема надає інтерфейс генератора, через який вебсервер може отримувати вихідне зображення. На стороні клієнта, вся комунікація з сервером відбувається через ApiService. У компонентах використовується ImageService, що є обгорткою над ApiService й використовує його методи для комунікації з бекендом. Окремо зображено залежності мережі від бібліотек PIL і scikit-image, що не входять до складу основного фреймворку Tensorflow.

2.2.2. Структура нейронної мережі

Усі три нейронні мережі побудовано за принципами об'єкто-орієнтованого програмування. Кожна модель (дискримінатор, генератор та власне GAN) представлена у вигляді окремого класу, що наслідує клас Model з пакету tf.keras. Крім того, до складу моделей входять користувацькі рівні, кожен з яких є класом-наслідником від tf.keras.layers.Layer.

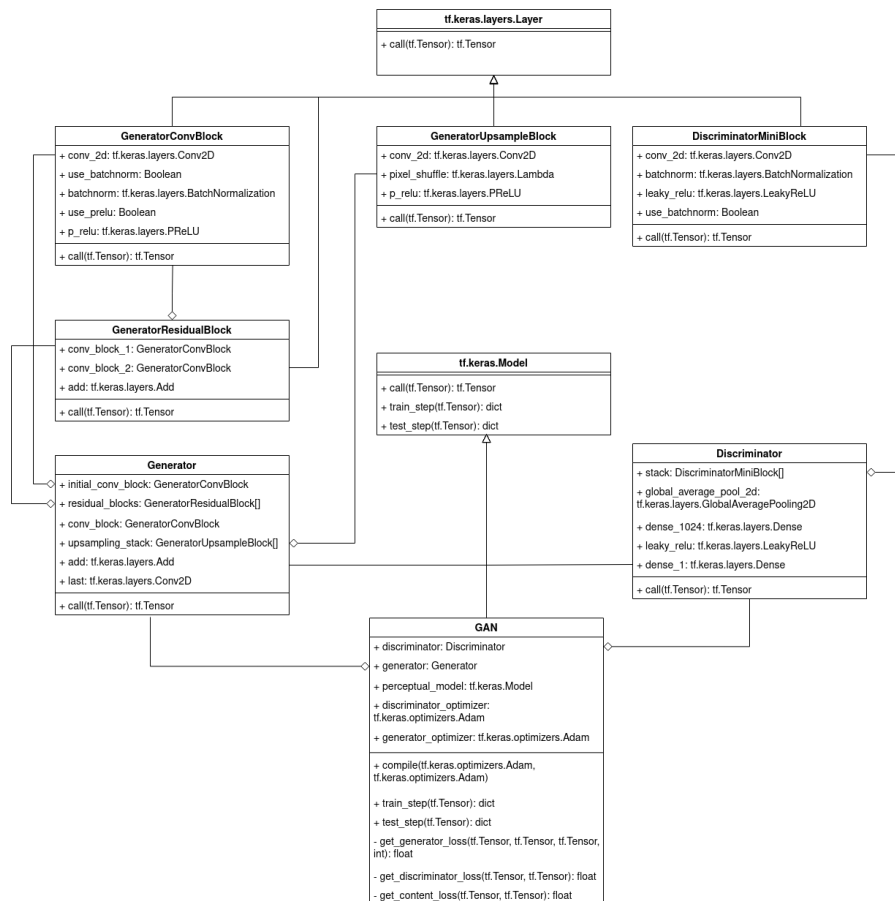


Рис. 2.4. UML-діаграма класів мережі SRGAN

На рис. 2.4 наведено UML-діаграму класів для мережі SRGAN. Як бачимо, для генератора було створено три класи з користувацькими рівнями, і один користувацький рівень для дискримінатора. І генератор, і дискримінатор є складовими частинами моделі GAN.

Модель GAN реалізує метод `train_step()` батьківського класу `tf.keras.Model` – цей метод представляє один навчальний крок. Деталі реалізації наведено в розділі 3 цього документу.

Подібну архітектуру має і модель Pix2pix.

2.2.3. Обґрунтування архітектурних рішень

Система має багаторівневу (3-шарову) архітектуру. Цей патерн було обрано з огляду на наступні його переваги.

- Поділ відповідальності та ізолюваність рівнів. Це означає, що кожен рівень відповідає лише за певні функції й не залежить від того, як влаштовано інші рівні. Таким чином, зміни в одному шарі системи не можуть призвести до збоїв у іншому. Це не лише підвищує стійкість системи, а й спрощує тестування.
- Тестованість. Як зазначено вище, рівні ізолювані та не мають спільного коду. Це значить, що для тестування достатньо замінити всі рівні, крім тестованого, на несправжні, і перевіряти кожен рівень індивідуально. Наприклад, вебінтерфейс користувача може бути протестований з використанням локального серверу із тестовими даними.
- Простота імплементації. Патерн досить інтуїтивний, а тому нескладний у реалізації. Крім того, саме цей підхід досить часто використовують найпопулярніші програмні фреймворки, й існує безліч ресурсів з порадами щодо організації коду та оглядами архітектур застосунків.

Код нейронних мереж організовано за принципами об'єктно-орієнтованого програмування (наслідування та абстракція).

Класи користувацьких рівнів є наслідниками від `tf.keras.layers.Layer` та імплементують його метод `call()`. Аналогічно, класи моделей наслідуються від `tf.keras.Model` та імплементують його методи. Це дозволяє

структурувати програмний код, а також використовувати вбудовані механізми фреймворку, як-от функцію `tf.keras.Model.fit()` для тренування мережі.

Прикладом абстракції можуть слугувати користувацькі рівні. Кожен такий рівень представляє певну часто повторювану групу операцій, як-от, наприклад, залишковий блок генератора в моделі SRGAN [3]: згортка з фільтром $3 \times 3 \rightarrow$ пакетна нормалізація $\rightarrow$ параметричний ReLU $\rightarrow$ згортка з фільтром $3 \times 3 \rightarrow$ пакетна нормалізація $\rightarrow$ поелементна сума із вхідним зображенням блоку. Прихованість деталей реалізації (наприклад, яким чином реалізовано ці операції) робить код більш простим до розуміння та гнучким до змін. Крім того, такі класи зручно перевикористовувати в декількох різних моделях.

3. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА ПІДПРОГРАМ

Цей розділ містить детальний опис програмної реалізації вебсервісу, у тому числі нейронних мереж.

3.1. Нейронні мережі для роботи з зображеннями

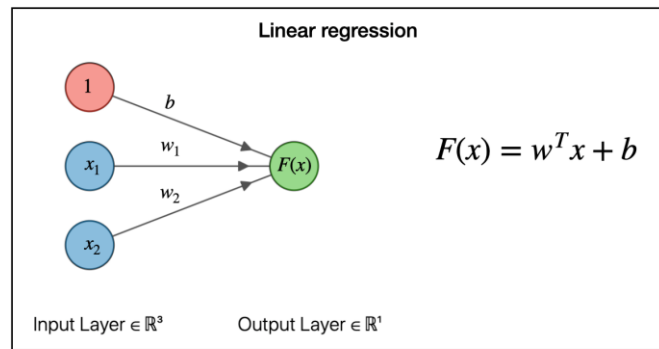
Нейронні мережі було написано на основі фреймворку Tensorflow, а саме – Keras API. Keras API надає високорівневі абстракції, як-от: шари (англ. layers, наприклад, операція згортки, ReLU, максимізаційне агрегування тощо), моделі, функції втрат, функції оптимізації тощо. Це значно спрощує розробку й дозволяє сфокусуватися на архітектурі моделей, а не на деталях реалізації.

Розглянемо спершу загальні поняття, які застосовують при розробці нейронних мереж.

Погляньмо на найпростішу нейронну мережу, що реалізує лінійну регресію. Зведемо задачу до математичних термінів. Нехай маємо вхідні параметри, подані у вигляді вектора: $\mathbf{x} = (x_1, x_2, \dots, x_n)$, і самостійний параметр y . Нехай відомо, що існує лінійна залежність між $\mathbf{x}$ та y , яку можна записати у вигляді функції (13):

$$y = f(\mathbf{x}) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b. \quad (13)$$

Числа w_i , або ваги, та число b , або зміщення, не відомі. Задачею дослідника є знайти ці числа (в літературі їх часто називають «тренованими параметрами», англ. trainable parameters, адже нейронна мережа шукає їх в процесі тренування). Найпростіша нейронна мережа для розв'язання цієї проблеми матиме два рівні: вхідний з параметрами $(x_1, x_2, \dots, x_n)$ та вихідний з числом y . Якщо провести аналогію з будовою мозку тварин, то параметри y та $\mathbf{x}$ будуть нейронами, а ваги w_i відображатимуть «міцність» зв'язку між парами нейронів.



$$F(x) = w_1 \cdot x_1 + w_2 \cdot x_2 + 1 \cdot b$$

Рис. 3.1. Найпростіша нейронна мережа [32]

Початково числа w_i та b обираються довільним чином (або за спеціальним алгоритмом у складніших задачах). До мережі подають безліч різноманітних комбінацій $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$ – число кроків може бути довільним, для задач обробки зображення воно нерідко доходить до 80 000–100 000. Після кожного такого кроку обчислюють $\hat{y} = f(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$. Для оцінки роботи мережі застосовують так звану функцію втрат, що порівнює справжнє значення y для заданого набору вхідних параметрів $(\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)$ та значення $\hat{y}$, отримане з мережі. Однією з поширених функцій втрат є середньоквадратична похибка, яку обчислюють за формулою (14):

$$L(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^N (\hat{y} - y)^2, \quad (14)$$

де N – число кроків тренування.

Мережа повинна підібрати значення w_i та b так, щоби мінімізувати функцію втрат $L(\hat{y}, y)$ – іншими словами, щоби передбачене значення $\hat{y}$ якомога менше відрізнялося від справжнього y . Таким чином, метою тренування мережі є розв’язати задачу оптимізації.

Одним із найбільш відомих оптимізаційних алгоритмів є градієнтний спуск та його модифікації.

Вибір алгоритму оптимізації – один із критичних при створенні мережі, адже «його ефективність напряду впливає на ефективність

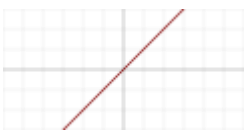
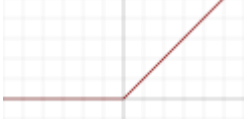
моделі» [33]. Незважаючи на простоту й популярність, градієнтний спуск не завжди є вдалим вибором для задач машинного навчання. Значно частіше використовують більш ефективний алгоритм Adam (англ. Adaptive Moment Estimation), що сьогодні рекомендований як стандартний варіант [34].

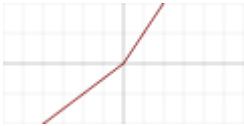
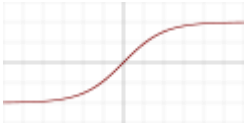
У прикладі вище бачимо, що вихідний результат мережі напряму залежить від $y = f(x)$. Проте на практиці до виходу кожного нейрону застосовують функцію активації. Це дозволяє додати нелінійність у роботу моделі й робить навчання більш ефективним.

Нижче наведено деякі з найбільш поширених функцій активації [35].

Таблиця 3.1

Найбільш поширені функції активації [35]

Назва	Графічна схема функції	Формула	Область можливих значень
Тотожна		$f(x) = x$	$(-\infty, \infty)$
Сигмоїда		$f(x) = \frac{1}{1+e^{-x}}$	$(0; 1)$
Випрямлений лінійний вузол (ReLU)		$f(x) = 0, x < 0$ $f(x) = x, x \geq 0$	$[0, \infty)$
Нещільний випрямлений лінійний вузол (Leaky ReLU)		$f(x) = \alpha x, x < 0$ $f(x) = x, x \geq 0$ $\alpha \in R, 0 < \alpha \ll 1$ Значення α стає впродовж тренування	$(-\infty, \infty)$

Назва	Графічна схема функції	Формула	Область можливих значень
Параметричний випрямлений лінійний вузол (PReLU)		$f(x) = \alpha x, x < 0$ $f(x) = x, x \geq 0$ $\alpha \in R$ Мережа змінює значення α у процесі навчання для оптимізації тренування	$(-\infty, \infty)$
Гіперболічний тангенс (TanH)		$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1; 1)$

3.1.1. Модель Pix2pix

Програмний код базовано на реалізації з офіційного репозиторію Tensorflow [13]. Однак, на відміну від джерела, що використовує принципи функціонального програмування, було використано об'єкто-орієнтований підхід, що включив наступні кроки.

1. Реалізація користувацьких рівнів. Tensorflow надає безліч готових операцій, що широко застосовуються в нейронних мережах. Наприклад, операція згортки представлена у вигляді `tf.keras.layers.Conv2D`. Для більш складних мереж, однак, може виникнути необхідність у рівні зі специфічною логікою. Крім того, користувацький рівень може об'єднати в собі часто повторювану групу операцій. Для реалізації такого рівня потрібно: створити клас-наслідник `tf.keras.layers.Layer`; імплементувати метод `call`; за необхідності імплементувати метод `build` (для встановлення певних параметрів рівня). У роботі для генератора та дискримінатора було створено два таких рівні: `Encoder` та `Decoder`.

2. Реалізація моделей генератора та дискримінатора. Кожна модель є окремим класом, що наслідується від батьківського `tf.keras.Model`. Моделі імплементують батьківський метод `call()`: у генераторі, цей метод повертає згенероване зображення, у дискримінаторі – булеву матрицю з оцінкою правдоподібності вхідного зображення.
3. Реалізація моделі GAN. Зауважмо, що генератор та дискримінатор є складовими частинами GAN. Як і ці мережі, GAN є класом-наслідником `tf.keras.Model`. GAN імплементує батьківський метод `train_step()`, що уособлює один навчальний крок. У цій функції відбувається обчислення втрат генератора та дискримінатора та оновлення їхніх вагів. Метод `train_step()` повертає мапу зі значеннями метрик тренування (наприклад, значення функцій втрат, SSIM). Крім того, GAN реалізує метод `test_step()` – по суті, аналог `train_step()`, проте без оновлення параметрів підмереж. Додатково, GAN містить `compile()` метод, у який передають оптимізатори для генератора та дискримінатора (у роботі – об'єкти `tf.keras.optimizers.Adam`).

Зауважимо наступні особливості моделі. У генераторі функція пакетної нормалізації застосовується до всіх рівнів, окрім першого та останнього рівнів кодування. Пакетна нормалізація покликана усталити тренування, проте, «застосована до всіх рівнів, вона може призвести до нестабільності моделі» [36].

Усі рівні кодування генератора, за винятком вхідного, використовують як функцію активації нещільний ReLU з параметром $\alpha = 0.02$, а всі рівні декодування – звичайний ReLU. Останній рівень, за рекомендаціями джерела [36], використовує гіперболічний тангенс. Дискримінатор використовує нещільний ReLU з параметром $\alpha = 0.02$ для всіх рівнів, крім останнього, що застосовує сигмоїду. Вище у табл. 3.1 наведено порівняння цих функцій активації.

І генератор, і дискримінатор застосовують алгоритм оптимізації Adam, а саме його реалізацію з пакету `tf.keras.optimizers` від Tensorflow Keras API.

Код моделі наведено в додатку 2.

Для тренування мереж було використано підхід із джерела [13]. У ньому застосовують спеціальний об'єкт `tf.GradientTape`.

Метод `gradient(target, sources)` (рядки 1-2) повертає градієнт функції `target`, обчислений для аргументів `sources`. У нашому випадку він обчислює градієнт для функцій втрат генератора й дискримінатора. Після цього обраний оптимізатор – алгоритм Adam для Pix2pix – оновлює ваги мереж таким чином, щоби мінімізувати функції втрат (рядки 3-4).

Модель Pix2pix була використана і для колоризації, і для реставрування зображень (для кожної із задач було створено й натреновано окрему мережу). Незначні зміни було застосовано лише до вхідних рівнів: для колоризації, додатковим вводом є 1-канальне чорно-біле зображення, а для реставрування – 3-канальне зображення з дефектами. Принцип роботи та тренування мереж в обох випадках той самий.

Обидві моделі працюють із зображеннями розмірності 512×512 .

Розглянемо загальний алгоритм тренування моделі. Перед тренуванням потрібно завантажити параметри мереж, збережені з попереднього тренування у форматі `tf.train.Checkpoint`. Після цього завантажуть датасети з тестовими та тренувальними зображеннями. Кількість епох тренування та кількість кроків за кожну епоху встановлює користувач. На кожному кроці тренування обчислюють втрати генератора та дискримінатора й оновлюють ваги мереж. Після кожної епохи значення втрат мереж та індекс SSIM зберігають у вигляді графіку та виводять користувачу. У кінці тренування генерують кілька тестових зображень для оцінки роботи мережі та зберігають параметри моделей.

Блок-схему алгоритму тренування наведено на рис. 3.2.

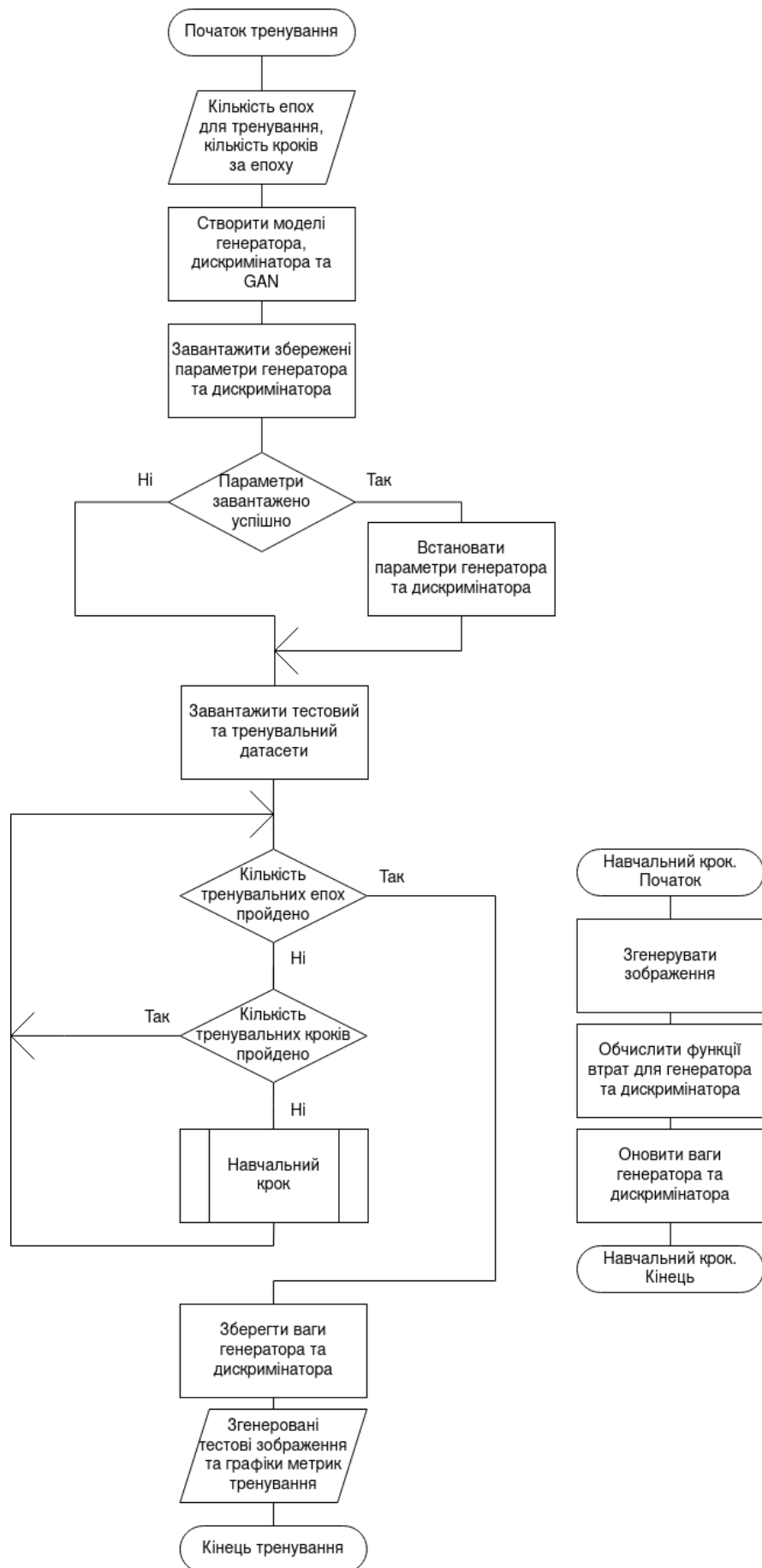


Рис. 3.2. Схема тренування мережі

Лістинг 3.1. Реалізація зворотного поширення помилки за допомогою об'єкта GradientTape. Узято з офіційного репозиторію Tensorflow [13]

```
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as
disc_tape:
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

        gradients_of_generator = gen_tape.gradient(gen_loss,
generator.trainable_variables) # 1
        gradients_of_discriminator = disc_tape.gradient(disc_loss,
discriminator.trainable_variables) # 2

        generator_optimizer.apply_gradients(zip(gradients_of_generator,
generator.trainable_variables)) # 3

    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables)) # 4
```

3.1.2. Модель SRGAN

Модель SRGAN, як і модель Pix2pix, побудована за принципами об'єктно-орієнтованого програмування. Деталі реалізації наведено у пункті 3.1.1.

Розглянемо особливості, притаманні SRGAN. Дискримінатор побудованої моделі складається із серії міні-блоків, кожен із яких у свою чергу включає такі операції: згортка з фільтром $3 \times 3 \rightarrow$ пакетна нормалізація $\rightarrow$ нещільний ReLU з параметром $\alpha = 0.02$. Пакетна нормалізація застосовується до всіх міні-блоків, окрім першого.

Генератор складається з 16 залишкових блоків (їхнє число обирається користувачем, було взято 16 за прикладом джерела [14]) та 2 блоки збільшення роздільної здатності. Кількість блоків збільшення роздільної здатності напрямку залежить від масштабу збільшення: для збільшення в два рази потрібен один такий блок, в чотири рази – два, у вісім разів – три [14].

