Міністерство освіти і науки України Національний технічний університет «Дніпровська політехніка»

Інститут електроенергетики
(інститут)

факультет інформаційних технологій
(факультет)

Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

Студента Аврахова Миколи Андрійовича
(ПІБ)

академічної групи 126м-23-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)

на тему Дослідження методів підвищення різкості зображень у складі Web-сервісу на базі мікросервісної архітектури
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	Гаркуша І.М.			
розділів:				

Рецензент	Ширін А.Л.			

Нормоконтролер	Коротенко Г.М.			

Дніпро 2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних систем та технологій
(повна назва)
_____Гнатушенко В.В.
(підпис) (прізвище, ініціали)

«_____»_____20__
року

ЗАВДАННЯ
на кваліфікаційну роботу ступеня
магістр
(бакалавра, спеціаліста, магістра)

студенту Аврахову М.А.. **академічної групи** 126м-23-1
(прізвище та ініціали) (шифр)

спеціальності 126 «Інформаційні системи та технології»

за освітньою-професійною програмою «Інформаційні технології та комп'ютерна інженерія»

на тему Дослідження методів підвищення різкості зображень у складі Web-сервісу на базі мікросервісної архітектури, затверджену наказом ректора НТУ «Дніпровська політехніка» від 17.10.2024 № 1388-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз теми та постановка задачі	1.10.20__ 31.10.20__
Розділ 2	Побудування моделі НМ для підвищення різкості зображень	1.11.20__ 30.11.20__
Розділ 3	Розробка інформаційної системи для розгортання веб-сервісу для підвищення різкості зображень на базі мікросервісної архітектури	1.12.20__ 21.12.20__

Завдання видано _____ _____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Аврахов М.А..
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 89 стор., 15 рис., 7 додатків, 11 джерел.

Об'єкт дослідження: розмиті растрові зображення різних розмірів

Предмет дослідження: методи та технології підвищення різкості зображень

Мета магістерської роботи: розробити Web-сервіс по підвищенню різкості зображень на базі обраного в процесі дослідження методу або технології обробки.

У вступі подано стан проблеми та виконана постановка задачі дослідження.

У першому розділі наведені основні відомості про засоби та технології побудування нейронних мереж (НМ) для обробки зображень, проведено порівняння технологій з розробкою веб – сервісів на базі мікросервісної архітектури.

У другому розділі наведена проектна складова вирішення завдання, обґрунтувано вибір інструментів програмної реалізації системи. Проведені випробування обраної НМ у різних конфігураціях, обґрунтувано вибір методів тренування.

У третьому розділі виконано проектування та реалізація інформаційної системи веб-сервісу для підвищення різкості зображень на базі навченої моделі.

Наукова новизна отриманих результатів кваліфікаційної роботи визначається тим, що удосконалено швидкість роботи нейронної мережі по підвищенню різкості зображень у порівнянні з наявними рішеннями.

Практична цінність результатів полягає в тому, що запропонована в роботі інформаційна технологія дозволяє гнучко розгортати інформаційну систему і використовувати її засобами автоматизації задач.

Ключові слова: НЕЙРОННА МЕРЕЖА, U-Net, МІКРОСЕРВІС, Web - СЕРВІС, РІЗКІСТЬ, ЗОБРАЖЕННЯ, Docker, РОЗМИТТЯ.

ABSTRACT

Explanatory note: 89 pages, 15 figures, 7 applications, 11 sources.

Object of research: blurred images of different resolutions

Subject of research methods and technologies of image sharpening

Purpose of Master's thesis: development of Web-service for image sharpening based of a selected method or technology.

In the introduction the status of the problem and the formulation of the research task are presented.

In the first section basic information about the tools and technologies of building neural networks for image processing, modern technologies of development of microservice-based web services are analyzed.

In the second section the design component of problem solution is described, the choice of tools for software implementation of the system is justified. Trials of various configurations of a chosen neural network are conducted, training methods are justified.

In the third section the design and implementation of the information system for image sharpening using microservice-based web service.

Originality of research is associated with the improvement of the processing time of the image sharpening neural network over the existing solutions.

Practical value of the results is that the information system offered in the work enables flexible deployment of the service and allows to use it with task automation technologies

NEURAL NETWORK, U-Net, MICROSERVICE, WEB-SERVICE, SHARPNESS, IMAGE, Docker, BLUR

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

ПЗ - програмне забезпечення

ІС - інформаційна система

НМ - нейронна мережа

ВСТУП

За останню декаду розвитку інформаційні технології зробили візуальний контент домінуючим в інтернеті. Згідно зі статистикою, опублікованою в журналі "Sproutworth", візуальний контент отримує на 94% більше, ніж текстовий. Значну частину цього контенту займають фотографії.

Кратно зростанню кількості фотографій, зростає і попит на підвищення різкості зображень. Разом із стрімким розвиненям НМ, з'явилась велика кількість інструментів з покращення візуальної якості зображень. Але домінуюча більщість цих інструментів знаходтся в одній з двох категорій:

1. Онлайн платформи для покращення зображень(Picsart, PicWish, Artguru та багато інших сервісів);
2. Модулі в складі професійного ПЗ для роботи з растровою графікою (Adobe Photoshop, Affinity photo та ін.).

Онлайн платформи не дозволяють працювати з файлами локально, вони можуть бути недоступними при слабкому або відсутньому інтернеті. Також поширеною практикою серед таких сервісів є розташування вотермарок на зображеннях, а також вимога платної підписки для завантаження покращених зображень.

Професійне ПЗ вимагає від користувача високого рівня навичок, може обмежувати користувачів підпискою на модуль з НМ.

Також це ПЗ є платним та у деяких випадках взагалі відсутнє на ОС Linux.

Обидві вище зазначені категорії бракують гнучкості у користуванні та налаштуванні. Також вони не надають можливостей для вивчення коду своїх НМ. Ще одна проблема - майже повна неможливість використання цих інструментів як модулів в інших ІС.

Тому актуальною є розробка системи підвищення різкості зображень у складі Web-сервісу на основі мікросервісної архітектури.

Мета роботи: розробка Web-сервісу по підвищенню різкості зображень на базі НМ, побудованого на базі мікросервісної архітектури.

Для досягнення поставленої мети необхідно виконати завдання:

1) запропонувати модель НМ для підвищення різкості зображення, цільову функцію для навчання моделі;
2) підготувати дані для навчання та навчити запропоновану модель на базі цільової функції на підготованих даних;
3) розробити ІС для підвищення різкості зображень, яка буде використовувати запропоновану модель;
4) провести тестування розробленої ІС з різними зображеннями та параметрами, зробити висновки.

Об'єкт дослідження – розмиті растрові зображення різних розмірів

Предмет дослідження - методи та технології підвищення різкості зображень.

Методи дослідження

Для дослідження ефективності методів навчання НМ, та структури НМ, було запропоновано порівняння за ключовими показниками на етапі навчання моделі та на етапі її використання.

Наукова новизна отриманих результатів кваліфікаційної роботи визначається тим, що удосконалено швидкість роботи нейронної мережі по підвищенню різкості зображень у порівнянні з наявними рішеннями

Практичне значення полягає в тому, що запропонована в роботі інформаційна технологія дозволяє гнучко розгортати інформаційну систему і використовувати її засобами автоматизації задач

В процесі проведення дослідження підготовлені тези доповіді на XII Міжнародній науково-технічній конференції студентів, аспірантів та молодих вчених [7].

1. АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

Оскільки тема кваліфікаційної роботи була обрана на поєднанні трьох напрямків ІТ, доцільно буде розглянути кожен з напрямків окремо.

Огляд стану області рішення задач було розглянуто у трьох напрямках:

1. НМ для підвищення різкості зображень;
2. Інструменти для розробки Web додатків;
3. Інструменти для будування ІС на базі мікросервісної архітектури.

1.1 Нейронна мережа для підвищення різкості зображень

Для задачі підвищення різкості зображень в даний момент широко використовують різноманітні НМ з різними конфігураціями шарів, кількості нейронів та іншими параметрами.

Оскільки в сфері машинного навчання нема чітко сформульованої практики щодо єдиного вірного підходу, було розглянуто найпоширеніші підходи, наведено їх переваги та недоліки.

1.1.1 Лінійна Згорткова нейронна мережа

Лінійні згорткові НМ складаються с декількох шарів згорткових нейронів, зазвичай переміжених шарами активації та нормалізації. Згорткові мережи зарекомендували себе як ефективний інструмент аналізу візуальних зображень.

Архітектура згоркової НМ бере за основу біологічну структуру нейронів зорової кори.

На відміну від щільного або повністю зв'язаного шару, згортковий шар є інваріантним відносно зсуву. В щільних шарах кожен нейрон попереднього шару з'єднаний с кожний нейроном наступного. В згорткових шарах

використовуються невеликі (3x3, 5x5, 7x7) відносно розміру зображень (зазвичай щонайменше 256x256) згорткові ядра.

Така структура дуже ефективна у розпізнаванні образів, вилученні ознак та інших характерних для роботи з зображеннями задач.

Також до переваг згорткових мереж можна віднести:

- зниження обчислювальної складності;
- менший обсяг вагових коефіцієнтів;
- менший ризик перенавчання.

