МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Програмний додаток підвищення роздільної здатності відео за допомогою нейронних мереж»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ-71-8 Захарченко Олександр Олександрович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____«__» _____2021 р.

Науковий керівник _____ к.т.н., доц., Марченко А. В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2021

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

_____ В. В. Шендрик
«__» _____ 2021 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Захарченко Олександр Олександрович

1 Тема роботи *Програмний додаток підвищення роздільної здатності відео за допомогою нейронних мереж*

керівник роботи *Марченко Анна Вікторівна, к.т.н., доцент*,

затверджені наказом по університету від «14» квітня 2021 р. №0181-VI

2 Строк подання студентом роботи «7» червня 2021 р.

3 Вхідні дані до роботи Архітектура нейронної мережі RDN, датасет зображень div2k

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області, 2) Проектування програмного додатку, 3) Розробка програмного додатку підвищення роздільної здатності відео за допомогою нейронних мереж

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність, цілі та задачі, огляд існуючих досліджень, огляд аналогів, вимоги, моделювання IDEF0, моделювання IDEF1, діаграми використання, архітектура нейронної мережі, стек технологій, демонстрація роботи, огляд результатів, порівняння з аналогом, висновки.

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання_____01.10.2020_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Оформлення планування робіт	До 02.04.2021	
2.	Оформлення технічного завдання	До 09.04.2021	
3.	Проведення аналізу предметної області	До 20.04.2021	
4.	Проведення структурно-функціонального моделювання процесів	До 30.04.2021	
5.	Розробка програмного додатку	До 20.05.2021	
6.	Тестування програмного додатку	До 25.05.2021	
7.	Порівняння програмного додатку з аналогами та аналіз результатів	До 30.05.2021	
8.	Здача пояснювальної записки та файлів розробленого додатку	До 06.06.2021	

Студент _____ Захарченко О.О.
(підпис)

Керівник роботи _____ к.т.н., доц. Марченко А.В.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Програмний додаток підвищення роздільної здатності відео за допомогою нейронних мереж».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 22 найменувань, додатків. Загальний обсяг роботи − 105 сторінок, у тому числі 43 сторінок основного тексту, 3 сторінки списку використаних джерел, 59 сторінок додатків.

Кваліфікаційну роботу бакалавра присвячено розробці програмного додатку підвищення роздільної здатності за допомогою нейронних мереж. В роботі проведено опис існуючих підходів для підвищення роздільної здатності, опис останніх досліджень та архітектур нейронних мереж для підвищення роздільної здатності, опис використаної архітектури нейронної мережі, наведені програмні додатки аналоги.

У роботі виконано реалізацію програмного додатку для підвищення роздільної здатності відео зображень, порівняння створеного додатку із аналогом, проведено аналіз отриманих результатів.

Результатом проведеної роботи є отриманий програмний додаток, який можна використовувати для покращення відео та зображень та інтегрувати у більш комплексні інформаційні системи. Знайдено вузьке місце всіх реконструкційних методів підвищення роздільної здатності, наведено припущення щодо їх рішення.

Практичне значення роботи полягає у можливості покращення відео та зображень користувачів, рекомендацій щодо покращення реконструкційних методів підвищення роздільної здатності.

Ключові слова: підвищення роздільної здатності, відео, зображення, реконструкційні методи обробки зображень, нейронна мережа, програмний додаток.

ЗМІСТ

ВСТУП

Використання штучного інтелекту у повсякденному житті є давньою мрією фантастів та частим інструментом руйнування у пост апокаліптичних книгах. Сучасний світ неминуче наближається до точки, коли більшість ручних професій буде заміщено автоматизованими роботами. Використання штучного інтелекту може допомогти позбутися людського фактору в таких важливих сферах життєдіяльності, як медицина, охорона, розвідка та наука в цілому. Наприклад, існує дослідження покращення зображень МРТ знімків серцевої системи [1], що дає можливість зменшити помилку при постановці діагнозу. Також існує дослідження збільшення роздільної здатності камер відеоспостереження із підключеною системою розпізнавання лиця та поведінки [2], що дозволяє реагувати на ще не почавшийся злочин.

Вирішенням таких задач займається область науки комп'ютерного бачення. Вона полягає у створенні штучних систем, що вміють отримувати інформацію із зображень. Досить часто отримані зображення мають низьку якість, що впливає на роботу всієї системи, тому системи підвищення роздільної здатності (далі — системи ПРЗ), що вміють збільшувати якість деградованого зображення, широко використовуються у вирішенні задач комп'ютерного бачення та в таких областях як: охорона [1], медицина [2], генерація зображень [3], астрономія [4] та інші. Фактично, така система може бути використана в будь–якій області, яка працює із зображенням.

Об'єкт дослідження — це процес підвищення роздільної здатності відео–зображення, або звичайного зображення.

Метою роботи є створення програмного додатку (далі – ПП) для ПРЗ відео–зображення шляхом використання нейронних мереж для відновлення втрачених деталей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Завдання ПРЗ зображення не є новинкою, в останні роки було представлено десятки різних підходів, проте вони рідко використовуються як незалежні рішення, а є доповненням до інших систем. Наприклад, більшість фото–редакторів мають можливість масштабування і використовують примітивні алгоритми ПРЗ, тому результат такого збільшення є малоефективним.

Для підвищення роздільної здатності використовують 2 підходи: інтерполяційний, що використовує значення сусідніх пікселів для генерації нових, та реконструкційний, що використовує нейронні мережі (далі — НМ) для відновлення деталей.

Інтерполяційні методи в порівнянні з реконструкційними є швидкими, доступними та звичними. Вони часто використовуються у відео–іграх та редакторах зображень, виконуючи дію згладжування «гострих» деталей, що покращує якість зображення, але роздільна здатність не змінюється.

Реконструкційні методи використовують НМ, які намагаються відновити деталі із деградованого зображення. Такі методи є трудомісткими, довгими, рідко точними, але результат якісно кращий за інтерполяційні, роздільна здатність зображення підвищується, зображення стає більш детальнішим.

До успішних методів, що використовують реконструкційний підхід можна віднести мережі: SRCNN [5], FSRCNN [6], VDSR [7], LapSRN [8], MemNet [9], EDSR [10], RCAN [11], MDSR [12], RDN [13]. Особливого успіху досягли мережі RCAN, MDSR, RDN. В наступному блоці аналізу аналогів, використаємо наведені вище методи для їх порівняння та знаходження недоліків.

1.2 Аналіз програмних продуктів – аналогів

Переважно, для оцінки результатів систем ПРЗ використовують декілька параметрів: пікове відношення сигналу до шуму (скорочено — PSNR) та індекс структурної подібності (скорочено — SSIM).

– PSNR знаходиться як відношення максимально можливої яскравості пікселя до обрахованої середньоквадратичної помилки (див. формулу 1);

– SSIM визначає наскільки отримане зображення є однаковим до еталонного. Індекс приймає значення від –1 до 1, де 1 — повна ідентичність (див. формулу 3).

Обидва параметри оцінюють наскільки система добре впоралась із завданням. На рисунку нижче представлено оцінювання існуючих систем кількісними параметрами. Зображення для покращення використовується із доступних в мережі наборів даних, в даному випадку — Urban100, зображення “119082” (перший рядок), та зображення “img_043” (другий рядок).

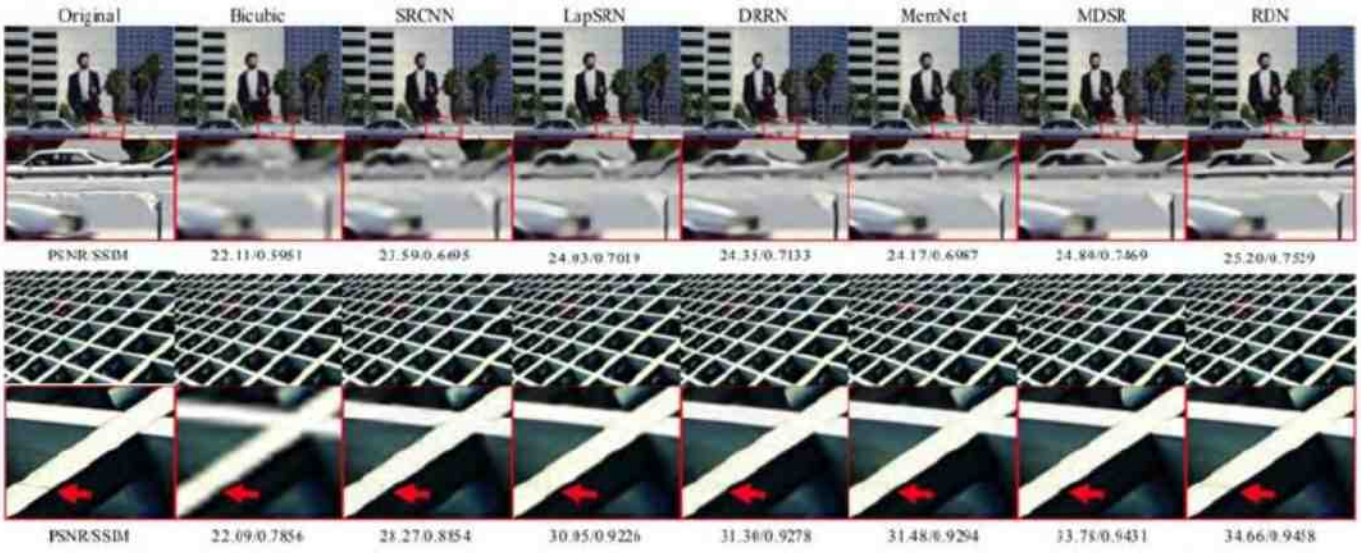


Рисунок 1.1 — Порівняння існуючих систем [13]

На рисунку 1.1 представлено результат підвищення роздільної здатності в 4 рази, для оцінки використано параметри (значення під зображеннями)

PSNR (див. формулу 1) та SSIM (див. формулу 2). Перший стовпчик відповідає оригінальному зображенню у великому розширенні. Другий стовпчик використовує бікубічний метод, що відноситься до інтерполяційного типу.

PSNR використовується для виміру рівня шумів у зжатому зображенні і визначається за формулою [19]:

$$PSNR = 10log_{10}\left(\frac{R^2}{MSE}\right), \quad (1)$$

де R^2 – максимальне коливання сигналу (255 для 8-бітового зображення)

$$MSE = \frac{\sum_{m,n}[I_1(m,n) - I_2(m,n)]^2}{M * N}, \quad (2)$$

де M, N – кількість стовпців і рядків в зображенні I.

SSIM використовується для визначення індексу структурної подібності, тобто, показує наскільки 2 зображення структурно подібні одне до одного, та обчислюється за формулою [20]:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}, \quad (3)$$

де μ_x, μ_y – локальні середні значення зображень, σ_x, σ_y – середньоквадратичне відхилення для зображень, σ_{xy} – коваріація [20].

Зображення отримане в результаті бікубічного методу складно назвати покращеним, проте його числові результати також не набагато гірші від конкурентів. Мережі SRCNN, LapSRN, DRRN мають великий рівень спотворень, прямі лінії стали хвилястими, що додає складності для сприйняття. Мережі MDSR

та RDN впорались значно краще, лише зображений бетонний блок на задньому плані втратив деталі, але в іншому все добре. Слід зазначити, що параметр SSIM все одно досить низький, що обумовлено кількістю деталей.

Додатково необхідно звернути увагу, що бетонний блок став схожий на намальований — це особливість нашого сприйняття, коли зображення має низьку кількість деталей, воно здається намальованим.

Зображення нижче, має меншу кількість деталей, тому системи впорались з ним краще. Індекс подібності досягає 0,945, що говорить майже про ідеальну схожість. Мережі MDSR та RDN мають різницю подібності в 0,0027 одиниці, але лише мережа RDN впоралась із відновленням ледве помітної лінії на балці.

Виходячи з отриманих результатів, можна підсумувати, що кількісні параметри є важливими в системах ПРЗ лише для глибокого аналізу результатів та покращення самої системи.

Представлені системи рідко використовуються як окремі рішення, вони часто виступають як компонент для більшої системи. До найближчого аналога продукта, можна віднести некомерційний проект waifu2x, що використовує мережу SRCNN для ПРЗ зображень [14]. Порівняння цієї мережі було виконано на рисунку 1.1. Також існує аналог waifu2x–caffe [15], що має візуальний інтерфейс та працює з Windows.

Основна мета аналогів — покращення якості художніх зображень. На мою думку, це обумовлено декількома факторами: використання мережі SRCNN з невисокими показниками, художні зображення мають менше деталей і їх легше покращити. Втрата деяких деталей на художньому зображенні не так впливає на результат, оскільки таке місце не буде сильно відрізнятися від загального зображення. До іншого недоліку таких продуктів можна віднести дуже жорстку прив’язку до апаратного забезпечення. Аналог waifu2x вимагає наявність відеокарти виробника Nvidia, без неї продукт взагалі працювати не буде. Аналог waifu2x–caffe може працювати або з відеокартою Nvidia, або з використанням ресурсів центрального процесору, що значно впливає на швидкодію.

Таблиця 1.1 – Порівнянна продуктів аналогів

Параметр	**Waifu2x**	**Waifu2x–caffe**	**Власне рішення**
Підтримка відео	–	–	+
Можливість навчання мережі	+	+	+
Зменшення рівня шуму	+	+	+
Підтримка Linux ОС	+	–	+
Підтримка Windows ОС	–	+	+
Жорстка прив’язка до апаратного забезпечення	+	–	–

Для вирішення проблем сумісності можна використати ПЗ docker [16], що дозволяє створювати своє власне середовище на хост системі. Наприклад, користувач може використовувати операційну систему Windows, встановивши docker та необхідне зображення (набір інструкцій для інсталяції та роботи якогось ПЗ), таким чином користувач зможе запустити ПЗ, яке розроблювалось лише під Linux систему. Тому для вирішення проблеми сумісності вирішено використовувати ПЗ docker. Для покращення якості отриманого результату вирішено використовувати мережу RDN, що має кращі показники за SRCNN, яку використано в аналогах.

1.3 Постановка задачі

Метою роботи є реалізація ПП підвищення роздільної здатності відео. Додаток повинен мати можливість:

– збільшити роздільну здатність відео–зображення або зображення на вказану користувачем величину та відновлювати втрачені деталі;

– зберігати налаштування користувача в конфігураційному файлі;

– автоматично інсталюватися на системі користувача;

– підтримувати можливість тренування мережі, використовуючи користувацькі набори даних;

– підтримувати збір статистики для перевірки кількісних показників успішності при тренуванні.

Задля реалізації поставлених задач, виділяються наступні етапи:

– пошук аналогів та їх аналіз;

– пошук існуючих мереж для ПРЗ та їх аналіз;

– вивчення архітектури обраної мережі;

– розробка модулю роботи з нейронною мережею;

– розробка модулю збору статистики;

– розробка інших модулів;

– збір рішення в контейнер docker;

– проведення тестування;

– створення супроводжуючої документації.

Для реалізації будуть використані наступні технології:

– мова програмування Python із використанням платформи TensorFlow [17], для моделювання роботи нейронної мережі, розбиття та подальшої збірки відео на кадри зображень, виконання покращення зображення;

– програмне забезпечення Docker [16], для забезпечення кросплатформеності та полегшення інсталяції для користувача.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Структурно–функціональне моделювання

Процес моделювання дозволяє краще зрозуміти бізнес–процеси, визначити ключові елементи в системі та пояснити роботу не вдаючись до низькорівневої імплементації. Контекстна діаграма IDEF0 описує саму систему, а також її взаємодію із середовищем. Діаграма складається із наступних елементів:

– вхідні дані: дані, що використовуються для отримання результату;

– вихідні дані: дані, що отримує користувач після виконання програми;

– управління: інформація, яка використовується системою під час роботи;

– механізми: ресурси, що використовуються при роботі системи.

В процесі роботи програмного продукту для ПРЗ відео–зображень будуть використані наступні дані:

– вхідні дані: деградоване відео–зображення або зображення;

– вихідні дані: покращене відео–зображення або зображення;

– управління: налаштування мережі, користувацькі налаштування, обрана якість;

– механізм: модуль роботи з нейронною мережею, архітектура мережі, IDE.

Таким чином була створена діаграма IDEF0, що представлена на рисунку 2.1.

Рисунок 2.1 — Діаграма в нотації IDEF0

Побудовану діаграму також називають діаграмою декомпозиції нульового рівня. Вона надає можливість отримати загальне представлення про систему, більш детальний опис роботи наведений в декомпозиції першого рівня на рисунку 2.2.

Перший під–процес полягає в розбитті відео на кадри зображень:

– вхід: відео у низькому розширенні;

– вихід: масив кадрів зображення, звукова доріжка;

– механізм: IDE.

Другий під–процес полягає в розбитті кадру на шари RGB кольорів:

– вхід: масив кадрів зображення;

– вихід: RGB шари, що містилися в кадрі;

– управління: налаштування мережі;

– механізм: IDE.

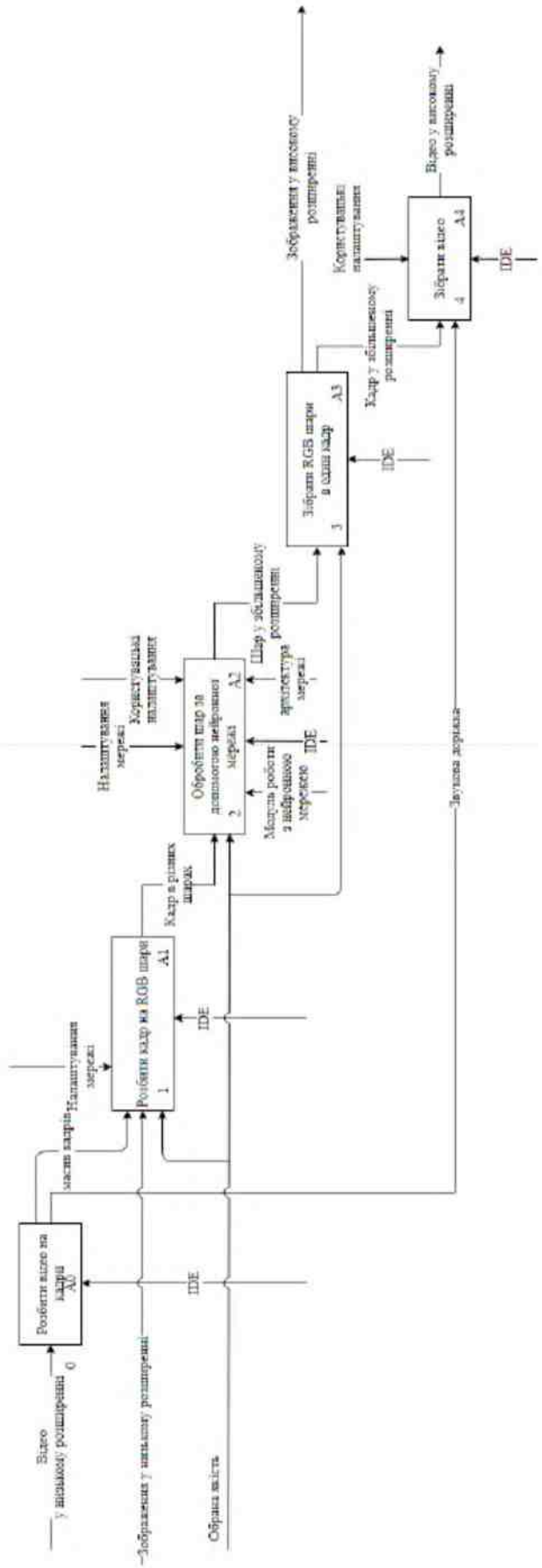


Рисунок 2.2 — Декомпозиція першого рівня

Третій під–процес полягає в обробці шару за допомогою нейронної мережі:

– вхід: RGB шари зображення, обрана якість;

– вихід: покращені RGB шари зображення;

– управління: налаштування мережі, користувацькі налаштування;

– механізм: Модуль роботи з нейронною мережею, архітектура мережі, IDE.

Четвертий під–процес полягає у збірці отриманих RGB шарів до цільного кадру зображення:

– вхід: покращені RGB шари зображення, обрана якість;

– вихід: покращений кадр зображення;

– управління: не має;

– механізм: IDE.

П’ятий під–процес полягає у збірці отриманих кадрів до відео:

– вхід: покращені кадри зображення, звукова доріжка;

– вихід: покращене відео–зображення;

– управління: користувацькі налаштування;

– механізм: IDE.

2.2 Моделювання варіантів використання програмного продукту

Моделювання варіантів використання (скорочено — ВВ) полягає у побудові взаємодії акторів та самих ВВ. Діаграма ВВ дозволяє показати весь функціонал, який може виконати система для користувача, а також визначає з чим саме взаємодіють різні актори в системі. До акторів можна віднести будь–яку сутність, яка взаємодіє із системою знаходячись поза її межами. В розроблюваному ПП виділяється 2 актори:

– Docker: програмне забезпечення, що використовується для запуску створеного ПП. Ретранслює команди користувача до ізольованого простору, в якому виконується створений ПП.

– Користувач: використовує систему для покращення відео–зображення чи зображення або тренування моделі НМ, може змінювати параметри системи, переглядати статистику;

Варіанти використання:

– покращити відео або зображення;

– навчити модель, використовуючи інші дані;

– перегляд статистики;

– зміна параметрів.

Схема варіантів використання зображена на рисунку 2.3.

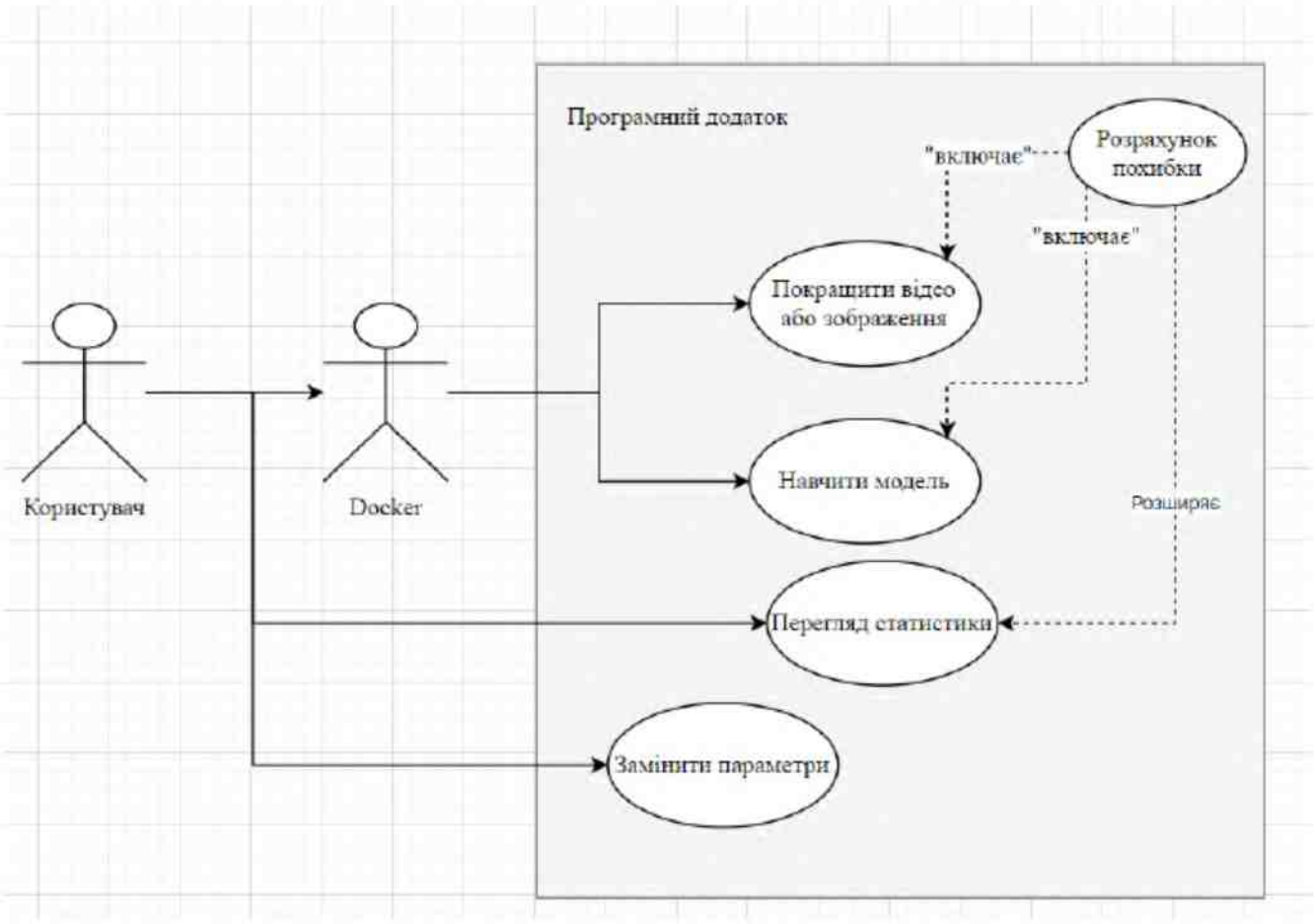


Рисунок 2.3 – Діаграма ВВ

2.3 Архітектура мережі RDN

Мережа RDN складається із чотирьох блоків: залишкові щільні блоки (далі – RDB, від англійської residual dense block), блоки об'єднання щільних ознак (далі – DFF, від англійської dense feature fusion), мережа збільшення роздільної здатності, мережа класифікації для витягування структурних ознак зображення.