Розроблена модель переводить зображення 128×128 у розмірність 512×512 , тобто виконує чотирикратне збільшення.

Для обчислення функції втрат було використано готову модель VGG19 з пакету `tf.keras.applications.vgg19` [14].

Процес тренування аналогічний такому ж для мережі Pix2pix (пункт 3.1.1).

3.2. Вебклієнт

Вебклієнт написано на основі фреймворку Angular з використанням бібліотеки компонентів Angular Material. Вебсайт включає наступні сторінки.

- Домашня сторінка. Містить короткий опис проєкту, список основних джерел, приклад результатів і контакти автора.
- Результати. Містить приклади роботи мереж та ілюстрації процесу тренування кожної з моделей (результат на тестових прикладах і графіки SSIM).
- Сторінка з віджетом. Користувач має змогу завантажити зображення й обрати один із сервісів: колоризація; відновлення від пошкоджень; покращення якості фотографії. Після цього результат обробки відображається поряд з вхідним фото і користувач має змогу завантажити його у вигляді PNG-файлу.
- Сторінка з використаними джерелами. Містить перелік усіх джерел, використаних у роботі: наукові та технічні статті, посилання на репозиторії з кодом, технічну документацію тощо.

Користувач може змінити мову веб-сайту на одну з доступних: англійська, французька, українська.

Вебклієнт було задеплоїно до AWS S3 Bucket. Для цього було використано інструмент GitLab CI/CD, що дозволяє налаштовувати автоматичні пайплайни й деплой застосунку. Для вебклієнту було створено пайплайн з наступними стадіями.

- Lint. На цій стадії здійснюється статична перевірка коду на відповідність налаштуванням ESLint.
- Test. Ця стадія запускає виконання unit-тестів.
- Build. На цій стадії відбувається збірка застосунку. Результатом цього кроку є директорія /dist, що містить головний файл index.html, JS-скрипти, файли стилів. Директорія /dist зберігається як артефакт для наступної стадії Deploy.
- Deploy. На відміну від усіх інших стадій, цей крок повинен бути запущений лише вручну. На цій стадії файли з директорії /dist (зібраний застосунок) завантажуються у сховище AWS S3 Bucket, звідки потім вони доступні за публічною IP-адресою.

У лістингу 3.2 наведено файл налаштувань пайплайну.

Механізм CI/CD значно спрощує розробку й процес розгортання застосунку, оскільки потребує найменшого втручання розробника.

На рис. 3.3 наведено скріншот із платформи GitLab з прикладами різних станів пайплайну («пройшов успішно», «помилка», «в прогресі»). Зауважмо, що при помилці на одній зі стадій, увесь пайплайн завершиться зі станом помилки. Це унеможливорює потрапляння неякісного або непрацюючого коду на production-середовище (у цьому випадку S3 Bucket).

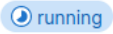
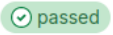
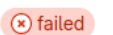
Status	Pipeline	Triggerer	Stages
 running	Update environment.prod.ts #900661947  master  812da7a0  latest		
 passed 00:07:36 15 hours ago	Update environment.prod.ts #899958878  master  46486a3a 		
 failed 00:03:23 18 hours ago	Update .gitlab-ci.yml file #899710491  master  df6b3d6d 		

Рис. 3.3. Різні стани CI/CD пайплайну. Скріншот із платформи GitLab

Лістинг 3.2. Файл налаштувань GitLab CI/CD пайплайну

```
image: node:latest

stages:
  - lint
  - test
  - build
  - deploy

before_script:
  - npm install -force

lint:
  stage: lint
  script:
    - npm run lint
  test:
    stage: test
    script:
      - npm run test

build:
  stage: build
  script:
    - npm run build --prod
  artifacts:
    paths:
      - $CI_PROJECT_DIR/dist

deploy:
  stage: deploy
  when: manual
  image: python:latest
  before_script:
    - pip install awscli
  needs: [ 'build' ]
  script:
    - aws configure set region $AWS_DEFAULT_REGION
    - aws configure set aws_access_key_id
      $AWS_ACCESS_KEY
    - aws configure set aws_secret_access_key
      $AWS_SECRET_KEY
    - aws s3 sync ./dist/colorizer s3://$BUCKET_NAME/ -
      -delete
    - echo "Deployed successfully"
  only: ['master']
```

Вебклієнт, розгорнутий у хмарному сховищі, доступний за посиланням: <http://colorizer-ui.s3-website-us-east-1.amazonaws.com/>.

3.3. Вебсервер

Для отримання обробленого зображення, клієнт робить POST-запит із вхідним фото на один із таких ендпоінтів: `/colorize` (колоризація зображення), `/restore` (відновлення зображення від пошкоджень), `/super_resolution` (підвищення якості зображення).

Результат обробки зберігається на стороні сервера у вигляді файлу, й після відповіді клієнту видаляється.

Для побудови веб-серверу було використано бібліотеку FastAPI, а для роботи із зображеннями – PIL (наприклад, для збереження фото у вигляді файлів, попередньої обробки зображень тощо).

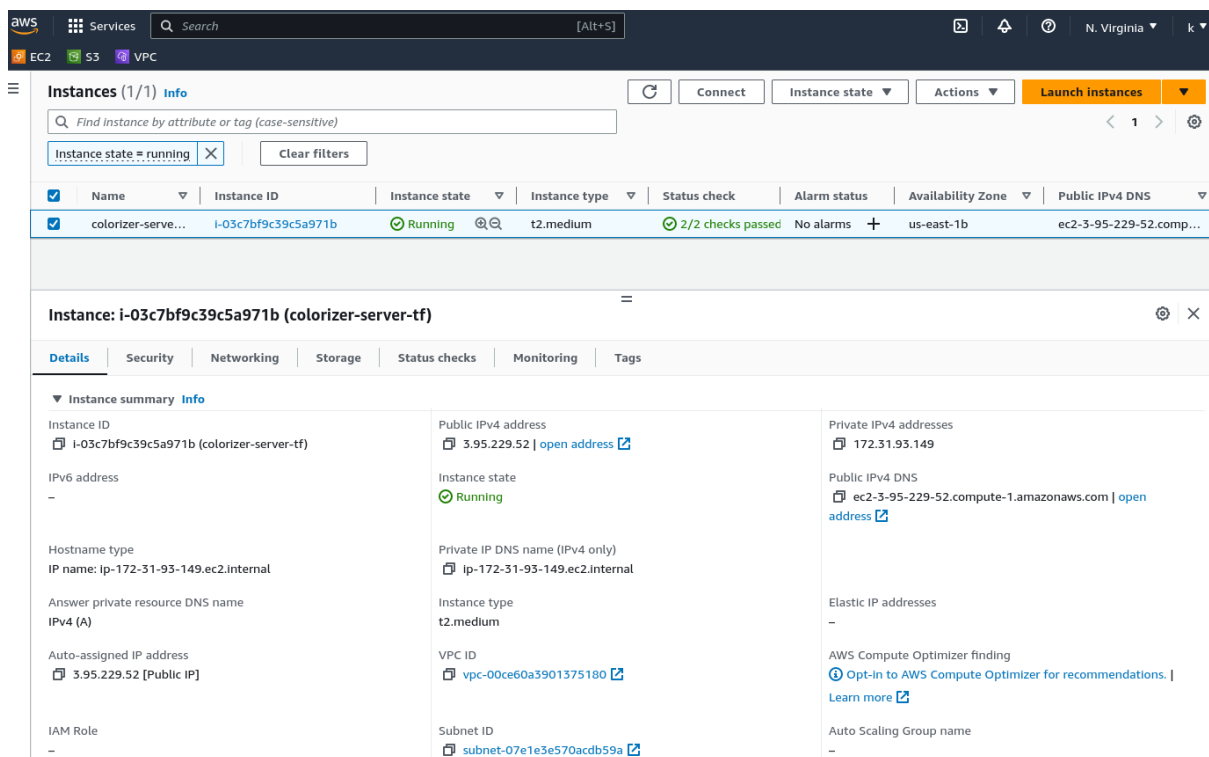


Рис. 3.4. Налаштування віртуальної машини Amazon EC2. Скріншот із консолі AWS

Як і вебклієнт, вебсервер було задеплоїно до хмарних сервісів AWS, а саме до Amazon EC2. На рис. 3.4 наведено налаштування віртуальної машини, що слугувала для розгортання серверу. Основні її параметри: тип – t2.medium, зона доступності (availability zone) – us-east-1b, 4 Гб оперативної пам'яті.

У лістингу 3.3 наведено скрипт для налаштування віртуальної машини та запуску сервера (виконується після підключення до машини через SSH). Окрім наведеного скрипту, налаштування також включило конфігурування Nginx (для забезпечення доступу до сервера за публічною IP-адресою машини).

Лістинг 3.3. Налаштування та розгортання серверу на віртуальній машині

```
# Завантаження пакету python-pip

sudo yum -y install python-pip

# Завантаження tensorflow

pip3 install tensorflow-cpu --no-cache-dir

# Завантаження усіх інших пакетів

pip install uvicorn
pip install matplotlib
pip install scikit-image
pip install "fastapi[all]"

# Завантаження git

sudo yum install git -y

# Клонування git-репозиторію

git clone https://gitlab.com/diploma\_colorizer/colorizer-server.git

# Перехід до кореневої папки проекту

cd colorizer-server

# Запуск вебсервера

uvicorn src.main:app
```

Для власне обробки зображень вебсервер передає вхідну фотографію моделі-генератору. Попередньо ваги моделей відновлюються із файлів формату `tf.Checkpoint`, збережених після тренування мереж. Зауважмо, що відновлення попередніх ваг відбувається лише один раз, при запуску сервера. Код та збережені параметри усіх трьох моделей зберігаються на сервері незалежно.

4. АНАЛІЗ РОЗРОБЛЕНОГО ВЕБСЕРВІСУ ДЛЯ КОЛОРИЗАЦІЇ ТА ВІДНОВЛЕННЯ ФОТОГРАФІЙ ВІД ПОШКОДЖЕНЬ

4.1. Особливості реалізації сервісу

Розділ містить опис основних деталей реалізації нейронних мереж.

4.1.1. Попередня обробка зображень

Перед поданням зображення на вхід до мережі, здійснюється його попередня обробка. Нижче наведено деякі деталі такої обробки.

Нормалізація значень пікселів зображень

У моделі RGB значення пікселів зображень знаходяться в діапазоні $[0, 255]$. При створенні нейронних мереж для роботи із зображеннями поширеною практикою є попереднє масштабування значень пікселів (нормалізація), найчастіше – до діапазону $[0, 1]$. Це підвищує ефективність тренування мережі. У цій роботі значення пікселів у кожній моделі було приведено до діапазону $[-1, 1]$.

Використання моделі YUV

Для задачі колоризації зображень деякі автори пропонують використовувати інші кольорові моделі, що мають лише два кольорові канали. Такими моделями є, наприклад, YUV (за яскравість відповідає канал Y, англ. luma), HSL (за яскравість відповідає канал L, англ. lightness), Lab (за яскравість відповідає канал L, англ. lightness). Ідея полягає в тому, що мережа повинна згенерувати лише два канали, базуючись на третьому, на противагу трьом кольоровим каналам для моделі RGB. Так, наприклад, у роботі «Image Colorization with Convolutional Neural Networks» [38] було використано модель Lab.

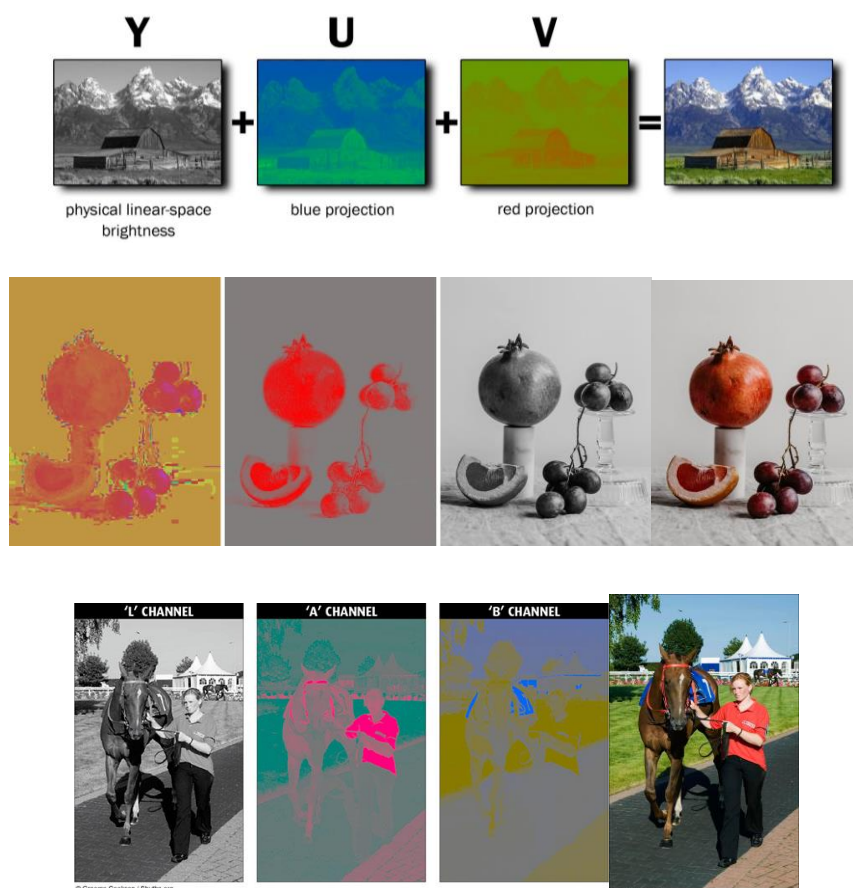


Рис. 4.1. Приклад зображень, розкладених на канали в різних кольорових моделях. Згори вниз: YUV [39], HSL [40], Lab [41]



Рис. 4.2. Спотворення, спричинені переведенням з моделі RGB до YUV (зліва направо: чорно-біле фото, оригінальне кольорове фото, зображення, згенероване мережею)

У першому варіанті мережі для колоризації зображення були переведені з моделі RGB до YUV. Проте було виявлено, що таке переведення спричиняє втрату даних, яке в свою чергу відображається на

результуючому фото. Наприклад, на темних ділянках зображень з'являлися яскраві кольорові плями (рис. 4.2). Такі ж результати було отримано і для моделі HSL. Тому було вирішено працювати із зображеннями в моделі RGB.

4.1.2. Особливості реалізації нейронних мереж

Для підвищення продуктивності програми до датасетів було застосовано операцію shuffle («перетасування») з розміров буфера 10 000 елементів. Таку операцію особливо рекомендовано використовувати для великих датасетів [42], причому розмір буфера варто обирати рівним або більшим за розмір датасета.

Лістинг 4.1. Штучне «погіршення» якості вхідного зображення

```
def apply_random_noise(image):
    noises = ['gaussian', 'poisson', 'salt', 'pepper', 's&p', 'speckle']
    noise = random.choice(noises)

    # Apply additional random parameters for some of the noises
    if noise == 'gaussian' or noise == 'speckle':
        variance = np.random.uniform(0.01, 0.4)
        return skimage.util.random_noise(image.numpy(), mode=noise,
var=variance)

    if noise == 'salt' or noise == 'pepper':
        amount = np.random.uniform(0.01, 0.4)
        return skimage.util.random_noise(image.numpy(), mode=noise,
amount=amount)

    if noise == 's&p':
        amount = np.random.uniform(0.01, 0.4)
        salt_vs_pepper = np.random.uniform(0, 1)
        return skimage.util.random_noise(image.numpy(), mode=noise,
amount=amount, salt_vs_pepper=salt_vs_pepper)
    return skimage.util.random_noise(image.numpy(), mode=noise)

def degrade_image(image, masks_dataset, noise_probability = 0.8):
    # Change brightness
    brightness_factor = np.random.uniform(-0.2, 0.2)
    result = tf.image.adjust_brightness(image, brightness_factor)
    # Change contrast
    contrast_factor = np.random.uniform(0.5, 0.8)
    result = tf.image.adjust_contrast(result, contrast_factor)
    # Apply random noise
    if np.random.uniform(0, 1) <= noise_probability:
        result = apply_random_noise(result)
    # Apply random number of scratches (0-5 including)
    n_scratches = np.random.randint(0, 5)
    for mask in masks_dataset.take(n_scratches):
        result = apply_mask_inverted(result, mask)

    return result
```

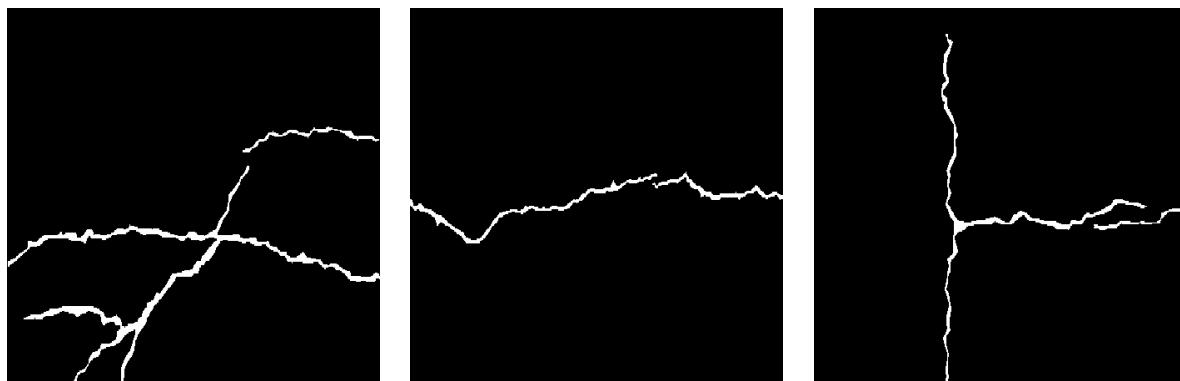


Рис. 4.3. Маски «тріщин». Датасет: Crack Segmentation Dataset [43]

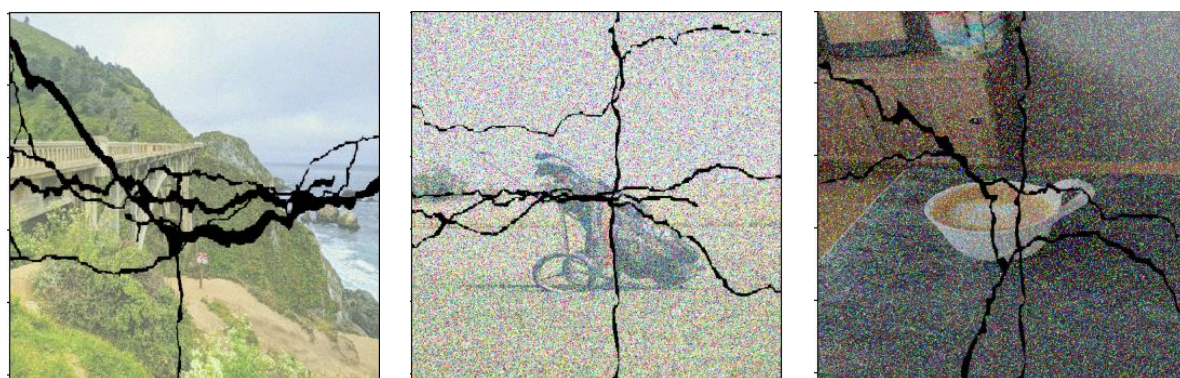


Рис. 4.4. Зображення із застосованими масками. Датасет із масками: Crack Segmentation Dataset [43], датасет із зображеннями: CLIC [44]

Для оцінки роботи мереж було використано SSIM (англ. structural similarity index measure). SSIM обраховують для двох зображень однакового розміру. Значення цієї метрики знаходяться на проміжку $[-1, 1]$, де -1 позначає відсутність жодної подібності, а 1 відповідає двом ідентичним зображенням. У процесі тренування SSIM повинно поступово підвищуватися – тоді це означатиме, що навчання ефективне.

У роботі для обчислення SSIM було використано метод `tf.image.ssim` з бібліотеки Tensorflow.

Для моделі реставрування зображень було реалізовано спеціальну функцію, що штучно «погіршує» якість вхідного зображення. А саме: додає випадковий шум на зображення, додає випадкову «тріщину», змінює яскравість та контраст зображення.

Для додання «подряпин» до зображення було застосовано датасет Crack Segmentation [43], що містить маски реальних тріщин. До кожного зображення із датасету було застосовано від 0 до 5 таких масок. На рис. 4.4 наведено кілька прикладів масок та зображень з доданими масками.

Для додання шуму було використано метод `random_noise` з бібліотеки `skimage.util`. Було використано такі види шумів: «gaussian», «poisson», «salt», «pepper», «s&p», «speckle». Додаткові налаштування (як-от відсоток зображення, що повинен мати шум) було також згенеровано випадково. До деяких зображень не було додано жодного шуму. Усе це дозволило мережі вчитися розрізняти зашумлені та «чисті» фото, а також розрізняти та правильно прибирати різні види шуму.

Нижче наведено приклад з різними видами шумів [45].

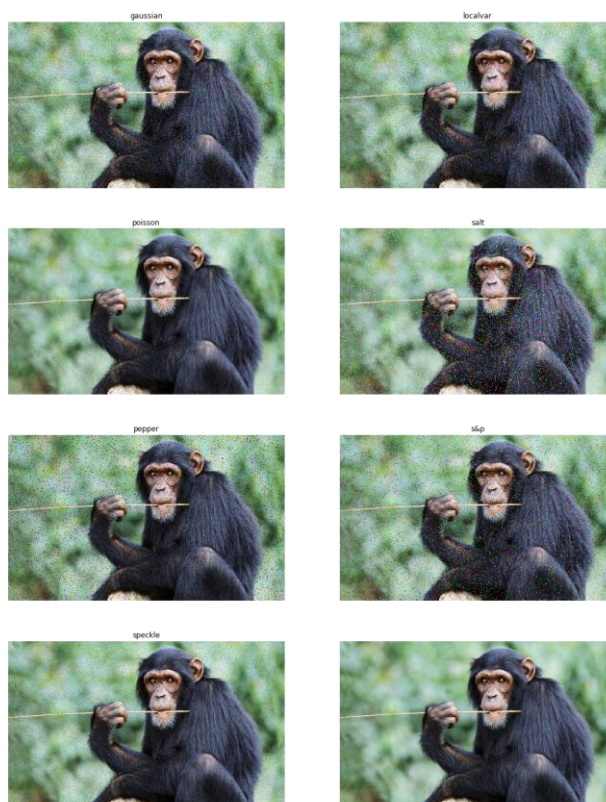


Рис. 4.5. Різні види шумів, застосовані до одного й того ж зображення.