На рис.1.1 зображено структуру згорткової НМ [6]

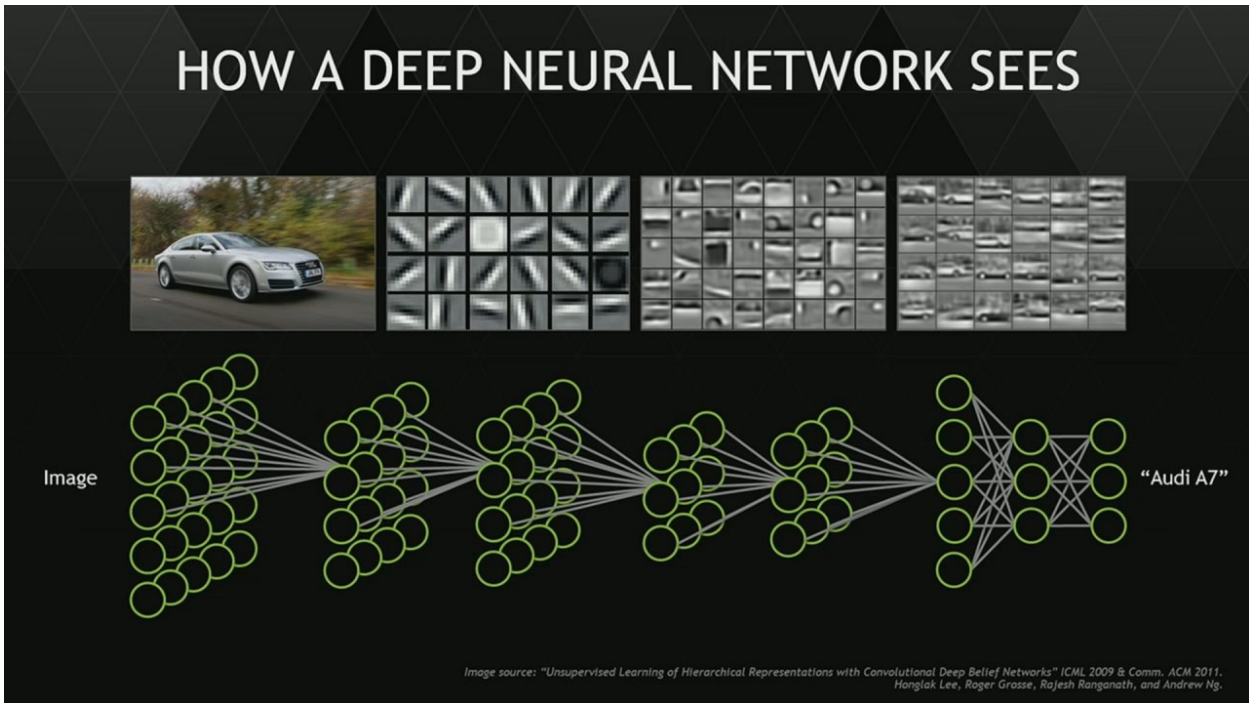


Рисунок 1.1 - Структура згорткової НМ

Незважаючи на всі вище зазначені переваги, побудова простої згорткової нейромережі без високорівневої архітектури має сенс лише для вивчення самої технології згорткових мереж. Отже, у наступних пунктах цього розділу будуть розглянуті архітектури згорткових мереж.

1.1.2 Архітектура U-net

Спочатку запропонована як НМ для класифікації медичних зображень [1], архітектура U-net виявилась дуже ефективною для вирішення великого кола задач у роботі з зображеннями:

- класифікація зображень;
- сегментація зображень;
- підвищення різкості зображення;
- очищення зображень від шумів;
- Super-resolution (підвищення візуальної якості зображення збільшенням його роздільної здатності);
- Inpainting (закінчення зображень з відсутніми сегментами).

Архітектура U-net складається з послідовності енкодерів та декодерів.

Енкодер - послідовність декількох згорткових шарів, переміжених функцією активації та, для деяких задач, шарами нормалізації.

Наприкінці кожного енкодеру дані проходять через max-pooling downsampling, що знижує роздільну здатність зображення. На кожному етапі кодування відбувається збільшення каналів та зменшення роздільної здатності зображення. Зазвичай, на кожному рівні удвічі збільшується кількість каналів та удвічі ж зменшується роздільна здатність. Таким чином, на кожному рівні кодування зменшується кількість інформації "Де це на зображені?", і збільшується кількість інформації "Що це на зображені?".

Після послідовності енкодерів дані проходять через шар - bottleneck, після якого MaxPooling не відбувається. Цей блок поєднує послідовності енкодерів та декодерів.

Декодер - послідовність згорткових та розширюючих (up-convolution) шарів, поєднана з інформацією з відповідного енкодеру.

Кожному енкодеру на шляху звуженя відповідає свій декодер, який зв'язаний с енкодером за допомогою так званих skip connections.

Skip connection - це конкатенація інформації з енкодера, яка містить на кожному етапі розширення все більше просторової інформації.

Ці зв'язки дозволяють не втрачати зв'язок між вхідним зображенням та вихідним. Наприкінці декодування знаходиться вихідний шар, який повертає зображення до первинної кількості каналів. Для деяких задач наприкінці можуть бути використані щільні шари або функції активації.

На рис.1.2 зображена схема оригінальної мережі U-net для класифікації медичних зображень.

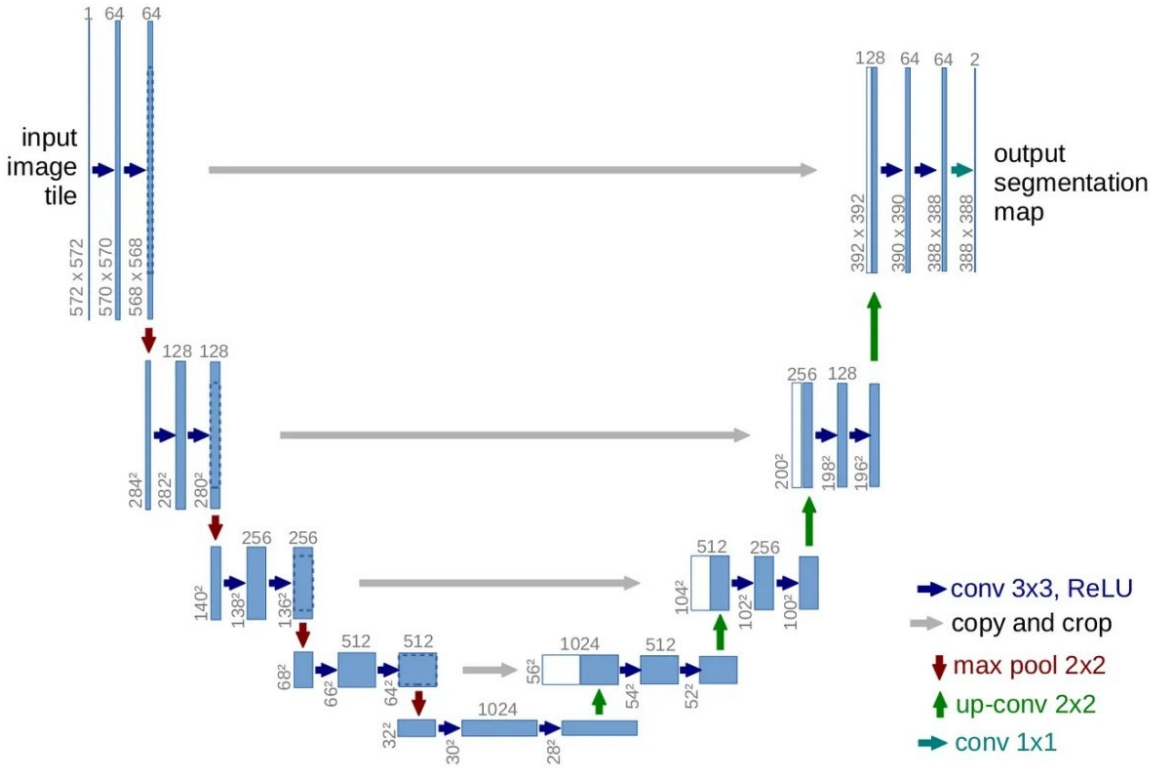


Рисунок 1.2 - Схема оригінальної мережі U-net для класифікації медичних зображень

Ця архітектура є дуже гнучкою, і може містити додаткові рівні глибини, шари активації та нормалізації.

До переваг мереж з архітектурою U-net відносяться:

- гарні показники навчання на невеликих датасетах;
- гарний рівень деталізації у вихідних зображеннях завдяки skip connections;
- відмінні показники в задачі регресії зображення-зображення.

До недоліків U-net відносять:

- високу обчислювальну складність через велику кількість параметрів, особливо у глибоких варіантах;
- відносно невелику глибину;
- велику кількість помилок при роботі з надто складними абстрактними даними.

1.1.3 Архітектура ResNet

Залишкова НМ – запропонована [2] для класифікації зображень архітектура глибокої(сотні шарів) НМ, яка складається з т.з. "залишкових блоків", проміж якими є skip connections.

Залишковий блок - комбінація шарів з активаціями, яка зокрема результатів власних обчислень передає на вхід до наступного блоку власні вхідні дані.

Таке архітектурне рішення дозволило ефективно боротися зі зникненням та "вибухом" градієнтів. Це явища, властиві глибоким нейронним мережам, у яких при проходженні багатьох шарів градієнт функції або дорівнює 0, або підіймається до аномальних значень, що значно ускладнює навчання.