Мережа класифікації, в нашому випадку – мережа VGG19 [21], що широко використовується для ідентифікації та класифікації структурних ознак зображення (найпростіший приклад – знайти та класифікувати горизонтальні лінії в зображенні, складніший – знайти лінії, що формують квадратний об'єкт). VGG19 побудована на основі набору із 14197122 зображень та є найпопулярнішим рішенням для рішення проблем класифікації зображень. В мережі RDN вона використовується при навчанні моделі, задає допустиме відхилення знайдених структурних ознак підвищеного зображення відносно еталонного. Архітектуру мережі (див. рисунок 2.4), а також формули, що описують її роботу взято із наукової статті дослідження мережі RDN [13].

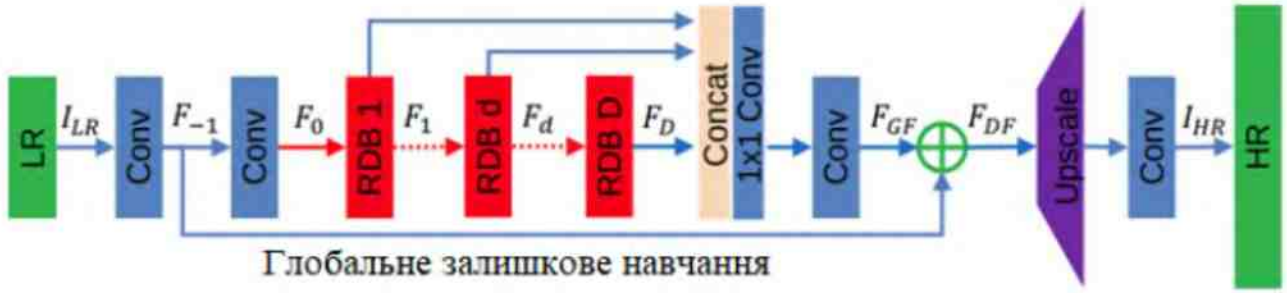


Рисунок 2.4 – Архітектура мережі RDN

Позначемо I_{LR} та I_{SR}, як входи та виходи мережі RDN. В мережі, для витягування структурних ознак, використовують 2 згорткових шари (далі – Conv шар, від англійської Convolutional). Перший шар витягує особливості F_{-1} із I_{LR} входу.

$$F_{-1} = H_{SFE1}(I_{LR}), \tag{4}$$

де H_{SFE1} – операція згортання.

В результаті формується мапа, або шар ознак. Процес формування шару ознак представлено на рисунку 2.5. F_{-1} потім використовується для подальшого витягування структурних ознак та глобального залишкового навчання:

$$F_0 = H_{SFE2}(F_{-1}), \tag{5}$$

де H_{SFE2} – операція згортання для формування другого шару, що подається на вхід блоків RDB.

Вихід блоку RDB можна обрахувати за формулою:

$$F_d = H_{RDB,d}(F_{d-1}) = H_{RDB,d}\left(H_{RDB,d-1}\left(\cdots\left(H_{RDB,1}(F_0)\right)\cdots\right)\right), \tag{6}$$

де $H_{RDB,d}$ – операції d–го RDB блоку.

До операцій RDB блоку необхідно віднести операції згортки та формування шару зрізаних лінійних вузлів (далі – ReLU, від англійської rectified linear unit), що замінєю на 0 негативні значення в шарі ознак, бо значення пікселя не може бути негативним.

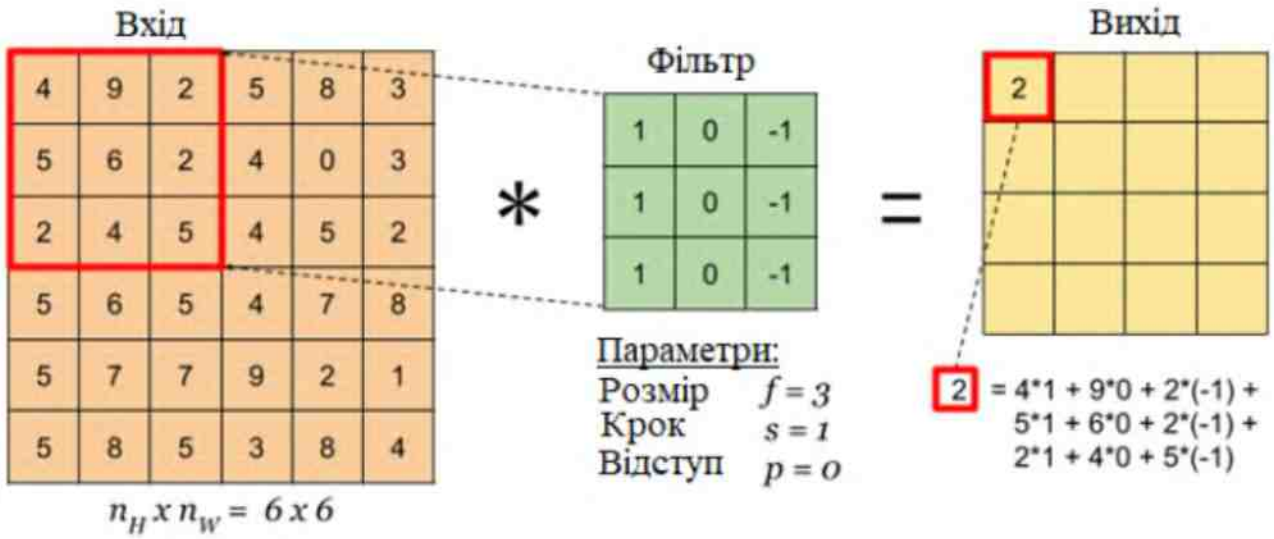


Рисунок 2.5 – Схема процесу формування шару ознак

Операція згортки формує шар ознак на кожний по кожному із кольорових шарів зображення, фільтр для формування може бути задано як автоматично випадковим чином, так і самостійно. В мережі RDN, під час навчання, він задається випадковим чином із використанням рівномірного розподілу, але в процесі навчання змінює свої значення на інші.

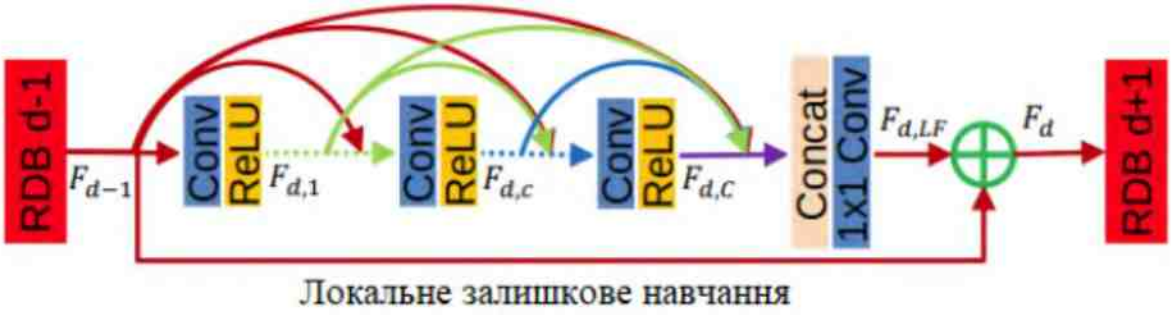


Рисунок 2.6 – Архітектура блоку RDB

Блок RDB (див. рисунок 2.6) складається із щільно пов'язаних слоїв, об'єднанні локальних ознак та локального залишкового навчання, що складають механізм неперервної пам'яті. Механізм неперервної пам'яті забезпечується тим, що кожний наступний блок RDB використовує стан попереднього блоку.

Оскільки F_d отриманий із використанням кожного згорткового шару в блоці, то його можна розглядати, як локальну ознаку.

Позначимо F_{d-1} та F_d, як вхід та вихід d–го RDB блоку, тоді вихід c–го згорткового шару можна розрахувати, як:

$$F_{d,c} = f\left(W_{d,c}\left[F_{d-1}, F_{d,1}, \cdots, F_{d,c-1}\right]\right), \tag{7}$$

де $f()$ – функція активації ReLU, $W_{d,c}$ – ваги згорткового шару c.

Потім використовується об'єднання локальних ознак за допомогою згорткового шару розмірністю 1x1:

$$F_{d,LF} = H^d_{LFF}\left(\left[F_{d-1}, F_{d,1}, \cdots, F_{d,c}, \cdots, F_{d,C}\right]\right), \tag{8}$$

де H^d_{LFF} – функція згорткового шару розмірності 1x1 в d-му блоці RDB.

Отримані результаті об'єднуються із результатами попереднього блоку:

$$F_d = F_{d-1} + F_{d,LF} \tag{9}$$

Таким чином створюються ієрархічні шари ознак, що об'єднуються між собою та формують процес глобального навчання, що виконується перед збільшенням зображення. Глобальне навчання можна виразити формулою:

$$F_{DF} = F_{-1} + F_{GF}, \tag{10}$$

де F_{GF} – об'єднання всіх шарів ознак із усіх блоків RDB

$$F_{GF} = H_{GFF}([F_1, \cdots, F_D]), \tag{11}$$

де H_{GFF} – функція згорткового шару розмірності 1x1.

Процес збільшення зображення показано на рисунку 2.7. Покращене зображення із збільшеною роздільною здатністю отримується в процесі обрахунку субпіксельного згорткового шару [22], який формує зображення із агрегованих шарів ознак. Процес субпіксельного згорткового шару полягає у заповнені матриці більшої розмірності із використанням отриманих шарів ознак.

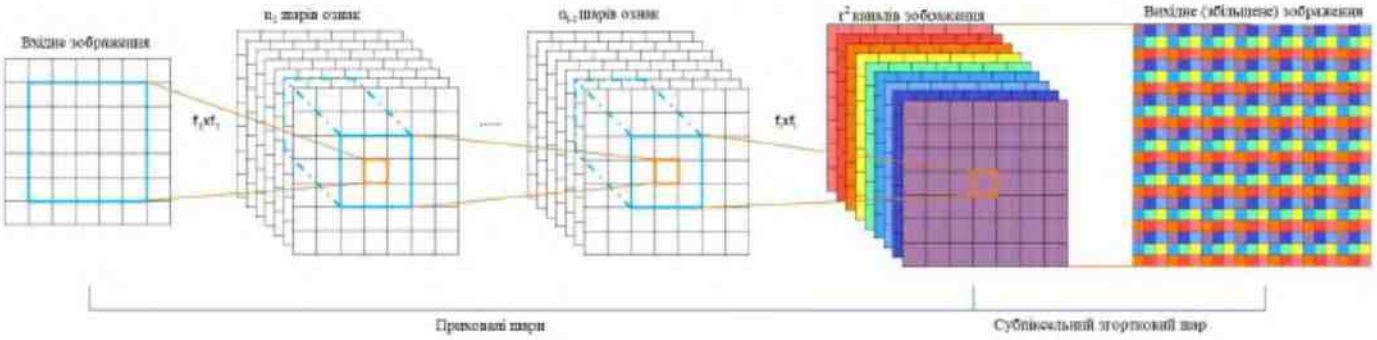


Рисунок 2.7 – Схема процесу збільшення роздільної здатності зображення

3 РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ПІДВИЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ВІДЕО ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ

3.1 Архітектура програмного додатку

Перед реалізацією програмного додатку, необхідно спроектувати його архітектуру: вибрати тип архітектури, головні модулі, інтерфейси їх поєднання.

Для реалізації було вирішено використовувати модульну монолітну архітектуру. Тобто, управління додатком може бути здійснено із одного модуля контролера, або ж використовувати кожен модуль окремо для досягнення певного результату (тренування моделі, виконання покращення відео).

Додаток включає в собі модуль для побудови нейронної мережі, модуль для тренування нейронної мережі, модуль для виконання класифікації нейронною мережею (виконання покращення відео чи зображення), допоміжні модулі для логування даних, розбиття зображення на менші ділянки, надання функцій обчислення помилок та валідації введених даних.

Модуль для побудови НМ використовує набір сторонніх методів із фреймворку Keras. Модуль тренування НМ використовує набір сторонніх методів із інструмента TensorBoard, для збереження та відображення даних у вигляді таблиць. Keras та TensorBoard використовуються за допомогою API. На рисунку 3.1 зображено схему архітектури додатку.

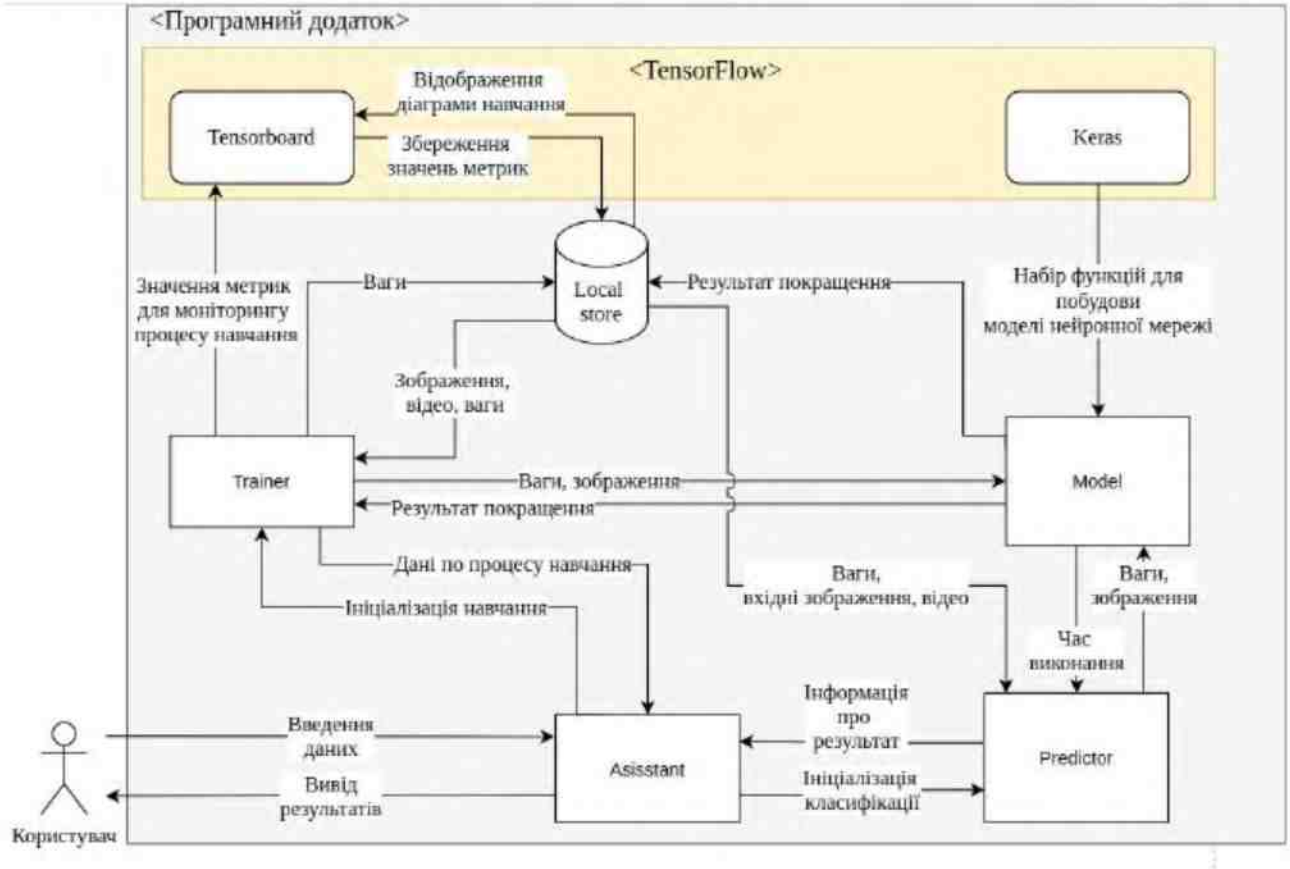


Рисунок 3.1 – Схема архітектури додатку

3.2. Програмна реалізація

В даному розділі представлено опис реалізації основних модулів системи, що не включають допоміжні модулі, які використовуються основними модулями. Допоміжні модулі включають функціонал логування, засоби конвертації даних, методи валідації вхідних даних. Весь програмний код представлено в додатку В.

3.2.1 Розробка модулю із архітектурою нейронної мережі

Модуль архітектури нейронної мережі повинен приймати на вхід декілька параметрів: кількість згорткових шарів в кожному залишковому щільному блоці

(RDB), кількість блоків RDB, кількість фільтрів в згорткових шарах, кількість вихідних шарів ознак в кожному блоці RDB, коефіцієнт збільшення розширення. Також є набір додаткових параметрів: розміри ділянки зображення (необхідно під час тренування), кількість кольорових каналів зображення, розмірність фільтрів для згортки, максимальне значення генератора випадкових чисел.

Модуль включає в себе метод make_model, що будує модель Keras модель заданої архітектури мережі.

```
def _build_rdn(self):
    LR_input = Input(shape=(self.patch_size, self.patch_size, 3), name='LR')
    F_m1 = Conv2D(…
    )(LR_input)
    F_0 = Conv2D(…
    )(F_m1)
    FD = self._RDBs(F_0)
    # Глобальне об'єднання ознак
    # 1x1 Conv of concat RDB layers -> G0 feature maps
    GFF1 = Conv2D(
        self.G0,
        kernel_size=1,
        padding='same',
        kernel_initializer=self.initializer,
        name='GFF_1',
    )(FD)
    GFF2 = Conv2D(…
    )(GFF1)
    # Глобальне залишкове навчання для щільних ознак
    FDF = Add(name='FDF')([GFF2, F_m1])
    # Збільшення роздільної здатності
    FU = self._UPN(FDF)
    # Збірка в зображення
    SR = Conv2D(…
    )(FU)

    return Model(inputs=LR_input, outputs=SR)
```

Рисунок 3.2 – Фрагмент коду побудови моделі НМ типу RDN

Під час побудови моделі викликаються методи Keras: Input (використовується для побудови об’єкта типу «тензор», це основний об’єкт, який використовується в TensorFlow та представляє собою n-мірний масив базових типів даних), Conv2d (об'єкт згорткового шару), Add (функція об’єднання двох тензорів в один).

Власні методи: _RDBs – виконує об’єднання шару ознак G_0 та шару отриманого із іншого залишкового щільного блока (див. формулу 6 та рисунки 2.6, 3.3), _UPN – функція збільшення роздільної здатності.

```
def _RDBs(self, input_layer):
    """RDB блоки. Returns: об'єднання шарів ознак RDB блоків із шарами ознак G0.
    """
    rdb_concat = list()
    rdb_in = input_layer
    for d in range(1, self.D + 1):
        x = rdb_in
        for c in range(1, self.C + 1):
            F_dc = Conv2D(
                self.G,
                kernel_size=self.kernel_size,
                padding='same',
                kernel_initializer=self.initializer,
                name='F_%d_%d' % (d, c),
            )(x)
            F_dc = Activation('relu', name='F_%d_%d_Relu' % (d, c))(F_dc)
            # об'єднати вхід та вихід ConvRelu блока
            # x = [input_layer,F_11(input_layer),F_12([input_layer,F_11(input_layer)]), F_13..]
            x = concatenate([x, F_dc], axis=3, name='RDB_Concat_%d_%d' % (d, c))
        # 1x1 згортка (Локальне злиття ознак)
        x = Conv2D(
            self.G0, kernel_size=1, kernel_initializer=self.initializer, name='LFF_%d' % (d)
        )(x)
        # Локальне залишкове навчання F_{i,LF} + F_{i-1}
        rdb_in = Add(name='LRL_%d' % (d))([x, rdb_in])
        rdb_concat.append(rdb_in)
    assert len(rdb_concat) == self.D
    return concatenate(rdb_concat, axis=3, name='LRLs_Concat')
```

Рисунок 3.3 – Фрагмент коду, метод _RDBs

Описаний метод проходить через кожний блок RDB та через кожний згортковий шар в блоці RDB, випрямляючи шар ознак функцією ReLU (замінює

негативні значення шару ознак на 0), та додає до отриманого шару ознак вхідний шар. Після отримання всіх шарів ознак із одного блоку, виконується локальне злиття ознак згортковим шаром розмірністю 1x1 та виконується локальне залишкове навчання шляхом об'єднання локальних ознак із шаром ознак, отриманих на виході попереднього блоку RDB.

```
def _UPN(self, input_layer):
    """ Шари збільшення."""

    x = Conv2D(
        64,
        kernel_size=5,
        strides=1,
        padding='same',
        name='UPN1',
        kernel_initializer=self.initializer,
    )(input_layer)
    x = Activation('relu', name='UPN1_Relu')(x)
    x = Conv2D(
        32, kernel_size=3, padding='same', name='UPN2', kernel_initializer=self.initializer
    )(x)
    x = Activation('relu', name='UPN2_Relu')(x)
    if self.upscaling == 'shuffle':
        return self._pixel_shuffle(x)
    elif self.upscaling == 'ups':
        return self._upsampling_block(x)
    else:
        raise ValueError('Невірний вибір шару збільшення роздільної здатності.')
```

Рисунок 3.4 – Фрагмент коду, метод _UPN

Метод _UPN є методом обгорткою, який виконує підготовку шарів ознак до виконання самого збільшення. Збільшення може бути виконано за допомогою субпіксельного згорткового шару (метод _pixel_shuffle), або ж за допомогою блоку Upsampling2D (метод _upsampling_block). Обидві функції надаються фрейморком Keras. Функція збільшення може бути задана користувачем в конфігураційному файлі, або ж використовується блок Upsampling2D за замовчуванням. Процес підготовки полягає у проходженні отриманого шару ознак, через додатковий шар згортки та випрямлення для встановлення більш чітких ознак, які передаються до

мережі підвищення роздільної здатності. Схему процесу збільшення роздільної здатності зображено на рисунку 2.7 із наведенням статті з детальним описом.

3.2.2. Розробка модулю класифікації «Predictor»

Основною задачею модуля класифікації є покращення зображення чи відео. Покращення відбувається в декілька етапів: завантажити ваги, створити вихідний каталог, відкрити зображення та покращити його, відкрити відео та покадрово покращити його. Метод, який виконує ці дії зображено на рисунку 3.5.

```
def get_predictions(self, model, weights_path):
    """ Виконує покращення. """

    self.model = model
    self.weights_path = Path(weights_path)
    weights_conf = self._load_weights()
    out_folder = self.output_dir / self._make_basename() / get_timestamp()
    self.logger.info('Результати в:\n > {}'.format(out_folder))
    if out_folder.exists():
        self.logger.warning('Каталог існує, можливо файли було перезаписано')
    else:
        out_folder.mkdir(parents=True)
    if weights_conf:
        yaml.dump(weights_conf, (out_folder / 'weights_config.yml').open('w'))
    # Класифікувати та зберегти
    for img_path in self.img_ls:
        output_path = out_folder / img_path.name
        self.logger.info('Оброблюю файл \n > {}'.format(img_path))
        start = time()
        lr_img = imageio.imread(img_path)
        sr_img = self._forward_pass(lr_img)
        end = time()
        self.logger.info('Витрачений час: {}s'.format(end - start))
        self.logger.info('Результати в: {}'.format(output_path))
        imageio.imwrite(output_path, sr_img)
    for video_path in self.video_ls:
```

Рисунок 3.5 – Фрагмент методу покращення вхідних даних

Для завантаження вагів використовується функція Keras, ваги мають мати розширення «hdf5». Вихідний каталог буте мати комплексну назву, яка складається із назви використаної архітектури мережі та її параметрів, в якому буде міститися інший каталог. Назва іншого каталогу буде згенеровано автоматично відповідно до системної дати та часу, в цьому каталозі і будуть вихідні дані.