Оригінальне зображення – нижній правий кут [45]

Нижче наведено приклад роботи мережі для реставрування фото.

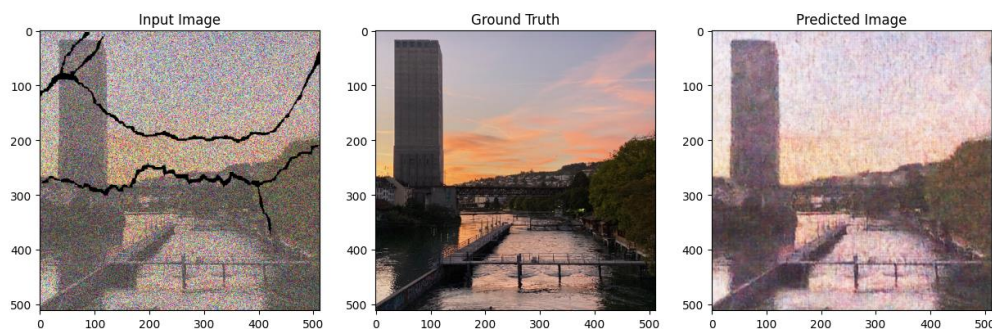


Рис. 4.6. Результат реставрування фото мережею. Зліва направо: штучно «погіршене» фото; оригінальне фото; зображення, відновлене мережею.

Датасет: CLIC [44]

4.2. Тренування нейронних мереж та тестування сервісу

Розділ містить опис процесу тренування мереж, а також тестування їхньої роботи.

4.2.1. Колоризація фотографій

Тренування моделі було здійснено на таких датасетах: DIV2K [31], Landscape color and grayscale images [46], Landscape Dataset [47], Portrait Dataset [48], Historical Color Image Database [49]. Із усіх датасетів було вилучено вручну чорно-білі фото. Модель пройшла 380 навчальних епох по 100 кроків кожна. Нижче наведено результати наприкінці тренування та графік SSIM за перші 180 епох.

З графіку видно, що спершу SSIM стрімко зростає і потім поступово уповільнюється. Зауважимо нелінійність навчання: на графіку присутні різкі стрибки, що можуть свідчити про нестабільність процесу. Наприкінці 180-ї епохи значення SSIM коливаються від 0.8 до 0.85.

Повний процес тренування наведено в додатку 5.

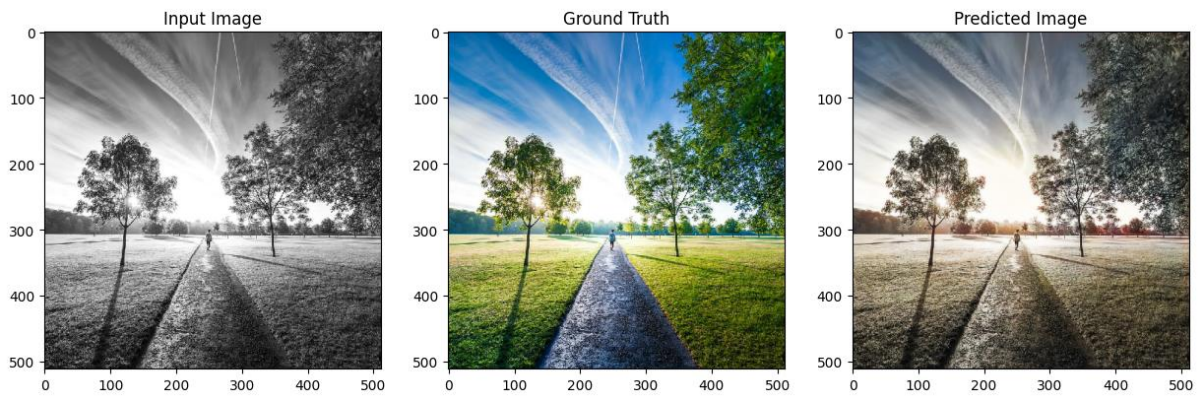


Рис. 4.7. Результати, отримані після 380-ї епохи. Зліва направо: вхідне зображення, оригінальне зображення, результат роботи мережі

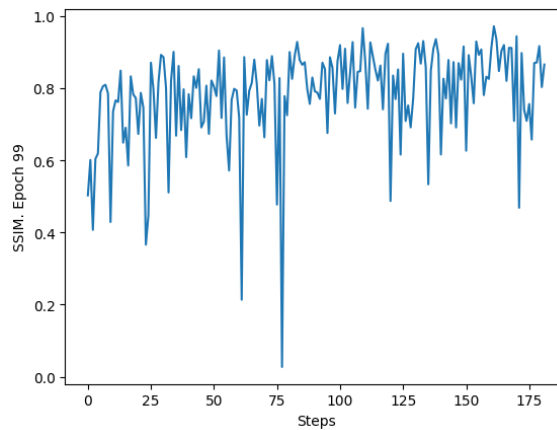


Рис. 4.8. Графік SSIM. Епохи 0-180

4.2.2. Відновлення фотографій від пошкоджень

Тренування моделі було здійснено на таких датасетах: DIV2K [31], CLIC [44], Crack Segmentation Dataset [43] (датасет із масками).

Модель пройшла 300 навчальних епох по 100 кроків кожна. Нижче наведено результати наприкінці тренування та графік SSIM за 300 епох. Як бачимо, робота мережі ще не стабільна на 300-й епосі й у графіку SSIM присутні значні стрибки.

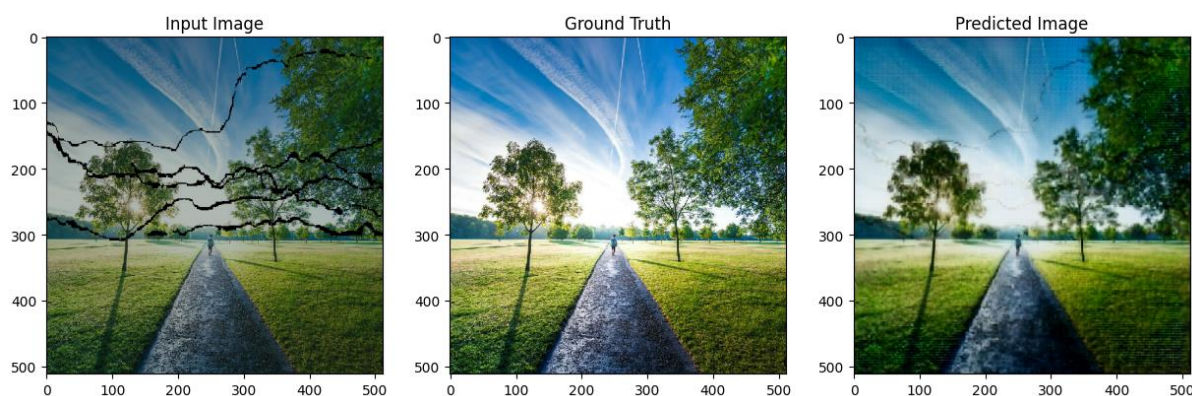


Рис. 4.9. Результати, отримані після 300-ї епохи. Зліва направо: вхідне зображення, оригінальне зображення, результат роботи мережі

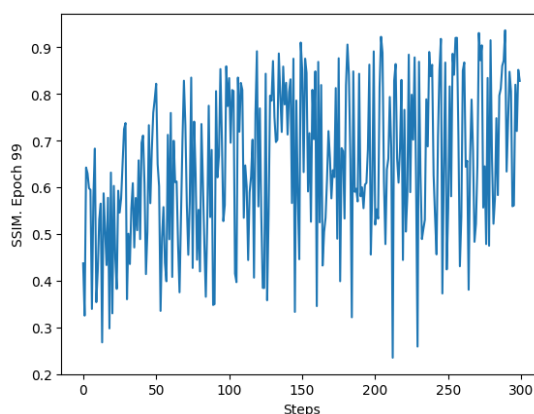


Рис. 4.10. Графік SSIM. Епохи 0-300

4.2.3. Покращення якості фотографій

Тренування моделі було здійснено на датасеті DIV2K [31].

Модель пройшла 200 навчальних епох по 100 кроків кожна. Нижче наведено результати наприкінці тренування та графік SSIM за 200 епох. Розмірність вхідного зображення — 128×128 , вихідного — 512×512 (вказано на шкалі зліва та справа від зображень).

Як бачимо, мережа потребує подальшого тренування, адже наприкінці 200-ї епохи значення SSIM коливається від 0.3 до 0.5.

Повний процес тренування наведено в додатку 5.

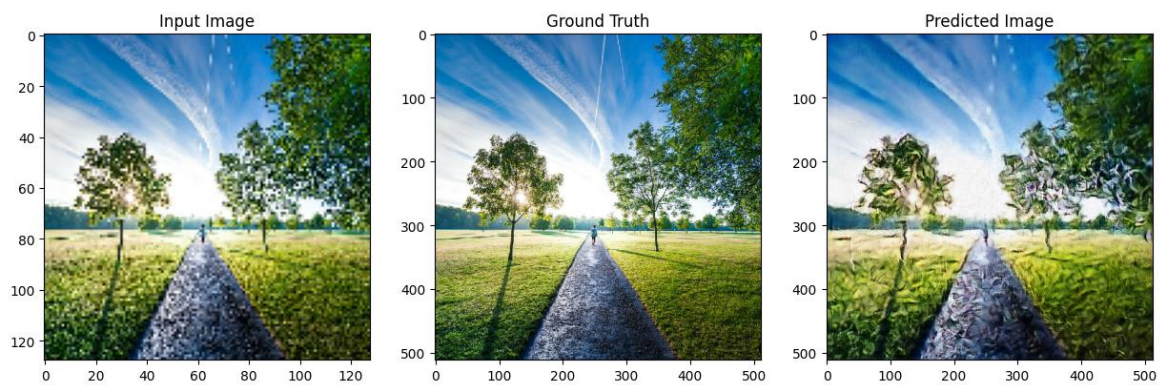


Рис. 4.11. Результати, отримані після 200-ї епохи тренування. Зліва направо: вхідне зображення, оригінальне зображення, результат роботи мережі

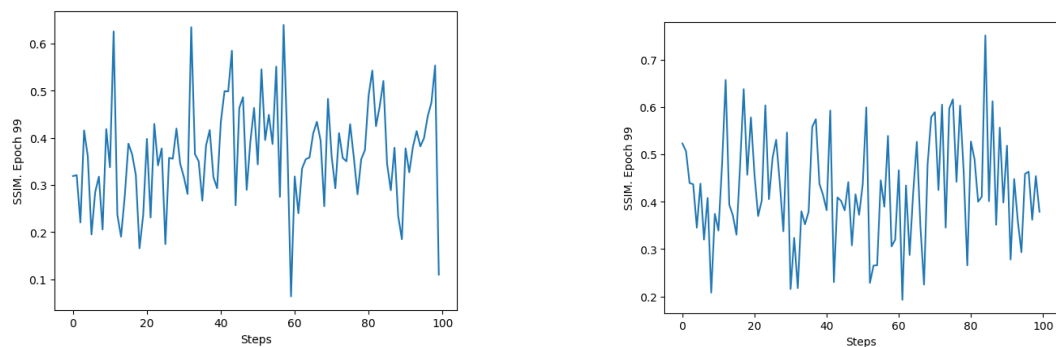


Рис. 4.12. Графіки SSIM. Епохи 0-100 (зліва) та 101-200 (справа)

4.3. Порівняння розробки з існуючими аналогами

Нижче наведено порівняння роботи мережі з деякими продуктами-аналогами на таких самих зображеннях.

4.3.1. Колоризація фотографій

Було перевірено роботу моделі на зображеннях пейзажів (джерело зображень: Landscape Dataset [47]). Мережа коректно розпізнає небо та зелень, однак результат часто виходить досить «сірим». Рішенням проблеми може бути подальше тренування мережі (до 500 епох кроків на противагу поточним 380).

Таблиця 4.1

Порівняння роботи мережі із сервісом Photomyne [29]

Вхідне зображення	Результат стороннього сервісу Photomyne [29]	Результат програми
		
		
		

Таблиця 4.2

Порівняння роботи мережі із сервісом img2go [28]

Вхідне зображення	Результат стороннього сервісу img2go [28]	Результат програми
		

Вхідне зображення	Результат стороннього сервісу img2go [28]	Результат програми
		

4.3.2. Відновлення фотографій від пошкоджень

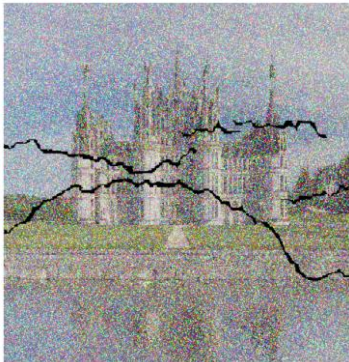
Розроблена програма коректно визнає шум, а також наявність «тріщини», і вирівнює тон зображення.

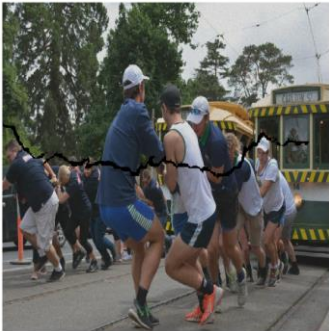
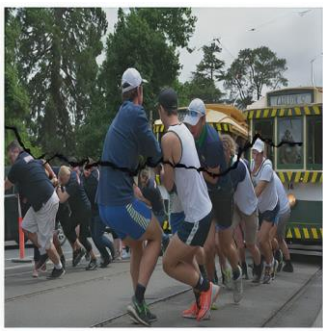
Результат, проте, часто виходить розмитим. Крім того, іноді на результуючому зображенні мережа лишає серйозні артефакти у вигляді яскравих кольорових плям. Найімовірніше, це можна пояснити недостатнім тренуванням мережі: 30000 навчальних кроків проти рекомендованих 50 000–100 000 [50].

Таблиця 4.3

Порівняння роботи мережі із сервісом Hotpot.ai [27].

Джерело зображень: DIV2K [31]

Вхідне зображення	Результат стороннього сервісу Hotpot.ai [27]	Результат програми
		

Вхідне зображення	Результат стороннього сервісу Hotpot.ai [27]	Результат програми
		
		

Сервіс Hotpot.ai [27] успішно прибирає шум із зображення, проте не здатний визначати наявність «тріщини». Крім того, сервіс не вирівнює яскравість та контраст фотографії. Такі результати можна пояснити тим, що мережа сервісу була натренована на дещо інших зображеннях, і тому порівняння з розробленою моделлю може бути некоректним.

4.3.3. Покращення якості фотографій

Обидві мережі (розроблена й зі стороннього сервісу) здійснюють 4-кратне збільшення зображення. Розроблена модель, однак, надає доволі «розмитий» результат. Це пояснюється недостатнім тренуванням мережі: 20 000 кроків проти рекомендованих 50 000–100 000 [50].

Джерело зображень: Landscape Dataset [47].

Порівняння роботи мережі із сервісом img2go [28]

Вхідне зображення	Результат стороннього сервісу img2go [28]	Результат програми
		
		
		

4.4. Рекомендації щодо подальшого вдосконалення

Як можна побачити з прикладів вище, основною проблемою мереж є недостатнє тренування. У статті «NVIDIA Research Achieves AI Training Breakthrough Using Limited Datasets» [50] рекомендується від 50 000 до 100 000 тренувальних зображень для навчання високоякісної мережі GAN. Для порівняння: у роботі один з найбільших датасетів CLIC містив 1633 тренувальні зображення [44].

Знайти або створити великий датасет із якісними зображеннями – часто серйозна проблема для розробників. Наприклад, для тренування мережі для колоризації необхідні «чисті» кольорові фотографії. Найбільш популярні датасети із зображеннями, однак, часто містять чорно-білі фото

або фото з візуальним ефектами. Такі зображення можуть порушити навчання мережі.

Одним із рішень є використовувати декілька готових датасетів, попередньо вручну відфільтрувавши зображення.

Іншою проблемою при тренуванні нейронних мереж є обмеженість ресурсів. У цій роботі навчання всіх моделей було проведено у середовищі Google Colab. Google Colab пропонує додаткові ресурси для пришвидшення виконання програми, як-от оперативна пам'ять, TPU, GPU, тощо. При використанні стандартної опції (12.7GB оперативної пам'яті) 1000 навчальних кроків займає більше 60 хв., і в цілому навчання мережі (50 000 кроків) може скласти більше 2 діб. Тому доцільним є застосування додаткових ресурсів. Наприклад, при додатковому використанні стандартної GPU 1000 навчальних кроків мережі для колоризації складає близько 8 хв., а мережі для покращення якості зображень – близько 20 хв.

Серед усіх трьох мереж найважчим було натренувати мережу для колоризації зображень – навіть після 38000 кроків вона надавала досить посередні результати. При цьому у випадку вузької тематики зображень (наприклад, портрети людей) модель вчилася доволі швидко та якісно.

Можливим рішенням є розділити мережу й тренувати окремо для кожної категорії зображень, приміром: портрети людей, пейзажі, архітектура, тварини тощо. При цьому необхідно буде зберігати параметри мережі для кожної з груп зображень. Також, такий підхід потребуватиме додаткового вводу від користувача.

Модель для відновлення фотографій від пошкоджень іноді залишає на результуючому зображенні артефакти у вигляді яскравих кольорових плям. Найімовірніше, мережа не розпізнає відсутність шуму та намагається «прибрати» його з зображення.

Рішенням проблеми є подальше тренування мережі на зашумлених та «чистих» фото, випадково перемішаних.

Модель для підвищення якості фотографій пройшла недостатнє навчання – наприкінці тренування (після 20 000 кроків) значення SSIM заледве сягають 0.5. Навчання моделі SRGAN у цілому виявилось більш повільним та нестабільним порівняно з Pix2pix. Ймовірно це пов'язано з ресурсоємною операцією «pixel shuffle». Рекомендовано продовжити тренування до 50 000 кроків для отримання кращих результатів.

ВИСНОВКИ

У результаті роботи було створено програму, що здійснює автоматичну фотографії, а саме виконує такі задачі:

- колоризація фотографій (надання чорно-білому зображенню кольору);
- відновлення фотографій від пошкоджень (низький контраст, зависока або занижка яскравість, шум, «тріщина» або «розрив»);
- покращення якості фотографій (збільшення роздільної здатності зображення).

Програма працює з фотографіями людей, природи, тварин, архітектури, інтер'єру, побутових предметів.

Для колоризації фотографій та відновлення зображень від пошкоджень було використано модель Pix2pix, запропоновану в роботі «Image-to-Image Translation with Conditional Adversarial Networks» [1]. Програмний код базовано на реалізації з офіційного репозиторію Tensorflow [13]. Для обох задач – колоризації та відновлення від пошкоджень – було проведено вичерпне навчання мережі й оцінено ефективність натренованих моделей.

Для підвищення якості зображень було використано модель SRGAN із роботи «Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network» [3], програмний код базовано на реалізації з джерела [14].

Для програмної реалізації обох моделей було використано фреймворк Tensorflow (Python) – один з найпотужніших та найбільш популярних фреймворків для у галузі машинного навчання.

На основі мереж було побудовано веб-застосунок, що дозволяє користувачу в інтерактивному форматі протестувати роботу моделей на власноруч обраних фото.

Для серверної частини було використано мову Python й бібліотеку FastAPI, а для фронтенд-частини – фреймворк Angular та бібліотеку компонентів Angular Material. Усі поставлені вимоги до функціональності було виконано. Окрім інтерактивного віджету для роботи з мережами, вебсайт містить подробиці про проєкт. Вебсайт доступний трьома мовами: англійською, українською, французькою.

Обидві моделі – Pix2pix та SRGAN – мають архітектуру GAN, тобто складаються з двох підмереж – генератора та дискримінатора. У процесі навчання ці підмережі «змагаються» між собою: генератор намагається згенерувати якомога більш правдоподібне зображення, у той час як дискримінатор учиться розрізняти згенеровані та справжні світлин.

Тренування мереж з такою архітектурою є достатньо непростим завданням. Частими явищами є колапс мережі, коли вона видає однаковий результат для різних вхідних даних, а також перенавчання, коли мережа генерує гарні результати лише для зображень, використаних для тренування.

Окремою складністю є пошук даних для тренування. У статті «NVIDIA Research Achieves AI Training Breakthrough Using Limited Datasets» [50] рекомендується від 50 000 до 100 000 тренувальних зображень для навчання високоякісної мережі GAN. Для порівняння: у роботі один з найбільших датасетів CLIC містив 1633 тренувальні зображення [44]. Пошук даних для тренування ускладнюється ще й специфікою задачі: наприклад, для колоризації потрібні «чисті» кольорові фото без жодних фільтрів, а для задачі відновлення від пошкоджень – старі та «вручну» відреставровані світлин. Для другого випадку потрібного датасету знайдено не було, тому навчання було проведене на основі «штучно» погіршених звичайних фото. «Погіршення» полягало в додаванні випадкового виду шуму, невеликій зміні яскравості або контрасту, додаванні «розриву» (для цього було взято маски реальних тріщин з датасету Crack Segmentation Dataset [43]).

Основні метрика, використана для оцінки ефективності тренування – SSIM, або індекс подібності. Для усіх трьох мереж подібність з оригінальним зображенням різко підвищувалася за перші 1000–5000 кроків й у подальшому сповільнювалась, доходячи до значення близько 0.5–0.8.