На рис. 1.3 зображена архітектура ResNet

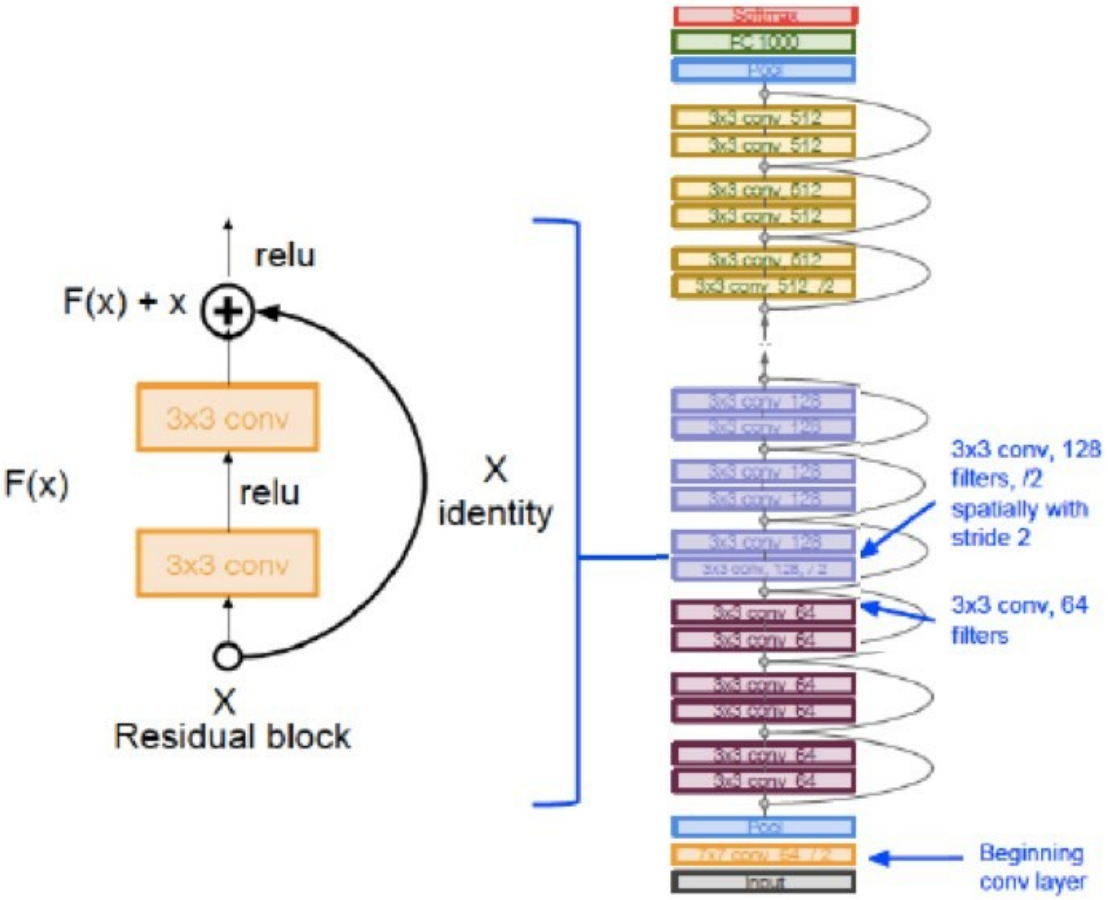


Рисунок 1.3 - Архітектура ResNet

До переваг мереж ResNet відноситься:

- можливість будувати мережі великої глибини (Resnet-152 має 152 шари);
- гарні показники у вивчені високорівневих ознак;
- гарні результати у вирішенні задач розпізнавання образів та сегментації зображень;
- полегшене навчання завдяки skip connections.

Недоліками цією архітектури є:

- низька ефективність в задачах регресії зображення-зображення;
- необхідність великого датасету для якісного навчання;
- великі вимоги до пам'яті через велику кількість шарів і необхідності тримати їх у відеопам'яті;

– через велику глибину та комплексність, мережі типу ResNet важко тонко підлаштовувати під конкретні задачі.

1.1.4 Генеративно-змагальна нейронна мережа

Генеративно-змагальні мережі (GAN) – запропонований [3] підхід до генеративних НМ, який передбачає тренування двух нейромереж: генератора та дискрімінатора.

Задача мережі - дискримінатора - визначити, чи було надане зображеня з тренувальної вибірки чи згенеровано генератором.

Задача генератора - максимізувати процент помилок дискримінатора.

Генератор та дискрімінатор можуть бути побудовані на базі однієї із зазначених вище архітектур.

Данний підхід дозволяє отримувати фотореалістичні результати генерації зображень.

На рис. 1.4 зображена схема взаємодії генератора та дискрімінатора.

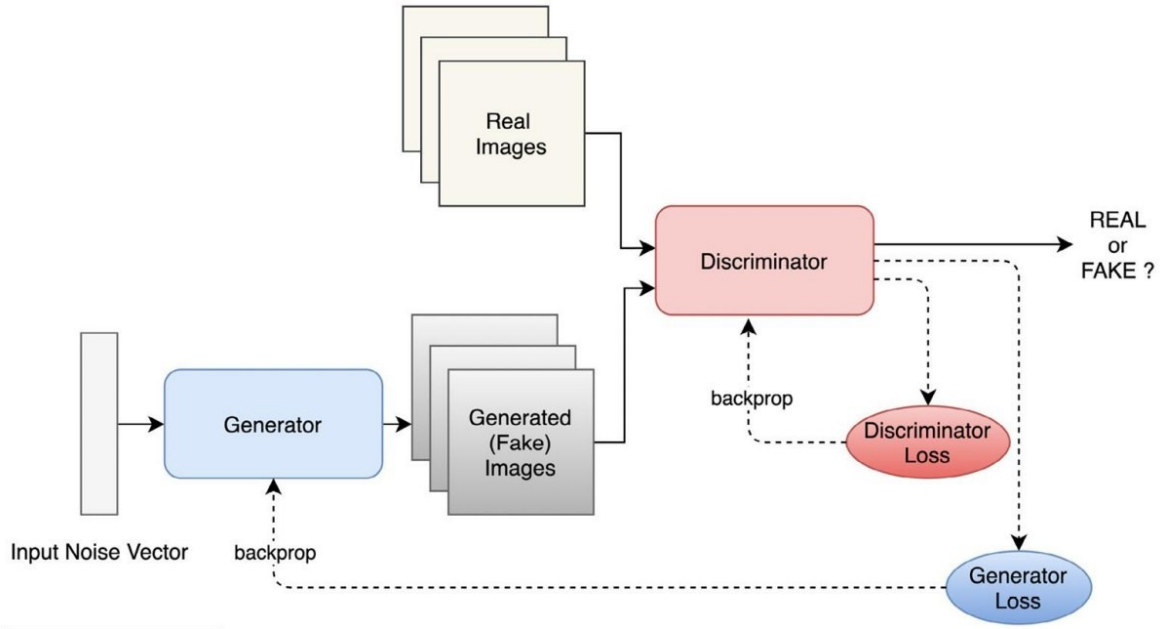


Рисунок 1.4 - Схема взаємодії генератора та дискрімінатора

До переваг генеративно-замгальних НМ відносять:

- потенціально дуже висока якість вихідних зображень;
- можливість навчання без вчителя;
- дуже висока чутливість до малих деталей.

До недоліків генеративно-замгальних НМ відносять:

- дуже висока обчислювальна складність як на етапі навчання, так і під час використання;
- нестабільність навчання;
- непередбачувані результати та артефакти.

1.2 Технології розробки Web-додатків

На сьогодні існує велика кількість інструментів, фреймворків та бібліотек для побудови Web сервісів.

Оскільки сама концепція Web сервісів існує стільки ж скільки й інтернет, у цьому напрямку сформувались стабільні практики, а основні інструменти є взаємозамінними.

Тим не менш, у даній області є декілька конкуруючих філософій щодо внутрішної структури додатків. Також фреймворки Web програмування мають характерні для їх мов програмування переваги та недоліки.

Насамперед треба зазначити, що всі перелічені фреймворки мають два набори інструментів:

1. Імперативний

   Цей підхід до програмування має більший контроль над станом програми;
2. Реактивний

   Цей підхід до програмування є більш абстрактним, він базується на подіях та зміні стану в даних.

1.2.1. NodeJS

NodeJS - інструмент, надаючий оточення для виконання JavaScript на базі рушія V8 від браузеру Chromium.

На базі NodeJS побудована велика кількість фреймворків для Web розробки, одні з найпопулярніших:

1. Express.JS;
2. Socker.IO;
3. Nest.js

Для огляду був обраний Nest.js, оскільки він є достатньо високорівневим, гнучким та універсальнім. Також він включає в себе код більш низькорівневих фреймворків, таких як зазначений Express.

Фреймворк дуже популярний завдяки відсутності язикового бар'єру для Front-end розробників, що прискорює розробку додатків у невеликих командах.

Фреймворк перейняв багато гарних практик та ідей з більш старих фреймворків, має модульну структуру і можливість інтеграції з багатьма сторонніми сервісами.

Недоліки фреймворк наслідує із своєї мови програмування JavaScript.

Це:

- відсутність типової безпеки;
- відсутність можливості роботи за багатьма потоками;
- можливість скритих помилок у роботі, які не призводять до виклику виключень або інших механізмів захисту.