Далі функція відкриє кожне зображення, яке було виявлено при ініціалізації об'єкту (див. рисунок 3.6), запустить таймер та запустить процес класифікації за допомогою API Keras. Як тільки результат буде отримано, відлік часу буде зупинено та записано в лог файл і в CLI. Після покращення зображень, функція перейде до обробки відео: для кожного відео файлу буде виконано відкриття файлу, запуск таймера, покадрове покращення, збереження у вихідному каталозі та вивід результатів.

Для відкриття зображень та відео використовується відкрита бібліотека «imageio-ffmpeg» [18]. Вона дозволяє зчитувати зображення у вигляді масивів даних, виконувати покадрове зчитування відео та багато іншого.

```
def __init__(self, input_dir, output_dir='./data/output', verbose=True):

    self.input_dir = Path(input_dir)
    self.data_name = self.input_dir.name
    self.output_dir = Path(output_dir) / self.data_name
    self.logger = get_logger(__name__)
    if not verbose:
        self.logger.setLevel(40)
    self.img_extensions = ('.jpeg', '.jpg', '.png')  # допустимі розширення зображень
    self.img_ls = [f for f in self.input_dir.iterdir() if f.suffix in self.img_extensions]
    self.video_extensions = ('.wmv', '.mkv', '.mp4', '.avi', '.mpeg')
    self.video_ls = [f for f in self.input_dir.iterdir() if f.suffix in self.video_extensions]
    if ( (len(self.img_ls) < 1) and (len(self.video_ls) < 1)):
        self.logger.error('Коректних зображень не знайшлося (перевірте конфігураційний файл).')
        raise ValueError('Коректних зображень не знайшлося (перевірте конфігураційний файл).')
    # Створити каталог для результатів
    if not self.output_dir.exists():
        self.logger.info('Створюю вихідний каталог:\n{}'.format(self.output_dir))
        self.output_dir.mkdir(parents=True)
```

Рисунок 3.6 – Фрагмент коду, конструктор класу Predictor

Конструктор задає поля об’єкта, в нашому випадку це вхідний та вихідний каталог, об’єкт логування, допустимі розширення відео та зображення, а також масиви шляхів до зображень та відео, які необхідно покращити.

Додаток працює із найпоширенішими розширеннями відео та зображень. Він може покращити зображення із розширенням «jpeg», «jpg», «png» та відео із розширенням «wmv», «mkv», «mp4», «avi», «mpeg». Список розширень лімітується підтримуваними форматами бібліотеки «imageio-ffmpeg» [18].

3.2.3 Розробка модуля тренування «Trainer»

Модуль тренування виконує навчання моделі для формування вагів, які потім будуть використовуватися користувачем для роботи з додатком. Процес навчання полягає у обробці зображення програмним додатком, порівнянням отриманого результату із еталонним за допомогою обчислення значень похибок та корегування вагів для покращення результатів. Даний процес виконується до досягнення бажаного результату (отримати похибку менше ніж задану), або певну кількість епох – ітерацій. Спрощену схему процесу навчання наведено на рисунку 3.7.

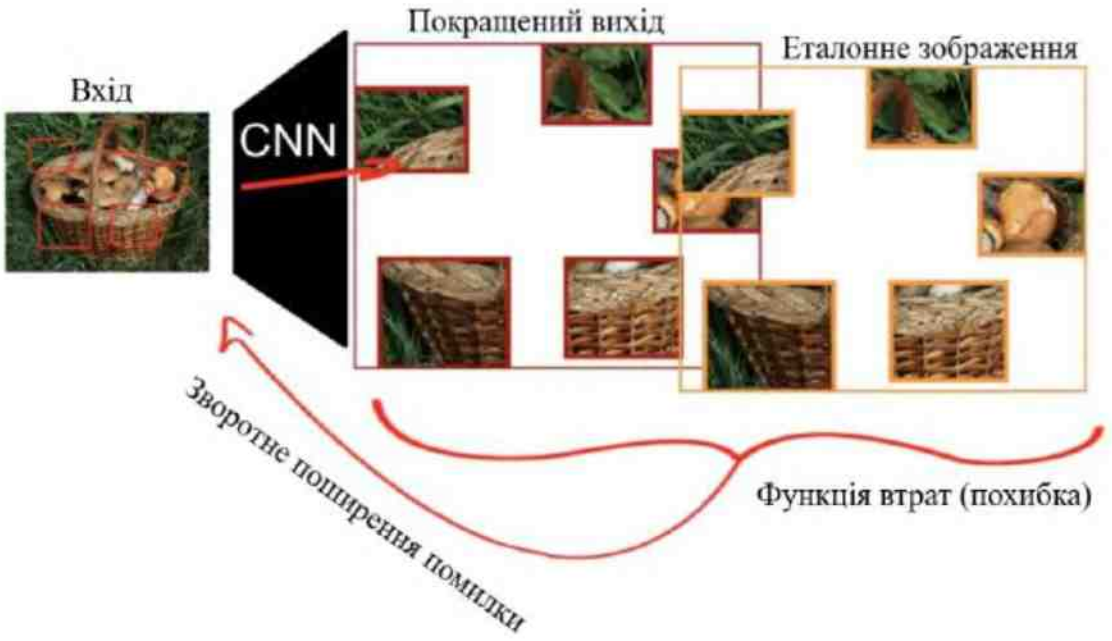


Рисунок 3.7. – Спрощена схема процесу навчання

Особливістю розроблюваного додатку є те, що при навчанні на вхід до моделі подається не повне зображення, а ділянки певного розміру витягнуті у випадковому місці із вхідного зображення. Таким чином одне зображення можна використати декілька разів, збільшуючи навчальну вибірку. Також такий підхід є ефективним з точки зору використовуваних ресурсів системи, оскільки не має необхідності зберігати та опрацьовувати величезні масиви даних. На рисунку 3.8 представлено фрагмент коду із функції навчання, що відповідає за отримання випадкових ділянок зображення, їх покращення, обрахування помилки та відображення результату.

```
epoch_start = time()
for step in tqdm(range(steps_per_epoch)):
    batch = self.train_dh.get_batch(batch_size, flatness=flatness)
    y_train = [batch['hr']]
    training_losses = {}
    model_losses = self.model.train_on_batch(batch['lr'], y_train)
    model_losses = self._format_losses('train_', model_losses, self.model.metrics_names)
    training_losses.update(model_losses)

    self.tensorboard.on_epoch_end(epoch * steps_per_epoch + step, training_losses)
    self.logger.debug('Похибки на кроці {s}:\n {l}'.format(s=step, l=training_losses))

elapsed_time = time() - epoch_start
self.logger.info('Епоха {} заняла {:10.1f}s'.format(epoch, elapsed_time))
```

Рисунок 3.8 – Фрагмент коду навчання

Даний фрагмент виконує опрацювання кроків епохи (від англ. – Epoch). Кроки епохи визначають скільки разів необхідно опрацювати різні ділянки за епоху для нормалізації похибки. Кількість кроків за епоху та кількість епох задає сам користувач.

Функція get_batch() виконує формування вибірки для навчання, що складається із випадкових ділянок вхідних даних (зображення в низькому розширенні) та ці ж ділянки у високому розширенні. Похибки та ваги обчислюються у методі train_on_batch, що надається фреймворком Keras. Після їх обчислення, похибки зберігаються за допомогою інструмента TensorBoard в

функції on_epoch_end. Окрім зберігання даних для відображення графіків TensorBoard, вона також зберігає нові ваги.

3.3 Використання програмного додатку

Перед використанням програмного додатку, користувачу необхідно провести інсталяцію на системі. Для цього у користувача є два варіанта: використати docker для простої та швидкої інсталяції, провести інсталяцію напряму.

Для інсталяції за допомогою docker, користувач повинен встановити docker на своїй системі, використовуючи інструкцію користувача додатку docker [16]. Потім користувачу необхідно запустити просту команду для проведення інсталяції (див. рисунок 3.9).

Для прямої інсталяції, користувачу необхідно запустити команду «python setup.py install», що встановить всі необхідні пакети для роботи додатку. Даний спосіб не є рекомендованим, оскільки можлива вірогідність виникнення конфліктів вже встановлених бібліотек python, які необхідно буде вирішувати самому, проте в цього метода є свої переваги: можливість інтегрувати існуючі класи до іншої системи.

```
user@user-To-be-filled-by-O-E-M:~/Desktop/SR_diplom$ sudo docker build -t vsr . -f Dockerfile.cpu
Sending build context to Docker daemon    143MB
Step 1/13 : FROM tensorflow/tensorflow:1.13.1-py3
 ---> 4ddde1227df8
Step 2/13 : RUN apt-get update && apt-get install -y --no-install-recommends       bzip2       g++       git
ennpl-bin       screen       wget &&     rm -rf /var/lib/apt/lists/*     apt-get upgrade
 ---> Using cache
 ---> bc407f70e121
Step 3/13 : ENV TENSOR_HOME /home/vsr
 ---> Using cache
 ---> d718eb02759b
 ---> 813d19121a12
Step 12/13 : ENV PYTHONPATH ./VSR/:$PYTHONPATH
 ---> Running in 98376e22b609
Removing intermediate container 98376e22b609
 ---> f2c24b0b3ab8
Step 13/13 : ENTRYPOINT ["sh", "./scripts/entrypoint.sh"]
 ---> Running in 81ede90b2f30
Removing intermediate container 81ede90b2f30
 ---> 5305b1b88350
Successfully built 5305b1b88350
Successfully tagged vsr:latest
```

Рисунок 3.9 – Приклад процесу інсталяції використовуючи docker

Після інсталяції користувач може використовувати додаток, але бажано провести першочергове налаштування. В каталозі додатку існує файл «config.yml», що містить всі необхідні параметри для роботи системи: модель мережі за замовчуванням, параметри для проведення навчання за замовчуванням, налаштування мережі за замовчуванням, шляхи до файлу логування, вагів, вхідного та вихідного каталогу.

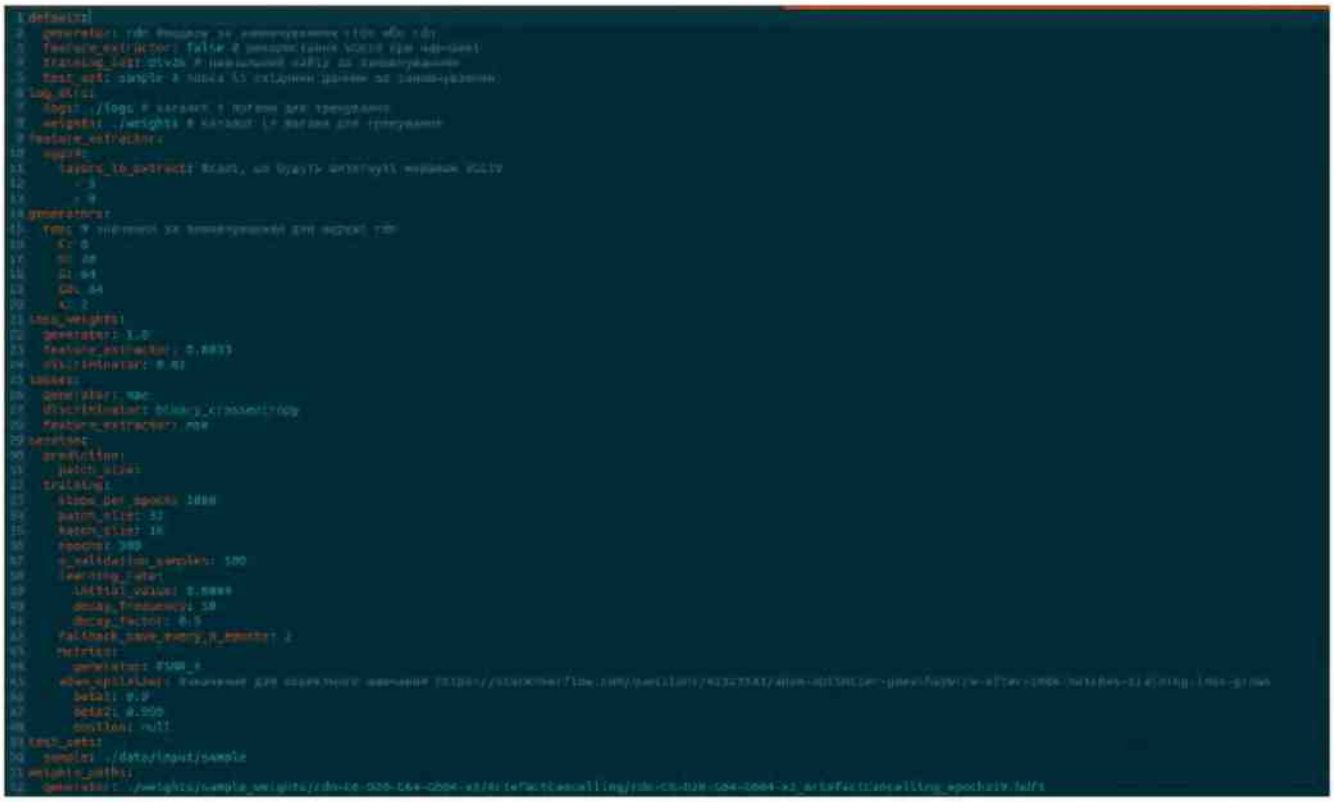


Рисунок 3.10 – Приклад файлу конфігурації

Після опціонального налаштування конфігурації, користувач може запускати та використовувати додаток. Якщо встановлення проводилось за допомогою docker, то користувачу необхідно запустити команду монтування каталогу із контейнера docker до файлової системи із встановленим додатком, та вказати ім'я контейнера (vsr – за замовчуванням, від Video Super Resolution) із вказанням ключа «c» та шляхом до файлу конфігурації (див. рисунок 3.11). Для простоти

використання було розроблено скрипт «assistant», який виконує взаємодію між компонентами системи.

Рисунок 3.11 – Приклад використання додатку із значеннями за замовчуванням

На рисунку вище зображено запуск процесу покращення із використанням параметрів за замовчуванням в контейнері docker.

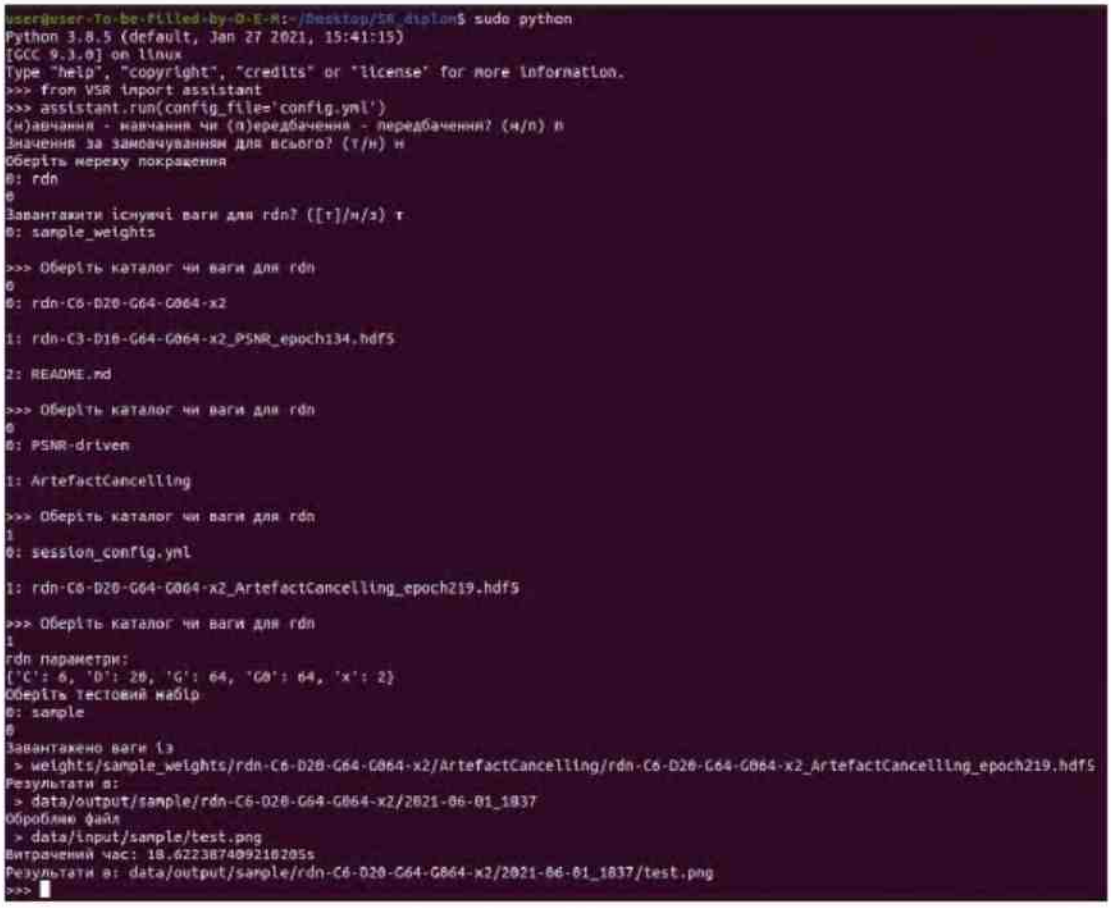


Рисунок 3.12 – Використання додатку із вказанням параметрів

На рисунку вище зображено процес використання додатку через скрипт python (використовується, коли додаток встановлений напряму). Для прикладу було обрано всі параметри самостійно. Також при прямій інсталяції є можливість використовувати існуючі класи напряму та інтегрувати додаток в будь-яку систему, якщо в цьому існує необхідність (див. рисунок 3.13).

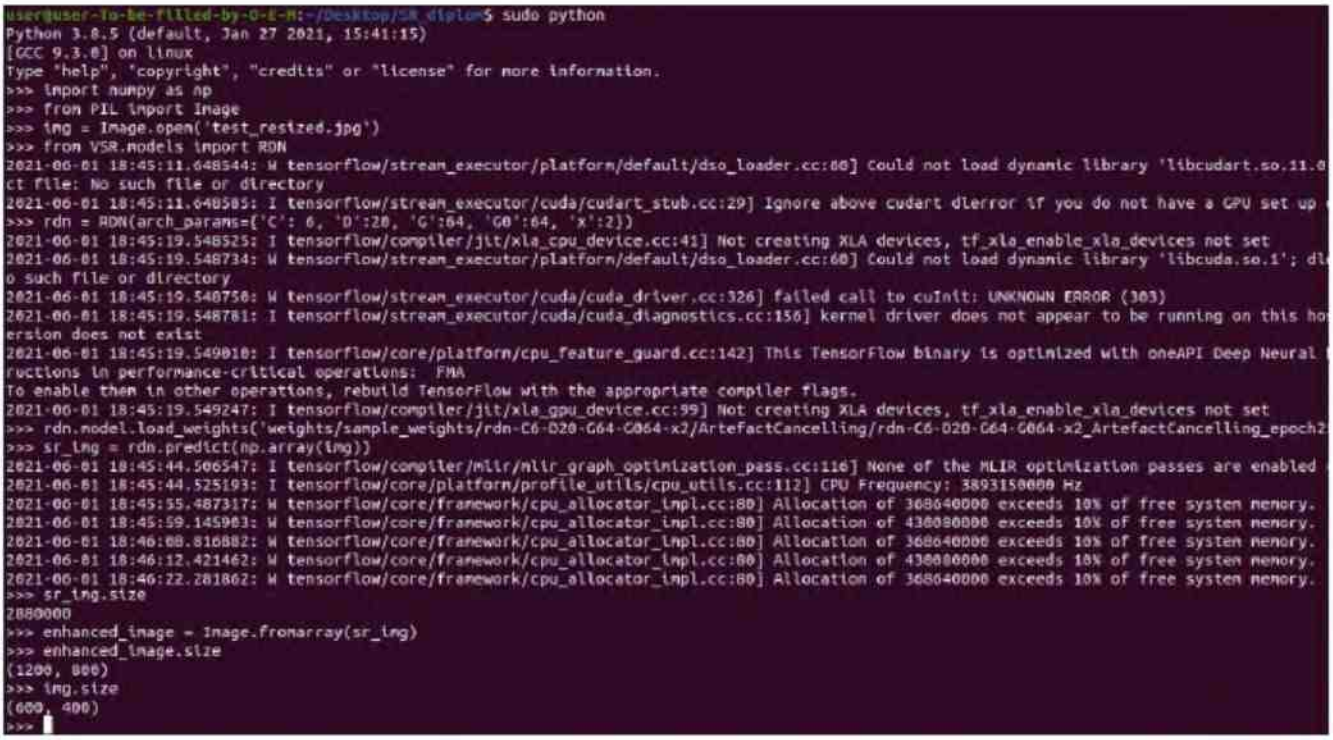


Рисунок 3.13 – Приклад прямого виклику методів існуючих модулів

На прикладі вище зображено процес відкриття зображення, ініціалізації моделі RDN, покращення зображення, перегляд розширення зображення до покращення і після. Також на зображені існує певна кількість повідомлень від tensorflow, які пов’язані з тим, що система з прикладу не має дискретної відеокарти, тому tensorflow видає повідомлення що відеокарти не знайдено і що для обчислень буде використано центральний процесор.

В процесі роботи додаток генерує лог файл, в якому зберігається історія всіх операцій та всіх опцій, що було введено користувачем (див. рисунок 3.14).

```
2021-06-01 18:37:43,021 - VSR.utils.utils - INFO - Оберіть мереху покращення
2021-06-01 18:37:43,021 - VSR.utils.utils - INFO - 0: rdn
2021-06-01 18:37:44,797 - VSR.utils.utils - INFO - 0: sample_weights

2021-06-01 18:37:45,558 - VSR.utils.utils - INFO - 0: rdn-C6-D20-G64-G064-x2

2021-06-01 18:37:45,558 - VSR.utils.utils - INFO - 1: rdn-C3-D10-G64-G064-x2_PSNR_epoch134.hdf5

2021-06-01 18:37:45,558 - VSR.utils.utils - INFO - 2: README.md

2021-06-01 18:37:50,709 - VSR.utils.utils - INFO - 0: PSNR-driven

2021-06-01 18:37:50,709 - VSR.utils.utils - INFO - 1: ArtefactCancelling

2021-06-01 18:37:53,145 - VSR.utils.utils - INFO - 0: session_config.yml

2021-06-01 18:37:53,145 - VSR.utils.utils - INFO - 1: rdn-C6-D20-G64-G064-
x2_ArtefactCancelling_epoch219.hdf5

2021-06-01 18:37:54,649 - VSR.utils.utils - INFO - rdn параметри:
2021-06-01 18:37:54,650 - VSR.utils.utils - INFO - {'C': 6, 'D': 20, 'G': 64, 'G0': 64, 'x': 2}
2021-06-01 18:37:54,650 - VSR.utils.utils - INFO - Оберіть тестовий набір
2021-06-01 18:37:54,650 - VSR.utils.utils - INFO - 0: sample
2021-06-01 18:37:59,556 - VSR.predict.predictor - INFO - Завантажено ваги із
 > weights/sample_weights/rdn-C6-D20-G64-G064-x2/ArtefactCancelling/rdn-C6-D20-G64-G064-
x2_ArtefactCancelling_epoch219.hdf5
2021-06-01 18:37:59,894 - VSR.predict.predictor - INFO - Результати в:
 > data/output/sample/rdn-C6-D20-G64-G064-x2/2021-06-01_1837
2021-06-01 18:37:59,897 - VSR.predict.predictor - INFO - Обробляю файл
 > data/input/sample/test.png
2021-06-01 18:38:18,520 - VSR.predict.predictor - INFO - Витрачений час: 18.6223874092102055
2021-06-01 18:38:18,520 - VSR.predict.predictor - INFO - Результати в: data/output/sample/rdn-C6-D20-G64-
G064-x2/2021-06-01_1837/test.png
```

Рисунок 3.14 – Приклад згенерованого лог файлу

На прикладах нижче зображено вхідні дані та отриманий результат із використанням програмного додатку.