Усі три мережі пройшли більше 20 000 навчальних кроків. У цілому, результати роботи мереж є достатньо добрими: мережа для колоризації коректно розпізнає основні кольори зображення (зелень, небо тощо), мережа для відновлення від пошкоджень усуває шум та тріщини, мережа для підвищення роздільної здатності робить різкішими краї об'єктів. Однак, було помічено й деякі недоліки в роботі моделей. Наприклад, мережа для колоризації часто «недофарбовує» зображення й воно все ще виглядає «сірим». Мережа для відновлення від пошкоджень іноді лишає сейрозі артефакти на зображенні у вигляді кольорових плям (найімовірніше, результат того, що модель намагається «прибрати» шум з зображення). Можливим рішенням проблем є збільшення тривалості тренування (до 80 000–100 000 кроків).

Значну частину роботи зайняла теоретична складова. Було розглянуто та описано принцип роботи найпростіших мереж, основні математичні поняття та алгоритми, що застосовуються в машинному навчанні. Крім цього, було досліджено деякі поширені архітектури мереж для роботи з зображеннями, а також варіанти їхнього застосування.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Image-to-Image Translation with Conditional Adversarial Networks [Електронний ресурс] / P.Isola, J. Zhu, T. Zhou, A. Efros. – 2018. – Режим доступу до ресурсу: <https://arxiv.org/abs/1611.07004> – (11.06.2023).
2. Kundu R. Deep Learning for Image Super-Resolution [incl. Architectures] [Електронний ресурс] / Rohit Kundu. – 2022. – Режим доступу до ресурсу: <https://www.v7labs.com/blog/image-super-resolution-guide> – (11.06.2023).
3. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network [Електронний ресурс] / [C. Ledig, L. Theis, F. Huszar та ін.]. – 2017. – Режим доступу до ресурсу: <https://arxiv.org/abs/1609.04802> – (11.06.2023).
4. Ronneberger O. U-Net: Convolutional Networks for Biomedical Image Segmentation [Електронний ресурс] / O. Ronneberger, P. Fischer, T. Brox. – 2015. – Режим доступу до ресурсу: <https://arxiv.org/abs/1505.04597> – (11.06.2023).
5. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network [Електронний ресурс] / [W. Shi, J. Caballero, F. Huszár та ін.]. – 2016. – Режим доступу до ресурсу: <https://arxiv.org/abs/1609.05158> – (11.06.2023).
6. Huber J. Batch normalization in 3 levels of understanding [Електронний ресурс] / Johann Huber. – 2020. – Режим доступу до ресурсу: <https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338> – (11.06.2023).
7. Cross entropy [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Cross_entropy – (11.06.2023).
8. Dive into Deep Learning. 7.3. Padding and Stride [Електронний ресурс] – Режим доступу до ресурсу: https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html – (11.06.2023).

9. Brodtman Z. The Importance and Reasoning behind Activation Functions [Електронний ресурс] / Zack Brodtman. – 2021. – Режим доступу до ресурсу: <https://towardsdatascience.com/the-importance-and-reasoning-behind-activation-functions-4dc00e74db41> – (11.06.2023).
10. Verma Y. A Complete Understanding of Dense Layers in Neural Networks [Електронний ресурс] / Yugesh Verma. – 2021. – Режим доступу до ресурсу: <https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/> – (11.06.2023).
11. Time-Travel Rephotography [Електронний ресурс] / [X. Luo, X. Zhang, P. Yoo та ін.]. – 2021. – Режим доступу до ресурсу: <https://arxiv.org/abs/2012.12261> – (11.06.2023).
12. Deep Nostalgia™ Goes Viral! [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://blog.myheritage.com/2021/02/deep-nostalgia-goes-viral/> – (11.06.2023).
13. pix2pix: Image-to-image translation with a conditional GAN [Електронний ресурс] // 2019 – Режим доступу до ресурсу: <https://github.com/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb> – (11.06.2023).
14. SRGAN: Super Resolution Generative Adversarial Networks [Електронний ресурс] // 2021 – Режим доступу до ресурсу: <https://blog.paperspace.com/super-resolution-generative-adversarial-networks/> – (11.06.2023).
15. FastAPI [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/> – (11.06.2023).
16. Why TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: <https://www.tensorflow.org/about> – (11.06.2023).
17. What is TypeScript? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/> – (11.06.2023).
18. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> – (11.06.2023).

19. What is Angular? [Электронный ресурс] – Режим доступа до ресурсу: <https://angular.io/guide/what-is-angular> – (11.06.2023).
20. RxJS - Introduction [Электронный ресурс] – Режим доступа до ресурсу: <https://rxjs.dev/guide/overview> – (11.06.2023).
21. Overview of GAN Structure [Электронный ресурс] – Режим доступа до ресурсу: https://developers.google.com/machine-learning/gan/gan_structure?hl=en – (11.06.2023).
22. Hesse C. Image-to-Image Translation in Tensorflow [Электронный ресурс] / Christopher Hesse. – 2017. – Режим доступа до ресурсу: <https://affinelayer.com/pix2pix/> – (11.06.2023).
23. What Makes Paris Look like Paris? [Электронный ресурс] – Режим доступа до ресурсу: <http://graphics.cs.cmu.edu/projects/whatMakesParis/> – (11.06.2023).
24. Bringing Old Photos Back to Life [Электронный ресурс] / [Z. Wan, B. Zhang, D. Chen та ін.]. – 2020. – Режим доступа до ресурсу: <https://arxiv.org/abs/2004.09484> – (11.06.2023).
25. PixelShuffle [Электронный ресурс] – Режим доступа до ресурсу: <https://paperswithcode.com/method/pixelshuffle> – (11.06.2023).
26. Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition [Электронный ресурс] / K. Simonyan, A. Zisserman. – 2015. – Режим доступа до ресурсу: <https://arxiv.org/abs/1409.1556> – (11.06.2023).
27. Hotpot.ai [Электронный ресурс] – Режим доступа до ресурсу: <https://hotpot.ai/> – (11.06.2023).
28. Img2go [Электронный ресурс] – Режим доступа до ресурсу: <https://www.img2go.com> – (11.06.2023).
29. Photomyne [Электронный ресурс] – Режим доступа до ресурсу: <https://photomyne.com/colorize-app> – (11.06.2023).
30. Cultural landscape - Wikipedia [Электронный ресурс] – Режим доступа до ресурсу: https://en.wikipedia.org/wiki/Cultural_landscape#/media/File:Neckertal_201

50527-6384.jpg – (11.06.2023).

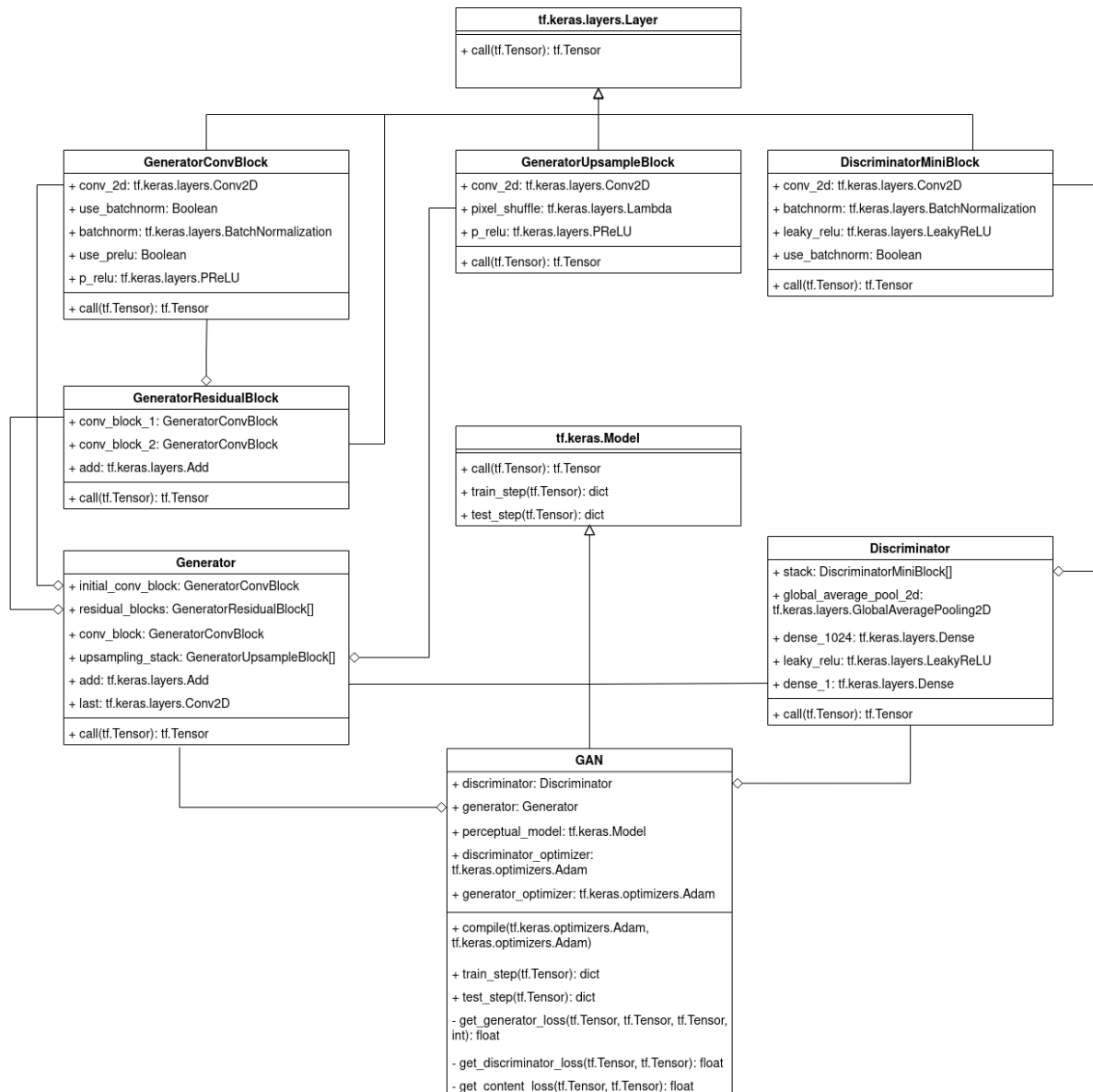
31. DIV2K Dataset [Електронний ресурс] – Режим доступу до ресурсу: <https://data.vision.ee.ethz.ch/cvl/DIV2K/> – (11.06.2023).
32. Goings J. Neural Networks by Analogy with Linear Regression [Електронний ресурс] / Joshua Goings – Режим доступу до ресурсу: <https://joshuagoings.com/2020/05/05/neural-network/> – (11.06.2023).
33. Dive into Deep Learning. 12. Optimization Algorithms [Електронний ресурс] – Режим доступу до ресурсу: http://www.d2l.ai/chapter_optimization/ – (11.06.2023).
34. CS231n Convolutional Neural Networks for Visual Recognition [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://cs231n.github.io/neural-networks-3/> – (11.06.2023).
35. Передавальна функція штучного нейрона [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Передавальна_функція_штучного_нейрона – (11.06.2023).
36. Radford A. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks [Електронний ресурс] / A. Radford, S. Chintala, L. Metz // 2016 – Режим доступу до ресурсу: <https://arxiv.org/abs/1511.06434> – (11.06.2023).
37. Deep Convolutional Generative Adversarial Network [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/tutorials/generative/dcgan#define_the_training_loop – (11.06.2023).
38. Melas-Kyriazi L. Image Colorization with Convolutional Neural Networks [Електронний ресурс] / Luke Melas-Kyriazi. – 2018. – Режим доступу до ресурсу: <https://lukemelas.github.io/image-colorization.html> – (11.06.2023).
39. What are RGB and YUV color spaces? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://dexonsystems.com/blog/rgb-yuv-color-spaces> – (11.06.2023).

40. Hsl extractor examples [Электронный ресурс] – Режим доступа до ресурсу: <https://onlinejpgtools.com/extract-hsl-channels-from-jpg> – (11.06.2023).
41. 'Lab' colour space [Электронный ресурс] – Режим доступа до ресурсу: <https://shutha.org/node/851> – (11.06.2023).
42. Performance tips [Электронный ресурс] – Режим доступа до ресурсу: https://www.tensorflow.org/datasets/performances?hl=en#shuffle_and_training – (11.06.2023).
43. Crack Segmentation Dataset [Электронный ресурс] – Режим доступа до ресурсу: <https://www.kaggle.com/datasets/lakshaymiddha/crack-segmentation-dataset> – (11.06.2023).
44. clic [Электронный ресурс] – Режим доступа до ресурсу: <https://www.tensorflow.org/datasets/catalog/clic> – (11.06.2023).
45. Skimage noises [Электронный ресурс] – Режим доступа до ресурсу: https://gist.github.com/Prasad9/28f6a2df8e8d463c6ddd040f4f6a028a?permalink_comment_id=2933012 – (11.06.2023).
46. Landscape color and grayscale images [Электронный ресурс] – Режим доступа до ресурсу: <https://www.kaggle.com/datasets/theblackmamba31/landscape-image-colorization?resource=download> – (11.06.2023).
47. Landscape Dataset [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/mahmoudnafifi/HistoGAN#landscape-dataset> – (11.06.2023).
48. Portrait Dataset [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/mahmoudnafifi/HistoGAN#portrait-dataset> – (11.06.2023).
49. Palermo F. <http://graphics.cs.cmu.edu/projects/historicalColor/> [Электронный ресурс] / F. Palermo, J. Hays, E. Alexei A.. – 2012. – Режим доступа до ресурсу: <http://graphics.cs.cmu.edu/projects/historicalColor/> – (11.06.2023).

50. Salian I. NVIDIA Research Achieves AI Training Breakthrough Using Limited Datasets [Электронный ресурс] / Isha Salian. – 2020. – Режим доступа до ресурсу: <https://blogs.nvidia.com/blog/2020/12/07/neurips-research-limited-data-gan/> – (11.06.2023).
51. image_denoise [Электронный ресурс] – Режим доступа до ресурсу: https://people.sc.fsu.edu/~jburkardt/c_src/image_denoise/image_denoise.html – (11.06.2023).
52. Noisy Photo Fix [Электронный ресурс] – Режим доступа до ресурсу: <http://www.thecreativitycenter.net/noisy> – (11.06.2023).

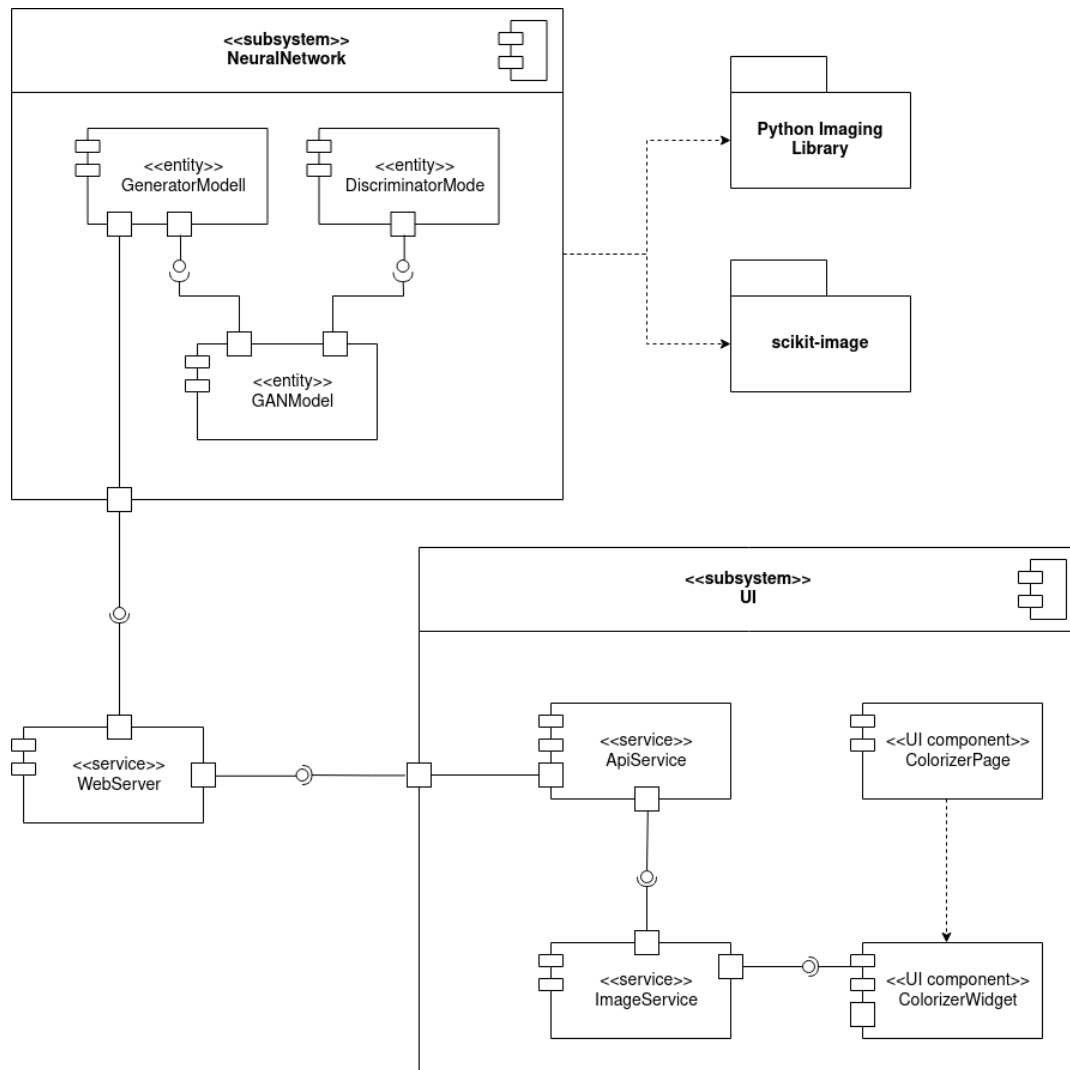
ДОДАТКИ

Додаток 1
Копії графічних елементів



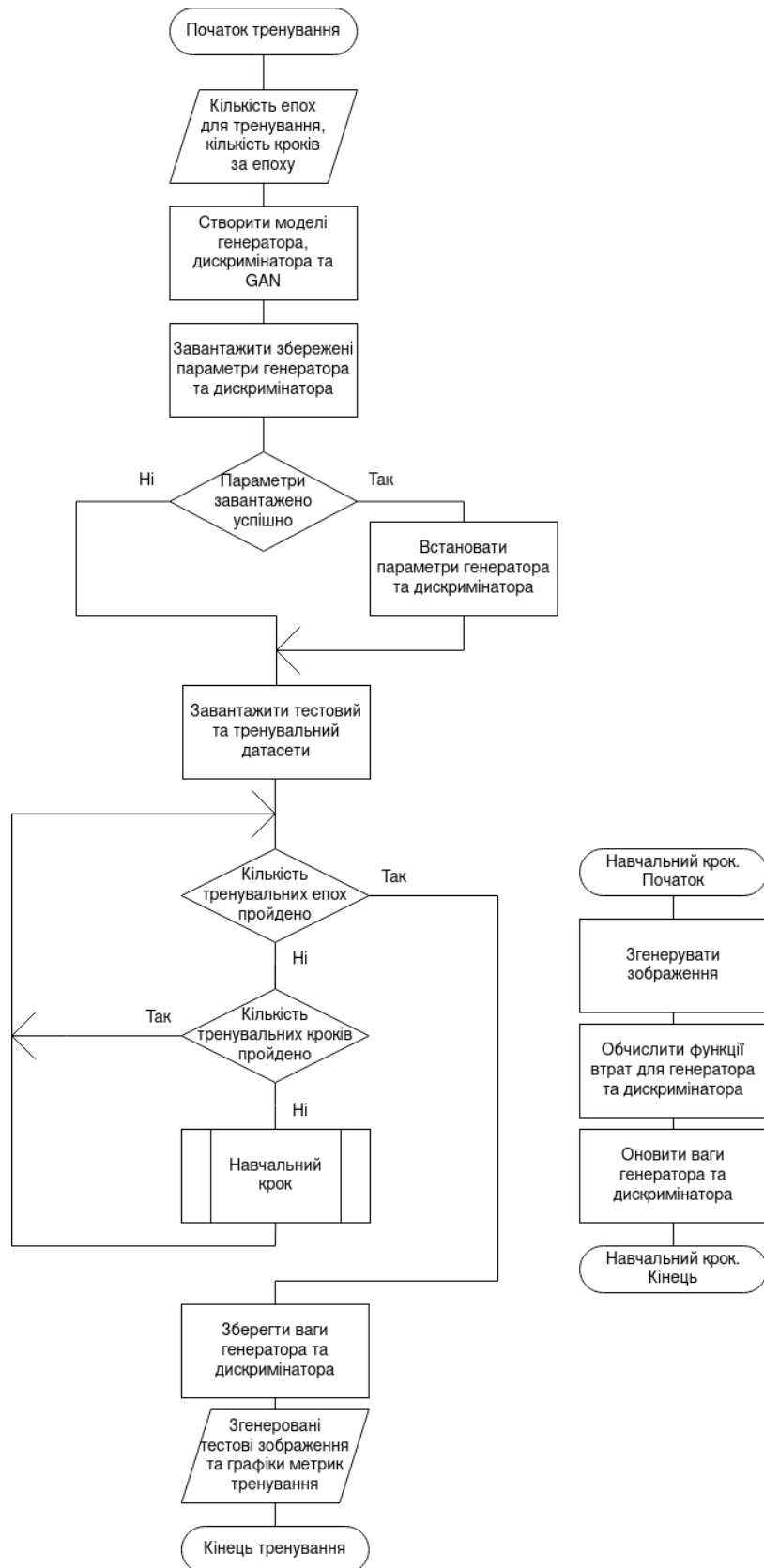
ДП.045480-06-99

Вебсервіс для реставрування та колоризації фотографій на основі нейронних мереж. Організація програмного коду нейронної мережі.
UML-діаграма класів