1.2.2 Django

Django - високорівневий фреймворк для розробки Web додатків мовою Python. Він розроблявся з метою надати можливість стрімкої розробки додатків.

Філософія Django - "batteries included". Він надає багато інструментів за замовченнями для широкого кола задач, серед яких є:

- автентикація та авторизація;
- ORM для роботи з базами даних;
- процедурно генерована панель адміністратора;
- інструменти масштабування;
- маршрутизація url;
- менеджер сесій;
- інструменти геолокації.

Такий підхід дозволяє швидко побудувати Web додаток для будь-яких цілей, але такий підхід має і певні недоліки:

- Монолітна структура
  Інша сторона філософії "Batteries included"- щільна інтеграція всіх модулів за замовченням. Це може бути проблемою при необхідності розбити сервіс на модулі;
- Фреймворк побудований навколо ідеї "Лише одне вірне рішення"
  Це може обмежити розробників у використанні
  альтернативних бібліотек.

1.2.3 Flask

Flask - ще один фрейморк для розробки мовою python. У певному сенсі він є протилежністю Django:

- мінімалістичний;
- гнучкий;
- відкритий до розширювань.

Flask краще підходить для розробки малих сервісів та прототипування, ніж для великих проектів рівня enterprise.

З його мінімалістичності та простоти виходять закономірні недоліки:

- необхідність ручного конфігурування багатьох елементів;
- відсутність нав'язаних стандартів та патернів проектування може призвести до хаотичної структури проекта.

1.2.4. Spring Framework

Spring Framework - фреймворк для розробки веб-додатків мовою Java. Він є одним з найстаріших фреймворків для Web розробки, є дуже стабільною платформою з розвиненою екосистемою.

Незважаючи на вік, Spring Framework є сучасним і дуже гнучким фреймворком. Завдяки проекту Spring Boot, Він має можливість швидкого розгортання єдиним .jar файлом, який містить в собі всі необхідні бібліотеки та вбудований контейнер сервлетів Tomcat Embedded. Spring має модулі для інтеграції із всіма можливими сервісами, протоколами та програмними продуктами.

Завдяки гнучкій системі автоконфігурації і пакетах-стартерах, Spring framework одночасно гнучкий у налаштуванні як Flask та надає велику кількість переконфігурованих модулів як Django.

Ще декілька переваг Spring Framework отримує від мови програмування Java:

- строгу статичну типізацію;
- повноцінне багатопоточне середовище;
- високу швидкість виконання байт-коду.

До недоліків можна віднести:

- повільнішу швидкість розробки через велику кількість вимог до коду від компілятора;
- великий розмір виконавчих файлів зібраного додатку.

1.3 Технології розробки мікросервісних інформаційних систем

В сфері побудови ІС з мікросервісною архітектурою основними технологіями є контейнерізація програмних додатків та управління розгорткою контейнерів. В області контейнерізації промисловим стандартом є Docker, який завдяки функції ізоляції в ядрі Linux дозволяє запускати програмні додатки ізольовано один від одного.

В сфері управління розгорткою контейнерів є декілька рішень, які відрізняються складністю, гнучкістю та сферою застосування.

1.3.1 Docker Compose

Docker Compose - утиліта контролю Docker контейнерів, яка дозволяє конфігурувати кожний з контейнерів:

- параметри мережі, відкриті порти;
- змінні оточення;
- розділи для зберігання даних;
- стратегію поведінки, перезапуску, health-check.

Даний інструментарій дозволяє швидко розгортувати ІС з декількох контейнерів та налаштовувати їх взаємодію відповідно до файлу конфігурації.

1.3.2 Kubernetes

Kubernetes - система оркестрації контейнерів, призначена для створення дубльованої гнучкої та децентралізованої ІС. Kubernetes об'єднує окремі машини в кластер, та дає можливості:

- гнучко розподіляти навантаження між нодами кластеру;
- керувати репликацією контейнерів в залежності від поточного навантаження;
- автоматично відновлювати роботу контейнерів у випадку виходу ноди з ладу;
- виконувати оновлення версії контейнерів без даунтайму.

В цілому Kubernetes призначений для роботи з великими ІС з високим ступенем.

Як неодноразово відзначалось [4], Kubernetes вивів великі Enterprise системи на новий еволюційний рівень гнучкості, надійності та відмовостійкості. складності.

1.4 Обґрунтування застосованих технологій

Розглянувши наявні на сьогодні методи та інструменти вирішення задач в предметній області, було вирішено використати наступний перелік технологій:

1. Для побудови НМ була обрана архітектура U-net, оскільки вона має одні з найкращіх показників щодо якості вихідного зображення у задачах регресії зображеня-зображеня. U-net може працювати з малими датасетами, не має тенденції до перенавчання. Також НМ на базі U-net буде значно ефективніша за GAN, оскільки вихідне зображення генерується за один прохід. В якості фреймворку для побудови НМ був обраний PyTorch

через простоту використання і компактний код. Ще одним аргументом на його користь стала підтримка обчислювального стеку ROCm для наявної відеокарти AMD;

2. Для побудови Web сервісу був обраний Spring Framweork. Критеріями для цього вибору стали:
   - наявний практичний досвід застосування;
   - поєднання великої кількості утиліт для взаємодії із зовнішніми сервісами з простотою та гнучкістю їх налаштування;
   - безпека та надійність мови програмування Java;
3. Для обслуговування розробленого сервісу та його зв'язку з розробленою моделлю були обрані наступні технології:
   - База даних PostgreSQL для зберігання інформації про завантажені фотографії та статус їх обробки;
   - Черга повідомлень RabbitMQ. Як зазначають [5], використання черг повідомлень на базі протоколу AMQP значно підвищує стабілність системи за рахунок гарантування доставки повідомлень та асинхронності. Використання черги повідомлень сильно спрощує розділення ІС на мікросервіси. Використання черги повідомлень сильно спрощує розділення ІС на мікросервіси. Черга гарантує доставку та обробку повідомлень, а також бере на себе функцію балансування навантаження при декількох споживачах;
   - Сховище об'єктів Minio для збереження завантажених фотографій та результатів їх обробки.

     Використання зовнішнього сховища також сприяє децентралізації архітектури розробляємої ІС. Крім того, дане сховище повністю сумісне зі сховищем AWS, що

дозволяє за необхідності перемкнутися на хмару лише заміною конфігурації, без змін в коді;

– Клієнт - обгортка моделі, написана мовою Python.

Оскільки сама модель використовує Python, для взаємодії з ним потрібен адаптер. Цей адаптер буде отримувати дані з RabbitMQ за допомогою бібліотеки pika, завантажувати, готувати і відправляти файли до моделі, повертати результати та інформувати про них чергу повідомлень.

2. ПОБУДУВАННЯ МОДЕЛІ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ПІДВИЩЕННЯ РІЗКОСТІ ЗОБРАЖЕНЬ

Для побудування НМ, яка буде ефективно виконувати завдання підвищення різкості зображень, необхідно обрати структуру НМ, метод її навчання, підготувати дані і провести порівняння різних конфігурацій.

2.1 Підготовка даних для нейронної мережі

Перед тим як почати конструювання НМ та її навчання, треба належним образом підготувати дані дня навчання.

Згорткові НМ будують під роботу із зображеннями фіксованого розміру, у багатьох випадках це - квадратне зображення із стороною, яка дорівнює ступеню двійки. Опрацювання зображень інших розмірів відбувається за допомогою сегментації та паддингу вхідного зображення.

Для навчання НМ був створений датасет квадратних зображень розміру 512x512 пікселів.

Базисом для тренувальної вибірки був обраний датасет dalle-mini/open-images [6], в якому містяться посилання на 9 мільйонів зображень у відкритому доступі.

Для формування тренувальної вибірки були обрані 500 випадкових зображень датасету.

Для навчання НМ необхідно створити пари зображень - еталон (Ground truth) та розмите, яке НМ повинна наблизити до Ground truth.

Кожне з обраних зображень було відмасштабовано та обрізано до розміру 512x512 пікселів засобами opencv.

Для отримання розмитого зображення на підготовлене Ground truth було накладено ефект розмиття Гаусса засобами opencv.

Розмиття Гаусса виконується за наступним алгоритмом:

1. За допомогою функції нормального розподілення Гаусса виконується обчислення матриці (ядра) розмиття із заданими параметрами.

Параметри розмиття:

- kernel_size - розмір ядра.
  Чим більше матриця, тим більше сусідніх пікселей будуть змішуватись;
- sigma - параметр, контролюючий ширину розподілення функції Гаусса.
  Підвищує долю ваги пікселя відносно його віддаленості від центру. При великих параметрах sigma ці ваги можуть майже зрівнятись. Таким чином отримано пари чіткого та розмитого зображень.

Елемент з координатами (0,0) знаходиться в центрі ядра, а розміри ядра мають бути непарними.

Вихідний піксель розмитого зображення обчислюється за допомогою накладання ядра на вхідне зображення, при якому елемент ядра (0,0) відповідає обчислюваному пікселю, множенню пікселів у зоні накладання на

їх відповідні коефіцієнти ядра, та підсумовуванню отриманих значень. У разі якщо піксель знаходиться близько краю, зображення, та частина накладеного ядра виходить за межі зображення, на місце відсутнього пікселя підставляється найближчий до нього.