Рисунок 3.15 – Приклад вхідного зображення

Рисунок 3.16 – Приклад отриманого результату

Користувач також має можливість самостійно створити власні ваги, для цього необхідно завантажити існуючий датасет зображень, або ж створити власний набір із зображень високого розширення та їх копій в низькому розширенні. Після чого запустити скрипт асистента (див. рисунок 3.17) , або ж звернутися до класу Trainer напряму.

```
from VSR import assistant
assistant.run(config_file='config.yml')
```

Рисунок 3.17 – Приклад запуску скрипту асистента

Після запуску асистента необхідно обрати пункт навчання, обрати модель та інші параметри навчання (див. рисунок 3.18). Після вказання всіх параметрів навчання автоматично почнеться, користувач буде бачити прогрес навчання (див. рисунок 3.19), також будуть створені лог файли для відображення графіків tensorboard. По закінченню навчання додаток автоматично створить нові ваги, які можна буде використовувати в додатку.

```
from VSR import assistant
assistant.run(config_file='config.yml')
```

```
(н)авчання - навчання чи (п)ередбачення - передбачення? (н/п) н
Значення за замовчуванням для всього? (т/н) н
Оберіть мережу покращення
0: rdn
0
Завантажити існуючі ваги для rdn? ([т]/н/з) н
Завантажити параметри за замовчуванням для rdn? (т/н) т
Параметри за замовчуванням rdn.
rdn параметри:
{'C': 6, 'D': 20, 'G': 64, 'G0': 64, 'x': 2}
Використовувати ваги за замовчуванням для компоненту втрат (loss)? (т/н) т
Використовувати конкурентну мережу? (т/н) н
Використосувати feature extractor? (т/н) н
Моніторити стандартні показники? (т/н) т
Оберіть набір для навчання
0: custom data
1: div2k
1
```

Рисунок 3.18 – Приклад визначення параметрів для навчання

Рисунок 3.19 – Приклад проведення навчання

3.4 Порівняльний аналіз програмного додатку із аналогом waifu2x

Порівняємо створений програмний продукт із продуктом аналогом – waifu2x. Для порівняння будемо використовувати зображення зображення університету

(див. рисунок 3.20), яке буде зжато в 2 рази з втратою якості 50% (див. рисунок 3.21). Отримане зображення будемо використовувати як вхідні дані до програмних продуктів. На рисунках буде представлено лише частку зображення, оскільки при представлені повного зображення буде важко розгледіти деталі. На рисунках нижче представлено еталонні дані, вхідні та вихідні дані із використанням власного додатку та додатку аналогу – waifu2x.

Рисунок 3.20 – Частина еталонного зображення

Рисунок 3.21 – Погіршене зображення (вхідні дані)

Рисунок 3.22 – Фрагмент покращеного зображення створеним додатком (вихідні дані)

Рисунок 3.23 – Фрагмент покращеного зображення продуктом аналогом (вихідні дані)

Значення метрик для зображення отриманого створеним додатком становлять: PSNR - 43.221, SSIM – 0.971. Значення метрик для зображення отриманого продуктом аналогом: PSNR - 41.623, SSIM – 0.961.

Продукт аналог виявився не набагато гіршим (різниця SSIM – 0.01), якщо взяти до уваги, що він використовує застарілу мережу. Даний факт можливо легко пояснити різницею в тренуванні. Не можна сказати точно як саме тренувалась мережа аналогу, але можна точно сказати, що з ресурсами доступними аналогу, використовувався значно більший набір даних та проводилось значно довше тренування мережі.

Проте, під час проведення експериментів та використання різних вагів в додатку, було виявлено що кількісні показники метрик, які широко використовуються в оцінці якості отриманого зображення є досить хибними з точки зору естетичного результату. Наприклад, на рисунку 3.24 зображено цей же самий фрагмент, але із використанням іншого набору вагів. Значення метрик для отриманого зображення: PSNR - 39.206, SSIM - 0.958.

Рисунок 3.24 – Фрагмент зображення із використанням невдалих вагів

Цього разу ми також маємо невелику різницю значень між зображеннями на рисунку 3.24 та рисунку 3.23, проте зображення на рисунку 3.23 візуально значно краще за тільки но отримане зображення. Різниця SSIM становить 0.003 одиниці, що менше ніж попереднє порівняння, але візуальна різниця є значно більшою. Тому, можна вважати, що методи оцінки якості отриманого зображення не є ефективними.

ВИСНОВКИ

Актуальність програмного додатку обумовлюється великою кількістю можливих підходів до рішення проблеми та незначною кількістю створених продуктів, що реалізують ці підходи. В результаті аналізу функціоналу аналогів було визначено, що вони використовують застарілий підхід, який можливо покращити з використанням нової архітектури нейронної мережі. Також можна додати можливість обробки відео–зображення, і зробити продукт платформонезалежним.

Було визначено актуальність рішень задач ПРЗ в областях охорони, медицини, астрономії та інших. Тому покращення в способі, архітектурі чи доступності вирішення таких задач із впровадженням в ці області технологій штучного інтелекту, дозволить в них збільшити набір інструментів для досягнення результатів.

Було визначено, що існуючі чисельні методи перевірки отриманого результату не співпадають із зоровим відчуттям людини, тобто отримані 2 зображення покращені різними способами з однаковими показниками не гарантують однакову візуальну якість зображення.

Майже всі реконструкційні методи, що базуються на нейронних мережах, використовують значення PSNR або SSIM під час навчання і є завідома хибними, що погіршує отриманий результат.

Для усунення цього недоліку необхідно розробити власний метод оцінювання, який буде визначати візуальну подібність зображення до еталонного. Огляд результатів публікацій з цього напряму вказує, що одним із способів досягнення цього результату може бути використання генеративно-змагальної мережі для оцінки отриманого зображення.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Shi W, Caballero J, Ledig C, Zhuang X, Bai W, Bhatia K, de Marvao AM, Dawes T, O'Regan D, Rueckert D. Cardiac image super–resolution with global correspondence using multi–atlas patchmatch. Med Image Comput Comput Assist Interv. 2013 p; C. 9–16. doi: 10.1007/978–3–642–40760–4_2. PMID: 24505738.

2. Zou WW, Yuen PC. Very low resolution face recognition problem. IEEE Trans Image Process. 2012 p. doi: 10.1109/TIP.2011.2162423. PMID: 21775262.

3. T. Karras, T. Aila, S. Laine, J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation [Електронний ресурс] / Tero Karras // International Conference on Learning Representations. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1710.10196.

4. K. G. Puschmann, F. Kneer. On super–resolution in astronomical imaging [Електронний ресурс] // A&A. – 2005. – Режим доступу до ресурсу: https://www.aanda.org/articles/aa/abs/2005/22/aa2320–04/aa2320–04.html.

5. C. Dong, C. C. Loy, K. He, X. Tang. Image Super–Resolution Using Deep Convolutional Networks [Електронний ресурс] // European Conference on Computer Vision. – 2014. – Режим доступу до ресурсу: https://arxiv.org/abs/1501.00092.

6. C. Dong, C. C. Loy, X. Tang. Accelerating the Super–Resolution Convolutional Neural Network [Електронний ресурс] // European Conference on Computer Vision. – 2016. – Режим доступу до ресурсу: https://arxiv.org/abs/1608.00367.

7. J. Kim, J. Kwon Lee, K. Mu Lee. Accurate Image Super–Resolution Using Very Deep Convolutional Networks [Електронний ресурс] // Conference on Computer Vision and Pattern Recognition. – 2016. – Режим доступу до ресурсу: https://arxiv.org/abs/1511.04587.

8. W.–S. Lai, J.–B. Huang, N. Ahuja, M.–H. Yang. Deep Laplacian Pyramid Networks for Fast and Accurate Super–Resolution [Електронний ресурс] // Conference

on Computer Vision and Pattern Recognition. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1704.03915.

9. Y. Tai, J. Yang, X. Liu, C. Xu. MemNet: A Persistent Memory Network for Image Restoration [Електронний ресурс] // International Conference on Computer Vision. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1708.02209.

10. B. Lim, S. Son, H. Kim, S. Nah, K. M. Lee. Enhanced Deep Residual Networks for Single Image Super–Resolution [Електронний ресурс] // Conference on Computer Vision and Pattern Recognition Workshops. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1707.02921.

11. Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., Fu, Y. Image Super–Resolution Using Very Deep Residual Channel Attention Networks [Електронний ресурс] // European Conference on Computer Vision. – 2018. – Режим доступу до ресурсу: https://arxiv.org/abs/1807.02758.

12. B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced Deep Residual Networks for Single Image Super–Resolution [Електронний ресурс] // Conference on Computer Vision and Pattern Recognition Workshops. – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1707.02921.

13. Z.Yulun, T.Yapeng, K.Yu, Z.Bineng, F.Yun. Residual Dense Network for Image Super–Resolution [Електронний ресурс] // Conference on Computer Vision and Pattern Recognition. – 2018. – Режим доступу до ресурсу: https://arxiv.org/abs/1802.08797.

14. Waifu2x [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/nagadomi/waifu2x.

15. Waifu2x–caffe [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/lltcggie/waifu2x–caffe.

16. Docker [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/.

17. Платформа TensorFlow [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org.

18. Відкрита бібліотека imageio-ffmpeg [Електронний ресурс] – Режим доступу до ресурсу: https://imageio.readthedocs.io/en/stable/format_ffmpeg.html.

19. Обчислення PSNR [Електронний ресурс] – Режим доступу до ресурсу: https://www.mathworks.com/help/vision/ref/psnr.html.

20. Обчислення SSIM [Електронний ресурс] – Режим доступу до ресурсу: https://www.mathworks.com/help/images/ref/ssim.html.

21. Опис архітектури мережі VGG19 [Електронний ресурс] – Режим доступу до ресурсу: https://iq.opengenus.org/vgg19-architecture.

22. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network [Електронний ресурс] / [W. Shi, J. Caballero, F. Huszar´ та ін.] // Computer Vision and Pattern Recognition. – 2016. – Режим доступу до ресурсу: https://arxiv.org/abs/1609.05158.

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
«Програмний додаток підвищення роздільної здатності відео за допомогою нейронних мереж»

А1 Призначення й мета створення програмного додатку

А1.1 Призначення програмного додатку

Програмний додаток має надавати можливість підвищувати роздільну здатність початкового відео запису до вказаного користувачем значення.

А1.2 Мета створення програмного додатку

Метою створення програмного додатку є аналіз та усунення типових недоліків існуючих додатків, що працюють над підвищенням якості зображення. Головні недоліки, які необхідно побороти: вузька спеціалізованість, швидкодія. Вузька спеціалізованість полягає в тому, що існуючі додатки добре справляються із покращенням зображення лише певного типу. Проблема швидкодії полягає в тому, що на обробку одного зображення може бути витрачено більше однієї хвилини часу.

А1.3 Цільова аудиторія

До цільової аудиторії можна віднести власників а також користувачів відео–сервісів, науковців в області штучного інтелекту, дослідників методів покращення якості зображення.

А2 Вимоги до програмного додатку

А2.1 Вимоги до програмного додатку в цілому

А2.1.1 Вимоги до структури й функціонування програмного додатку

Програмний додаток повинен надавати можливість тренування моделі штучної мережі, збільшувати роздільну здатність відео та зображення, підтримувати користувацькі налаштування якості, робити розрахунки похибки відносно еталонного зображення. Працездатність самої програми не повинна залежати від апаратного забезпечення, має бути використана технологія докеризації для забезпечення мобільності та крос–платформеності.

А2.1.2 Вимоги до персоналу

Від персоналу не має вимагатися особливих технічних навичок для підтримки й експлуатації додатку, окрім загальних навичок роботи з персональним комп'ютером.

А2.1.3 Вимоги до збереження інформації

Інформація необхідна для функціонування програмного додатку повинна зберігатися на файловій системі користувача. Інформація, що є результатом роботи програмного додатку повинна бути збережена на файловій системі користувача поза межами внутрішнього захищеного середовища.

А2.1.4 Вимоги до розмежування доступу

Розроблюваний додаток має бути загальнодоступним та не виділяти різні права доступу.

А2.2 Структура програмного додатку

А2.2.1 Загальна інформація про структуру програмного додатку

Структура додатку являє собою набір файлів, які реалізують логіку роботи програмного додатку. До таких файлів можна віднести:

Конфігураційний файл docker — файл із налаштуваннями для успішної інсталяції та запуску програми.

Конфігураційний файл системи — файл із налаштуваннями поточних параметрів програми.

Скрипти для роботи із відео — файли із інструкціями, що реалізують основні команди для роботи із відео зображенням.

Скрипти для роботи із звуком — файли із інструкціями, що реалізують основні команди для роботи зі звуком.

Скрипти для роботи із штучним інтелектом — файли із інструкціями, що реалізують можливість створення, побудову та налаштування нейронної мережі.

Скрипт контролер — файл із інструкціями, що реалізує основну логіку роботи програми.

Файли початкових тренувальних даних — набір даних, що будуть використовуватись для тренування штучного інтелекту.

А2.2.2 Навігація

Навігація надається стандартними засобами операційної системи користувача. Користувач повинен мати можливість вказати каталог із кінцевими даними.

А2.2.3 Наповнення програмного додатку (контент)

Програмний додаток повинен включати в собі набір вагів, які можуть бути використані для покращення вхідних даних. Набір вагів повинен бути створений розробником програмного додатку. Користувач повинен мати можливість обрати ваги для покращення та створити свої власні.

А2.2.4 Дизайн та структура програмного додатку

Додаток повинен містити документацію по методам та класам. Назви методів повинні бути осмислені та відповідати їх функціоналу. Приклад схеми внутрішньої структури додатку наведено нижче. Він слугує для візуального представлення структури додатку, тому створені модулі та їх взаємозв'язки можуть відрізнятися від представлених.

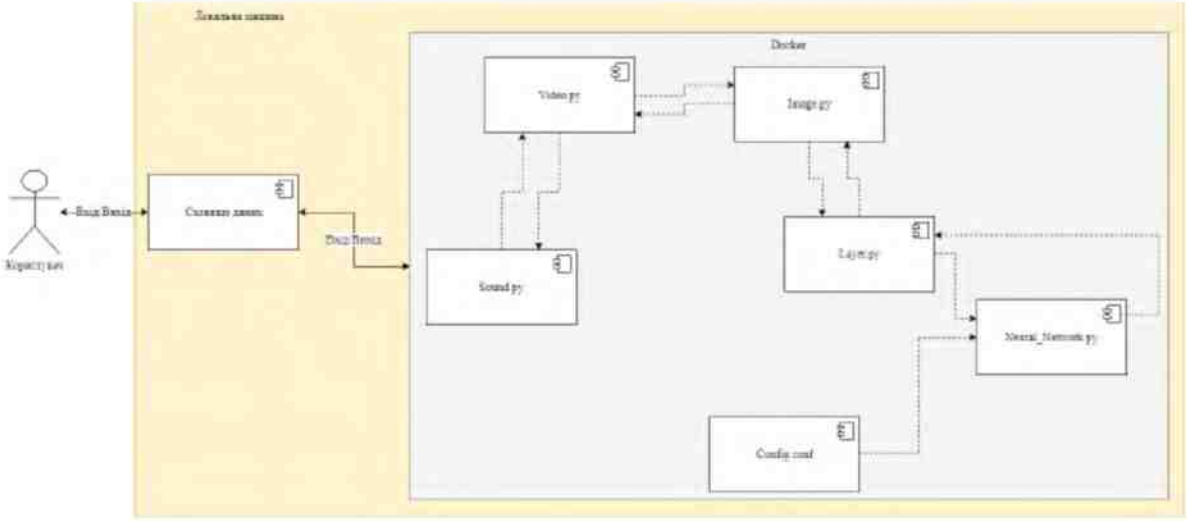


Рисунок А.1 – Приклад структури додатку

А2.3 Вимоги до функціонування програмного додатку

А2.3.1 Потреби користувача

Потреби користувача, визначені на основі рішення замовника, представлені у таблиці А.1.

Таблиця А.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN–01	Завантаження тренувальних даних	Користувач
UN–02	Тренування нейронної мережі	Користувач
UN–03	Збільшення роздільної здатності зображення	Користувач
UN–04	Збільшення роздільної здатності відео	Користувач
UN–05	Отримання статистичних даних про результат тренування	Користувач
UN–06	Отримання статистичних даних про результат обробки	Користувач
UN–07	Налаштування бажаних параметрів обробки (вибір роздільної здатності, якості)	Користувач
UN–08	Отримання підсказок на будь–якому етапі роботи з системою	Користувач

А2.3.2 Функціональні вимоги

На основі потреб користувача були визначені такі функціональні вимоги:

– Підтримка користувацьких налаштувань в конфігураційному файлі;

– Збільшення роздільної здатності відео та зображення;

– Тренування нейронної мережі;

– Збір даних та оцінка процесу навчання за допомогою PSNR, SSIM.

А2.3.3 Системні вимоги

Даний розділ визначає, розподіляє та вказує на системні вимоги, визначені розробником. Їх перелік наведений в таблиці А.2.

Таблиця А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR–01	Модуль збіру статистики	M	Надає можливість збирати статистичні дані по прогресу навчання та оцінити успішність навчання за допомогою PSNR, SSIM.
SR–02	Модуль штучного інтелекту	M	Надає можливість побудови, тренування та використання нейронної мережі
SR–03	Модуль роботи із відео	M	Надає можливість розбити відео на зображення та звукову доріжку
SR–04	Модуль роботи із зображенням	M	Надає можливість роботи із зображенням та конвертуванням даних для подачі їх до штучного інтелекту
SR–05	Модуль контроллера	M	Керує роботою інших модулів
SR–06	Модуль збірки	S	Збирає отримані зображення та звук у відео контейнер

Продовження таблиці А.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR–07	Модуль конфігуратора	M	Відповідає за поточні налаштування системи
SR–08	Форма інтерфейсу користувача	C	Використання форми програми, замість терміналу
SR–09	API	C	Надати можливість зовнішнім розробникам використовувати внутрішні методи.

Умовні позначення в таблиці А.2:

Must have (M) – вимоги, які повинні бути реалізовані в додатку;

Should have (S) – вимоги, які мають бути виконані, але вони можуть почекати своєї черги;

Could have (C) – вимоги, які можуть бути реалізовані, але вони не є центральною ціллю проекту.

А2.4 Вимоги до видів забезпечення

А2.4.1 Вимоги до інформаційного забезпечення

Стек технологій, що може бути використаний для реалізації:

- C#
- Phyton та надбудови для роботи із нейронними мережами
- MySQL
- Docker
- Docker compose

А2.4.2 Вимоги до лінгвістичного забезпечення

Програмний додаток має бути виконаний українською мовою.

А2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення користувача повинне задовольняти наступним вимогам:

- Windows 10 64–bit (build 17134+);
- Підтримка Windows Hyper–V та Containers

Або:

- 64–bit Linux
- Linux kernel version 3.10+
- iptables version 1.4+
- git version 1.7+
- XZ Utils 4.9+

А2.4.4 Вимоги до апаратного забезпечення

Апаратне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- 4GB RAM;
- 64 bit processor із підтримкою Second Level Address Translation;
- увімкнена віртуалізація на рівні BIOS;
- опціонально – наявність відеокарти Nvidia для використання cuda ядер, інакше буде використано ресурси центрального процесора.

А3 Склад і зміст робіт зі створення програмного додатку

Докладний опис етапів роботи зі створення програмного додатку наведено в таблиці А.3.

Таблиця А.3 – Етапи створення програмного додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Визначення аналогів	1 день
2	Визначення недоліків	2 дня
3	Визначення цілей	7 днів
4	Розробка технічного завдання	5 днів
5	Проектування додатку	10 днів
6	Реалізація програмного додатку	15 днів
7	Тестування програмного додатку	15 днів
8	Підтримка додатку (створення супровідної документації)	5 днів
	Загальна тривалість робіт	60 днів

А4 Вимоги до складу й змісту робіт із введення програмного додатку в експлуатацію

Для використання додатку, необхідно впевнитися, що клієнтська система відповідає вимогам пунктів 2.4.3–2.4.4. Після чого, необхідно розмістити файли програмного додатку на файловій системі користувача. Розмістивши файли, необхідно обрати потрібні параметри інсталяції у файлі типу yml (користуючись інструкцією, наведеною в файлах системи), або ж просто запустити команду збірки, без користувацьких налаштувань. Після збірки користувач може за допомогою консольних команд або графічного інтерфейсу обирати файли для обробки, змінювати налаштування мережі та здійснювати її тренування.

ДОДАТОК Б
ПЛАНУВАННЯ РОБІТ

Для правильного визначення мети проекту, доцільно використовувати метод SMART. Правильно визначена мета дозволяє краще визначити кінцевий продукт та забезпечить визначення очікуваного результату. SMART:

– S (specific) – конкретність, специфічність. Система підвищення розширення вхідного зображення та його якості;

– M (measurable) – вимірюваність. Для виміру системи можна взяти еталонне зображення — зображення високої якості. Після чого понизити його якість, і відповідно знову покращити за допомогою створюваного програмного додатку. Використовуючи інформаційну систему, можна порівняти покращене зображення із еталонним для отримання конкретних даних. Систему можна вважати успішною, якщо покращене зображення з 720p до 1080p (кількість пікселів збільшено в двічі) буде відповідати більше ніж на 50% до еталонного;

– A (achievable) – досяжність. На даний момент існує декілька наукових статей по дослідженню штучного інтелекту в області модифікації зображень. Використовуючи їх, можливо виділити основні моменти та ввібрати краще з різних підходів для генерації нового підходу;

– R (relevant) – актуальність. Окрім користі для окремих груп людей, система несе наукову цінність для дослідження штучного інтелекту в області модифікації зображень;

– T (time–framed) – обмеженість в часі. Термін досягнення мети становить 90 днів. Початок проекту 18.01.21 та кінець 21.05.21.

Опис функціонування продукту проекту

Виходячи з мети можна виділити основні характеристики проекту:

– Створена система повинна функціонувати (приймати відео контейнер із вхідним розширенням зображення та видавати відео контейнер із збільшеним розширенням);

– Відсоток подібності із еталонним зображенням щонайменше 50%;

– Система повинна бути документованою та порівняною із актуальними аналогами.

Планування змісту структури робіт IT—проекту.

Ієрархічна структура робіт WBS, або структура декомпозиції, являє собою схему, де завдання проекту відображають їхнє ставлення один до одного і до проекту в цілому. Процес часто описується як структура відгалуження, яка охоплює всі етапи проекту в організованому порядку.

Структуру декомпозиції використовують щоб структурувати і ділити проекти на легко керовані компоненти. Вони, в свою чергу, поділяються до тих пір, поки вони не призначаються конкретному фахівцю в команді.

Кожен нижчий рівень структури є деталізацію елемента вищого рівня проекту. Елементом проекту може бути як продукт, послуга так і пакет робіт або робота.

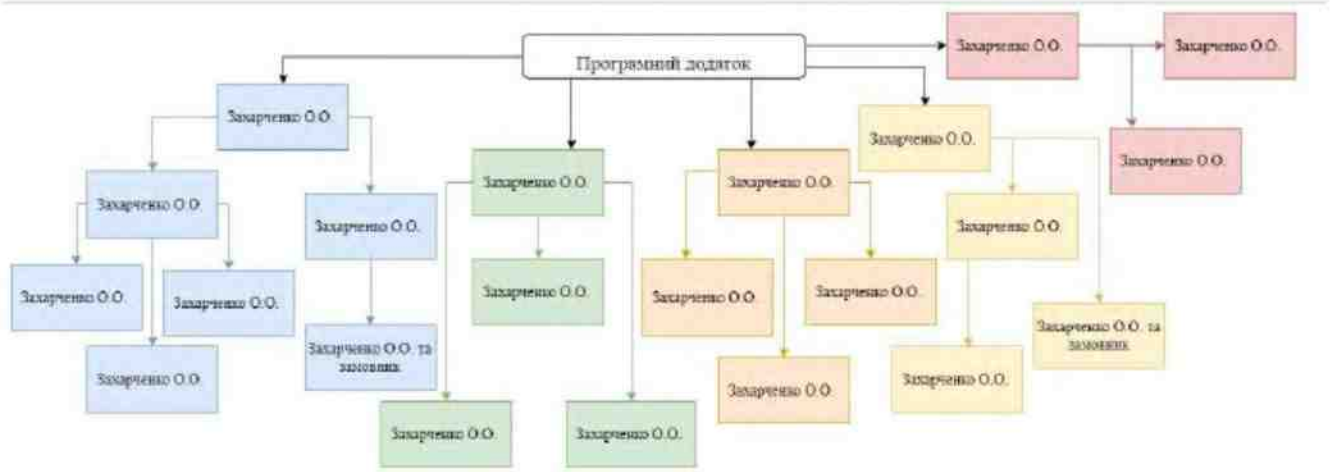
На рисунку Б1.1 представлено WBS схему з розробки програмного додатку.