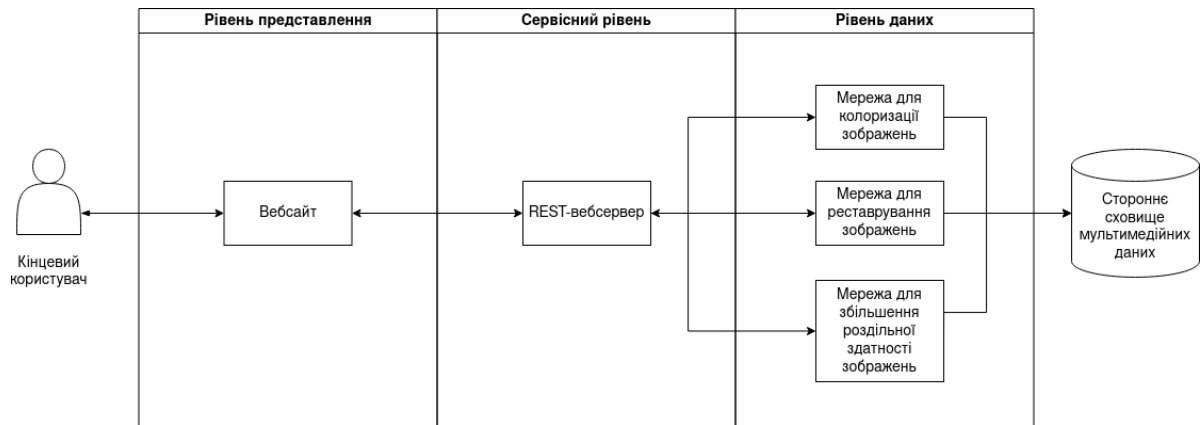


ДП.045480-07-99

Вебсервіс для реставрування та колоризації фотографій на основі нейронних мереж. Архітектура програми. UML-діаграма компонентів



Блок-схема тренування мережі
Тарелкіна К.О., група КП-93



Багаторівнева архітектура системи
Тарелкіна К.О., група КП-93

Додаток 2
Програмна реалізація мереж

Лістинг 1. Мережа Pix2pix. Програмний код базовано на реалізації з офіційного репозиторію Tensorflow: [13]

```
## Models

def random_normal_initializer():
    return tf.random_normal_initializer(0., 0.02)

class Encoder(tf.keras.layers.Layer):
    def __init__(self, n_filters,
                  kernel_size = 4, strides = 2,
                  use_leaky_relu = True, use_batchnorm = True,
                  **kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size,
            strides,
            activation = 'relu',
            kernel_initializer = random_normal_initializer(),
            padding = 'same' # padding='same' resolves error Negative dimension
size
        )

        self.use_batchnorm = use_batchnorm

        if use_batchnorm:
            self.batchnorm = tf.keras.layers.BatchNormalization()

        self.use_leaky_relu = use_leaky_relu

        if use_leaky_relu:
            self.leaky_relu = tf.keras.layers.LeakyReLU(alpha = 0.2)

    def call(self, inputs):
        x = self.conv_2d(inputs)

        if self.use_batchnorm:
            x = self.batchnorm(x)

        if self.use_leaky_relu:
            x = self.leaky_relu(x)

        return x

class Decoder(tf.keras.layers.Layer):
    def __init__(self, n_filters,
                  kernel_size = 4, strides = 2,
                  dropout = 0, use_batchnorm = True, use_relu = True,
                  **kwargs):
        super().__init__(**kwargs)

        self.conv_2d_transpose = tf.keras.layers.Conv2DTranspose(
            n_filters,
            kernel_size = kernel_size,
            kernel_initializer = random_normal_initializer(),
            strides = strides,
            padding = 'same'
        )

        self.use_batchnorm = use_batchnorm

        if self.use_batchnorm:
            self.batchnorm = tf.keras.layers.BatchNormalization()
```

```

        self.use_dropout = dropout > 0

        if self.use_dropout:
            self.dropout = tf.keras.layers.Dropout(dropout)

        self.use_relu = use_relu

        if self.use_relu:
            self.relu = tf.keras.layers.ReLU()

    def call(self, inputs):
        x = self.conv_2d_transpose(inputs)

        if self.use_batchnorm:
            x = self.batchnorm(x)

        if self.use_dropout:
            x = self.dropout(x)

        if self.use_relu:
            x = self.relu(x)

        return x

### Generator

# Author(s): The TensorFlow Authors
# Date: Mar 9, 2023
# Title of program/source code: pix2pix: Image-to-image translation with a
conditional GAN
# Type: source code
# Availability:
https://github.com/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb

class Generator(tf.keras.Model):
    def __init__(self, n_filters = 64, **kwargs):
        super().__init__(**kwargs)

        self.encode_stack = [
            Encoder(n_filters, use_leaky_relu = False, use_batchnorm = False),
            Encoder(n_filters * 2),
            Encoder(n_filters * 4),
            Encoder(n_filters * 8),
            Encoder(n_filters * 8),
            Encoder(n_filters * 8),
            Encoder(n_filters * 8),
            Encoder(n_filters * 8, use_batchnorm = False),
        ]

        self.decode_stack = [
            Decoder(n_filters * 8, dropout = 0.5),
            Decoder(n_filters * 8, dropout = 0.5),
            Decoder(n_filters * 8, dropout = 0.5),
            Decoder(n_filters * 8),
            Decoder(n_filters * 4),
            Decoder(n_filters * 2),
            Decoder(n_filters)
        ]

        self.concatenate = tf.keras.layers.Concatenate()

        self.last = Decoder(N_OUTPUT_CHANNELS, use_batchnorm = False,
            use_relu = False)

        self.activation = tf.keras.layers.Activation('tanh')

```



```

def call(self, inputs):
    skip_connections = []

    x = inputs

    # Downsampling

    for encode_layer in self.encode_stack:
        x = encode_layer(x)
        skip_connections.insert(0, x)

    # Upsampling

    for decode_layer, skip_connection in zip(self.decode_stack,
skip_connections[1:]):
        x = decode_layer(x)
        x = self.concatenate([x, skip_connection])

    x = self.last(x)

    x = self.activation(x)

    return x

generator = Generator()

### Discriminator

class Discriminator(tf.keras.Model):
    def __init__(self, n_filters = 64, **kwargs):
        super().__init__(**kwargs)

        self.encoding_stack = [
            Encoder(n_filters, kernel_size = 4, use_leaky_relu = False,
use_batchnorm = False),
            Encoder(n_filters * 2),
            Encoder(n_filters * 4, strides = 1),
            Encoder(1, strides = 1, use_batchnorm = False)
        ]

        self.concatenate = tf.keras.layers.Concatenate()

        self.activation = tf.keras.layers.Activation('sigmoid')

    def call(self, generated, ground_truth):
        x = self.concatenate([generated, ground_truth])

        for layer in self.encoding_stack:
            x = layer(x)

        x = self.activation(x)

        return x

discriminator = Discriminator()

```

Лістинг 2. Мережа SRGAN. Програмний код базовано на реалізації з таких джерел: [13] (офіційний репозиторій Tensorflow), [14]

```
## Models

def random_normal_initializer():
    return tf.random_normal_initializer(0., 0.02)

### Discriminator

class DiscriminatorMiniBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters, strides = 1, use_batchnorm = True,
**kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size = 3,
            strides = strides,
            kernel_initializer = random_normal_initializer(),
            padding = 'same'
        )

        self.use_batchnorm = use_batchnorm

        if use_batchnorm:
            self.batchnorm = tf.keras.layers.BatchNormalization()

        self.leaky_relu = tf.keras.layers.LeakyReLU(alpha = 0.2)

    def call(self, inputs):
        x = self.conv_2d(inputs)

        if self.use_batchnorm:
            x = self.batchnorm(x)

        x = self.leaky_relu(x)

        return x

class Discriminator(tf.keras.Model):
    def __init__(self, n_filters = 64, **kwargs):
        super().__init__(**kwargs)

        self.stack = stack = [
            DiscriminatorMiniBlock(n_filters, use_batchnorm = False),
            DiscriminatorMiniBlock(n_filters, strides = 2),
            DiscriminatorMiniBlock(n_filters * 2),
            DiscriminatorMiniBlock(n_filters * 2, strides = 2),
            DiscriminatorMiniBlock(n_filters * 4),
            DiscriminatorMiniBlock(n_filters * 4, strides = 2),
            DiscriminatorMiniBlock(n_filters * 8),
            DiscriminatorMiniBlock(n_filters * 8, strides = 2),
        ]

        self.global_average_pool_2d =
tf.keras.layers.GlobalAveragePooling2D()

        self.dense_1024 = tf.keras.layers.Dense(1024, activation='sigmoid')
        self.leaky_relu = tf.keras.layers.LeakyReLU(alpha = 0.2)
        self.dense_1 = tf.keras.layers.Dense(1, activation='sigmoid')

    def call(self, inputs):
        x = inputs
```

```

        for layer in self.stack:
            x = layer(x)

        x = self.global_average_pool_2d(x)

        x = self.dense_1024(x)
        x = self.leaky_relu(x)
        x = self.dense_1(x)

        return x

discriminator = Discriminator()

### Generator

# Author(s): Bharath K
# Date: 2021
# Title of program/source code: SRGAN: Super Resolution Generative
Adversarial Networks
# Type: source code
# Availability:
https://blog.paperspace.com/super-resolution-generative-adversarial-networks/

N_UPSAMPLES_PER_SCALE = {
    2: 1,
    4: 2,
    8: 3
}

def pixel_shuffle(scale):
    return lambda x: tf.nn.depth_to_space(x, scale)

class GeneratorConvBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters, kernel_size = 3, use_prelu = False,
use_batchnorm = False, batchnorm_momentum = 0.99, **kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size = 3,
            strides = 1,
            kernel_initializer = random_normal_initializer(),
            padding = 'same'
        )

        self.use_batchnorm = use_batchnorm

        if use_batchnorm:
            self.batchnorm = tf.keras.layers.BatchNormalization(momentum =
batchnorm_momentum)

        self.use_prelu = use_prelu

        if use_prelu:
            self.p_relu = tf.keras.layers.PReLU()

    def call(self, inputs):
        x = self.conv_2d(inputs)

        if self.use_batchnorm:
            x = self.batchnorm(x)

        if self.use_prelu:
            x = self.p_relu(x)

        return x

```

```

class GeneratorResidualBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters, **kwargs):
        super().__init__(**kwargs)

        self.conv_block_1 = GeneratorConvBlock(
            n_filters,
            use_batchnorm = True, batchnorm_momentum = 0.8,
            use_prelu = True
        )

        self.conv_block_2 = GeneratorConvBlock(
            n_filters,
            use_batchnorm = True, batchnorm_momentum = 0.8
        )

        self.add = tf.keras.layers.Add()

    def call(self, inputs):
        x = self.conv_block_1(inputs)
        x = self.conv_block_2(x)
        x = self.add([inputs, x])
        return x

class GeneratorUpsampleBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters, **kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size = 3,
            kernel_initializer = random_normal_initializer(),
            padding = 'same'
        )

        self.pixel_shuffle = tf.keras.layers.Lambda(pixel_shuffle(scale =
2))

        self.p_relu = tf.keras.layers.PReLU()

    def call(self, inputs):
        x = self.conv_2d(inputs)
        x = self.pixel_shuffle(x)
        x = self.p_relu(x)
        return x

class Generator(tf.keras.Model):
    def __init__(self, n_filters = 64, n_residual_blocks = 16, **kwargs):
        super().__init__(**kwargs)

        self.initial_conv_block = GeneratorConvBlock(n_filters, kernel_size
= 9, use_prelu = True)

        # Residual blocks

        self.residual_blocks = []

        for _ in range(n_residual_blocks):
            self.residual_blocks.append(GeneratorResidualBlock(n_filters))

        self.conv_block = GeneratorConvBlock(n_filters, use_batchnorm =
True)

        self.add = tf.keras.layers.Add()

        # Upsampling

        n_upsamples = N_UPSAMPLES_PER_SCALE[SCALE]

```

```

self.upsampling_stack = []

for _ in range(n_upsamples):
    self.upsampling_stack.append(GeneratorUpsampleBlock(n_filters * 4))

# Last layer

self.last = tf.keras.layers.Conv2D(
    3,
    kernel_size = 9,
    kernel_initializer = random_normal_initializer(),
    padding = 'same',
    activation='tanh'
)

def call(self, inputs):
    x = self.initial_conv_block(inputs)

    initial_skip_connection = x

    # Residual blocks

    for residual_block in self.residual_blocks:
        x = residual_block(x)

    x = self.conv_block(x)
    x = self.add([initial_skip_connection, x])

    # Upsampling

    for upsample in self.upsampling_stack:
        x = upsample(x)

    # Last layer

    x = self.last(x)

    return x

generator = Generator()

```

Додаток 3
Копія презентації

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



ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
КОМП'ЮТЕРНИХ СИСТЕМ

ВЕБСЕРВІС ДЛЯ РЕСТАВРУВАННЯ ТА КОЛОРИЗАЦІЇ ФОТОГРАФІЙ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Виконала: Тарелкіна Катерина Олексіївна, КП-93

Керівник: д.т.н., доцент Сулема Євгенія Станіславівна

Київ – 2023

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити програмне забезпечення для автоматичної колоризації та реставрування фотографій.

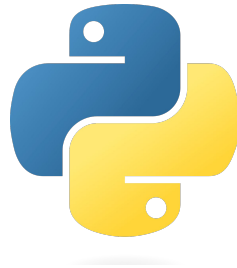
Завдання:

1. Опрацювання літературних джерел.
2. Аналіз існуючих архітектур нейронних мереж.
3. Аналіз продуктів-аналогів.
4. Побудова програмного застосунку на основі відомих архітектур нейронних мереж.
5. Тестування застосунку.

АКТУАЛЬНІСТЬ

- Обмеження традиційних алгоритмів (наприклад, матричних фільтрів).
- Широке застосування нейронних мереж.
- Перспективність галузі.

ЗАСОБИ РОЗРОБКИ



Python,
фреймворк
TensorFlow



TypeScript,
фреймворк
Angular

АРХІТЕКТУРА СИСТЕМИ

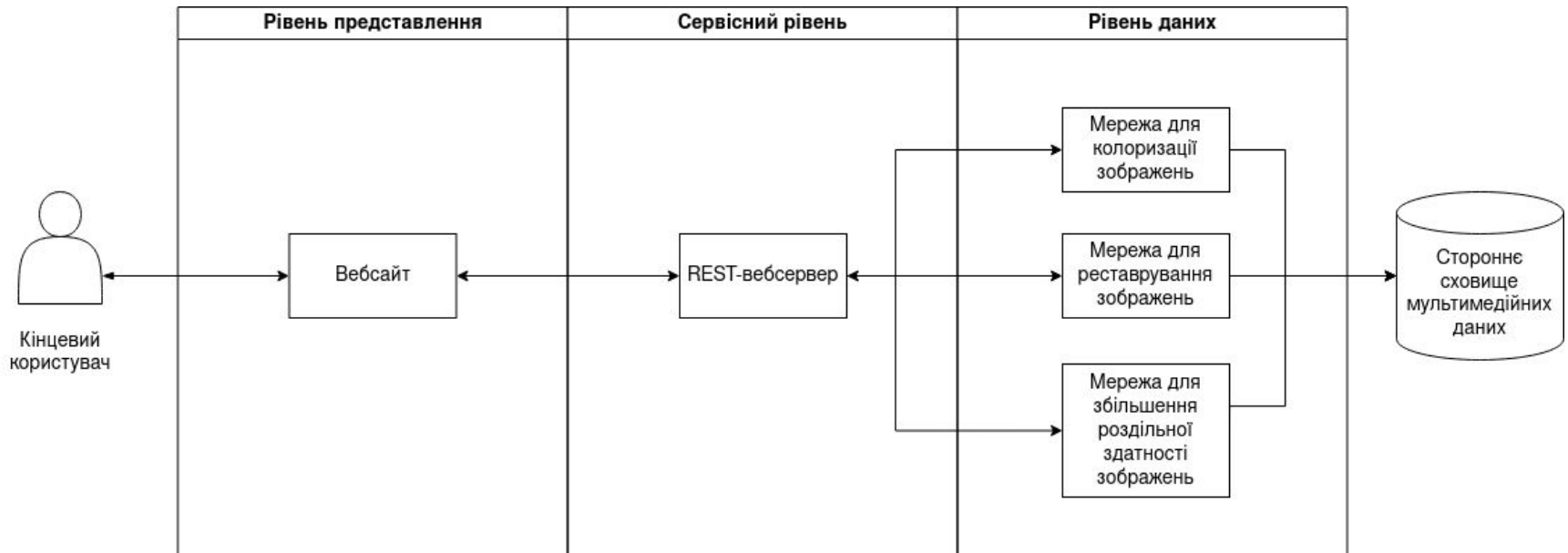


Схема багаторівневої архітектури програми

АРХІТЕКТУРА НЕЙРОННИХ МЕРЕЖ. GAN (generative adversarial network)

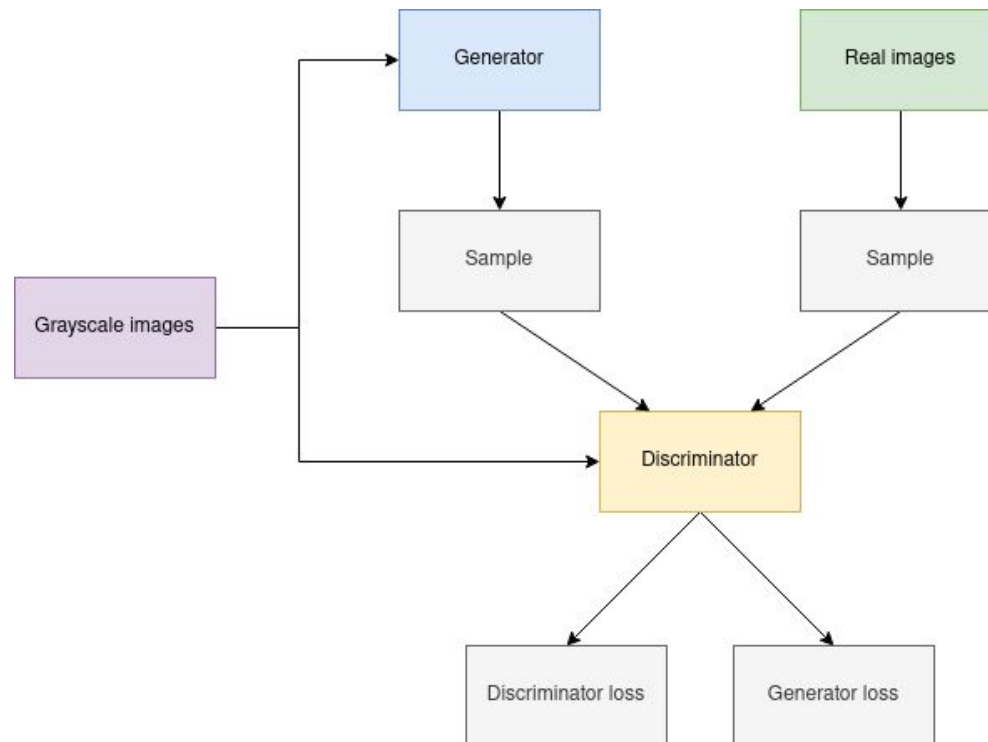
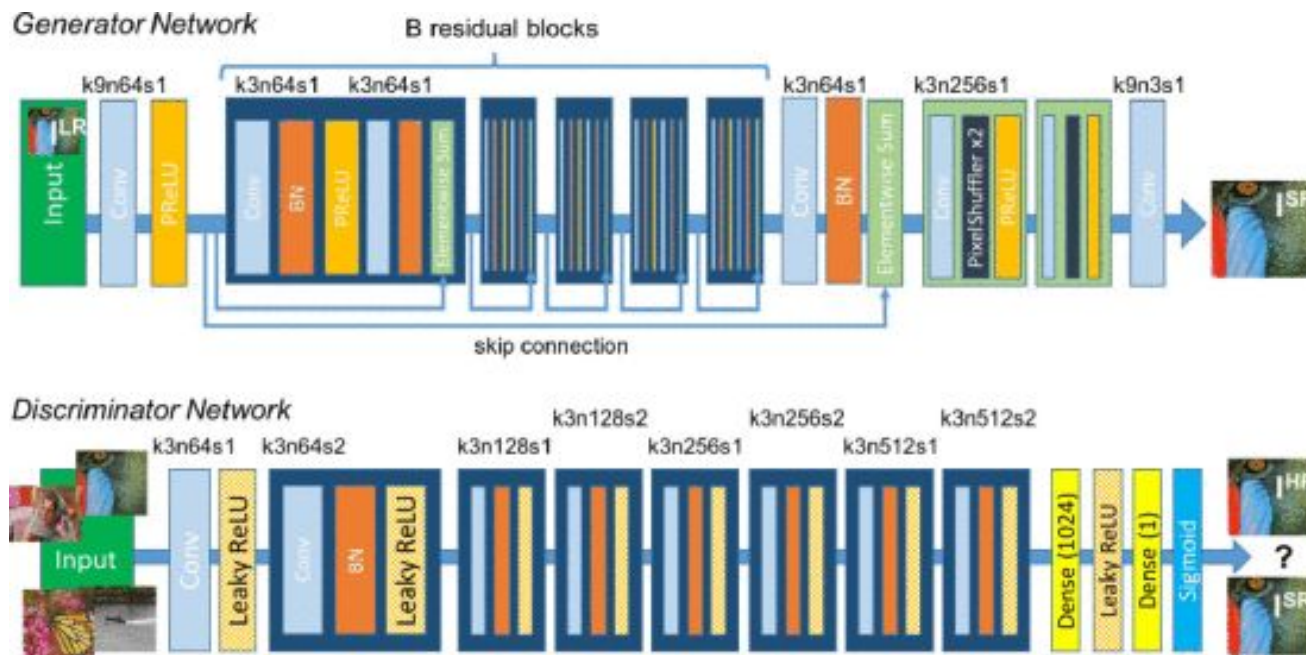