Приклад розмиття з різними параметрами зображено на рис. 2.1

Рисунок 2.1 - Результат гаусівського розмиття

2.1.1 - Зображення до розмиття

2.1.2 - kernel_size 5x5, sigma 1.5

2.1.3 - kernel_size 5x5, sigma 5

2.1.4 - kernel_size 7x7, sigma 5

Серед наведених для демонстрації та ще кількох комбінаціях параметрів розмиття, найбільш доцільними для тренування булі обрані наступні:

- kernel size: 5x5;

- sigma: 1.5.

При такому розмитті візуально помітна значна деградація зображення, проте на зображені все ще присутні залишки ознак, які можуть бути знайдені та відновлені при роботі НМ.

Таким чином отримано пари чіткого та розмитого зображень.

На рис.2.2 зображено приклад чіткого та розмитого зображень для навчання НМ.

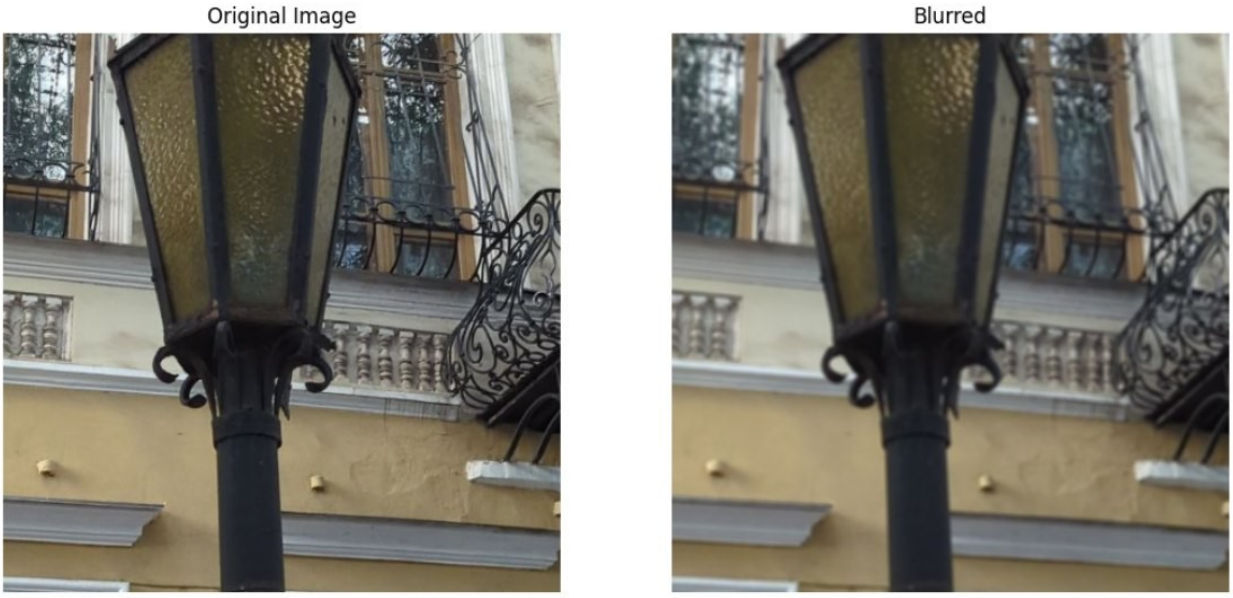


Рисунок 2.2 - Приклад чіткого та розмитого зображень для навчання НМ

Накладення такого розмиття успішно імітує реальний кадр, зроблений не в фокусі камери.

2.2 Архітектура нейронної мережі

Для порівняльного аналізу підходів до структури НМ та її навчання, була розроблена динамічна модель на базі архітектури U-net. Дана модель отримує на вхід кількість енкодерно-декодерних пар, а також початкову кількість каналів, яка буде подвоюватись на кожному етапі кодування.

Структурно НМ складається з трьох вивід блоків: енкодер, боттлнек, декодер.

Всі блоки при створенні мають наступний набір параметрів:

- in_channels - кількість каналів, які входять до блоку;
- out_channels - кількість каналів, які блок віддає на виході;
- use_bn - флаг, індикуючий потрібність додати шари нормалізації.

Якщо встановлений до False, замість шару нормалізації буде додано прозорий шар nn.Identity, який пропускає через себе дані без змін.

Параметри усіх шарів типу nn.Conv2d, зокрема вихідного шару:

- kernel_size: 3 - цей параметр є гарною відправною точкою, оскільки при більших розмірах ядра згорткові шари втрачають чутливість до дрібних деталей, що може негативно вплинути на показники підвищення різкості, яка цілком залежить від якості передачі дрібних деталей зображенням;
- padding: 1 - цей параметр потрібен для збереження оригінальних розмірів зображення.

Оскільки ядро згорткового шара не може проходити по крайнім рядкам та кутам зображення, навколо оригінала додається рамка з нульових пікселів.

Параметри усіх шарів типу ConvTranspose2d:

- kernel_size: 2;
- stride: 2.

Ці параметри зазначені в оригінальній праці про U-net і використовуються в дослідженні як опорне рішення.

Вихідний шар НМ має параметр kernel_size 1, оскільки його задача не знаходити ознаки, а привести вже знайдені ознаки до RGB діапазону окремо для кожного пікселя.

2.2.1 Структура енкодера

Кожний енкодер - блок з двох згорткових шарів, переміжених шарами активації ReLu. У процесі дослідження були також додані опціональні шари нормалізації. Наприкінці блоку доданий шар MaxPooling, який зменшує роздільну здатність зображення, обираючи тільки найбільш виражені ознаки.

При роботі блок повертає два значення: skip та pool.

skip - повертає значення до роботи MaxPooling, щоб зберегти більше ознак у просторі, які будуть використані при декодуванні.

pool - повертає дані після даунсемплінгу з найбільш вираженими ознаками, використовується як вхід до наступного енкодеру або боттлнеку.

Структура енкодеру зображена на рис. 2.3

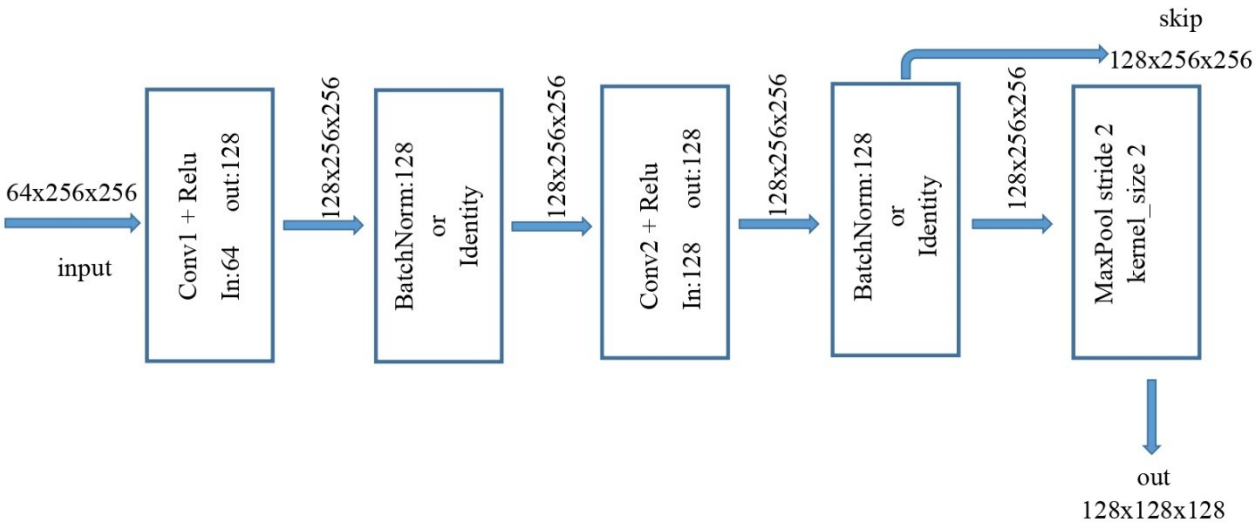


Рисунок 2.3 – Структура енкодеру

2.2.2 Структура bottleneck шару

Bottleneck - Структурно схожий на енкодер блок, найглибша частина НМ, в якій дані є найбільш абстрактними.

Також має 2 шари, переміжені шарами активації та нормалізації. Вихідні дані передаються в перший декодер.

Оскільки задачою роботи НМ є підвищення різкості зображення, у якій дуже велике значення має максимальна точність на границях зображення, у блоці bottleneck, як і в енкодері, відбувається подвоєння кількості каналів, що має дати можливість вивчати комплексні абстрактні ознаки, ціною підвищеної обчислювальної складової.

Структура боттлнеку зображена на рис. 2.4

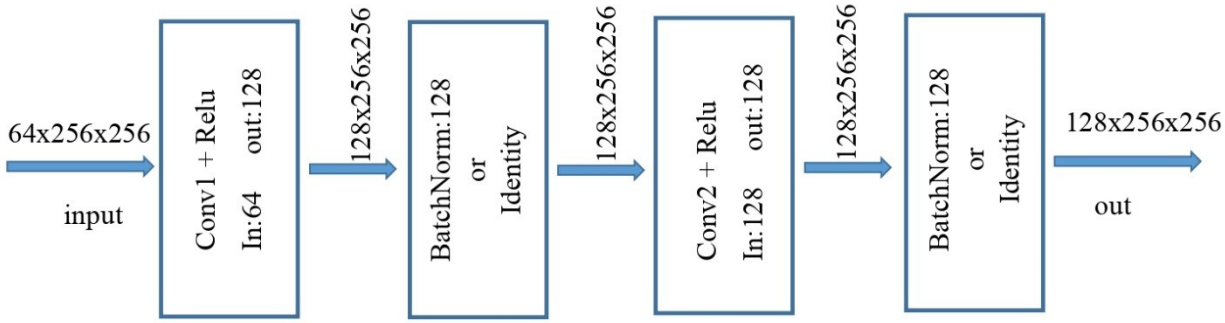


Рисунок 2.4 - Структура боттлнеку

2.2.3 Структура декодеру

Декодер - блок шарів, призначений декодувати дані з енкодерів до початкового вигляду зображення.