Рисунок Б1.1 – WBS схема

Планування структури організації.

Подальшим етапом в створення проекту є опис організаційної структури проекту або OBS, яка являє собою ієрархічну структурою управління проектом і показує відносини між учасниками проекту.

На рисунку Б1.2 представлено організаційну структуру проекту.


Програмний додаток
Захарченко О.О.
Захарченко О.О.
Захарченко О.О.

Таблиця Б.1 – Матриця відповідальності проекту

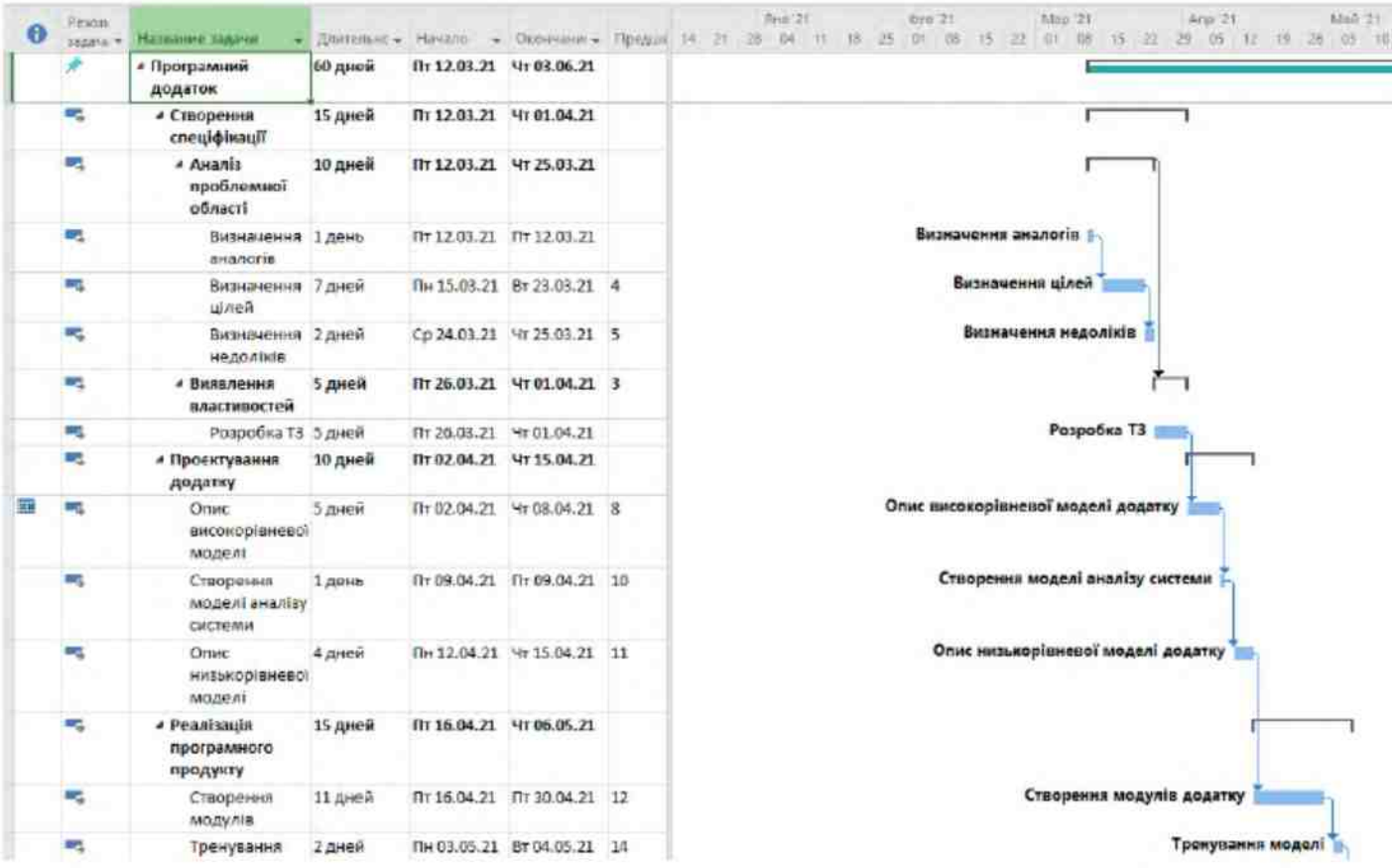
№	Задача	Захарченко О.О.	Марченко А.В.	Аналітична система	Замовник
1	Визначення аналогів	Р	К		
2	Визначення недоліків	Р	К		
3	Визначення цілей	Р	К		
4	Розробка ТЗ	Р	К		У
5	Опис високорівневої моделі додатку	Р	К		
6	Створення моделі аналізу системи	Р	К		
7	Опис низькорівневої моделі додатку	Р	К		
8	Створення модулів додатку	Р	К		
9	Тренування моделі	Р	К	А	
10	Створення docker скриптів	Р	К		
11	Крос-платформне тестування	Р	К		
12	Функціональне тестування	Р	К	А	
13	Створення інструкції інсталяції та користування	Р	К		
14	Опис лімітацій	Р			

Розробка PDM мережі

Управління даними про виріб (PDM) – це система для управління даними про продукти і процеси в рамках єдиної централізованої системи. PDM–системи збирають дані про продукт, керують ними і надають користувачам потрібну інформацію в правильному контексті протягом всього життєвого циклу виробу.

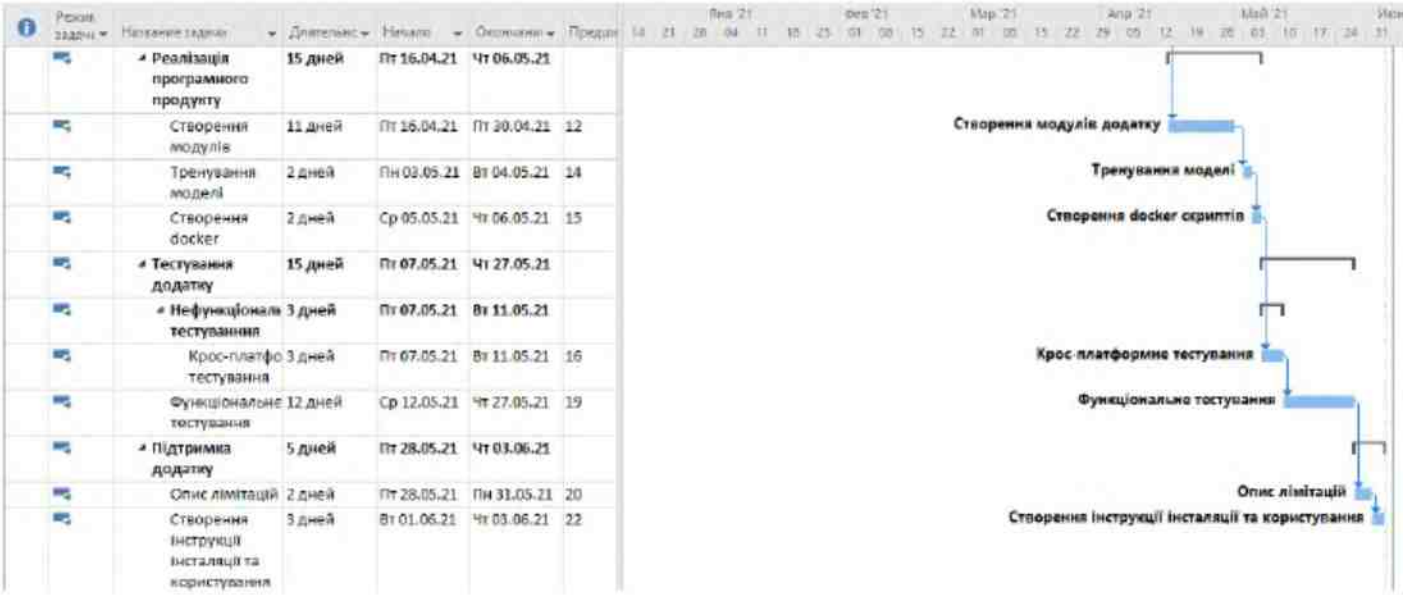
По суті, система PDM пропонує рішення для безпечного управління даними, увімкнення процесів та управління конфігурацією.

Календарний графік з виконання роботи представлений на рисунку Б.1.3–Б.1.4.



	Название задачи	Длительнос	Начало	Окончани	Предшес
	Програмний додаток	**60 дней**	**Пт 12.03.21**	**Чт 03.06.21**	
	Створення специфікацІТ	**15 дней**	**Пт 12.03.21**	**Чт 01.04.21**	
	Аналіз проблемної області	**10 дней**	**Пт 12.03.21**	**Чт 25.03.21**	
	Визначення аналогів	1 день	Пт 12.03.21	Пт 12.03.21	
	Визначення цілей	7 дней	Пн 15.03.21	Вт 23.03.21	4
	Визначення недоліків	2 дней	Ср 24.03.21	Чт 25.03.21	5
	Виявлення властивостей	**5 дней**	**Пт 26.03.21**	**Чт 01.04.21**	3
	Розробка ТЗ	5 дней	Пт 26.03.21	Чт 01.04.21	
	Проектування додатку	**10 дней**	**Пт 02.04.21**	**Чт 15.04.21**	
	Опис високорівневої моделі	5 дней	Пт 02.04.21	Чт 08.04.21	8
	Створення моделі аналізу системи	1 день	Пт 09.04.21	Пт 09.04.21	10
	Опис низькорівневої моделі	4 дней	Пн 12.04.21	Чт 15.04.21	11
	Реалізація програмного продукту	**15 дней**	**Пт 16.04.21**	**Чт 06.05.21**	
	Створення модулів	11 дней	Пт 16.04.21	Пт 30.04.21	12
	Тренування	2 дней	Пн 03.05.21	Вт 04.05.21	14

Рисунок Б.1.3 – Календарний графік



Режим задачи	Название задачи	Длительность	Начало	Окончание	Предшественники
	Реалізація програмного продукту	**15 дней**	**Пт 16.04.21**	**Чт 06.05.21**	
	Створення модулів	11 дней	Пт 16.04.21	Пт 30.04.21	12
	Тренування моделі	2 дней	Пн 03.05.21	Вт 04.05.21	14
	Створення docker	2 дней	Ср 05.05.21	Чт 06.05.21	15
	Тестування додатку	**15 дней**	**Пт 07.05.21**	**Чт 27.05.21**	
	Нефункціональ тестування	**3 дней**	**Пт 07.05.21**	**Вт 11.05.21**	
	Крос-платфо тестування	3 дней	Пт 07.05.21	Вт 11.05.21	16
	Функціональне тестування	12 дней	Ср 12.05.21	Чт 27.05.21	19
	Підтримка додатку	**5 дней**	**Пт 28.05.21**	**Чт 03.06.21**	
	Опис лімітацій	2 дней	Пт 28.05.21	Пн 31.05.21	20
	Створення інструкції інсталяції та користування	3 дней	Вт 01.06.21	Чт 03.06.21	22

Рисунок Б.1.4 – Календарний графік

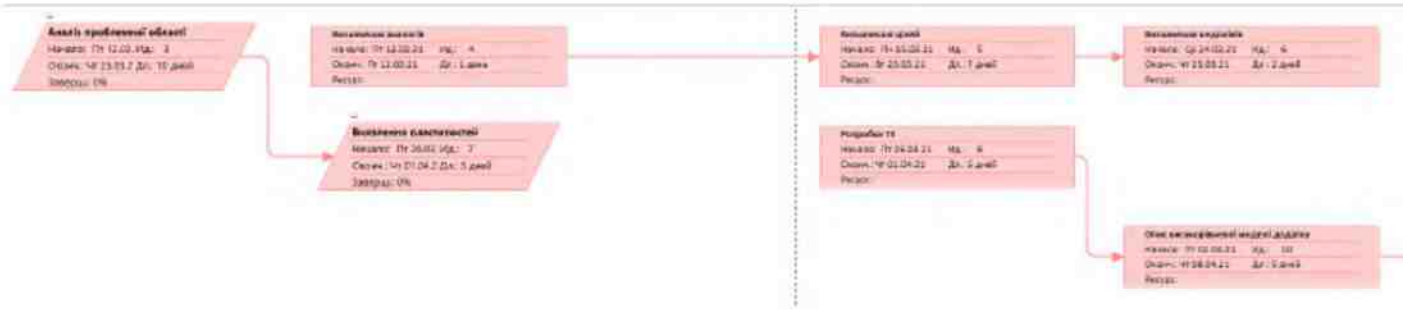


Рисунок Б.1.5 – Фрагмент PDM діаграми

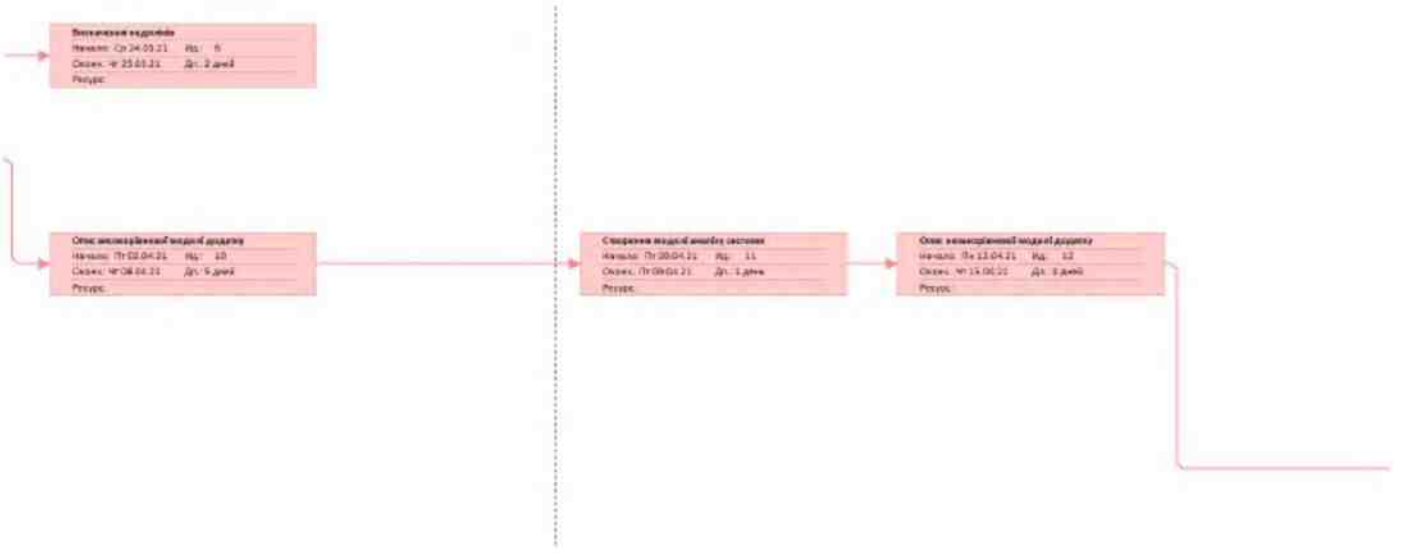


Рисунок Б.1.6 – Продовження PDM діаграми №1

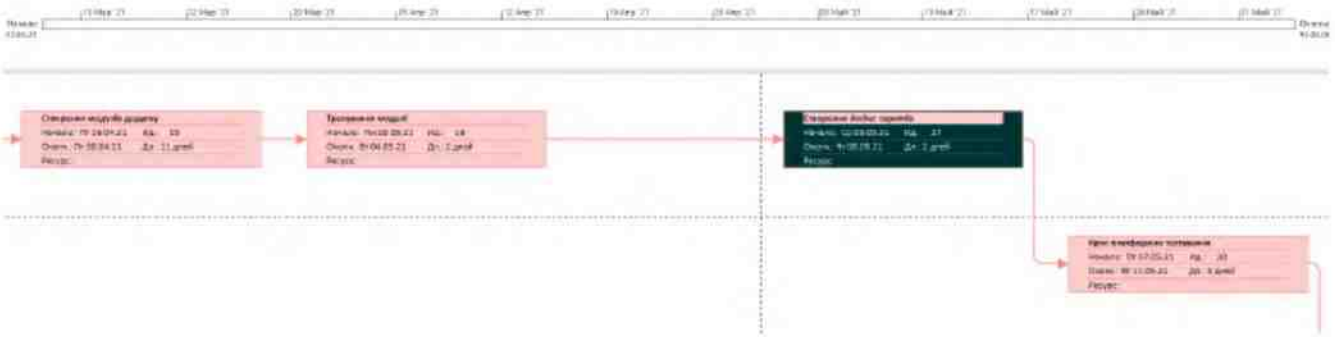


Рисунок Б.1.7 – Продовження PDM діаграми №2

Рисунок Б.1.8 – Продовження фрагменту PDM діаграми №3

Управління ризиками проекту

Ризик – це можливість настання подій з негативними наслідками в результаті певних рішень чи дій.

Аналіз ризиків – це процес, який допомагає виявити та управляти потенційними проблемами, які можуть підірвати проект.

Для проведення аналізу ризиків потрібно спочатку визначити можливі загрози, з якими стикаєтесь, а потім оцінити ймовірність реалізації цих загроз. Першим кроком в аналізі ризиків є виявлення існуючих та можливих загроз, з якими ви можете зіткнутися. Після того, як загрози були визначені, необхідно розрахувати яка ймовірність реалізації цих загроз, так і їх можливий вплив.

Після того, як значення ризиків було визначено, можна почати шукати шляхи управління ними.

У таблиці Б.2 проведена класифікація ризиків.

Таблиця Б.2 – Ризики проекту

Ризик	Назва ризику	Опис ризику
1	Невдала реалізація мережі	Реалізована система не досягла цілей проекту
2	Зміна вимог до системи	Зміна об'ємів, вартості чи терміну робіт замовником
3	Зміна технологій	Поява нових проривних технологій обробки зображень
4	Проблеми фінансування	Недотримання термінів чи обсягів фінансування
5	Низька сумісність	Створена система підтримується лише на обмеженій кількості операційних систем

У таблицях Б.3 – Б.4 описана реєстрація ризиків та матриця впливу відповідно.

Таблиця Б.3 – Реєстрація ризиків

Ризик	Вірогідність виникнення	Ступінь впливу	Значення ризику
1	4	5	20
2	2	3	6
3	2	5	10
4	1	5	5
5	3	3	9

Таблиця Б.4 – Матриця впливу

Вірогідність виникнення	**Матриця впливу**		
5		5	
4		4	
3		3	
2		2	

Після оцінки ризиків було складено план реагування на ризики, який зображено у таблиці Б.5.

Таблиця Б.5 – Реакція на ризики проекту

Ризик	Назва ризику	Реакція на ризик
1	Невдала реалізація мережі	Створення нової мережі іншого типу, тестування та аналіз результатів. Вибір мережі кращого конкурента.
2	Зміна вимог до системи	Зміна часу необхідних для реалізації проекту
3	Зміна технологій	Зміна часу та вимог до реалізації проекту
4	Проблеми фінансування	Припинення розробки до відновлення фінансування, зменшення функціоналу системи
5	Низька сумісність	Використання технологій контейнеризації для забезпечення роботи на більшості систем

ДОДАТОК В
ЛІСТИНГ ПРОГРАМНОГО КОДУ ДОДАТКА

setup.py

```
from setuptools import setup, find_packages

long_description = '''
VSR (Video Super-Resolution) is a library to upscale and improve the quality of low resolution
videos and images.
'''

setup(
    name='VSR',
    version='1.0.0',
    author='Oleksandr Zakharchenko',
    description='Video Super Resolution',
    long_description=long_description,
    license='Apache 2.0',
    install_requires=['imageio', 'imageio-ffmpeg', 'numpy', 'tensorflow==2.*', 'tqdm',
'pyaml'],
    extras_require={
        'gpu': ['tensorflow-gpu==2.*'],
        'dev': ['bumpversion==0.5.3'],
    },
    classifiers=[
        'Development Status :: 4 - Beta',
        'Intended Audience :: Science/Research',
        'License :: OSI Approved :: Apache Software License',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.6',
        'Topic :: Software Development :: Libraries',
        'Topic :: Software Development :: Libraries :: Python Modules',
    ],
    packages=find_packages(exclude=('tests',)),
)
```

Dockerfile.cpu

```
FROM tensorflow/tensorflow:1.13.1-py3

# Install system packages
RUN apt-get update && apt-get install -y --no-install-recommends \
      bzip2 \
      g++ \
      git \
      graphviz \
      libgl1-mesa-glx \
      libhdf5-dev \
```

```
      openmpi-bin \
      screen \
      wget && \
    rm -rf /var/lib/apt/lists/* \
    apt-get upgrade

ENV TENSOR_HOME /home/vsr
WORKDIR $TENSOR_HOME

COPY VSR ./VSR
COPY scripts ./scripts
COPY weights ./weights
COPY config.yml ./
COPY setup.py ./

RUN pip install --upgrade pip
RUN pip install -e .

ENV PYTHONPATH ./VSR/:$PYTHONPATH
ENTRYPOINT ["sh", "./scripts/entrypoint.sh"]
```