Рисунок створено на основі схеми із
[https://developers.google.com/machine-learning/gan/gan_structu
re?hl=en](https://developers.google.com/machine-learning/gan/gan_structure?hl=en)

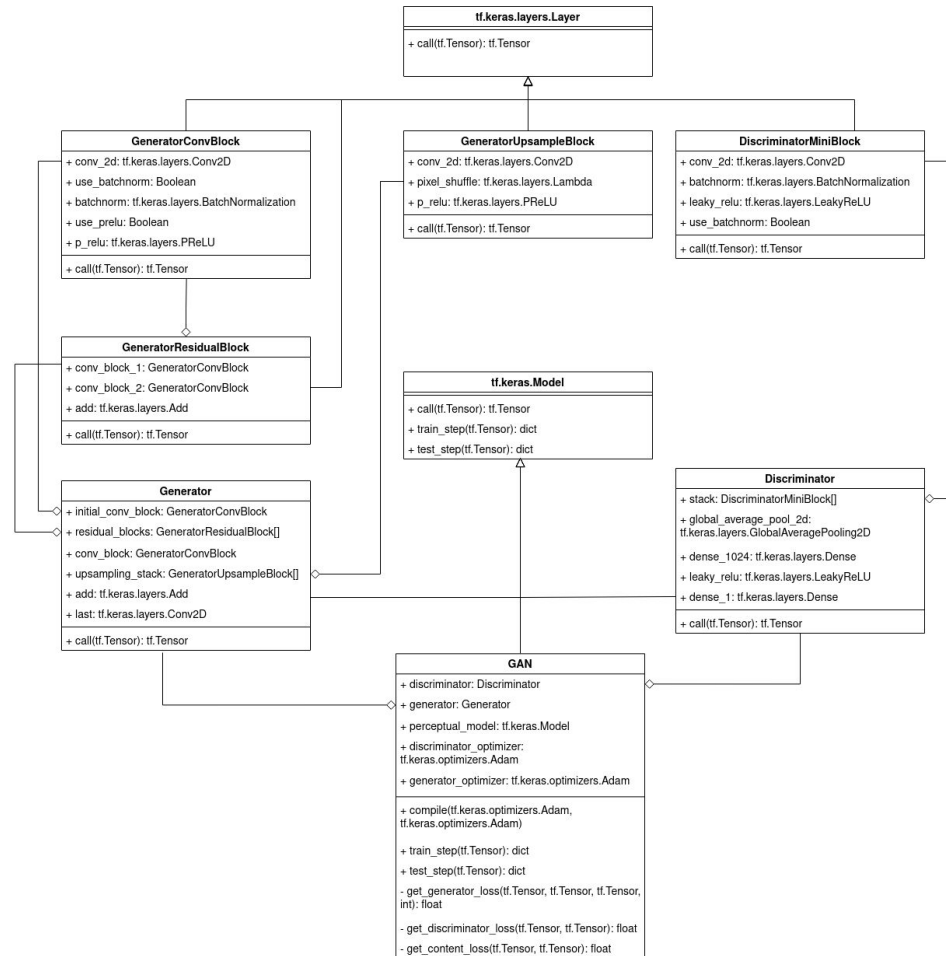
АРХИТЕКТУРА МЕРЕЖІ SRGAN



Генератор (згори) та дискримінатор (внизу).

Джерело: <https://arxiv.org/abs/1609.04802>

АРХИТЕКТУРА МЕРЕЖІ SRGAN



UML-діаграма класів

ПРОГРАМНА РЕАЛІЗАЦІЯ МЕРЕЖІ SRGAN

```
class DiscriminatorMiniBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters, strides = 1,
use_batchnorm = True, **kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size = 3,
            strides = strides,
            kernel_initializer =
random_normal_initializer(),
            padding = 'same'
        )

        self.use_batchnorm = use_batchnorm

        if use_batchnorm:
            self.batchnorm =
tf.keras.layers.BatchNormalization()

        self.leaky_relu = tf.keras.layers.LeakyReLU(alpha activation='sigmoid')
= 0.2)

    def call(self, inputs):
        x = self.conv_2d(inputs)

        if self.use_batchnorm:
            x = self.batchnorm(x)

        x = self.leaky_relu(x)

        return x

class Discriminator(tf.keras.Model):
    def __init__(self, n_filters = 64, **kwargs):
        super().__init__(**kwargs)

        self.stack = [
            DiscriminatorMiniBlock(n_filters, use_batchnorm =
False),
            DiscriminatorMiniBlock(n_filters, strides = 2),
            DiscriminatorMiniBlock(n_filters * 2),
            DiscriminatorMiniBlock(n_filters * 2, strides = 2),
            DiscriminatorMiniBlock(n_filters * 4),
            DiscriminatorMiniBlock(n_filters * 4, strides = 2),
            DiscriminatorMiniBlock(n_filters * 8),
            DiscriminatorMiniBlock(n_filters * 8, strides = 2),
        ]

        self.global_average_pool_2d =
tf.keras.layers.GlobalAveragePooling2D()

        self.dense_1024 = tf.keras.layers.Dense(1024,
            self.leaky_relu = tf.keras.layers.LeakyReLU(alpha =
0.2)

        self.dense_1 = tf.keras.layers.Dense(1,
activation='sigmoid')

        # ...
```

АРХІТЕКТУРА МЕРЕЖІ PIX2PIX

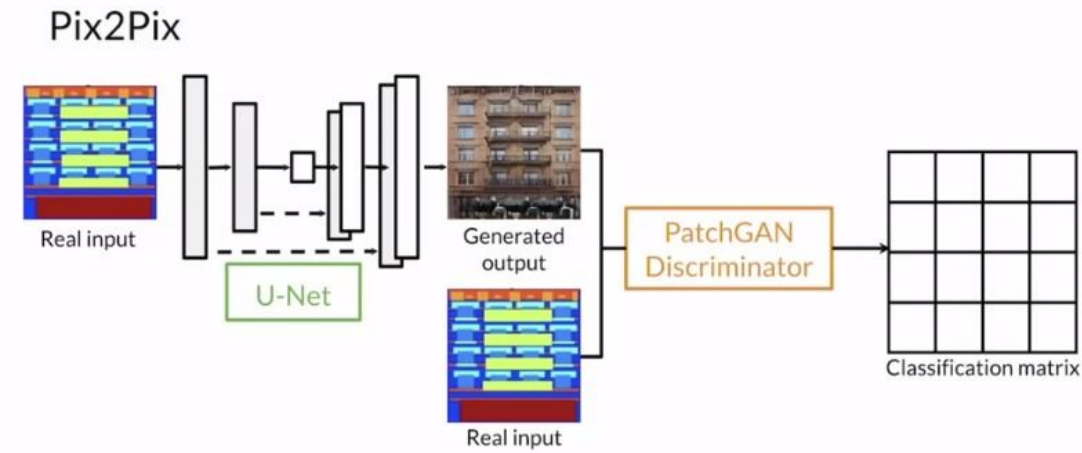


Image available from: <https://arxiv.org/abs/1611.07004>

Джерело:

<https://www.haikutechcenter.com/2020/10/pix2pix-gan-architecture-for-image-to.html>

ПРОГРАМНА РЕАЛІЗАЦІЯ МЕРЕЖІ PИX2PИX

```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, n_filters,
                kernel_size = 4, strides = 2,
                use_leaky_relu = True,
                use_batchnorm = True,
                **kwargs):
        super().__init__(**kwargs)

        self.conv_2d = tf.keras.layers.Conv2D(
            n_filters,
            kernel_size,
            strides,
            activation = 'relu',
            kernel_initializer =
random_normal_initializer(),
            padding = 'same' # padding='same' resolves error
Negative dimension size
        )

        self.use_batchnorm = use_batchnorm

        if use_batchnorm:
            self.batchnorm =
tf.keras.layers.BatchNormalization()

        self.use_leaky_relu = use_leaky_relu

        if use_leaky_relu:
            self.leaky_relu =
tf.keras.layers.LeakyReLU(alpha = 0.2)

# ...
```

```
class Discriminator(tf.keras.Model):
    def __init__(self, n_filters = 64, **kwargs):
        super().__init__(**kwargs)

        self.encoding_stack = [
            Encoder(n_filters, kernel_size = 4, use_leaky_relu =
False, use_batchnorm = False),
            Encoder(n_filters * 2),
            Encoder(n_filters * 4, strides = 1),
            Encoder(1, strides = 1, use_batchnorm = False)
        ]

        self.concatenate = tf.keras.layers.Concatenate()

        self.activation = tf.keras.layers.Activation('sigmoid')

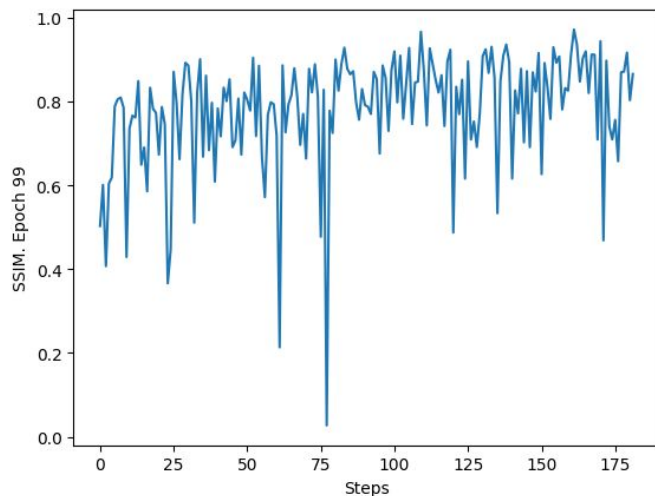
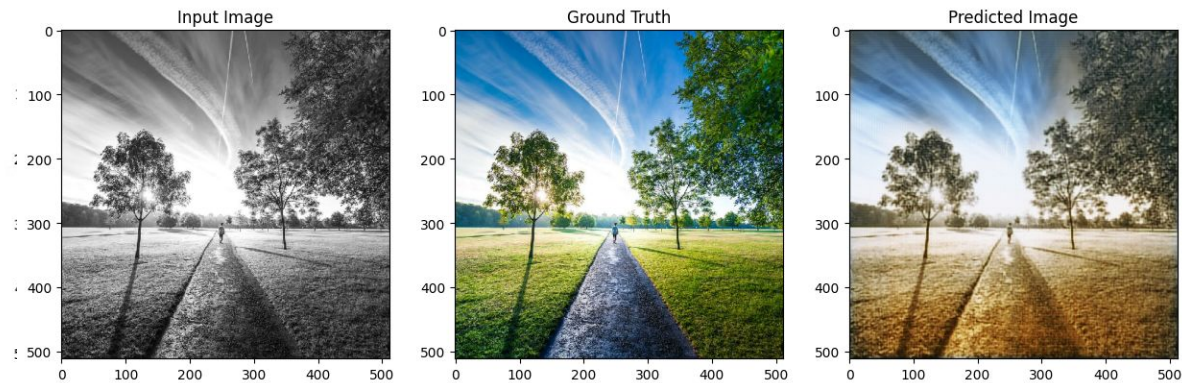
    def call(self, generated, ground_truth):
        x = self.concatenate([generated, ground_truth])

        for layer in self.encoding_stack:
            x = layer(x)

        x = self.activation(x)

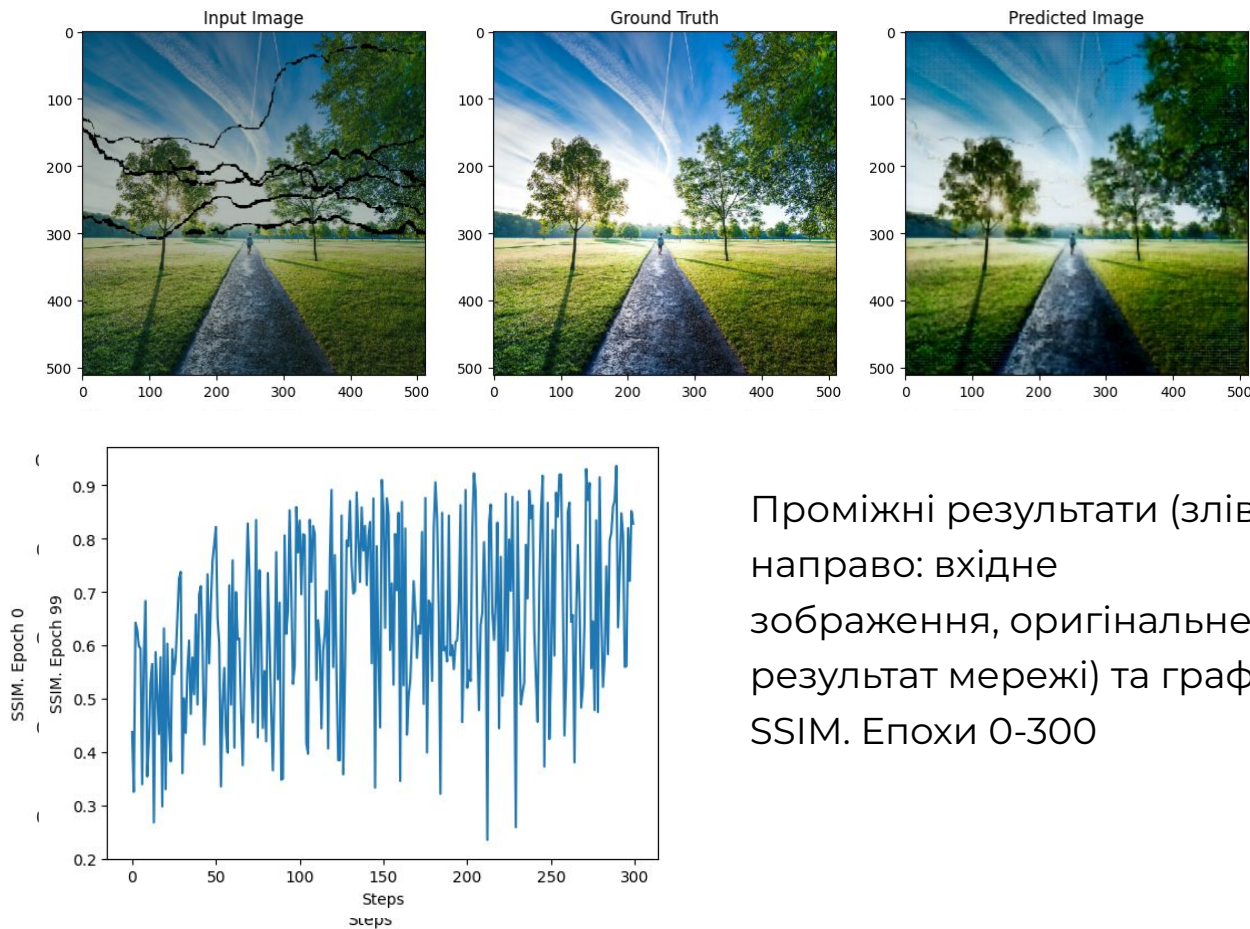
        return x
```

НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ. КОЛОРИЗАЦІЯ ЗОБРАЖЕНЬ



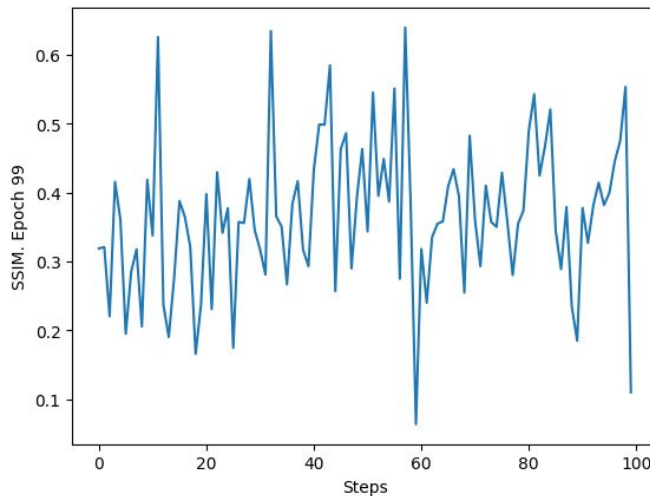
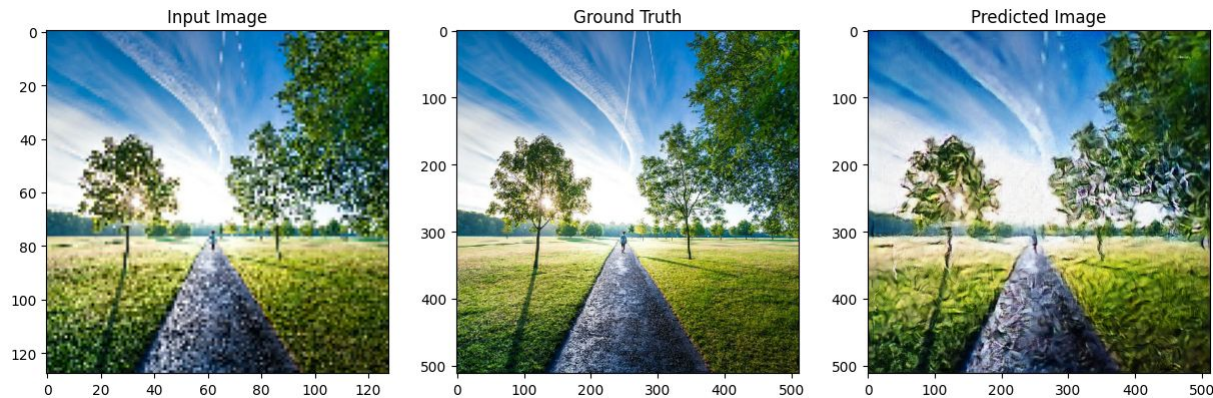
Проміжні результати (зліва
направо: вхідне
зображення, оригінальне,
результат мережі) та графік
SSIM. Епохи 0-180

НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ. ВІДНОВЛЕННЯ ЗОБРАЖЕНЬ ВІД ПОШКОДЖЕНЬ



Проміжні результати (зліва
направо: вхідне
зображення, оригінальне,
результат мережі) та графік
SSIM. Епохи 0-300

НАВЧАННЯ НЕЙРОННИХ МЕРЕЖ. ЗБІЛЬШЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ

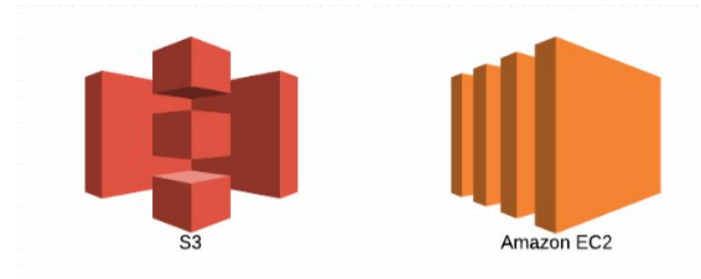


Проміжні результати (зліва
направо: вхідне
зображення, оригінальне,
результат мережі) та графік
SSIM. Епохи 0-100

ТЕСТУВАННЯ ВЕБСЕРВЕРУ

1. Метрики для оцінки мереж: SSIM, проміжні результати, час проходження навчального кроку.
2. Тестування кожної мережі на тестових зображеннях. Візуальна оцінка результатів, зняття метрик: SSIM, MSE, час обробки зображення.
3. Unit-тестування клієнтської частини.
4. Ручне тестування веб-інтерфейсу.

ВЕБСЕРВІС НА ХМАРНІЙ ПЛАТФОРМІ AWS



<http://colorizer-ui.s3-website-us-east-1.amazonaws.com/>

Instances (1/1) info

Find instance by attribute or tag (case-sensitive)

Instance state: running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
colorizer-serve...	i-03c7bf9c39c5a971b	Running	t2.medium	2/2 checks passed	No alarms	us-east-1b

Instance: i-03c7bf9c39c5a971b (colorizer-server-tf)

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Instance summary info

Instance ID: i-03c7bf9c39c5a971b (colorizer-server-tf)

Public IPv4 address: 3.95.229.52 | open address

Private IPv4 address: ec2-3-95-229-52.compute-1.amazonaws.com | open address

Instance state: Running

Private IP DNS name (IPv4 only): ip-172-31-93-149.ec2.internal

Instance type: t2.medium

VPC ID: vpc-00ce60a3901375180

Auto-assigned IP address: 3.95.229.52 [Public IP]

Amazon S3 > Buckets > colorizer-ui

colorizer-ui info

Publicly accessible

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (8)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 Inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder

Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
3rdpartylicenses.txt	txt	June 15, 2023, 17:47:18 (UTC+03:00)	15.4 KB	Standard
assets/	Folder	-	-	-
favicon.ico	ico	June 15, 2023, 17:47:20 (UTC+03:00)	948.0 B	Standard
index.html	html	June 15, 2023, 17:47:20 (UTC+03:00)	10.5 KB	Standard
main.a7b147fac3167a06.js	js	June 15, 2023, 17:47:20 (UTC+03:00)	764.6 KB	Standard
polyfills.c0dd865ae3e63aab.js	js	June 15, 2023, 17:47:20 (UTC+03:00)	33.1 KB	Standard
runtime.0c2641156b5fa202.js	js	June 15, 2023, 17:47:20 (UTC+03:00)	1.1 KB	Standard
styles.26ef3c9cf0f3bf61.css	css	June 15, 2023, 17:47:20 (UTC+03:00)	169.4 KB	Standard

ВИСНОВКИ

- Виконано аналіз існуючих рішень.
- Розроблено завершений програмний продукт на основі відомих архітектур нейронних мереж.
- Проаналізовано тренування та результати роботи мереж.
- Проведено тестування програми різноманітними засобами.

Дякую за увагу!