Декодер поєднує абстрактні висококанальні дані, які пройшли через весь ланцюг енкодерів та ботлнек, та поєднує їх з первинними даними до кодування. Таке рішення дозволяє НМ не втрачати контекст, що часто відбувається у лінійних, особливо глибоких мережах.

На вход блок отримує два параметри:

- дані input з виходу попереднього декодера або ботлнеку;
- дані skip, які повернув відповідний цьому декодеру енкодер.

Декодер виконує деконіолюцію даних skip через шар nn.ConvTranspose2d, збільшуючи роздільну здатність та поменшуючи кількість каналів. Більш ефективною для обчислювань міг стати ненавчаємий шар nn.Upsample, який робить цю операцію шляхом білінейноїінтерполяції або інтерполяції найближчого сусіда. Але це може негативно вплинути на точність результатів роботи НМ, що є критичним.

Після деконволюції, до даних input конкатенуются дані skip, що поєднує ознаки, знайдені енкодерами, з даними про місцезнаходження цих ознак на зображені.

Після конкатенації об'єднані дані проходять через два згорткові шари з активацією та нормалізацією, після чого предаються до наступного декодеру.

На рис. 2.5 зображена структура декодера

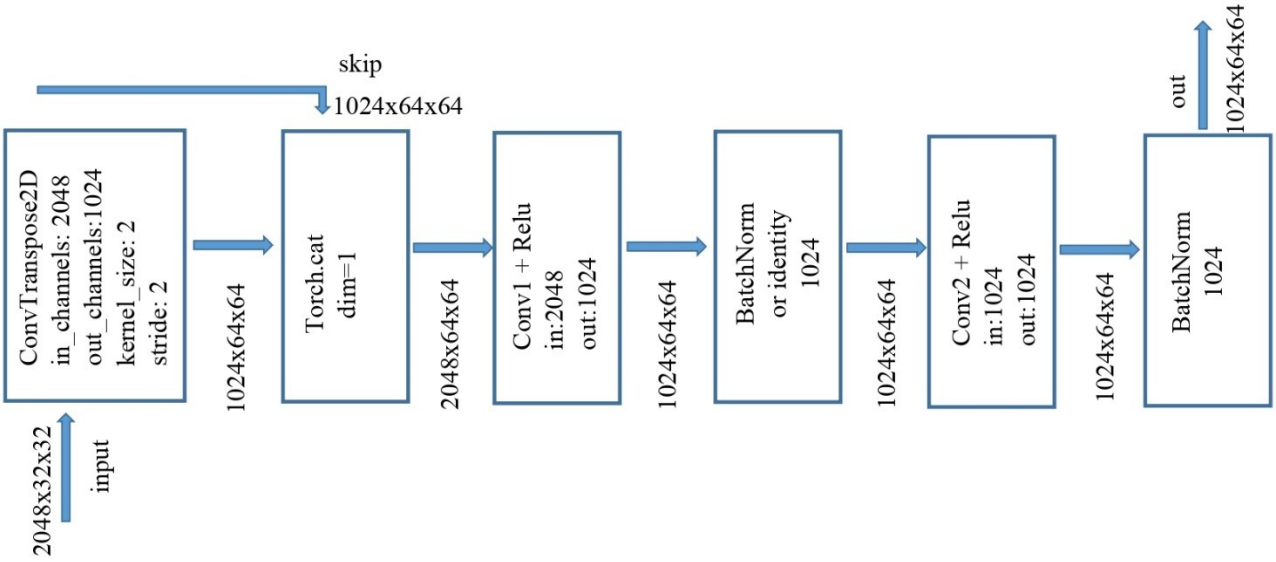


Рисунок 2.5 - Структура декодера

На виході з останнього енкодера розташований вихідний згортковий шар, перетворюючий дані до первинного вигляду. Цей шар має кількість вхідних каналів, дорівнюючу кількості вхідних каналів моделі. Кількість виходних каналів дорвінює трьом, що є RGB каналами вихідного зображення.

Первинний аналіз ефективності НМ проводився для НМ с 3, 4 та 5 енкодерно-декодерними парами.

Була спроба натренувати НМ з 6 парами, але через нестачу відеопам'яті на наявному графічному прискорювачі ця спроба була невдалою.

Таким чином, була побудована класична U-net модель для задачі регресії зображення-зображення.

Повний вихідний код моделі та окремих її модулів наведено у додатку А.

2.3 Функція втрат

Коректно зазначена функція втрат для навчання НМ є критичною для якості результатів.

Найпоширенішими функціями втрат є середньоквадратичне (L2) та середнє абсолютне відхилення (L1). Функція L2 була використана на початковому етапі навчання НМ, для визначення оптимальної кількості енкодерно-декодерних пар НМ.

В якості функцій втрат було розглянуто 2 варіанти:

1. Запропонований в тезах [7] метод модифікації функції MSE на основі оператора Canny для пріоритезації границь зображення в роботі НМ.

   Описаний в тезах метод був покращений за допомогою розширення області пріоритетизації.

   Для розширення цієї області на результат роботи оператора Canny накладено ефекти розмиття Гаусса з параметрами:

   – kernel size: 3x3;

   – sigma: 1.5.

Також було додано множник λ для більшої гнучкості у штрафуванні пікселів на та поряд з границями зображення.

Підсумковий математичний вираз має вигляд::

$$F_{loss} = \frac{\sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \left[f(x,y) - f_{pred}(x,y) \right]^2 \cdot \left[CannyCoef_{xy} + 1 \right]}{N \cdot M},$$

де F_{loss} – модифікована функція втрат;
N, M – кількість відповідно рядків та стовпців зображення;
$f(x, y)$ – значення яскравості пікселя вхідного зображення;
$f_{pred}(x, y)$ – значення яскравості пікселя зображення, яке передбачене;
$CannyCoef_{xy}$ – коефіцієнт втрат на базі результатів обробки оператора Canny;
λ – коефіцієнт додаткової ваги пікселя визначеного оператором Canny.

На рис.2.6 зображено "зону інтересу" НМ, в якій містяться границі зображення.

2. Використання шарів виявлення ознак натренованої НМ VGG16 для порівняння ознак вхідного та вихідного зображень.

3. Перцептуальна функція втрат порівнює виходи трьох перших блоків виявлення ознак мережі VGG16 за допомогою функції L2.

Рисунок 2.6 - "Зона інтересу" НМ, в якій містяться границі зображення.

2.4 Тренування нейронної мережі

Тренування усіх моделей відбувалось на ПК с наступною конфігурацією:

- CPU Ryzen 7 5800x;
- RAM 64 GB DDR4-3200;
- GPU AMD Radeon RX 6750XT 12 GB GDDR6.

Для об'єктивної оцінки результатів роботи НМ використовувався показник SSIM, який добре корелює з суб'єктивною оцінкую різкості зображень. Запропонований в 2004 [8] році і детально розібраний у 2020 [9], показник SSIM використовується для оцінки схожесті зображень. На відміну від таких показників як MSE, MAE, Laplacian Variance або PSNR, цей показник значно краще корелює з суб'єктивним сприйняттям якості зображень, оскільки враховує велику кількість факторів, таких як яскравість, контрастність, структуру.

Ця метрика є метрикою full-reference, що не дозволить використовувати її у процесі експлуатації НМ, але дасть гарний показник якості для навчання і валідації.

Для об'єктивної оцінки результатів роботи НМ використовувався показник SSIM, який добре корелює з суб'єктивною оцінкую різкості зображень.

Також для оцінки навчених НМ були використані наступні параметри:

- загальній час навчання;
- значення функції втрат на валідаційному датасеті;
- розмір вагових коефіцієнтів отриманої НМ.

Параметри всіх навчених НМ:

- Initial channels: 64;
- Optimizer: Adam;
- Learning Rate: 0.0002;
- Training datased: 500 512x512 RGB images;
- Number of epohcs: 20.

Оптимізатором для НМ був обраний запропонований [10] в 2014 Adam. Цей оптимізатор продемонстрував значні переваги понад аналогами як швідкістю, так і точністю тренування. Adam демонструє перевагу від

десятків процентів до декількох разів над іншими оптимізаторами в швидкості тренування.

Серед інших переваг оптимізатора автори приводять:

- Адапитвне тренування за допомогою суб-градієнтів. Adam тренує кожен параметр окремо на базі його градієнтної історії. Оскільки в розроблюємій НМ велика кількість параметрів, ця перевага оптимізатору Adam є дуже вагомою;
- Використання "Інерції". Adam бере у розрахунок градіенти минулиї операцій для корекції швидкості навчання кожногоз параметрів;
- Гарні показники в роботі з великими датасетами та багатомивірними даними;
- Ефективність використовуємої пам'яті. Наявний для тренувать графічний прискорючач має досить середній для сучасних моделей обсяг відеопам'яті. Хоча це не проблема для побутового використання, цього не завжди достатньо для тренування великих глибоких моделей, опитмізація Адамом пам'яті є дуже вагомою перевагою.