Dockerfile.gpu

```
FROM tensorflow/tensorflow:1.13.1-gpu-py3

# Install system packages
RUN apt-get update && apt-get install -y --no-install-recommends \
      bzip2 \
      g++ \
      git \
      graphviz \
      libgl1-mesa-glx \
      libhdf5-dev \
      openmpi-bin \
      screen \
      wget && \
    rm -rf /var/lib/apt/lists/* \
    apt-get upgrade

ENV TENSOR_HOME /home/vsr
WORKDIR $TENSOR_HOME

COPY VSR ./VSR
COPY scripts ./scripts
COPY weights ./weights
COPY config.yml ./
COPY setup.py ./

RUN pip install --upgrade pip
RUN pip install -e ".[gpu]" --ignore-installed

ENV PYTHONPATH ./VSR/:$PYTHONPATH
ENTRYPOINT ["sh", "./scripts/entrypoint.sh"]
```

entrypoint.sh

```
#!/usr/bin/env bash
script_dir="$( cd "$(dirname "$0")" ; pwd -P )"
main_dir="$(dirname $script_dir)"

echo "Привіт. Програму асистента запущено."
echo ""

train_flag=""
prediction_flag=""
default_flag=""
config_file=""

print_usage() {
  printf "Використання:"
  printf "-c : шлях до файлу конфігурації "
  printf "-t : сессія тренування "
  printf "-p : сессія передбачення "
  printf "-d : завантажити значення за замовчуванням із файлу конфігурації "
}

while getopts 'ptdc:' flag; do
  case "${flag}" in
    c) config_file="--config ${OPTARG}" ;;
    t) train_flag="--training" ;;
    p) prediction_flag="--prediction" ;;
    d) default_flag="--default" ;;
    *) print_usage
       exit 1 ;;
  esac
done


python3 $main_dir/VSR/assistant.py $train_flag $prediction_flag $default_flag $config_file
```

assistant.py

```
import os
from importlib import import_module

import numpy as np

from VSR.utils.utils import setup, parse_args
from VSR.utils.logger import get_logger


def _get_module(generator):
    return import_module('VSR.models.' + generator)


def run(config_file, default=False, training=False, prediction=False):
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
    logger = get_logger(__name__)
    session_type, generator, conf, dataset = setup(config_file, default, training, prediction)

    lr_patch_size = conf['session'][session_type]['patch_size']
    scale = conf['generators'][generator]['x']

    module = _get_module(generator)
    gen = module.make_model(conf['generators'][generator], lr_patch_size)
```

```
    if session_type == 'prediction':
        from VSR.predict.predictor import Predictor

        pr_h = Predictor(input_dir=conf['test_sets'][dataset])
        pr_h.get_predictions(gen, conf['weights_paths']['generator'])

    elif session_type == 'training':
        from VSR.train.trainer import Trainer

        hr_patch_size = lr_patch_size * scale
        if conf['default']['feature_extractor']:
            from VSR.models.cut_vgg19 import Cut_VGG19

            out_layers = conf['feature_extractor']['vgg19']['layers_to_extract']
            f_ext = Cut_VGG19(patch_size=hr_patch_size, layers_to_extract=out_layers)
        else:
            f_ext = None

        if conf['default']['discriminator']:
            from VSR.models.discriminator import Discriminator

            discr = Discriminator(patch_size=hr_patch_size, kernel_size=3)
        else:
            discr = None

        trainer = Trainer(
            generator=gen,
            discriminator=discr,
            feature_extractor=f_ext,
            lr_train_dir=conf['training_sets'][dataset]['lr_train_dir'],
            hr_train_dir=conf['training_sets'][dataset]['hr_train_dir'],
            lr_valid_dir=conf['training_sets'][dataset]['lr_valid_dir'],
            hr_valid_dir=conf['training_sets'][dataset]['hr_valid_dir'],
            learning_rate=conf['session'][session_type]['learning_rate'],
            loss_weights=conf['loss_weights'],
            losses=conf['losses'],
            dataname=conf['training_sets'][dataset]['data_name'],
            log_dirs=conf['log_dirs'],
            weights_generator=conf['weights_paths']['generator'],
            weights_discriminator=conf['weights_paths']['discriminator'],
            n_validation=conf['session'][session_type]['n_validation_samples'],
            flatness=conf['session'][session_type]['flatness'],
            fallback_save_every_n_epochs=conf['session'][session_type][
                'fallback_save_every_n_epochs'
            ],
            adam_optimizer=conf['session'][session_type]['adam_optimizer'],
            metrics=conf['session'][session_type]['metrics'],
        )
        trainer.train(
            epochs=conf['session'][session_type]['epochs'],
            steps_per_epoch=conf['session'][session_type]['steps_per_epoch'],
            batch_size=conf['session'][session_type]['batch_size'],
            monitored_metrics=conf['session'][session_type]['monitored_metrics'],
        )

    else:
        logger.error('Невірний вибір.')


if __name__ == '__main__':
    args = parse_args()
    np.random.seed(1000)
```

```
    run(
        config_file=args['config_file'],
        default=args['default'],
        training=args['training'],
        prediction=args['prediction'],
    )
```

imagemodel.py

```
import numpy as np

from VSR.utils.image_processing import (
    process_array,
    process_output,
    split_image_into_overlapping_patches,
    stich_together,
)


class ImageModel:
    """Батьківський клас моделі.

    Містить типові функції для всіх SR моделей.
    """

    def predict(self, input_image_array, by_patch_of_size=None, batch_size=10,
padding_size=2):
        """
        Оброблює вхідний масив до необхідного формату та трансформує вихід мережі

        Args:
            input_image_array: масив вхідного зображення.
            by_patch_of_size: інтерфейс для великих зображень. Ділить зображення на ділянки
вхідного розміру.
            padding_size: інтерфейс для великих зображень. Відступи між ділянками.
            batch_size: інтерфейс для великих зображень. Кількість оброблюваних ділянок за
раз.
        Returns:
            sr_img: зображення.
        """

        if by_patch_of_size:
            lr_img = process_array(input_image_array, expand=False)
            patches, p_shape = split_image_into_overlapping_patches(
                lr_img, patch_size=by_patch_of_size, padding_size=padding_size
            )
            # повертає ділянки зображення
            for i in range(0, len(patches), batch_size):
                batch = self.model.predict(patches[i: i + batch_size])
                if i == 0:
                    collect = batch
                else:
                    collect = np.append(collect, batch, axis=0)

            scale = self.scale
            padded_size_scaled = tuple(np.multiply(p_shape[0:2], scale)) + (3,)
            scaled_image_shape = tuple(np.multiply(input_image_array.shape[0:2], scale)) +
(3,)
            sr_img = stich_together(
```

```
                collect,
                padded_image_shape=padded_size_scaled,
                target_shape=scaled_image_shape,
                padding_size=padding_size * scale,
            )

        else:
            lr_img = process_array(input_image_array)
            sr_img = self.model.predict(lr_img)[0]

        sr_img = process_output(sr_img)
        return sr_img
```

RDN.py

```
import tensorflow as tf
from tensorflow.keras.initializers import RandomUniform
from tensorflow.keras.layers import concatenate, Input, Activation, Add, Conv2D, Lambda,
UpSampling2D
from tensorflow.keras.models import Model

from VSR.models.imagemodel import ImageModel

def make_model(arch_params, patch_size):
    """ Повертає модель.

    Використовується для вибору моделі.
    """

    return RDN(arch_params, patch_size)

class RDN(ImageModel):
    """Імплементація моделі RDN для ПРЗ.

    Мережа описана в https://arxiv.org/abs/1802.08797 (Zhang et al. 2018).
    rdn = RDN(arch_params={'C': 5, 'D':16, 'G':48, 'G0':52, 'x':3})
    rdn.model.load_weights('PATH/TO/WEIGHTS')

    Args:
        arch_params: dictionary, містить параметри мережі C, D, G, G0, x.
        patch_size: integer або None, визначає вхідний розмір. Необхідна лише під час
навчання.
        c_dim: integer, кількість каналів вхідного зображення.
        kernel_size: integer, звичний розмір ядра (фільтра) для згорток.
        upscaling: string, 'ups' або 'shuffle', визначає імплементацію шару збільшення.
        init_extreme_val: максимальні значення для ініціалізації RandomUniform.

    Attributes:
        C: integer, кількість шарів згорток в кожному залишковому щільному блоці (RDB).
        D: integer, кількість блоків RDB.
        G: integer, кількість вихідних фільтрів згорток в блоках RDB.
        G0: integer, кількість вихідних фільтрів кожного RDB.
        x: integer, коефіцієнт збільшення.
        model: Keras модель для RDN.
        name: назва для ідентифікації мережі збільшення під час навчання.
        model._name: ідентифікує дану мережу як мережу генератор в комплексній моделі, що
створюється класом навчання.
    """
```

```
def __init__(
        self,
        arch_params={},
        patch_size=None,
        c_dim=3,
        kernel_size=3,
        upscaling='ups',
        init_extreme_val=0.05,
        weights=''
):

    self.params = arch_params
    self.C = self.params['C']
    self.D = self.params['D']
    self.G = self.params['G']
    self.G0 = self.params['G0']
    self.scale = self.params['x']
    self.patch_size = patch_size
    self.c_dim = c_dim
    self.kernel_size = kernel_size
    self.upscaling = upscaling
    self.initializer = RandomUniform(
        minval=-init_extreme_val, maxval=init_extreme_val, seed=None
    )
    self.model = self._build_rdn()
    self.model._name = 'generator'
    self.name = 'rdn'

def _upsampling_block(self, input_layer):
    """ Блок Upsampling для старих вагів. """

    x = Conv2D(
        self.c_dim * self.scale ** 2,
        kernel_size=3,
        padding='same',
        name='UPN3',
        kernel_initializer=self.initializer,
    )(input_layer)
    return UpSampling2D(size=self.scale, name='UPsample')(x)

def _pixel_shuffle(self, input_layer):
    """ Імплементація PixelShuffle для шару підвищення. """

    x = Conv2D(
        self.c_dim * self.scale ** 2,
        kernel_size=3,
        padding='same',
        name='UPN3',
        kernel_initializer=self.initializer,
    )(input_layer)
    return Lambda(
        lambda x: tf.nn.depth_to_space(x, block_size=self.scale, data_format='NHWC'),
        name='PixelShuffle',
    )(x)

def _UPN(self, input_layer):
    """ Шари збільшення."""

    x = Conv2D(
        64,
        kernel_size=5,
        strides=1,
```

```
            padding='same',
            name='UPN1',
            kernel_initializer=self.initializer,
        )(input_layer)
        x = Activation('relu', name='UPN1_Relu')(x)
        x = Conv2D(
            32, kernel_size=3, padding='same', name='UPN2',
kernel_initializer=self.initializer
        )(x)
        x = Activation('relu', name='UPN2_Relu')(x)
        if self.upscaling == 'shuffle':
            return self._pixel_shuffle(x)
        elif self.upscaling == 'ups':
            return self._upsampling_block(x)
        else:
            raise ValueError('Невірний вибір шару збільшення роздільної здатності.')

    def _RDBs(self, input_layer):
        """RDB блоки.

        Args:
            input_layer: вхідний шар RDB блоку (другий шар згортки F_0).

        Returns:
            об'єднання шарів ознак RDB блоків із шарами ознак G0.
        """
        rdb_concat = list()
        rdb_in = input_layer
        for d in range(1, self.D + 1):
            x = rdb_in
            for c in range(1, self.C + 1):
                F_dc = Conv2D(
                    self.G,
                    kernel_size=self.kernel_size,
                    padding='same',
                    kernel_initializer=self.initializer,
                    name='F_%d_%d' % (d, c),
                )(x)
                F_dc = Activation('relu', name='F_%d_%d_Relu' % (d, c))(F_dc)
                # об'єднати вхід та вихід ConvRelu блока
                # x = [input_layer,F_11(input_layer),F_12([input_layer,F_11(input_layer)]),
F_13..]
                x = concatenate([x, F_dc], axis=3, name='RDB_Concat_%d_%d' % (d, c))
            # 1x1 згортка (Локальне злиття ознак)
            x = Conv2D(
                self.G0, kernel_size=1, kernel_initializer=self.initializer, name='LFF_%d' %
(d)
            )(x)
            # Локальне залишкове навчання F_{i,LF} + F_{i-1}
            rdb_in = Add(name='LRL_%d' % (d))([x, rdb_in])
            rdb_concat.append(rdb_in)

        assert len(rdb_concat) == self.D

        return concatenate(rdb_concat, axis=3, name='LRLs_Concat')

    def _build_rdn(self):
        LR_input = Input(shape=(self.patch_size, self.patch_size, 3), name='LR')
        F_m1 = Conv2D(
            self.G0,
            kernel_size=self.kernel_size,
            padding='same',
```

```
            kernel_initializer=self.initializer,
            name='F_m1',
        )(LR_input)
        F_0 = Conv2D(
            self.G0,
            kernel_size=self.kernel_size,
            padding='same',
            kernel_initializer=self.initializer,
            name='F_0',
        )(F_m1)
        FD = self._RDBs(F_0)
        # Глобальне об'єднання ознак
        # 1x1 Conv of concat RDB layers -> G0 feature maps
        GFF1 = Conv2D(
            self.G0,
            kernel_size=1,
            padding='same',
            kernel_initializer=self.initializer,
            name='GFF_1',
        )(FD)
        GFF2 = Conv2D(
            self.G0,
            kernel_size=self.kernel_size,
            padding='same',
            kernel_initializer=self.initializer,
            name='GFF_2',
        )(GFF1)
        # Глобальне залишкове навчання для щільних ознак
        FDF = Add(name='FDF')([GFF2, F_m1])
        # Збільшення роздільної здатності
        FU = self._UPN(FDF)
        # Збірка в зображення
        SR = Conv2D(
            self.c_dim,
            kernel_size=self.kernel_size,
            padding='same',
            kernel_initializer=self.initializer,
            name='SR',
        )(FU)

        return Model(inputs=LR_input, outputs=SR)
```

cut_vgg19.py

```
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19

from VSR.utils.logger import get_logger


class Cut_VGG19:
    """
    Клас що отримує мережу VGG19 з Keras, навчану на датасеті imagenet, та визначає
    <layers_to_extract> як вихідні шари.

    Args:
        layers_to_extract: список вихідних шарів.
        patch_size: integer, визначає розмір входу (patch_size x patch_size).
```

```
    Attributes:
        loss_model: vgg архітектура з <layers_to_extract>, як вихідні шари.
    """

    def __init__(self, patch_size, layers_to_extract):
        self.patch_size = patch_size
        self.input_shape = (patch_size,) * 2 + (3,)
        self.layers_to_extract = layers_to_extract
        self.logger = get_logger(__name__)

        if len(self.layers_to_extract) > 0:
            self._cut_vgg()
        else:
            self.logger.error('Невірна ініціалізація VGG: витягнуті шари повинні бути > 0')
            raise ValueError('Невірна ініціалізація VGG: витягнуті шари повинні бути > 0')

    def _cut_vgg(self):
        """
        Завантажує навчану VGG, вихідні шари - self.layers_to_extract.
        """

        vgg = VGG19(weights='imagenet', include_top=False, input_shape=self.input_shape)
        vgg.trainable = False
        outputs = [vgg.layers[i].output for i in self.layers_to_extract]
        self.model = Model([vgg.input], outputs)
        self.model._name = 'feature_extractor'
        self.name = 'vgg19'  # використовується в назві вагів
```

discriminator.py

```
from tensorflow.keras.layers import Input, Activation, Dense, Conv2D, BatchNormalization, \
    LeakyReLU
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam


class Discriminator:
    """
    Імплементація мережі дискримінатора для генеративно-змагального компоненту функції втрат

    Args:
        patch_size: integer, визначає вхідний розмір як (patch_size, patch_size, 3).
        kernel_size: розмір ядра (фільтра) блоків згортки.

    Attributes:
        model: Keras модель.
        name: назва для класифікації дискримінатора під час навчання GAN.
        model._name: ідентифікує дану мережу як дискримінатор в комплексній моделі створеній
класом навчання.
        block_param: dictionary, визначає кількість фільтрів та кроків для кожного блоку
згортки.

    """

    def __init__(self, patch_size, kernel_size=3):
        self.patch_size = patch_size
        self.kernel_size = kernel_size
        self.block_param = {}
        self.block_param['filters'] = (64, 128, 128, 256, 256, 512, 512)
```

```
        self.block_param['strides'] = (2, 1, 2, 1, 1, 1, 1)
        self.block_num = len(self.block_param['filters'])
        self.model = self._build_disciminator()
        optimizer = Adam(0.0002, 0.5)
        self.model.compile(loss='binary_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
        self.model._name = 'discriminator'
        self.name = 'srgan-large'

    def _conv_block(self, input, filters, strides, batch_norm=True, count=None):
        """ Шар згортки + Leaky ReLU + умовний BN. """

        x = Conv2D(
            filters,
            kernel_size=self.kernel_size,
            strides=strides,
            padding='same',
            name='Conv_{}'.format(count),
        )(input)
        x = LeakyReLU(alpha=0.2)(x)
        if batch_norm:
            x = BatchNormalization(momentum=0.8)(x)
        return x

    def _build_disciminator(self):
        """ Об'єднує шари дискримінатора. """

        HR = Input(shape=(self.patch_size, self.patch_size, 3))
        x = self._conv_block(HR, filters=64, strides=1, batch_norm=False, count=1)
        for i in range(self.block_num):
            x = self._conv_block(
                x,
                filters=self.block_param['filters'][i],
                strides=self.block_param['strides'][i],
                count=i + 2,
            )
        x = Dense(self.block_param['filters'][-1] * 2, name='Dense_1024')(x)
        x = LeakyReLU(alpha=0.2)(x)
        # x = Flatten()(x)
        x = Dense(1, name='Dense_last')(x)
        HR_v_SR = Activation('sigmoid')(x)

        discriminator = Model(inputs=HR, outputs=HR_v_SR)
        return discriminator
```

predictor.py

```
from time import time

import imageio
import yaml
import numpy as np
from pathlib import Path

from VSR.utils.logger import get_logger
from VSR.utils.utils import get_timestamp


class Predictor:
```

```
    """Клас класифікатор для передбачення, використовуючи вхідну модель.

    Завантажує зображення та відео із вхідного каталогу, виконує класифікацію за допомогою
    моделі та зберігає результат у вихідному каталозі.
    Може отримати шлях до вагів або надати користувачу можливість обрати шлях.

    Args:
        input_dir: string, шлях до вхідного каталогу.
        output_dir: string, шлях до вихідного каталогу.
        verbose: bool.

    Attributes:
        img_extensions: список дозволених розширень зображення.
        img_ls: список файлів зображень у вхідному каталозі.
        video_extensions: список дозволених розширень відео.
        video_ls: список файлів відео у вхідному каталозі.

    Methods:
        get_predictions: виконати класифікацію файлів із вхідного каталогу
        із використанням моделі та вагів та зберегти результати у вихідному каталозі
    """

    def __init__(self, input_dir, output_dir='./data/output', verbose=True):

        self.input_dir = Path(input_dir)
        self.data_name = self.input_dir.name
        self.output_dir = Path(output_dir) / self.data_name
        self.logger = get_logger(__name__)
        if not verbose:
            self.logger.setLevel(40)
        self.img_extensions = ('.jpeg', '.jpg', '.png')  # допустимі розширення зображень
        self.img_ls = [f for f in self.input_dir.iterdir() if f.suffix in self.img_extensions]
        self.video_extensions = ('.wmv', '.mkv', '.mp4', '.avi', '.mpeg')
        self.video_ls = [f for f in self.input_dir.iterdir() if f.suffix in
self.video_extensions]
        if ( (len(self.img_ls) < 1) and (len(self.video_ls) < 1)):
            self.logger.error('Коректних зображень не знайшлося (перевірте конфігураційний
файл).')
            raise ValueError('Коректних зображень не знайшлося (перевірте конфігураційний
файл).')
        # Створити каталог для результатів
        if not self.output_dir.exists():
            self.logger.info('Створюю вихідний каталог:\n{}'.format(self.output_dir))
            self.output_dir.mkdir(parents=True)

    def _load_weights(self):
        """ Invokes the model's load weights function if any weights are provided. """
        if self.weights_path is not None:
            self.logger.info('Завантажено ваги із \n > {}'.format(self.weights_path))
            # loading by name automatically excludes the vgg layers
            self.model.model.load_weights(str(self.weights_path))
        else:
            self.logger.error('Помилка: Шлях до вагів не вказаний (перевірте конфігураційний
файл).')
            raise ValueError('Шлях до вагів не вказаний (перевірте конфігураційний файл).')

        session_config_path = self.weights_path.parent / 'session_config.yml'
        if session_config_path.exists():
            conf = yaml.load(session_config_path.read_text(), Loader=yaml.FullLoader)
        else:
            self.logger.warning('Не вдалося знайти ваги конфігурації навчання')
            conf = {}
```

```
    conf.update({'pre-trained-weights': self.weights_path.name})
    return conf

def _make_basename(self):
    """ Поєднує назву генератора та параметри архітектури. """

    params = [self.model.name]
    for param in np.sort(list(self.model.params.keys())):
        params.append('{g}{p}'.format(g=param, p=self.model.params[param]))
    return '-'.join(params)

def get_predictions(self, model, weights_path):
    """ Виконує покращення. """

    self.model = model
    self.weights_path = Path(weights_path)
    weights_conf = self._load_weights()
    out_folder = self.output_dir / self._make_basename() / get_timestamp()
    self.logger.info('Результати в:\n > {}'.format(out_folder))
    if out_folder.exists():
        self.logger.warning('Каталог існує, можливо файли було перезаписано')
    else:
        out_folder.mkdir(parents=True)
    if weights_conf:
        yaml.dump(weights_conf, (out_folder / 'weights_config.yml').open('w'))
    # Класифікувати та зберегти
    for img_path in self.img_ls:
        output_path = out_folder / img_path.name
        self.logger.info('Оброблюю файл \n > {}'.format(img_path))
        start = time()
        lr_img = imageio.imread(img_path)
        sr_img = self._forward_pass(lr_img)
        end = time()
        self.logger.info('Витрачений час: {}s'.format(end - start))
        self.logger.info('Результати в: {}'.format(output_path))
        imageio.imwrite(output_path, sr_img)
    for video_path in self.video_ls:
        output_path = out_folder / video_path.name
        self.logger.info('Оброблюю файл \n > {}'.format(video_path))
        start = time()
        reader = imageio.get_reader(video_path)
        fps = reader.get_meta_data()['fps']
        duration = reader.get_meta_data()['duration']
        writer = imageio.get_writer(output_path, fps=fps)
        for lr_img in reader:
            sr_img = self._forward_pass(lr_img)
            writer.append_data(sr_img)
        writer.close()
        end = time()
        self.logger.info('Витрачений час: {}s'.format(end - start))
        self.logger.info('Результати в: {}'.format(output_path))

def _forward_pass(self, lr_img):
    if lr_img.shape[2] == 3:
        sr_img = self.model.predict(lr_img)
        return sr_img
    else:
        self.logger.error('{} не є зображенням із трьох каналів.'.format(file_path))
```