Додаток 4

Приклад роботи продуктів-аналогів

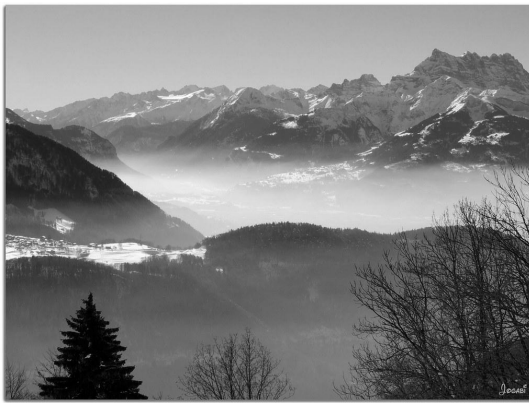


Рис. 1. Колоризація фотографій
Джерело зображень: Landscape Dataset [47].
Результат роботи сервісу img2go [28]

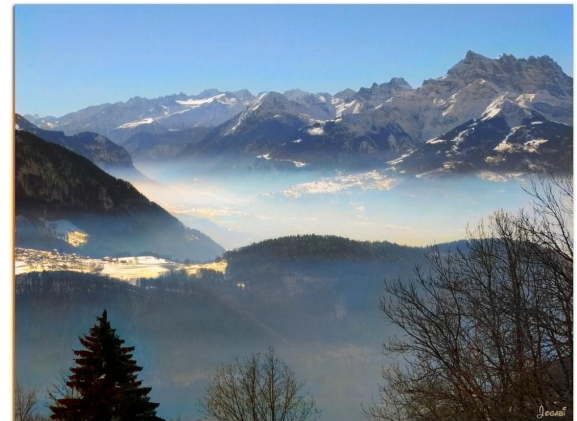


Рис. 2. Колоризація фотографій
Джерело зображень: Landscape Dataset [47].
Результат роботи сервісу Photomyne [29]

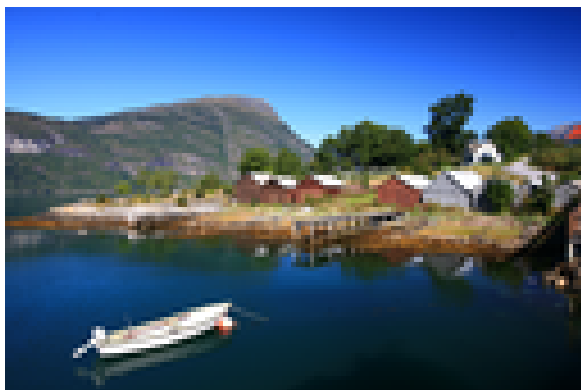
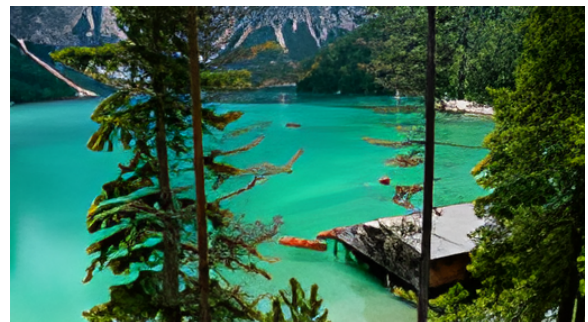
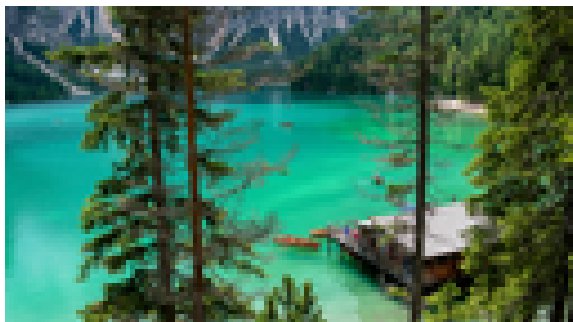


Рис. 3. Покращення якості фотографій
Джерело зображень: Landscape Dataset [47].
Результат роботи сервісу img2go [28]

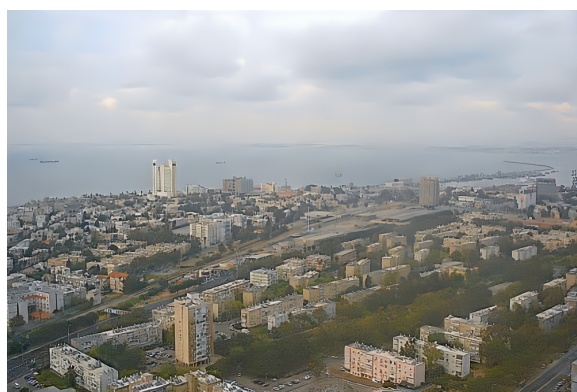
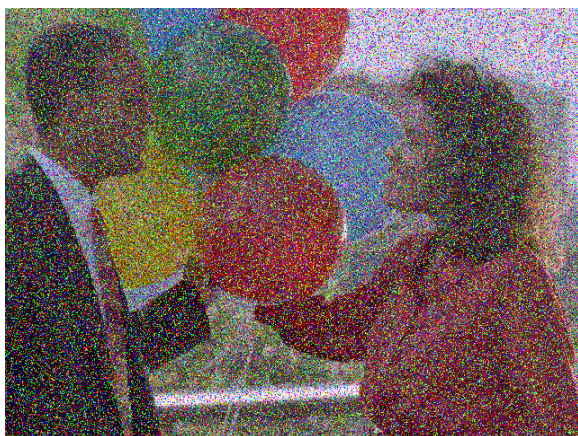


Рис. 4. Реставрування фотографій

Джерела фотографій: [51], [52].

Результат роботи сервісу Hotpot.ai [27]

Додаток 5
Тренування мереж

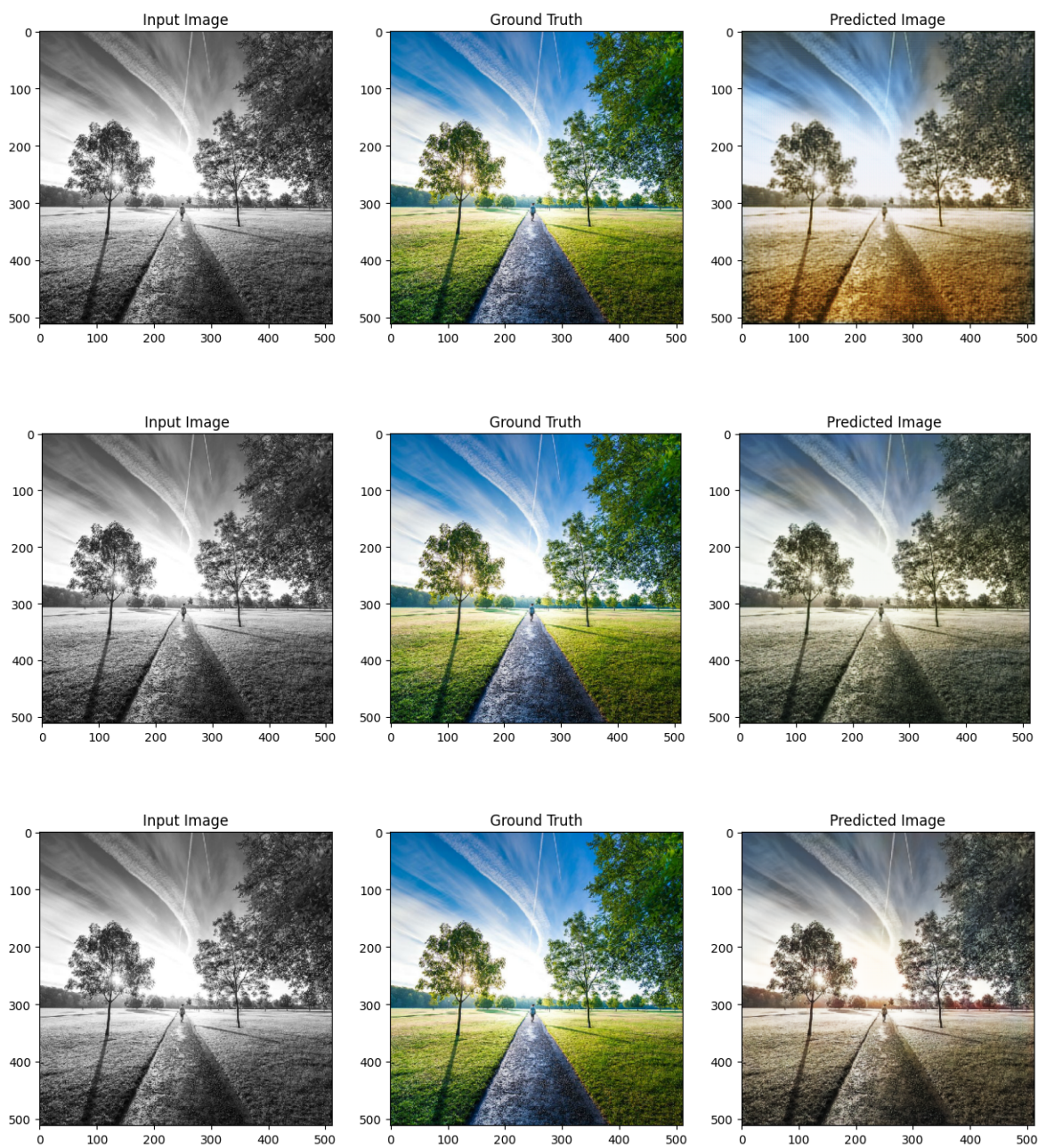


Рис. 1. Тренування мережі для колоризації зображень.

Результат роботи генератора, згори вниз: після 0-ї епохи, після 180-ї епохи, після 380-ї епохи (зліва направо: вхідне, оригінальне, згенероване зображення)

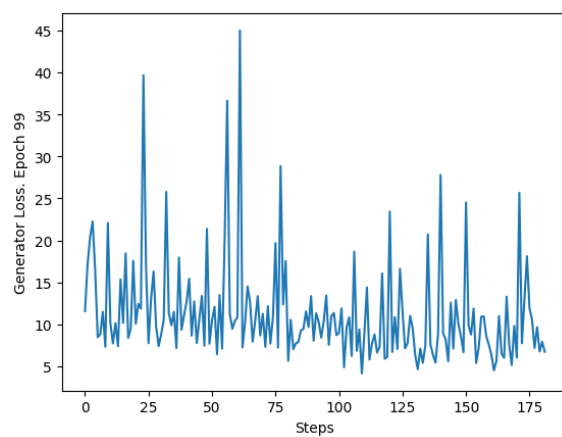
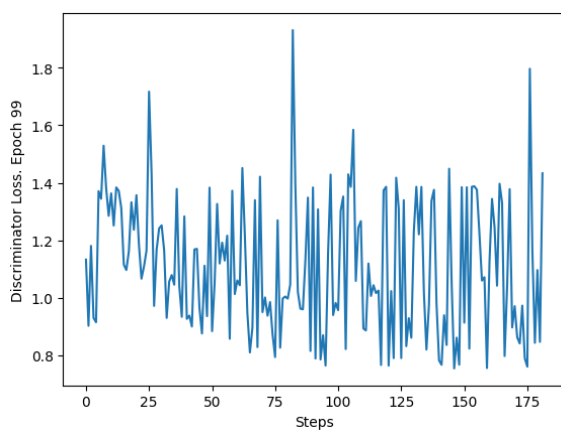
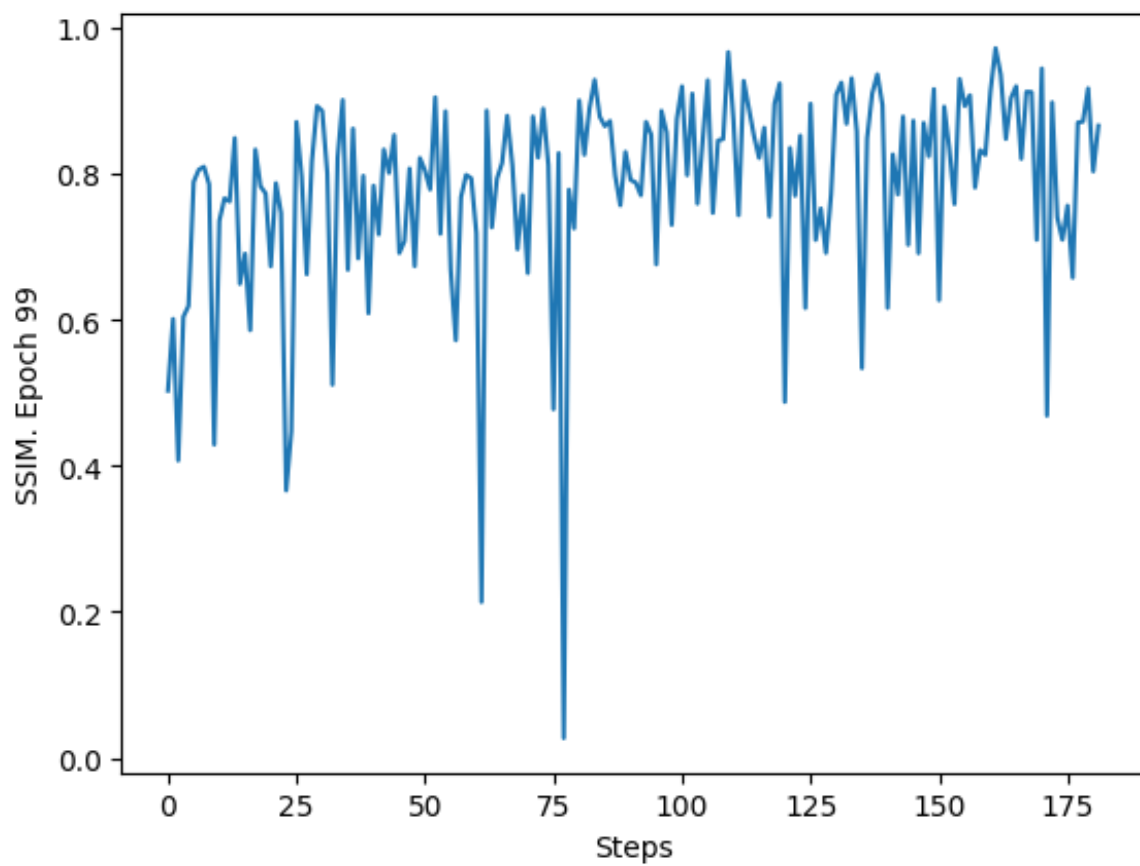


Рис. 2. Тренування мережі для колоризації зображень.

Епохи 0-180. SSIM, втрати генератора (справа) та дискримінатора (зліва)

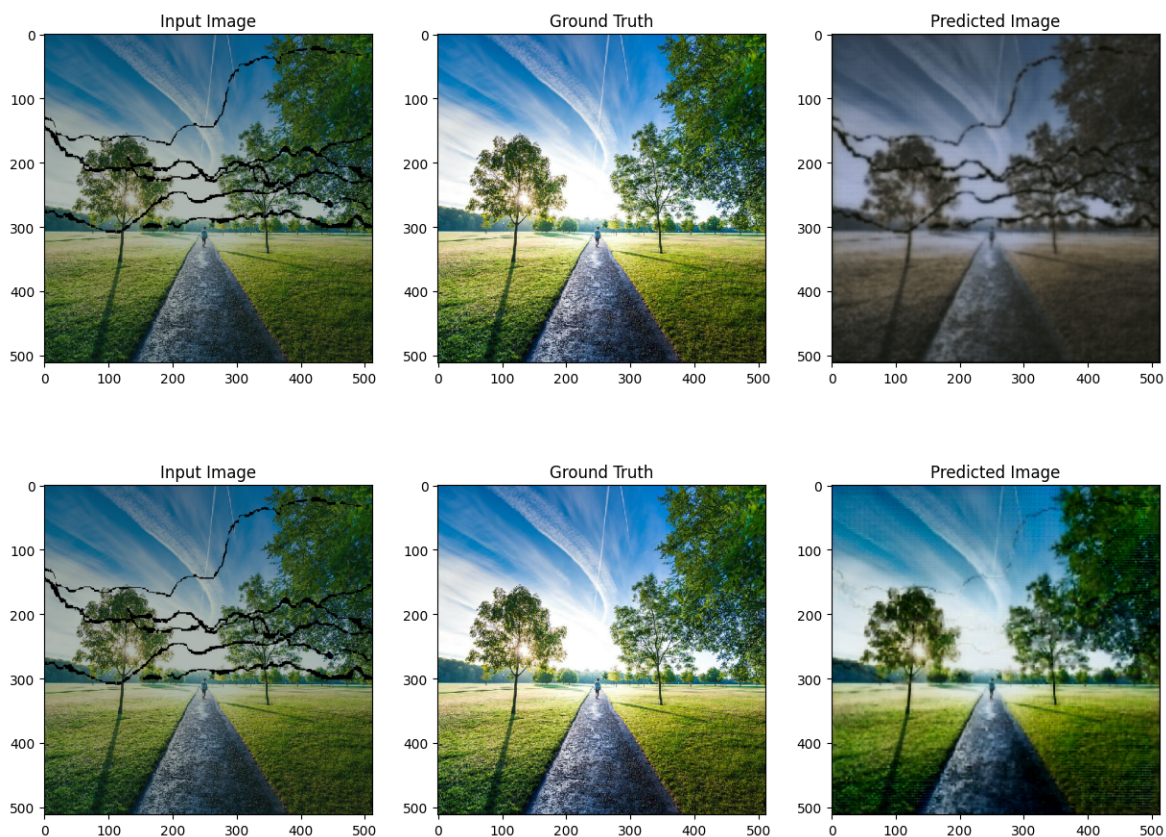


Рис. 3. Тренування мережі для реставрування зображень.

Результат роботи генератора, згори вниз: після 0-ї епохи, після 300-ї епохи
(зліва направо: вхідне, оригінальне, згенероване зображення)

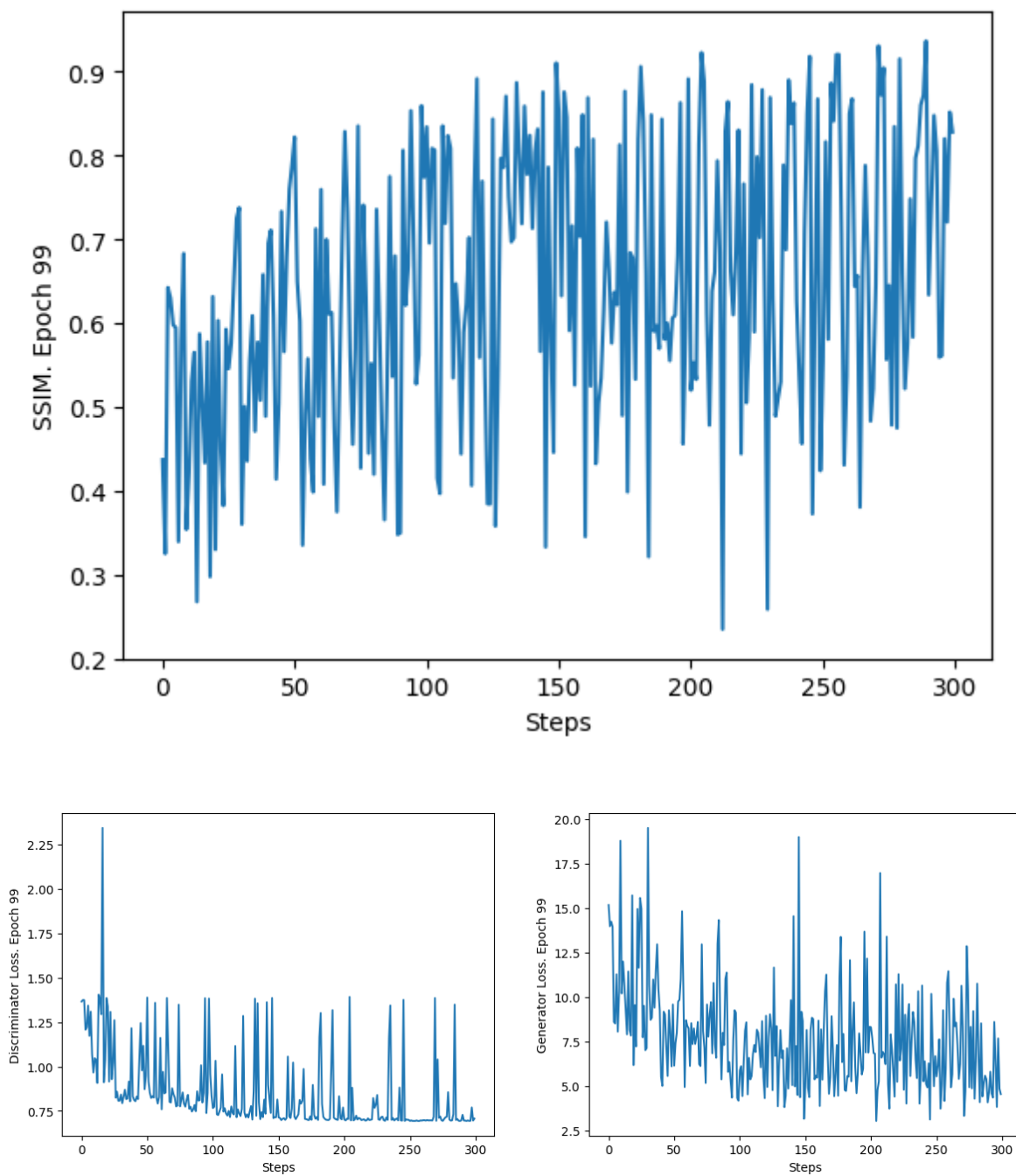


Рис. 4. Тренування мережі для реставрування зображень.
Епохи 0-300. SSIM, втрати генератора (справа) та дискримінатора (зліва)