Для навчання НМ було підготовлено два датасети даних:

- тренувальний датасет з 500 зображень, зведених до розміру 512x512 пікселів, випадково обраних з датасету dallee/open-images на приблизно 9 млн зображень;
- валідаційний датасет з 15 зображень, зведених до розміру 512x512 пікселів, використаних у якості тренувального від час написання тез [7].

Через велику обчислювальну складність та час тренування навіть за умови завантаження датасету до відопам'яті, початкові параметри тренування були обрані наступні: 20 епох на модель, learning rate 0.0002.

Оскільки під час підготовки тез перша версія моделі при learning_rate=0.001 демонструвала певну тенденцію до перенавчання, learning_rate було дещо знижено. Оскільки обчисленя однієї епохи для

датасету на 500 зображень займає від 60 до 90с, в цілях економії часу кількість епох було знижено до 20.

Після вибору аріхітектури мережі та функції втрат для тренування, були проведені тренування певної кількості моделей:

- три моделі з функцією втрат L2 для визначення оптимальної кількості енкодерно-декодерних пар. Були навчені моделі 3, 4 та 5 парами. Була спроба здійснити тренування моделі с 6 парами, але здійснити це не вдалось через нестачу відеопам'яті;
- дві моделі з виявленеою оптимальною глибиною: з модифікованою функцією MSE та з перцептуальною функцією на базі моделі VGG16.

2.4.1. Пошук оптимальної глибини

Результати навчання трьох НМ з різною глибиною та MSE лоссом наведені в табл.2.1

Таблиця 2.1

	SSIM_ SIMPLE	SSIM_ MEDIUM	SSIM_ HARD	TRAINING_ TIME	MEAN_ VALIDATION_ LOSS	WEIGHTS_ SIZE
MSE_ L3	0.84786	0.77824	0.66612	22min 29s	1.24023	88.12 MB
MSE_ L4	0.83680	0.77125	0.66136	26min 29s	1.36453	363.7 MB
MSE_ L5	0.87139	0.80183	0.70375	30min 48s	2.21484	1423 MB

Позначення:

SSIM_SIMPLE - показник SSIM для зображення з невеликою кількістю малих деталей та границь

SSIM_MEDIUM - показник SSIM для зображення з середнею кількістю малих деталей та границь

TRAINING_TIME - час тренування НМ при заданих параметрах.

SSIM_HARD - показник SSIM для зображення з великою кількістю малих деталей та границь

MEAN_VALIDATION_LOSS - середній показник втрат на валідаційному датасеті

WEIGHTS_SIZE - розмір файлу з ваговими коефіцієнтами НМ.

Як видно з наведеної таблиці, збільшеня глибини НМ призводить до покращення показниківя вихідних зображень.

Виходячи з цього, для подальшого навчання була обрана НМ з глибиною 5 енкодерно-декодерних пар.

На рис. 2.7 зображена схема побудованої НМ з обраною оптимальною глибиною без нормалізації даних.

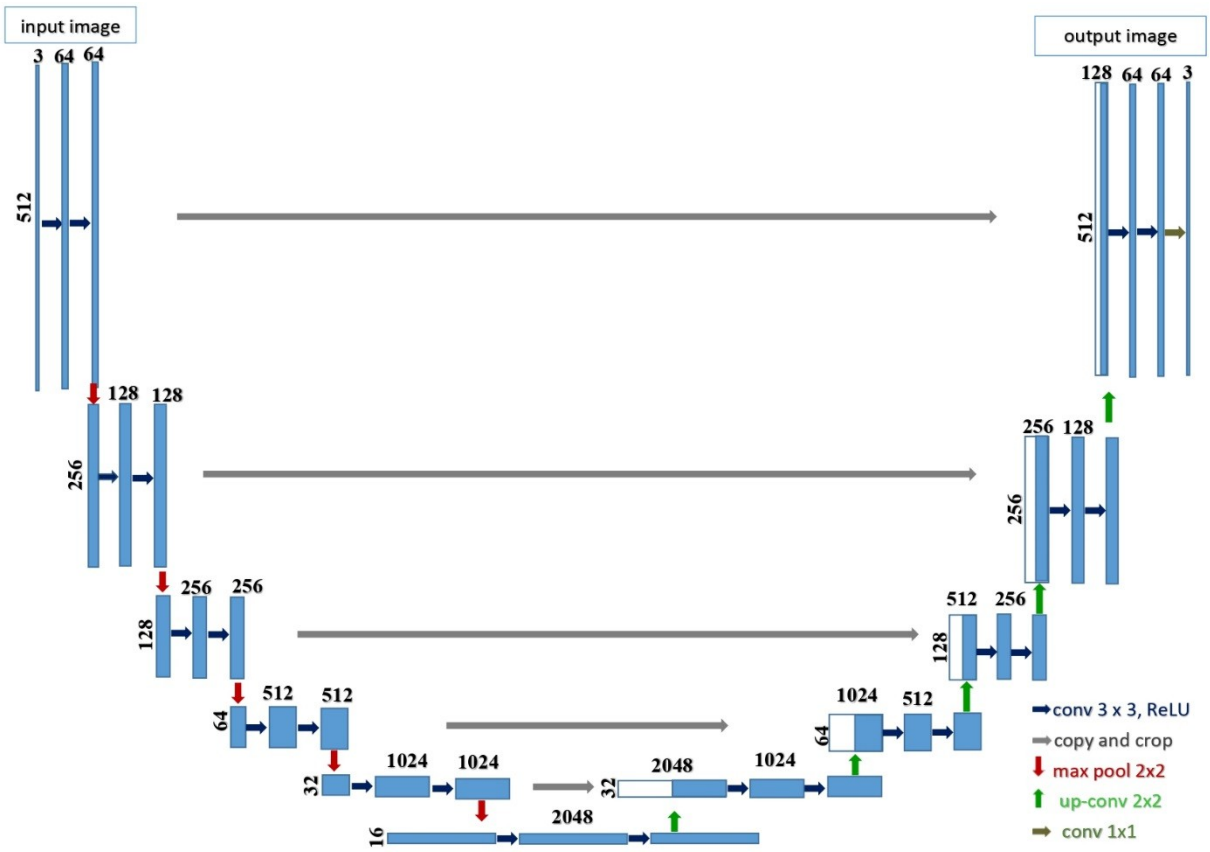
Результати роботи кожної с натренованих моделей для можливості візуальної оцінки роботи НМ див. у додатку Г. Повний код тренування НМ наведено у додатку Д.

2.4.2 Навчання нейронної мережі зі специалізованими функціями втрат

Після вияіленя опитмальної глибини для НМ, були натреновані дві НМ із зазначеною глибиною.

Результати навчання двох НМ наведені в табл.2.2

Таблиця 2.2



	SSIM_	SSIM_	SSIM_	TRAINING_	MEAN_

Рисунок 2.7 - схема побудованої НМ з обраною оптимальною глибиною без нормалізації даних.

					LOSS
MSE_L5	0.87139	0.80183	0.70375	30min 48s	2.21484
CANNY_ MSE	0.87077	0.79627	0.70091	31min 31s	1.43245
VGG_ PERC	0.88028	0.81032	0.72089	33min 17s	1.66416

MSE_L5 - показники натренованої на функіці MSE НМ з глибиною 5.

Через той факт, що усі дані для тренування були завантажені в пам'ять відеокарти, час виконання перцептуальної втрати за допомогою VGG16 навіть швидший ніж у MSE. Але якщо є необхідність розрахувати loss на процесорі, обробка зображень засобами VGG може займате декілька секунд на зображення. MSE та його модифікації значно швидше працюють на CPU, що дозволяє звільнити частину відеопам'яті для інших потреб.

Оскільки показники НМ, натренованої CANNY_MSE віявились нижчими за звичайний MSE на більшому датасеті, додатково була зроблена перевірка показника laplacian variance.

Результати перевірки наведені в табл.2.3

Таблиця 2.3

	LAPLACIAN_ORIGINAL	LAPLACIAN_OUTPUT
MSE_L5_SIMPLE	9.890	5.187
MSE_L5_MEDIUM	18.134	11.008
MSE_L5_HARD	31.570	20.517
MSE_CANNY_SIMPLE	9.890	12.704
MSE_CANNY_MEDIUM	18.134	19.903
MSE_CANNY_HARD	31.570	43.261
VGG_SIMPLE	9.890	7.423
VGG_MEDIUM	18.134	10.714

VGG_HARD	31.570	24.507

Результати роботи трьох зазначених НМ можно побачити у додатку Г.

Для зручності читання показники помножені на 1000, десяткова частина скорочена до 3 знаків після коми.

Як можна побачити, показники вихідних даних для НМ, яка тренувалась на запропонованому методі, перевищує показники вхідних зображень. З цього можно зробити висновок, що запропонована функція має слабку кореляцію з суб'єктивним сприйняттям різкості.

Як можна побачити, вихідні дані НМ, особливо на зображені з великою кількістю деталей, мають ефект "пікселізації" - на дрібних деталях присутній шум на границях. Для боротьби з цим ефектом було запроваджено нормазізацію даних проміж шарами, окрім першого енкодера та останнього декодера.