trainer.py

```
from time import time

import numpy as np
from tqdm import tqdm
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras import backend as K

from VSR.utils.datahandler import DataHandler
from VSR.utils.train_helper import TrainerHelper
from VSR.utils.metrics import PSNR
from VSR.utils.metrics import PSNR_Y
from VSR.utils.logger import get_logger
from VSR.utils.utils import check_parameter_keys


class Trainer:
    """Клас для налаштування та проведення навчання.

    Приймає на вхід генератор(модель) та створює SR зображення

    Args:
        generator: Keras модель.
        discriminator: Keras model, мережа дискримінатора для генеративно-змагальної мережі
        feature_extractor: Keras модель VGG19.
        lr_train_dir: шлях до каталогу вхідних зображень для навчання (низьке розширення).
        hr_train_dir: шлях до каталогу вхідних зображень для навчання (високе розширення).
        lr_valid_dir: шлях до каталогу вхідних зображень для валідації (низьке розширення).
        hr_valid_dir: шлях до каталогу вхідних зображень для валідації (високе розширення).
        learning_rate: float.
        loss_weights: dictionary, використовується як ваги до компонентів функцій втрат.
            Містить 'generator' для компоненту втрат генератора, може містити 'discriminator'
та 'feature_extractor'
            для дискримінатора та VGG19.
        logs_dir: шлях до каталогу де логи tensorboard будуть збережені.
        weights_dir: шлях до каталогу де будуть збережені ваги.
        dataname: string, використовується для ідентифікації датасету під час сессії
тренування.
        weights_generator: шлях до вагів генератора, для продовження навчання.
        weights_discriminator: шлях до вагів дискримінатора, для продовження навчання.
        n_validation:integer, кількість прикладів валідації, що використовується під час
навчання.
        flatness: dictionary. Визначає поріг рівності для тренувальних ділянок.
        lr_decay_frequency: integer, кількість епох після яких зменшується коефіцієнт
навчання.
        lr_decay_factor: 0 < float <1, коефіцієнт зменшення рівня навчання.

    Methods:
        train: поєднує мережі та починає навчання із встановленими параметрами.

    """

    def __init__(
            self,
            generator,
            discriminator,
            feature_extractor,
            lr_train_dir,
            hr_train_dir,
            lr_valid_dir,
```

```
            hr_valid_dir,
            loss_weights={'generator': 1.0, 'discriminator': 0.003, 'feature_extractor': 1 /
12},
            log_dirs={'logs': 'logs', 'weights': 'weights'},
            fallback_save_every_n_epochs=2,
            dataname=None,
            weights_generator=None,
            weights_discriminator=None,
            n_validation=None,
            flatness={'min': 0.0, 'increase_frequency': None, 'increase': 0.0, 'max': 0.0},
            learning_rate={'initial_value': 0.0004, 'decay_frequency': 100, 'decay_factor':
0.5},
            adam_optimizer={'beta1': 0.9, 'beta2': 0.999, 'epsilon': None},
            losses={
                'generator': 'mae',
                'discriminator': 'binary_crossentropy',
                'feature_extractor': 'mse',
            },
            metrics={'generator': 'PSNR_Y'},
    ):
        self.generator = generator
        self.discriminator = discriminator
        self.feature_extractor = feature_extractor
        self.scale = generator.scale
        self.lr_patch_size = generator.patch_size
        self.learning_rate = learning_rate
        self.loss_weights = loss_weights
        self.weights_generator = weights_generator
        self.weights_discriminator = weights_discriminator
        self.adam_optimizer = adam_optimizer
        self.dataname = dataname
        self.flatness = flatness
        self.n_validation = n_validation
        self.losses = losses
        self.log_dirs = log_dirs
        self.metrics = metrics
        if self.metrics['generator'] == 'PSNR_Y':
            self.metrics['generator'] = PSNR_Y
        elif self.metrics['generator'] == 'PSNR':
            self.metrics['generator'] = PSNR
        self._parameters_sanity_check()
        self.model = self._combine_networks()

        self.settings = {}
        self.settings['training_parameters'] = locals()
        self.settings['training_parameters']['lr_patch_size'] = self.lr_patch_size
        self.settings = self.update_training_config(self.settings)

        self.logger = get_logger(__name__)

        self.helper = TrainerHelper(
            generator=self.generator,
            weights_dir=log_dirs['weights'],
            logs_dir=log_dirs['logs'],
            lr_train_dir=lr_train_dir,
            feature_extractor=self.feature_extractor,
            discriminator=self.discriminator,
            dataname=dataname,
            weights_generator=self.weights_generator,
            weights_discriminator=self.weights_discriminator,
            fallback_save_every_n_epochs=fallback_save_every_n_epochs,
        )
```

```
        self.train_dh = DataHandler(
            lr_dir=lr_train_dir,
            hr_dir=hr_train_dir,
            patch_size=self.lr_patch_size,
            scale=self.scale,
            n_validation_samples=None,
        )
        self.valid_dh = DataHandler(
            lr_dir=lr_valid_dir,
            hr_dir=hr_valid_dir,
            patch_size=self.lr_patch_size,
            scale=self.scale,
            n_validation_samples=n_validation,
        )

    def _parameters_sanity_check(self):
        """ Parameteres sanity check. """

        if self.discriminator:
            assert self.lr_patch_size * self.scale == self.discriminator.patch_size
            self.adam_optimizer
        if self.feature_extractor:
            assert self.lr_patch_size * self.scale == self.feature_extractor.patch_size

        check_parameter_keys(
            self.learning_rate,
            needed_keys=['initial_value'],
            optional_keys=['decay_factor', 'decay_frequency'],
            default_value=None,
        )
        check_parameter_keys(
            self.flatness,
            needed_keys=[],
            optional_keys=['min', 'increase_frequency', 'increase', 'max'],
            default_value=0.0,
        )
        check_parameter_keys(
            self.adam_optimizer,
            needed_keys=['beta1', 'beta2'],
            optional_keys=['epsilon'],
            default_value=None,
        )
        check_parameter_keys(self.log_dirs, needed_keys=['logs', 'weights'])

    def _combine_networks(self):
        """
        Створює комбіновану модель, що містить мережу генератора, дискримінатора,
        VGG19, якщо вони є.
        """

        lr = Input(shape=(self.lr_patch_size,) * 2 + (3,))
        sr = self.generator.model(lr)
        outputs = [sr]
        losses = [self.losses['generator']]
        loss_weights = [self.loss_weights['generator']]

        if self.discriminator:
            self.discriminator.model.trainable = False
            validity = self.discriminator.model(sr)
            outputs.append(validity)
            losses.append(self.losses['discriminator'])
```

```
            loss_weights.append(self.loss_weights['discriminator'])
        if self.feature_extractor:
            self.feature_extractor.model.trainable = False
            sr_feats = self.feature_extractor.model(sr)
            outputs.extend([*sr_feats])
            losses.extend([self.losses['feature_extractor']] * len(sr_feats))
            loss_weights.extend(
                [self.loss_weights['feature_extractor'] / len(sr_feats)] * len(sr_feats)
            )
        combined = Model(inputs=lr, outputs=outputs)
        # https://stackoverflow.com/questions/42327543/adam-optimizer-goes-haywire-after-200k-
batches-training-loss-grows
        optimizer = Adam(
            beta_1=self.adam_optimizer['beta1'],
            beta_2=self.adam_optimizer['beta2'],
            lr=self.learning_rate['initial_value'],
            epsilon=self.adam_optimizer['epsilon'],
        )
        combined.compile(
            loss=losses, loss_weights=loss_weights, optimizer=optimizer, metrics=self.metrics
        )
        return combined

    def _lr_scheduler(self, epoch):
        """ Планувальник оновлень коефіцієнту навчання. """

        n_decays = epoch // self.learning_rate['decay_frequency']
        lr = self.learning_rate['initial_value'] * (self.learning_rate['decay_factor'] **
n_decays)
        # no lr below minimum control 10e-7
        return max(1e-7, lr)

    def _flatness_scheduler(self, epoch):
        if self.flatness['increase']:
            n_increases = epoch // self.flatness['increase_frequency']
        else:
            return self.flatness['min']

        f = self.flatness['min'] + n_increases * self.flatness['increase']

        return min(self.flatness['max'], f)

    def _load_weights(self):
        """
        Завантажує ваги із шляху
        """

        if self.weights_generator:
            self.model.get_layer('generator').load_weights(str(self.weights_generator))

        if self.discriminator:
            if self.weights_discriminator:

self.model.get_layer('discriminator').load_weights(str(self.weights_discriminator))
                self.discriminator.model.load_weights(str(self.weights_discriminator))

    def _format_losses(self, prefix, losses, model_metrics):
        """ Створює хеш для відслідковування в tensorboard. """

        return dict(zip([prefix + m for m in model_metrics], losses))

    def update_training_config(self, settings):
```

```
        """ Генералізує налаштування навчання. """

        _ = settings['training_parameters'].pop('weights_generator')
        _ = settings['training_parameters'].pop('self')
        _ = settings['training_parameters'].pop('generator')
        _ = settings['training_parameters'].pop('discriminator')
        _ = settings['training_parameters'].pop('feature_extractor')
        settings['generator'] = {}
        settings['generator']['name'] = self.generator.name
        settings['generator']['parameters'] = self.generator.params
        settings['generator']['weights_generator'] = self.weights_generator

        _ = settings['training_parameters'].pop('weights_discriminator')
        if self.discriminator:
            settings['discriminator'] = {}
            settings['discriminator']['name'] = self.discriminator.name
            settings['discriminator']['weights_discriminator'] = self.weights_discriminator
        else:
            settings['discriminator'] = None

        if self.discriminator:
            settings['feature_extractor'] = {}
            settings['feature_extractor']['name'] = self.feature_extractor.name
            settings['feature_extractor']['layers'] = self.feature_extractor.layers_to_extract
        else:
            settings['feature_extractor'] = None

        return settings

    def train(self, epochs, steps_per_epoch, batch_size, monitored_metrics):
        """
        Підтримує навчання для заданої кількості епох, надсилає втрати до Tensorboard

        Args:
            epochs: кількість епох для тренування.
            steps_per_epoch: кількість кроків за епоху.
            batch_size: кількість зображень на крок епохи.
            monitored_metrics: dictionary, ключі - це метрики для відслідковування вагів та їх
зберігання.
            Ключі - режим метрик для зберігання вагів ('min' або 'max').
        """

        self.settings['training_parameters']['steps_per_epoch'] = steps_per_epoch
        self.settings['training_parameters']['batch_size'] = batch_size
        starting_epoch = self.helper.initialize_training(
            self
        )  # load_weights, creates folders, creates basename

        self.tensorboard = TensorBoard(log_dir=str(self.helper.callback_paths['logs']))
        self.tensorboard.set_model(self.model)

        # validation data
        validation_set = self.valid_dh.get_validation_set(batch_size)
        y_validation = [validation_set['hr']]
        if self.discriminator:
            discr_out_shape = list(self.discriminator.model.outputs[0].shape)[1:4]
            valid = np.ones([batch_size] + discr_out_shape)
            fake = np.zeros([batch_size] + discr_out_shape)
            validation_valid = np.ones([len(validation_set['hr'])] + discr_out_shape)
            y_validation.append(validation_valid)
        if self.feature_extractor:
            validation_feats = self.feature_extractor.model.predict(validation_set['hr'])
```

```
            y_validation.extend([*validation_feats])

        for epoch in range(starting_epoch, epochs):
            self.logger.info('Епоха {e}/{tot_eps}'.format(e=epoch, tot_eps=epochs))
            K.set_value(self.model.optimizer.lr, self._lr_scheduler(epoch=epoch))
            self.logger.info('Поточний рівень навчання:
{}'.format(K.eval(self.model.optimizer.lr)))

            flatness = self._flatness_scheduler(epoch)
            if flatness:
                self.logger.info('Поточний поріг рівності: {}'.format(flatness))

            epoch_start = time()
            for step in tqdm(range(steps_per_epoch)):
                batch = self.train_dh.get_batch(batch_size, flatness=flatness)
                y_train = [batch['hr']]
                training_losses = {}

                ## Discriminator training
                if self.discriminator:
                    sr = self.generator.model.predict(batch['lr'])
                    d_loss_real = self.discriminator.model.train_on_batch(batch['hr'], valid)
                    d_loss_fake = self.discriminator.model.train_on_batch(sr, fake)
                    d_loss_fake = self._format_losses(
                        'train_d_fake_', d_loss_fake, self.discriminator.model.metrics_names
                    )
                    d_loss_real = self._format_losses(
                        'train_d_real_', d_loss_real, self.discriminator.model.metrics_names
                    )
                    training_losses.update(d_loss_real)
                    training_losses.update(d_loss_fake)
                    y_train.append(valid)

                ## Generator training
                if self.feature_extractor:
                    hr_feats = self.feature_extractor.model.predict(batch['hr'])
                    y_train.extend([*hr_feats])

                model_losses = self.model.train_on_batch(batch['lr'], y_train)
                model_losses = self._format_losses('train_', model_losses,
self.model.metrics_names)
                training_losses.update(model_losses)

                self.tensorboard.on_epoch_end(epoch * steps_per_epoch + step, training_losses)
                self.logger.debug('Похибки на кроці {s}:\n {l}'.format(s=step,
l=training_losses))

            elapsed_time = time() - epoch_start
            self.logger.info('Епоха {} заняла {:10.1f}s'.format(epoch, elapsed_time))

            validation_losses = self.model.evaluate(
                validation_set['lr'], y_validation, batch_size=batch_size
            )
            validation_losses = self._format_losses(
                'val_', validation_losses, self.model.metrics_names
            )

            if epoch == starting_epoch:
                remove_metrics = []
                for metric in monitored_metrics:
                    if (metric not in training_losses) and (metric not in validation_losses):
                        msg = ' '.join([metric, 'не є серед метрик моделі, видаляю.'])
```

```
                    self.logger.error(msg)
                    remove_metrics.append(metric)
            for metric in remove_metrics:
                _ = monitored_metrics.pop(metric)

        # should average train metrics
        end_losses = {}
        end_losses.update(validation_losses)
        end_losses.update(training_losses)

        self.helper.on_epoch_end(
            epoch=epoch,
            losses=end_losses,
            generator=self.model.get_layer('generator'),
            discriminator=self.discriminator,
            metrics=monitored_metrics,
        )
        self.tensorboard.on_epoch_end(epoch, validation_losses)
    self.tensorboard.on_train_end(None)
```

config.yml

```
default:
  generator: rdn #модель за замовчуванням
  feature_extractor: false
  discriminator: false
  training_set: div2k # навчальний набір за замовчуванням
  test_set: sample # папка із вхідними даними за замовчуванням
log_dirs:
  logs: ./logs # каталог з логами для тренування
  weights: ./weights # каталог із вагами для тренування
dirs:
  weights: ./weights # каталог із вагами для передбачення
feature_extractor:
  vgg19:
    layers_to_extract: #слої, що будуть витягнуті мережею VGG19
      - 5
      - 9
generators:
  rdn: # значення за замовчуванням для мережі rdn
    C: 6 # number of conv layer inside each residual dense blocks
    D: 20 # number of RDBs.
    G: 64 # number of convolution output filters inside the RDBs.
    G0: 64 # number of output filters of each RDB.
    x: 2
loss_weights:
  generator: 1.0
  feature_extractor: 0.0833
  discriminator: 0.01
losses:
  generator: mae # mean absolute error
  discriminator: binary_crossentropy
  feature_extractor: mse #mean square error
session:
  prediction:
    patch_size:
  training:
    steps_per_epoch: 1000
    patch_size: 32
```

```
    batch_size: 16
    epochs: 300
    n_validation_samples: 100
    learning_rate:
      initial_value: 0.0004
      decay_frequency: 50
      decay_factor: 0.5
    fallback_save_every_n_epochs: 2
    flatness:
      min: 0.0
      increase_frequency: null
      increase: 0.0
      max: 0.0
    metrics:
      generator: PSNR_Y
    monitored_metrics:
      val_loss: min
      val_PSNR_Y: max
    adam_optimizer: #значення для коректного навчання
https://stackoverflow.com/questions/42327543/adam-optimizer-goes-haywire-after-200k-batches-
training-loss-grows
      beta1: 0.9
      beta2: 0.999
      epsilon: null
test_sets:
  sample: ./data/input/sample
training_sets:
  custom data:
    lr_train_dir: ./data/custom/lr/train
    hr_train_dir: ./data/custom/hr/train
    lr_valid_dir: ./data/custom/lr/validation
    hr_valid_dir: ./data/custom/hr/validation
    data_name: custom
  div2k:
    lr_train_dir: ./data/DIV2K/DIV2K_train_LR_bicubic/X2
    hr_train_dir: ./data/DIV2K/DIV2K_train_HR
    lr_valid_dir: ./data/DIV2K/DIV2K_valid_LR_bicubic/X2
    hr_valid_dir: ./data/DIV2K/DIV2K_valid_HR
    data_name: div2k
weights_paths:
  discriminator:
  generator: ./weights/sample_weights/rdn-C6-D20-G64-G064-x2/ArtefactCancelling/rdn-C6-D20-
G64-G064-x2_ArtefactCancelling_epoch219.hdf5
```

datahandler.py

```
import os

import imageio
import numpy as np

from VSR.utils.logger import get_logger


class DataHandler:
    """
    DataHandler генерує додаткові партії зображень для навчання та валідації.

    Args:
```

```
    lr_dir: каталог із зображеннями низької якості.
    hr_dir: каталог із еталонними зображеннями.
    patch_size: integer, розмір ділянок отриманих із зображень.
    scale: integer, коефіцієнт підвищення роздільної здатності.
    n_validation_samples: integer, розмір партії валідації.
"""

def __init__(self, lr_dir, hr_dir, patch_size, scale, n_validation_samples=None):
    self.folders = {'hr': hr_dir, 'lr': lr_dir}  # image folders
    self.extensions = ('.png', '.jpeg', '.jpg')  # admissible extension
    self.img_list = {}  # list of file names
    self.n_validation_samples = n_validation_samples
    self.patch_size = patch_size
    self.scale = scale
    self.patch_size = {'lr': patch_size, 'hr': patch_size * self.scale}
    self.logger = get_logger(__name__)
    self._make_img_list()
    self._check_dataset()

def _make_img_list(self):
    """ Створює хеш з лістингом lr, hr зображень із відповідних каталогів. """

    for res in ['hr', 'lr']:
        file_names = os.listdir(self.folders[res])
        file_names = [file for file in file_names if file.endswith(self.extensions)]
        self.img_list[res] = np.sort(file_names)

    if self.n_validation_samples:
        samples = np.random.choice(
            range(len(self.img_list['hr'])), self.n_validation_samples, replace=False
        )
        for res in ['hr', 'lr']:
            self.img_list[res] = self.img_list[res][samples]

def _check_dataset(self):
    """ Перевірка датасету. """

    # the order of these asserts is important for testing
    assert len(self.img_list['hr']) == self.img_list['hr'].shape[0], 'UnevenDatasets'
    assert self._matching_datasets(), 'Input/LabelsMismatch'

def _matching_datasets(self):
    """ Приблизне співставлення назв в директоріях lr та hr. """
    # LR_name.png = HR_name+x+scale.png
    # or
    # LR_name.png = HR_name.png
    LR_name_root = [x.split('.')[0].rsplit('x', 1)[0] for x in self.img_list['lr']]
    HR_name_root = [x.split('.')[0] for x in self.img_list['hr']]
    return np.all(HR_name_root == LR_name_root)

def _not_flat(self, patch, flatness):
    """
    Визначає складність ділянки зображення. Поріг рівності визначає flatness.
    """

    if max(np.std(patch, axis=0).mean(), np.std(patch, axis=1).mean()) < flatness:
        return False
    else:
        return True

def _crop_imgs(self, imgs, batch_size, flatness):
    """
```

```
        Отримує випадкові координати верхнього лівого кута в LR зображені, помножене на
        коефіцієнт збільшення для отримання координат в HR.
        Отримує batch_size + n можливих координат.
        Приймає набір лише із стандартним відхиленням інтенсивності пікселів більшим за
границю, або
        якщо ділянки більше неможливо видкинути (n вже було відкинуто)
        Ріже зображення на квадратні ділянки розміром patch_size із обраного верхнього лівого
кута
        """

        slices = {}
        crops = {}
        crops['lr'] = []
        crops['hr'] = []
        accepted_slices = {}
        accepted_slices['lr'] = []
        top_left = {'x': {}, 'y': {}}
        n = 50 * batch_size
        for i, axis in enumerate(['x', 'y']):
            top_left[axis]['lr'] = np.random.randint(
                0, imgs['lr'].shape[i] - self.patch_size['lr'] + 1, batch_size + n
            )
            top_left[axis]['hr'] = top_left[axis]['lr'] * self.scale
        for res in ['lr', 'hr']:
            slices[res] = np.array(
                [
                    {'x': (x, x + self.patch_size[res]), 'y': (y, y + self.patch_size[res])}
                    for x, y in zip(top_left['x'][res], top_left['y'][res])
                ]
            )

        for slice_index, s in enumerate(slices['lr']):
            candidate_crop = imgs['lr'][s['x'][0]: s['x'][1], s['y'][0]: s['y'][1],
slice(None)]
            if self._not_flat(candidate_crop, flatness) or n == 0:
                crops['lr'].append(candidate_crop)
                accepted_slices['lr'].append(slice_index)
            else:
                n -= 1
            if len(crops['lr']) == batch_size:
                break

        accepted_slices['hr'] = slices['hr'][accepted_slices['lr']]

        for s in accepted_slices['hr']:
            candidate_crop = imgs['hr'][s['x'][0]: s['x'][1], s['y'][0]: s['y'][1],
slice(None)]
            crops['hr'].append(candidate_crop)

        crops['lr'] = np.array(crops['lr'])
        crops['hr'] = np.array(crops['hr'])
        return crops

    def _apply_transform(self, img, transform_selection):
        """ Повертає та перевертає вхідне зображення відповідно до transform_selection. """

        rotate = {
            0: lambda x: x,
            1: lambda x: np.rot90(x, k=1, axes=(1, 0)),  # rotate right
            2: lambda x: np.rot90(x, k=1, axes=(0, 1)),  # rotate left
        }
```

```
        flip = {
            0: lambda x: x,
            1: lambda x: np.flip(x, 0),  # flip along horizontal axis
            2: lambda x: np.flip(x, 1),  # flip along vertical axis
        }

        rot_direction = transform_selection[0]
        flip_axis = transform_selection[1]

        img = rotate[rot_direction](img)
        img = flip[flip_axis](img)

        return img

    def _transform_batch(self, batch, transforms):
        """ Трансформує кожне зображення в наборі індивідуально. """

        t_batch = np.array(
            [self._apply_transform(img, transforms[i]) for i, img in enumerate(batch)]
        )
        return t_batch

    def get_batch(self, batch_size, idx=None, flatness=0.0):
        """
        Повертає хеш з ключами ('lr', 'hr'), що містять тренувальні набори
        зображень низького та високого розширення

        Args:
            batch_size: integer.
            flatness: float в діапазоні [0,1], порогове значення рівності ділянки.
                Визначає який рівень деталізації ділянка повинна мати. Значення 0 означає
будь-яка.
        """

        if not idx:
            # randomly select one image. idx is given at validation time.
            idx = np.random.choice(range(len(self.img_list['hr'])))
        img = {}
        for res in ['lr', 'hr']:
            img_path = os.path.join(self.folders[res], self.img_list[res][idx])
            img[res] = imageio.imread(img_path) / 255.0
        batch = self._crop_imgs(img, batch_size, flatness)
        transforms = np.random.randint(0, 3, (batch_size, 2))
        batch['lr'] = self._transform_batch(batch['lr'], transforms)
        batch['hr'] = self._transform_batch(batch['hr'], transforms)

        return batch

    def get_validation_batches(self, batch_size):
        """ Повертає набір для кожного зображення в сетах валідації. """

        if self.n_validation_samples:
            batches = []
            for idx in range(self.n_validation_samples):
                batches.append(self.get_batch(batch_size, idx, flatness=0.0))
            return batches
        else:
            self.logger.error(
                'Розмір сету валідації не визначено. (не працює з сетом валідації?)'
            )
            raise ValueError(
                'Розмір сету валідації не визначено. (не працює з сетом валідації?)'
```

```
            )

    def get_validation_set(self, batch_size):
        """
        Повертає набір для кожного зображення в сеті валідації
        Розрівнює та ділить їх для Keras функції model.evaluate.
        """

        if self.n_validation_samples:
            batches = self.get_validation_batches(batch_size)
            valid_set = {'lr': [], 'hr': []}
            for batch in batches:
                for res in ('lr', 'hr'):
                    valid_set[res].extend(batch[res])
            for res in ('lr', 'hr'):
                valid_set[res] = np.array(valid_set[res])
            return valid_set
        else:
            self.logger.error(
                'Розмір сету валідації не визначено. (не працює з сетом валідації?)'
            )
            raise ValueError(
                'Розмір сету валідації не визначено. (не працює з сетом валідації?)'
            )
```

image_processing.py

```
import numpy as np


def process_array(image_array, expand=True):
    """ Оброблює 3-вимірний масив в збільшену, 4 вимірну вибірку розміром 1. """

    image_batch = image_array / 255.0
    if expand:
        image_batch = np.expand_dims(image_batch, axis=0)
    return image_batch


def process_output(output_tensor):
    """ Трансформує 4-вимірний вихід тензора в зображення. """

    sr_img = output_tensor.clip(0, 1) * 255
    sr_img = np.uint8(sr_img)
    return sr_img


def pad_patch(image_patch, padding_size, channel_last=True):
    """ Додає відступ до image_patch із padding_size значенням від границь. """

    if channel_last:
        return np.pad(
            image_patch,
            ((padding_size, padding_size), (padding_size, padding_size), (0, 0)),
            'edge',
        )
    else:
        return np.pad(
            image_patch,
```

```
            ((0, 0), (padding_size, padding_size), (padding_size, padding_size)),
            'edge',
        )


def unpad_patches(image_patches, padding_size):
    return image_patches[:, padding_size:-padding_size, padding_size:-padding_size, :]


def split_image_into_overlapping_patches(image_array, patch_size, padding_size=2):
    """ Ділить зображення в частково перекриті ділянки.

    Ділянки перекриваються за padding_size значенням в пікселях.
    Args:
        image_array: масив numpy вхідного зображення.
        patch_size: розмір ділянки із оригінального зображення.
        padding_size: розмір перекритої ділянки.
    """

    xmax, ymax, _ = image_array.shape
    x_remainder = xmax % patch_size
    y_remainder = ymax % patch_size

    # modulo here is to avoid extending of patch_size instead of 0
    x_extend = (patch_size - x_remainder) % patch_size
    y_extend = (patch_size - y_remainder) % patch_size

    # make sure the image is divisible into regular patches
    extended_image = np.pad(image_array, ((0, x_extend), (0, y_extend), (0, 0)), 'edge')

    # add padding around the image to simplify computations
    padded_image = pad_patch(extended_image, padding_size, channel_last=True)

    xmax, ymax, _ = padded_image.shape
    patches = []

    x_lefts = range(padding_size, xmax - padding_size, patch_size)
    y_tops = range(padding_size, ymax - padding_size, patch_size)

    for x in x_lefts:
        for y in y_tops:
            x_left = x - padding_size
            y_top = y - padding_size
            x_right = x + patch_size + padding_size
            y_bottom = y + patch_size + padding_size
            patch = padded_image[x_left:x_right, y_top:y_bottom, :]
            patches.append(patch)

    return np.array(patches), padded_image.shape


def stich_together(patches, padded_image_shape, target_shape, padding_size=4):
    """ Реконструює зображення із перекритих ділянок.

    Args:
        patches: ділянки отримані із split_image_into_overlapping_patches
        padded_image_shape: форма зображень отриманих в split_image_into_overlapping_patches
        target_shape: форма фінального зображення
        padding_size: розмір перекритої ділянки.
    """

    xmax, ymax, _ = padded_image_shape
```

```
    patches = unpad_patches(patches, padding_size)
    patch_size = patches.shape[1]
    n_patches_per_row = ymax // patch_size

    complete_image = np.zeros((xmax, ymax, 3))

    row = -1
    col = 0
    for i in range(len(patches)):
        if i % n_patches_per_row == 0:
            row += 1
            col = 0
        complete_image[
        row * patch_size: (row + 1) * patch_size, col * patch_size: (col + 1) * patch_size, :
        ] = patches[i]
        col += 1
    return complete_image[0: target_shape[0], 0: target_shape[1], :]
```