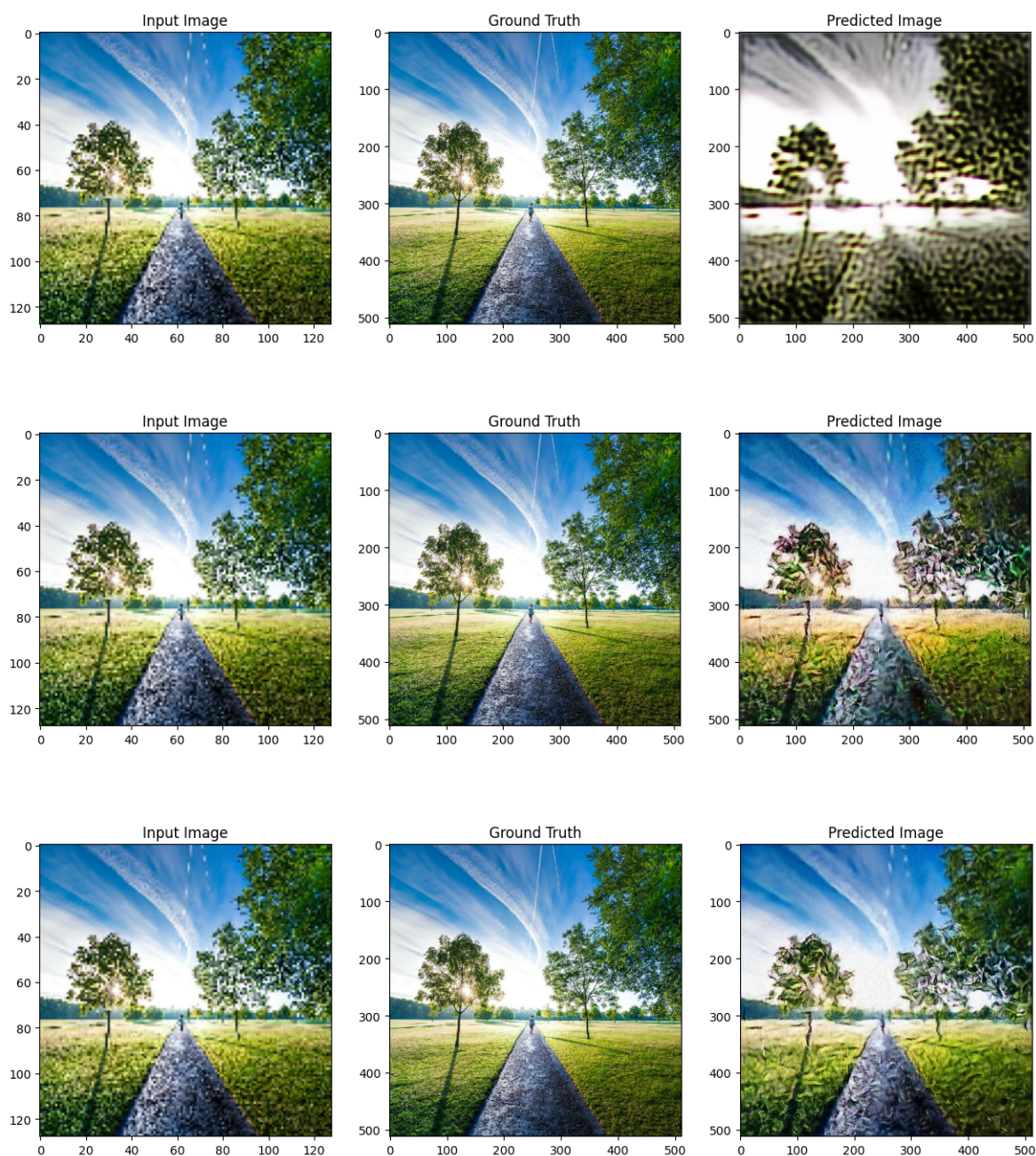


Рис. 5. Тренування мережі для покращення якості зображень.

Результат роботи генератора, згори вниз: після 0-ї епохи, після 100-ї епохи, після 200-ї епохи (зліва направо: вхідне, оригінальне, згенероване зображення)

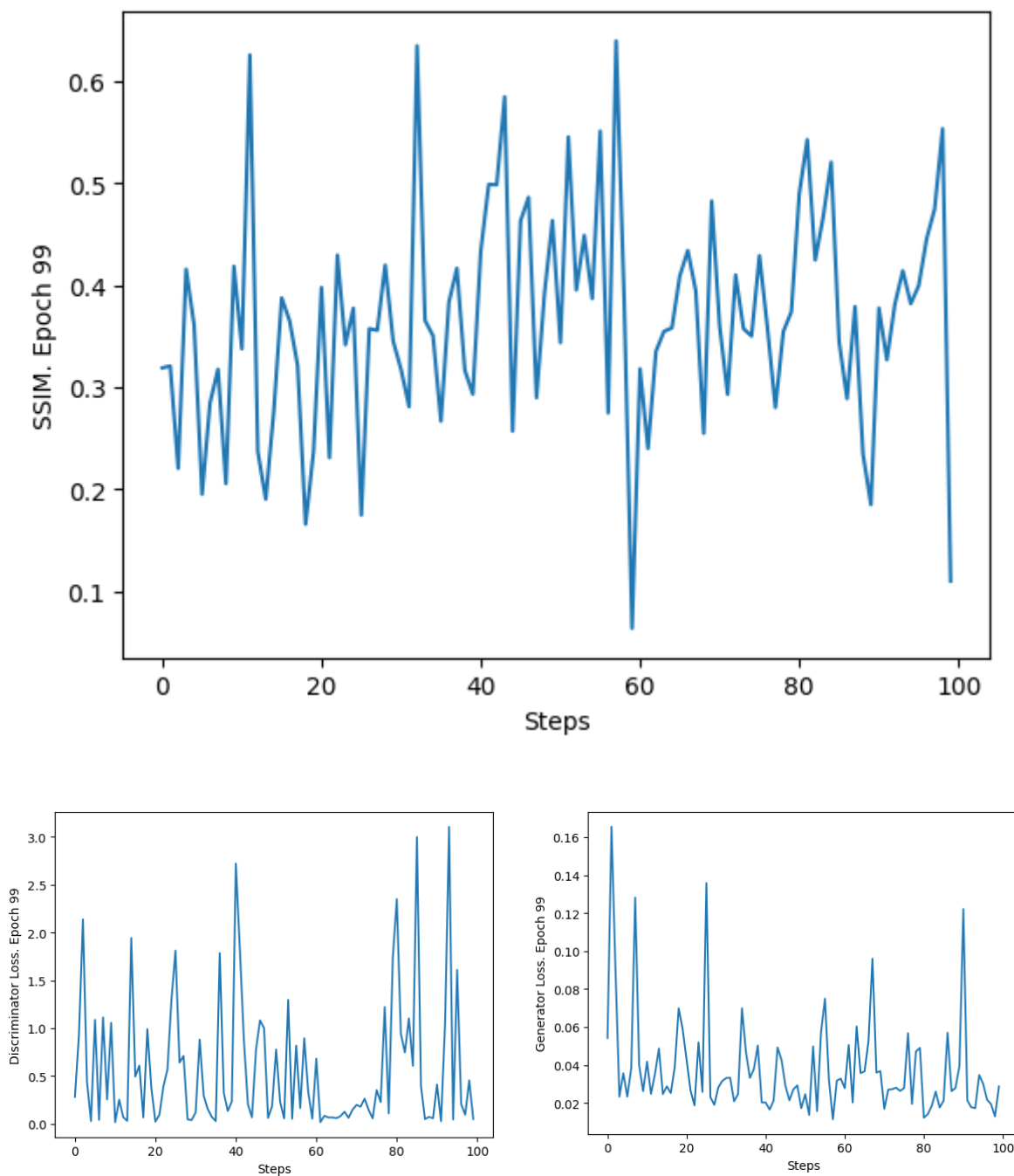


Рис. 6. Тренування мережі для покращення якості зображень.
Епохи 0-100. SSIM, втрати генератора (справа) та дискримінатора (зліва)

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2022 р.

ВЕБСЕРВІС ДЛЯ РЕСТАВРУВАННЯ ТА КОЛОРИЗАЦІЇ
ФОТОГРАФІЙ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Програма та методика тестування

ДП.045480-04-51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Євгенія СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Катерина ТАРЕЛКІНА

ЗМІСТ

1. Об'єкт випробувань	3
2. Мета тестування	3
3. Методи тестування	3
4. Засоби та порядок тестування	4

1. ОБ'ЄКТ ВИПРОБУВАНЬ

Вебсервер для автоматичних реставрування та колоризації фотографій на основі нейронних мереж, написаний мовою Python, фреймворк Tensorflow (серверна частина) та мовою TypeScript, фреймворк Angular (клієнтська частина).

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступні пункти.

- Функціональні характеристики програми (якість вихідних зображень, ступінь їхньої подібності до оригінальних зображень).
- Нефункціональні характеристики програми, перелічені нижче.
 - Швидкість обробки одного зображення.
 - Швидкість тренування мережі (кількість секунд на один навчальний крок).
 - Дизайн та адаптивність сторінок вебсайту відповідно до вимог технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування повинне бути виконане методами White Box Testing (автоматизоване тестування клієнтської частини) та Black Box Testing (ручне тестування серверної та клієнтської частин).

Використовуються наступні методи.

1. Функціональне автоматизоване тестування за допомогою unit-тестів.
2. Ручне функціональне тестування (шляхом візуального порівняння оригінальних та вихідних зображень та зняття метрик: SSIM, MSE, значення функцій втрат мереж).
3. Ручне тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується за допомогою бібліотек scikit-image, засобів Tensorflow для серверної частини застосунку, бібліотеки Jest для клієнтської частини (автоматизоване тренування) та панелі інструментів розробника в браузері (ручне тестування). Тестування повинно включити наступні кроки.

- Зняття метрик під час тренування мереж: SSIM, проміжні результати роботи мережі, час проходження однієї епохи та одного навчального кроку.
- Тестування кожної нейронної мережі на принаймні десятих тестових зображеннях. Візуальна оцінка результатів, зняття метрик: SSIM, MSE, час обробки зображення.
- Unit-тестування клієнтської частини.
- Ручне тестування веб-інтерфейсу: адаптивність, підтримка кількох мов (англійська, українська, французька), відповідність дизайну технічному завданню.

Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Євгенія СУЛЕМА

“ ____ ” _____ 2023 р.

ВЕБСЕРВІС ДЛЯ РЕСТАВРУВАННЯ ТА КОЛОРИЗАЦІЇ
ФОТОГРАФІЙ НА ОСНОВІ НЕЙРОННИХ МЕРЕЖ

Керівництво користувача

ДП.045480-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Євгенія СУЛЕМА

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Катерина ТАРЕЛКІНА

ЗМІСТ

1. Опис сторінок вебсайту.....	3
2. Алгоритм розгортання вебсервісу.....	7
3. Робота з мережами.....	7

1. Опис сторінок вебсайту

Головна сторінка містить загальний опис проєкту; посилання на основні джерела, використані при розробці; отримані результати.

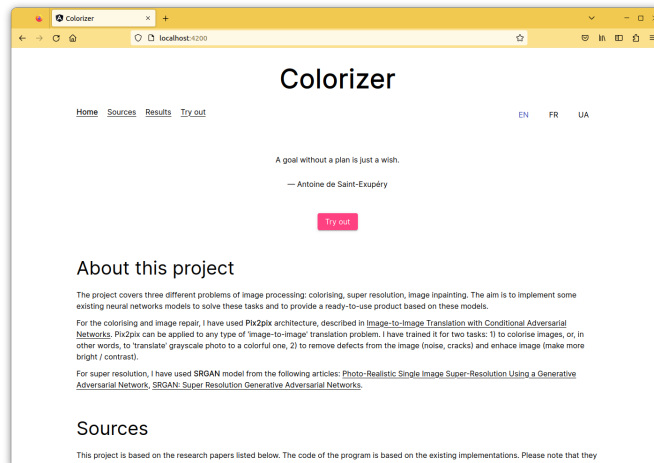


Рис. 1. Головна сторінка

Сторінка «Джерела» містить детальний список усіх джерел, використаних при розробці кожної з мереж, з посиланнями на ці джерела.

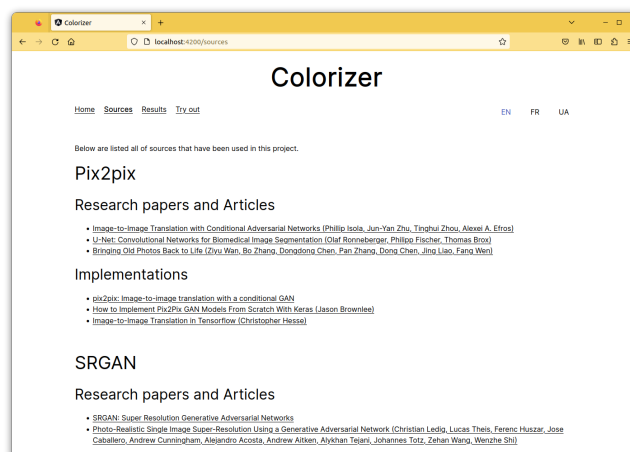


Рис. 2. Сторінка «Джерела»

Сторінка «Результати» містить графіки втрат генератора, дискримінатора та зміни у в процесі навчання, а також результати роботи мереж на тестових зображеннях.

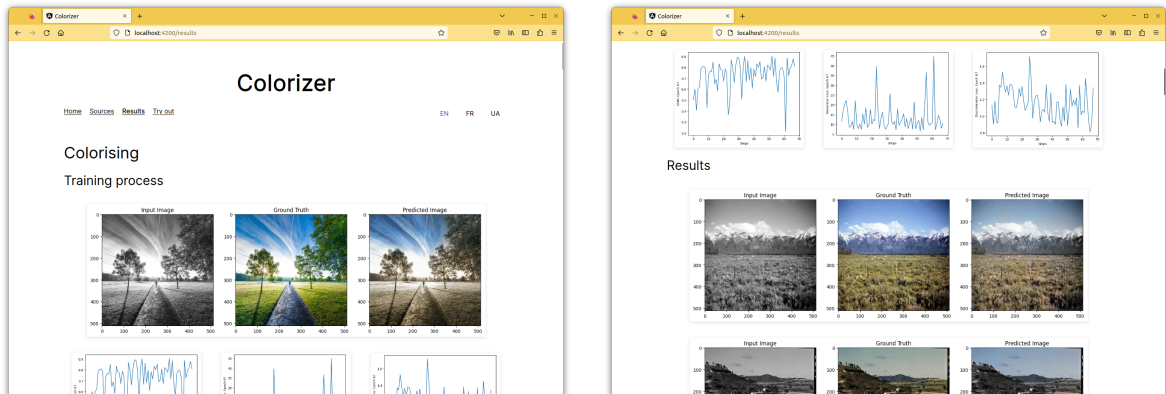


Рис. 3. Сторінка «Результати»

Сторінка «Colorizer» містить інтерактивний віджет з такими елементами:

- Кнопка «Завантажити файл». При натисканні з'являється діалогове вікно менеджера файлів, де користувач може обрати файл одного з форматів: PNG, JPG, JPEG.
- Меню зі списком сервісів. Користувач може обрати лише один сервіс за раз:
 - колоризація фотографій (надання чорно-білому зображенню кольору);
 - відновлення фотографій від пошкоджень (низький контраст, зависока або занижка яскравість, шум, «тріщина» або «розрив»);
 - покращення якості фотографій (збільшення роздільної здатності зображення).
- Кнопка «Підтвердити». Кнопка недоступна, якщо користувач не обрав жодного сервісу.

- Віконце «Поточна фотографія». Містить фотографію, завантажену користувачем. У випадку, якщо користувач ще не завантажив жодної фотографії, віконце містить напис: «No image yet».
- Віконце «Результат». Залежно від статусу містить:
 - завантажувач (елемент, який відображає те, що запит в обробці);
 - повідомлення про помилку (якщо під час обробки зображення сталася помилка. Повідомлення містить текст помилки);
 - результуюче зображення (у випадку, якщо обробка пройшла успішно);
 - напис «No image yet» у всіх інших випадках.
- Кнопка «Завантажити результат». Після натискання користувач повинен обрати папку, в яку завантажити результат. Після цього повинне почнутися завантаження в цю папку.

Користувач може змінити мову вебсайту у верхньому правому куті.

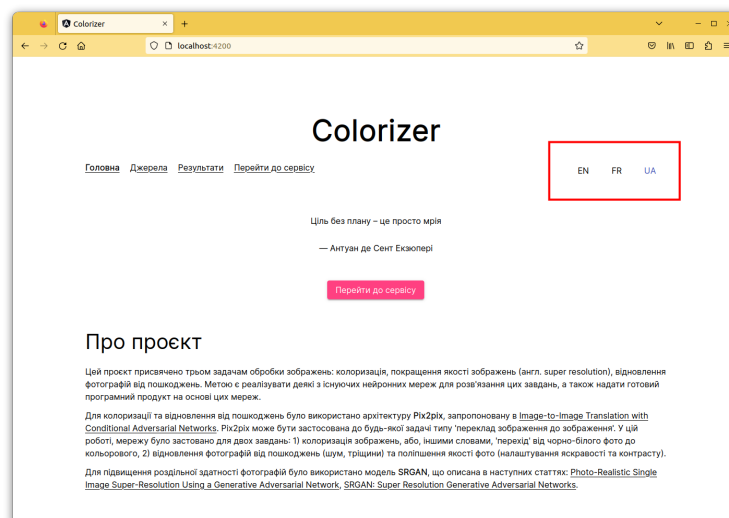


Рис. 4. Зміна мови вебсайту

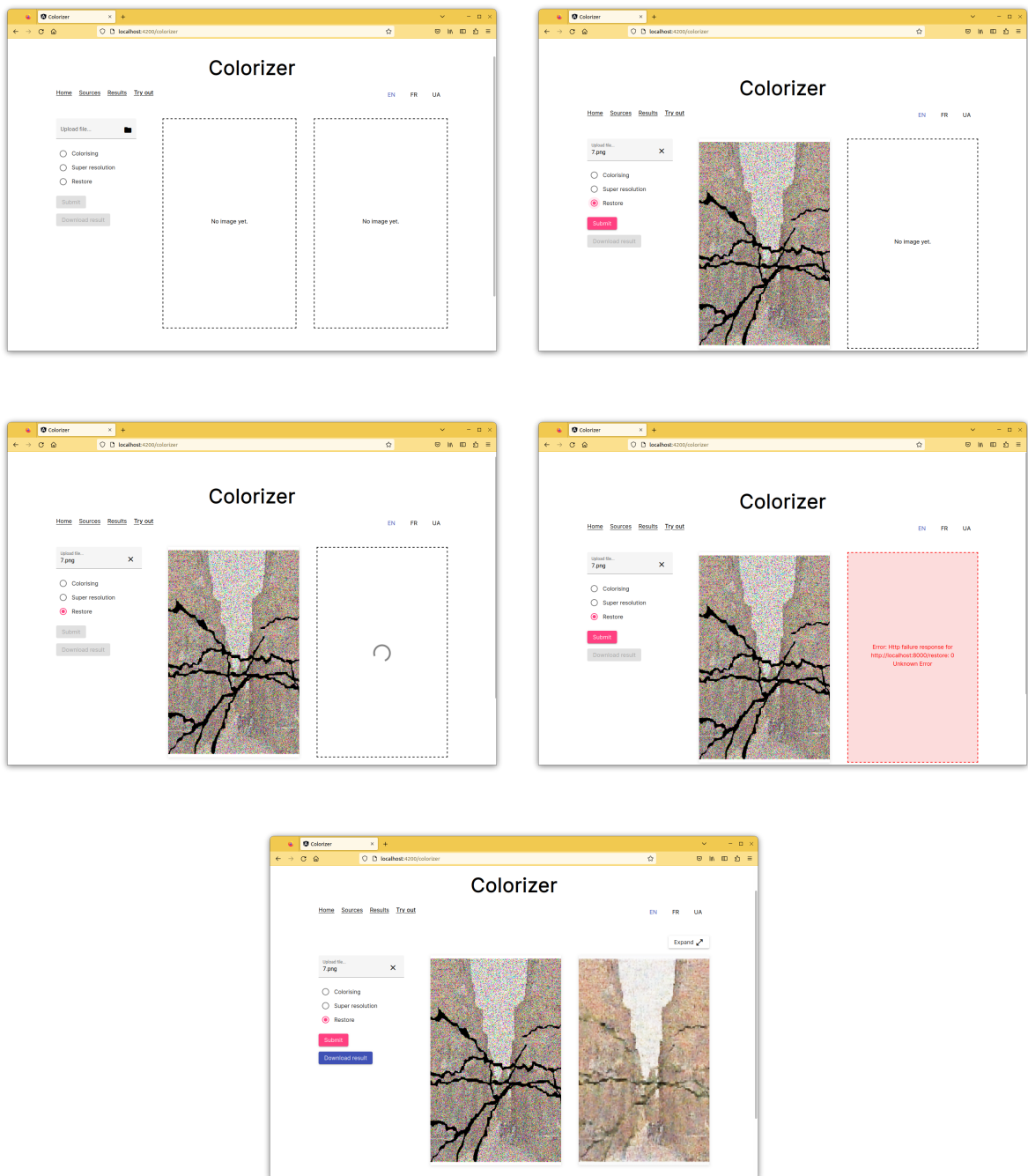


Рис. 5. Сторінка «Colorizer». Інтерактивний віджет та його різні стани (порожній, завантаження, помилка тощо)

2. Алгоритм розгортання вебсервісу

Для розгортання сервісу локально потрібно:

- 1) у папці клієнтської частини проєкту запустити наступну команду:
`npm run start`;
- 2) у папці серверної частини проєкту запустити команду: `uvicorn src.main:app --reload`.

Попередньо потрібно встановити всі необхідні пакети (npm, Angular CLI, Pip, Tensorflow).

3. Робота з мережами

Користувач може самостійно натренувати мережі на власноруч обраних датасетах. Для цього потрібно скористуватися інструкціями, наведеними в Google Colab notebook (додається до файлів проєкту).

Після завершення навчання результати потрібно зберегти у форматі `tf.Checkpoint` та завантажити у відповідну папку вебсерверу:

- `colorizer_discriminator_checkpoints`,
- `colorizer_generator_checkpoints`,
- `restoration_discriminator_checkpoints`,
- `restoration_generator_checkpoints`,
- `super_resolution_discriminator_checkpoints`,
- `super_resolution_generator_checkpoints`.