logger.py

```
import logging
import os


def get_logger(name, job_dir='.'):
    """ Повертає лог, що друкує до данні в консоль. """

    logger = logging.getLogger(name)
    logger.setLevel(logging.DEBUG)
    if not logger.handlers:
        # stream handler ensures that logging events are passed to stdout
        ch = logging.StreamHandler()
        ch.setLevel(logging.INFO)
        ch_formatter = logging.Formatter('%(message)s')
        ch.setFormatter(ch_formatter)
        logger.addHandler(ch)

        # file handler ensures that logging events are passed to log file
        if not os.path.exists(job_dir):
            os.makedirs(job_dir)

        fh = logging.FileHandler(filename=os.path.join(job_dir, 'log_file'))
        fh.setLevel(logging.DEBUG)
        fh_formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
        fh.setFormatter(fh_formatter)
        logger.addHandler(fh)

    return logger
```

metrics.py

```
import tensorflow.keras.backend as K
```

```
def PSNR(y_true, y_pred, MAXp=1):
    """
    Визначає значення PSNR:
        PSNR = 20 * log10(MAXp) - 10 * log10(MSE).

    Args:
        y_true: еталон.
        y_pred: покращене.
        MAXp: максимальне значення пікселя (default=1).
    """
    return -10.0 * K.log(K.mean(K.square(y_pred - y_true))) / K.log(10.0)


def RGB_to_Y(image):
    """ Зображення має значення від 0 до 1. """

    R = image[:, :, :, 0]
    G = image[:, :, :, 1]
    B = image[:, :, :, 2]
    # https://ru.wikipedia.org/wiki/YCbCr
    Y = 16 + (65.738 * R) + (129.057 * G) + (25.064 * B)
    return Y / 255.0


def PSNR_Y(y_true, y_pred, MAXp=1):
    """
    Визначає значення PSNR по каналу Y:
        PSNR = 20 * log10(MAXp) - 10 * log10(MSE).

    Args:
        y_true: еталон.
        y_pred: покращене зображеня.
        MAXp: максимальне значення пікселя в діапазоні (default=1).
    """
    y_true = RGB_to_Y(y_true)
    y_pred = RGB_to_Y(y_pred)
    return -10.0 * K.log(K.mean(K.square(y_pred - y_true))) / K.log(10.0)
```

train_helper.py

```
import yaml
import numpy as np
from pathlib import Path

from VSR.utils.logger import get_logger
from VSR.utils.utils import get_timestamp


class TrainerHelper:
    """Колекція корисних функцій для менеджменту навчання.

    Args:
        generator: Keras модель.
        logs_dir: шлях до каталогу де логи tensorboard будуть збережені.
        weights_dir: шлях до каталогу де будуть збережені ваги.
        lr_train_dir: шлях до каталогу вхідних зображень для навчання (низьке розширення).
        feature_extractor: Keras модель VGG19.
        discriminator: Keras model, мережа дискримінатора для генеративно-змагальної мережі
```

```
        dataname: string, використовується для ідентифікації датасету під час сессії
тренування.
        weights_dictionary містить шлях до вагів генератора та дискримінатора.
        fallback_save_every_n_epochs: integer, визначає через скільки епох зберігати ваги,
навіть коли метрики не покращились.
        max_n_best_weights: максимальне значення вагів, які зберігаються для кращих метрик.
        max_n_other_weights: максимальне значення не кращих вагів, які зберігаються.


    Methods:
        print_training_setting.
        on_epoch_end.
        epoch_n_from_weights_name.
        initialize_training.

    """

    def __init__(
            self,
            generator,
            weights_dir,
            logs_dir,
            lr_train_dir,
            feature_extractor=None,
            discriminator=None,
            dataname=None,
            weights_generator=None,
            weights_discriminator=None,
            fallback_save_every_n_epochs=2,
            max_n_other_weights=5,
            max_n_best_weights=5,
    ):
        self.generator = generator
        self.dirs = {'logs': Path(logs_dir), 'weights': Path(weights_dir)}
        self.feature_extractor = feature_extractor
        self.discriminator = discriminator
        self.dataname = dataname

        if weights_generator:
            self.pretrained_generator_weights = Path(weights_generator)
        else:
            self.pretrained_generator_weights = None

        if weights_discriminator:
            self.pretrained_discriminator_weights = Path(weights_discriminator)
        else:
            self.pretrained_discriminator_weights = None

        self.fallback_save_every_n_epochs = fallback_save_every_n_epochs
        self.lr_dir = Path(lr_train_dir)
        self.basename = self._make_basename()
        self.session_id = self.get_session_id(basename=None)
        self.session_config_name = 'session_config.yml'
        self.callback_paths = self._make_callback_paths()
        self.weights_name = self._weights_name(self.callback_paths)
        self.best_metrics = {}
        self.since_last_epoch = 0
        self.max_n_other_weights = max_n_other_weights
        self.max_n_best_weights = max_n_best_weights
        self.logger = get_logger(__name__)

    def _make_basename(self):
```

```
    """ Комбінує назву генератора та параметрів архітектури. """

    gen_name = self.generator.name
    params = [gen_name]
    for param in np.sort(list(self.generator.params.keys())):
        params.append('{g}{p}'.format(g=param, p=self.generator.params[param]))
    return '-'.join(params)

def get_session_id(self, basename):
    """ Повертає унікальне значення сессії. """

    time_stamp = get_timestamp()

    if basename:
        session_id = '{b}_{ts}'.format(b=basename, ts=time_stamp)
    else:
        session_id = time_stamp
    return session_id

def _get_previous_conf(self):
    """ Перевіряє наявність конфігурації сессії session_config.yml для вагів. """

    if self.pretrained_generator_weights:
        session_config_path = (
                self.pretrained_generator_weights.parent / self.session_config_name
        )
        if session_config_path.exists():
            return yaml.load(session_config_path.read_text(), Loader=yaml.FullLoader)
        else:
            self.logger.warning('Не вдалось знайти попередню конфігурацію')
            return {}

    return {}

def update_config(self, training_settings):
    """
    Додає до існуючих налаштувань поточні значення.
    """

    session_settings = self._get_previous_conf()
    session_settings.update({self.session_id: training_settings})

    return session_settings

def _make_callback_paths(self):
    """ Створює шлях, що використовуєтья для менеджменту лог файлів та вагів. """

    callback_paths = {}
    callback_paths['weights'] = self.dirs['weights'] / self.basename / self.session_id
    callback_paths['logs'] = self.dirs['logs'] / self.basename / self.session_id
    return callback_paths

def _weights_name(self, callback_paths):
    """ Будує назву вагів тренувальної сессії. """

    w_name = {
        'generator': callback_paths['weights']
                     / (self.basename + '{metric}_epoch{epoch:03d}.hdf5')
    }
    if self.discriminator:
        w_name.update(
            {
```

```
                    'discriminator': callback_paths['weights']
                                     / (self.discriminator.name +
'{metric}_epoch{epoch:03d}.hdf5')
                }
            )
        return w_name

    def print_training_setting(self, settings):
        """ Друкує налаштування навчання. """

        self.logger.info('\nДеталі навчання:')
        for k in settings[self.session_id]:
            if isinstance(settings[self.session_id][k], dict):
                self.logger.info('  {}: '.format(k))
                for kk in settings[self.session_id][k]:
                    self.logger.info(
                        '    {key}: {value}'.format(
                            key=kk, value=str(settings[self.session_id][k][kk])
                        )
                    )
            else:
                self.logger.info(
                    '  {key}: {value}'.format(key=k, value=str(settings[self.session_id][k]))
                )

    def _save_weights(self, epoch, generator, discriminator=None, metric=None, best=False):
        """ Зберігає ваги існуючих моделей. """

        if best:
            gen_path = self.weights_name['generator'].with_name(
                (self.weights_name['generator'].name).format(
                    metric='_best-' + metric, epoch=epoch + 1
                )
            )
        else:
            gen_path = self.weights_name['generator'].with_name(
                (self.weights_name['generator'].name).format(metric='', epoch=epoch + 1)
            )
        # CANT SAVE MODEL DUE TO TF LAYER INSIDE LAMBDA (PIXELSHUFFLE)
        generator.save_weights(gen_path.as_posix())
        if discriminator:
            if best:
                discr_path = self.weights_name['discriminator'].with_name(
                    (self.weights_name['discriminator'].name).format(
                        metric='_best-' + metric, epoch=epoch + 1
                    )
                )
            else:
                discr_path = self.weights_name['discriminator'].with_name(
                    (self.weights_name['discriminator'].name).format(metric='', epoch=epoch +
1)
                )
            discriminator.model.save_weights(discr_path.as_posix())
        try:
            self._remove_old_weights(self.max_n_other_weights,
max_best=self.max_n_best_weights)
        except Exception as e:
            self.logger.warning('Не вдалося видалити ваги: {}'.format(e))

    def _remove_old_weights(self, max_n_weights, max_best=5):
        """
        Сканує каталог вагів та видаляє все, окрім:
```

```
            - max_best кращі нові ваги.
            - max_n_weights інші ваги.
        """

        w_list = {}
        w_list['all'] = [w for w in self.callback_paths['weights'].iterdir() if '.hdf5' in
w.name]
        w_list['best'] = [w for w in w_list['all'] if 'best' in w.name]
        w_list['others'] = [w for w in w_list['all'] if w not in w_list['best']]
        # remove older best
        epochs_set = {}
        epochs_set['best'] = list(
            set([self.epoch_n_from_weights_name(w.name) for w in w_list['best']])
        )
        epochs_set['others'] = list(
            set([self.epoch_n_from_weights_name(w.name) for w in w_list['others']])
        )
        keep_max = {'best': max_best, 'others': max_n_weights}
        for type in ['others', 'best']:
            if len(epochs_set[type]) > keep_max[type]:
                epoch_list = np.sort(epochs_set[type])[::-1]
                epoch_list = epoch_list[0: keep_max[type]]
                for w in w_list[type]:
                    if self.epoch_n_from_weights_name(w.name) not in epoch_list:
                        w.unlink()

    def on_epoch_end(self, epoch, losses, generator, discriminator=None, metrics={}):
        """
        Після кожної епохи перевіряє метрики, зберігає ваги, виконує логування.
        """

        self.logger.info(losses)
        monitor_op = {'max': np.greater, 'min': np.less}
        extreme = {'max': -np.Inf, 'min': np.Inf}
        for metric in metrics:
            if metric in losses.keys():
                if metric not in self.best_metrics.keys():
                    self.best_metrics[metric] = extreme[metrics[metric]]

                if monitor_op[metrics[metric]](losses[metric], self.best_metrics[metric]):
                    self.logger.info(
                        '{} покращення від {:10.5f} до {:10.5f}'.format(
                            metric, self.best_metrics[metric], losses[metric]
                        )
                    )
                    self.logger.info('Зберігаю ваги')
                    self.best_metrics[metric] = losses[metric]
                    self._save_weights(epoch, generator, discriminator, metric=metric,
best=True)
                    self.since_last_epoch = 0
                    return True
                else:
                    self.logger.info('{} не покращились.'.format(metric))
                    if self.since_last_epoch >= self.fallback_save_every_n_epochs:
                        self.logger.info('Все одно зберігаю ваги.')
                        self._save_weights(epoch, generator, discriminator, best=False)
                        self.since_last_epoch = 0
                        return True

            else:
                self.logger.warning('{} не відстежується, не можливо зберігти
ваги.'.format(metric))
```

```
        self.since_last_epoch += 1
        return False

    def epoch_n_from_weights_name(self, w_name):
        """
        Витягує останнє значення епохи із назви вагів.
        """
        try:
            starting_epoch = int(w_name.split('epoch')[1][0:3])
        except Exception as e:
            self.logger.warning(
                'Не можливо отримати початкову епогу із назви вагів: \n{}'.format(w_name)
            )
            self.logger.error(e)
            starting_epoch = 0
        return starting_epoch

    def initialize_training(self, object):
        """Виконується до навчання.

        завантажує ваги, генерує назви сессій та вагів, створює каталоги та друкує дані по
сессії навчання.
        """

        object.weights_generator = self.pretrained_generator_weights
        object.weights_discriminator = self.pretrained_discriminator_weights
        object._load_weights()
        w_name = object.weights_generator
        if w_name:
            last_epoch = self.epoch_n_from_weights_name(w_name.name)
        else:
            last_epoch = 0

        self.callback_paths = self._make_callback_paths()
        self.callback_paths['weights'].mkdir(parents=True)
        self.callback_paths['logs'].mkdir(parents=True)
        object.settings['training_parameters']['starting_epoch'] = last_epoch
        self.settings = self.update_config(object.settings)
        self.print_training_setting(self.settings)
        yaml.dump(
            self.settings, (self.callback_paths['weights'] /
self.session_config_name).open('w')
        )
        return last_epoch
```

utils.py

```
import os
import argparse
from datetime import datetime

import numpy as np
import yaml

from VSR.utils.logger import get_logger

logger = get_logger(__name__)
```

```
def _get_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('--prediction', action='store_true', dest='prediction')
    parser.add_argument('--training', action='store_true', dest='training')
    parser.add_argument('--summary', action='store_true', dest='summary')
    parser.add_argument('--default', action='store_true', dest='default')
    parser.add_argument('--config', action='store', dest='config_file')
    return parser


def parse_args():
    """ Зчитує CLI аргументи. """

    parser = _get_parser()
    args = vars(parser.parse_args())
    if args['prediction'] and args['training']:
        logger.error('Оберіть тільки "prediction" - передбачення, чи "training" - навчання.')
        raise ValueError('Оберіть тільки "prediction" чи "training".')
    return args


def get_timestamp():
    ts = datetime.now()
    time_stamp = '{y}-{m:02d}-{d:02d}_{h:02d}{mm:02d}'.format(
        y=ts.year, m=ts.month, d=ts.day, h=ts.hour, mm=ts.minute
    )
    return time_stamp


def check_parameter_keys(parameter, needed_keys, optional_keys=None, default_value=None):
    if needed_keys:
        for key in needed_keys:
            if key not in parameter:
                logger.error('{p} не вказано ключ {k}'.format(p=parameter, k=key))
                raise
        object.settings['training_parameters']['starting_epoch'] = last_epoch
        self.settings = self.update_config(object.settings)
    if optional_keys:
        for key in optional_keys:
            if key not in parameter:
                logger.info('Встановлення {k} в {p} до {d}'.format(k=key, p=parameter,
d=default_value))
                parameter[key] = default_value


def get_config_from_weights(w_path, arch_params, name):
    """
    Витягує параметри архітектури із файлу або назви вагів
    """

    w_path = os.path.basename(w_path)
    parts = w_path.split(name)[1]
    parts = parts.split('_')[0]
    parts = parts.split('-')
    new_param = {}
    for param in arch_params:
        param_part = [x for x in parts if param in x]
        param_value = int(param_part[0].split(param)[1])
        new_param[param] = param_value
    return new_param
```

```
def select_option(options, message='', val=None):
    """ вибір наданої CLI опцій. """

    while val not in options:
        val = input(message)
        if val not in options:
            logger.error('Невірний вибір.')
    return val


def select_multiple_options(options, message='', val=None):
    """ Вибір декілької CLI опцій. """

    n_options = len(options)
    valid_selections = False
    selected_options = []
    while not valid_selections:
        for i, opt in enumerate(np.sort(options)):
            logger.info('{}: {}'.format(i, opt))
        val = input(message + ' (вибірка поділена пробілом)\n')
        vals = val.split(' ')
        valid_selections = True
        for v in vals:
            if int(v) not in list(range(n_options)):
                logger.error('Невірний вибір.')
                valid_selections = False
            else:
                selected_options.append(options[int(v)])

    return selected_options


def select_bool(message=''):
    """ CLI зчитування логічної змінної. """

    options = ['т', 'н']
    message = message + ' (' + '/'.join(options) + ') '
    val = None
    while val not in options:
        val = input(message)
        if val not in options:
            logger.error('Введіть т (так) чи н (ні).')
    if val == 'т':
        return True
    elif val == 'н':
        return False


def select_positive_float(message=''):
    """ CLI не негативне значення float. """

    value = -1
    while value < 0:
        value = float(input(message))
        if value < 0:
            logger.error('Невірний вибір.')
    return value


def select_positive_integer(message='', value=-1):
    """ CLI не негативне значення int. """
```

```
    while value < 0:
        value = int(input(message))
        if value < 0:
            logger.error('Невірний вибір.')
    return value


def browse_weights(weights_dir, model='generator'):
    """ Вибірка вагів із CLI. """

    exit = False
    while exit is False:
        weights = np.sort(os.listdir(weights_dir))[::-1]
        print_sel = dict(zip(np.arange(len(weights)), weights))
        for k in print_sel.keys():
            logger_message = '{item_n}: {item} \n'.format(item_n=k, item=print_sel[k])
            logger.info(logger_message)

        sel = select_positive_integer('>>> Оберіть каталог чи ваги для {}\n'.format(model))
        if weights[sel].endswith('hdf5'):
            weights_path = os.path.join(weights_dir, weights[sel])
            exit = True
        else:
            weights_dir = os.path.join(weights_dir, weights[sel])
    return weights_path


def setup(config_file='config.yml', default=False, training=False, prediction=False):
    """Інтерфейс CLI для встановлення навчання чи передбачення (класифікації).

    Зчитує шлях до конфігурації та аргументів із CLI.
    """

    conf = yaml.load(open(config_file, 'r'), Loader=yaml.FullLoader)

    if training:
        session_type = 'training'
    elif prediction:
        session_type = 'prediction'
    else:
        message = '(н)авчання - навчання чи (п)ередбачення - передбачення? (н/п) '
        session_type = {'н': 'training', 'п': 'prediction'}[select_option(['н', 'п'],
message)]
    if default:
        all_default = 'y'
    else:
        all_default = select_bool('Значення за замовчуванням для всього?')

    if all_default:
        generator = conf['default']['generator']
        if session_type == 'prediction':
            dataset = conf['default']['test_set']
            conf['generators'][generator] = get_config_from_weights(
                conf['weights_paths']['generator'], conf['generators'][generator], generator
            )
        elif session_type == 'training':
            dataset = conf['default']['training_set']

        return session_type, generator, conf, dataset

    logger.info('Оберіть мережу покращення')
    generators = {}
```

```
    for i, gen in enumerate(conf['generators']):
        generators[str(i)] = gen
        logger.info('{}: {}'.format(i, gen))
    generator = generators[select_option(generators)]

    load_weights = input('Завантажити існуючі ваги для {}? ([т]/н/з) '.format(generator))
    if load_weights == 'н':
        default = select_bool('Завантажити параметри за замовчуванням для
{}?'.format(generator))
        if not default:
            for param in conf['generators'][generator]:
                value = select_positive_integer(message='{}:'.format(param))
                conf['generators'][generator][param] = value
        else:
            logger.info('Параметри за замовчуванням {}.'.format(generator))
    elif (load_weights == 'з') and (conf['weights_paths']['generator']):
        logger.info('Завантаження вагів за замовчуванням для {}'.format(generator))
        logger.info(conf['weights_paths']['generator'])
        conf['generators'][generator] = get_config_from_weights(
            conf['weights_paths']['generator'], conf['generators'][generator], generator
        )
    else:
        conf['weights_paths']['generator'] = browse_weights(conf['dirs']['weights'],
generator)
        conf['generators']['generator'] = get_config_from_weights(
            conf['weights_paths']['generator'], conf['generators'][generator], generator
        )
    logger.info('{} параметри:'.format(generator))
    logger.info(conf['generators'][generator])

    if session_type == 'training':
        default_loss_weights = select_bool('Використовувати ваги за замовчуванням для
компоненту втрат (loss)?')
        if not default_loss_weights:
            conf['loss_weights']['generator'] = select_positive_float(
                'Введіть коефіцієнт генератора попіксельного компонента втрат '
            )
        use_discr = select_bool('Використовувати конкурентну мережу?')
        if use_discr:
            conf['default']['discriminator'] = True
            discr_w = select_bool('Використовувати існуючі ваги дескримінатора?')
            if discr_w:
                conf['weights_paths']['discriminator'] = browse_weights(
                    conf['dirs']['weights'], 'discriminator'
                )
            if not default_loss_weights:
                conf['loss_weights']['discriminator'] = select_positive_float(
                    'Введіть коефіцієнт для компоненту втрат конкуретної мережі '
                )

        use_feature_extractor = select_bool('Використосувати feature extractor?')
        if use_feature_extractor:
            conf['default']['feature_extractor'] = True
            if not default_loss_weights:
                conf['loss_weights']['feature_extractor'] = select_positive_float(
                    'Введіть коефіцієнт для функції втрат компонента conv '
                )
        default_metrics = select_bool('Моніторити стандартні показники?')
        if not default_metrics:
            suggested_list = suggest_metrics(use_discr, use_feature_extractor)
            selected_metrics = select_multiple_options(
                list(suggested_list.keys()), message='Оберіть показники для моніторингу.'
```

```
            )

            conf['session']['training']['monitored_metrics'] = {}
            for metric in selected_metrics:
                conf['session']['training']['monitored_metrics'][metric] =
suggested_list[metric]
            print(conf['session']['training']['monitored_metrics'])

    dataset = select_dataset(session_type, conf)

    return session_type, generator, conf, dataset


def suggest_metrics(discriminator=False, feature_extractor=False, loss_weights={}):
    suggested_metrics = {}
    if not discriminator and not feature_extractor:
        suggested_metrics['val_loss'] = 'min'
        suggested_metrics['train_loss'] = 'min'
        suggested_metrics['val_PSNR'] = 'max'
        suggested_metrics['train_PSNR'] = 'max'
    if feature_extractor or discriminator:
        suggested_metrics['val_generator_loss'] = 'min'
        suggested_metrics['train_generator_loss'] = 'min'
        suggested_metrics['val_generator_PSNR'] = 'max'
        suggested_metrics['train_generator_PSNR'] = 'max'
    if feature_extractor:
        suggested_metrics['val_feature_extractor_loss'] = 'min'
        suggested_metrics['train_feature_extractor_loss'] = 'min'
    return suggested_metrics


def select_dataset(session_type, conf):
    """ вибір датасету для навчання через CLI. """

    if session_type == 'training':
        logger.info('Оберіть набір для навчання')
        datasets = {}
        for i, data in enumerate(conf['training_sets']):
            datasets[str(i)] = data
            logger.info('{}: {}'.format(i, data))
        dataset = datasets[select_option(datasets)]

        return dataset
    else:
        logger.info('Оберіть тестовий набір')
        datasets = {}
        for i, data in enumerate(conf['test_sets']):
            datasets[str(i)] = data
            logger.info('{}: {}'.format(i, data))
        dataset = datasets[select_option(datasets)]

        return dataset
```

ДОДАТОК Г
ОПРИЛЮДНЕННЯ РЕЗУЛЬТАТІВ РОБОТИ

Інформаційна система «Підвищення роздільної здатності відео за допомогою нейронних мереж»

Захарченко О.О., *студент*; Марченко А.В., *доцент*
Сумський державний університет, м. Суми, Україна

Інформаційні технології стрімко розвиваються і вже давно стали невід'ємною частиною життєдіяльності людини. Завдяки штучним нейронним мережам, які здатні до навчання та самовдосконалення, всі сфери діяльності суспільства від освіти до бізнесу вийшли на новий рівень. Сьогодні штучний інтелект може виконати певні завдання навіть краще за людину, а саме завдання обробки, редагування та класифікації даних. Креативність - це найбільш слабка сторона НМ. Все більше набуває популярності застосування нейроних мереж для виконання редагування графіки та відеоматеріалів: додання кольору на чорно-білі фотознімки, ідентифікація центральних об'єктів та розмиття фону, поліпшення якості та генерації фотореалістичних зображень. Забезпечивши мережу мінімальним набором вхідних даних для роботи, можливо згенерувати фото людини, яка ніколи не існувала. Штучний інтелект має свої обмеження, але наукова цінність розроблених програм важливіша за кінцевий продукт.

Тому метою даного дослідження є розробка інформаційної системи для підвищення роздільної здатності відео, яка мала б більш широку спеціалізацію та кращу швидкодію. Програмний продукт призначений для власників і користувачів відео-сервісів та науковців сфери штучного інтелекту.

Після виконання аналізу аналогів програмного продукту було виділено наступні вимоги до майбутнього додатку: надавати можливість тренування моделі штучної мережі; збільшувати роздільну здатність відео та зображення; забезпечити підтримку користувацьких налаштувань якості; підтримувати розрахунки похибки відносно еталонного зображення; працездатність самої системи не повинна залежати від апаратного забезпечення забезпечити використання технології докеризації для забезпечення мобільності та крос-платформності.

В результаті розроблено інформаційну систему підвищення роздільної здатності відео за допомогою нейронних мереж, що містить розширені користувацькі налаштування та необхідні функціональні доповнення.