Результати навчання із зазначеними параметрами, та порівняння з результатами НМ без нормалізації наведено в таблиці 2.4

Таблиця 2.4

	VGG_PERC	VGG_PERC_NORM	REAL_DATA
LAPLACIAN_SIMPLE	7.423	7.302	9.890
LAPLACIAN_MEDIUM	10.714	11.077	18.134
LAPLACIAN_HARD	24.507	16.070	31.570
SSIM_SIMPLE	0.88028	0.87634	—
SSIM_MEDIUM	0.81032	0.81281	—
SSIM_HARD	0.72089	0.72526	—

Результат роботи НМ з нормалізацією даних у додатку Г.

Як можна побачити з показників нормалізованої НМ, показники дещо знизились на зображені з малою кількістю деталей, але зросли на більш складних зображеннях. На складному зображені можна побачити значне зниження лапласівської дисперсії при підвищені показника SSIM, що може свідчити про значне зниження пікселізації. Для подальшого покращення результатів була натренована ще одна модель на вдвічу більшій кількості епох.

При тренуванні НМ з нормалізацією на 40 епох були отримані результати, які наведені у таблиці 2.5

Таблиця 2.5

	VGG_PERC_NORM_40	REAL_DATA
LAPLACIAN_SIMPLE	13.367	9.890
LAPLACIAN_MEDIUM	18.944	18.134
LAPLACIAN_HARD	30.642	31.570
SSIM_SIMPLE	0.87742	—
SSIM_MEDIUM	0.80375	—
SSIM_HARD	0.72921	—

Час тренування моделі на 40 епох: 1h 3min 41s

У додатку Г зображено результат роботи моделі при тренуванні на 40 епох.

Я можна побачити з метрик, показник SSIM дещо впав у порівнянні з моделлю на 20 епох, проте показник лапласівської дисперсії максимально наблизився до показників оригінальних чітких зображень.

На даному етапі можна зробити висновок, що дана конфігурація НМ досягла ліміту ефективності, подальше покращення вимагатиме перезбірки моделі, глибокої модифікації енкодерів та декодерів.

Оскільки найкращі показники у підвищені різкості отримала НМ з перцептульною функцією втрат на основі моделі VGG, з нормалізацією даних, вона була обрана для побудови ІС.

3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ВЕБ-СЕРВІСУ ДЛЯ ПІДВИЩЕННЯ РІЗКОСТІ ЗОБРАЖЕНЬ НА БАЗІ НАВЧЕНОЇ МОДЕЛІ

Для створення ІС з мікросервісною архітектурою треба визначити її головні компоненти та способи їх взаємодії.

Виконанням бізнес-логіки займаються 2 компоненти:

- Компонент з НМ, який приймає розмиті зображення та віддавати у відповідь різкі;
- Компонент з Web-сервісом, який приймає запити від користувача, зберігає завантажені файли та дозволяє отримати результати роботи НМ та переглянути їх статус.

Для обслуговування цих компонентів були додані технічні компоненти:

1. База даних PostgreSQL, яка зберігає історію завантажених файлів, посилання на них, статус процесів НМ;
2. Черга повідомлень RabbitMQ, яка відповідає за комунікацію між двома бізнес-компонентами.
   Бере на себе роль балансира та маршрутизатора даних, коли є необхідність додати декілька компонентів з НМ;
3. Сховище файлів minio.
   Зберігає завантажені файли та результати підвищення їх різкості.
   Дозволяє без змін в коді перемкнутись на хмарне сховище Amazon S3, що підвищує гнучкість ІС.
   Архітектура побудованої ІС зображена на рис.3.1

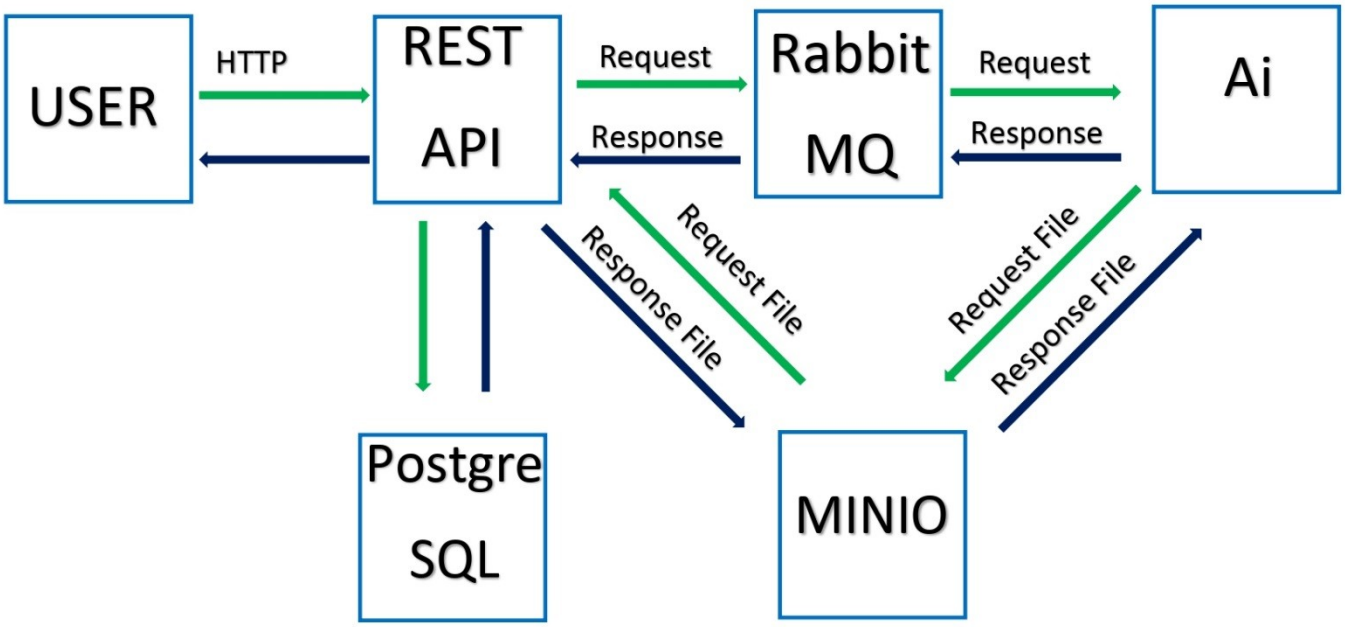


Рисунок 3.1 - Архітектура побудованої ІС

Архітектурою передбачено наступний порядок дій:

4. Користувач завантажує файл зображення на ендпоінт POST /;
5. Web-сервіс завантажує файл до сховища minio та створює записи про це в таблицях file та image в базі даних PostgreSQL, повертає користувачу id його файла;
6. Користувач робить запит POST /submit, до якого передає отриманий id;
7. Web-сервіс перевіряє наявність файла з таким ID, формує заявку на підвищення різкості, створює запис в базі даних, відправляє заявку до черги заказів RabbitMQ, повертає користувачу номер заявки;
8. Скрипт-обгортка НМ отримує заявку з черги заказів, завантажує відповідний файл, передає його НМ. НМ передає до обгортки вихідний файл;
9. Скрипт завантажує вихідний файл до minio, та надсилає повідомлення до черги відповідей RabbitMQ;
10. Web-сервіс отримує відповідь від модуля НМ, сворює для вихідного зображення записи в базі даних, оновлює статус заявки;

11. Клієнт, маючи id своєї заявки, звертається до Web-сервісу, отримає його актуальний статус, і, у випадку завершення, завантажує покращене зображення.

3.1 Структура Web-сервісу

Архітектурно Web-сервіс складається з трьох рівней:

1. Рівень даних.

   На цьому рівні знаходяться репозиторії, взаємодіючі з базою даних. В сервісі присутні три репозиторії з їх відповідними сутностями.

   В розробленому Web-сервісі це:

   - FileRepository та File.

     Сутність має поля:

     - ✓ id: String, автоматично згенерований uuid4;
     - ✓ path: String, ключ, за яким тіло файла доступно в сховищі minio;
     - ✓ filename: String, оригінальна назва завантаженого файлу;
     - ✓ status: Enum, статус файла.

       Може бути CREATED та UPLOADED.

   - ImageRepository, Image.

     Сутність має поля:

     - ✓ id: String, автоматично згенерований uuid4;
     - ✓ file: File.

       Файл зображення, завантажений до середовища minio.

       Зв'язок з File один до одного;
     - ✓ laplacian: double.

       Метрика Лапласівської дисперсії. Дозволяє оцінити ступінь підвищення різкості між вхідним та вихідним зображеннями. Обчислюється на стороні модуля НМ.

   - SumbissionRepository, Submission.

Сутність має поля:

- ✓ id: String, автоматично згенерований uuid4;
- ✓ inputImage: Image.
  Вхідне зображення, передане для підвищення різкості. Зв'язок з Image багато до одного;
- ✓ outputImage: Image.
  Вихідне зображення, згенероване НМ. Зв'язок з Image багато до одного.

Усі репозиторії наслідують інтерфейс CrudRepository, та надають стандартний набір функцій Create, Read, Update, Delete.

Підключення до бази даних відбувається за допомогою бібліотеки ORM Hibernate, що робить використання SQL в базових операціях непотрібним.

На рис. 3.2 зображено схему бази даних для Web сервісу.

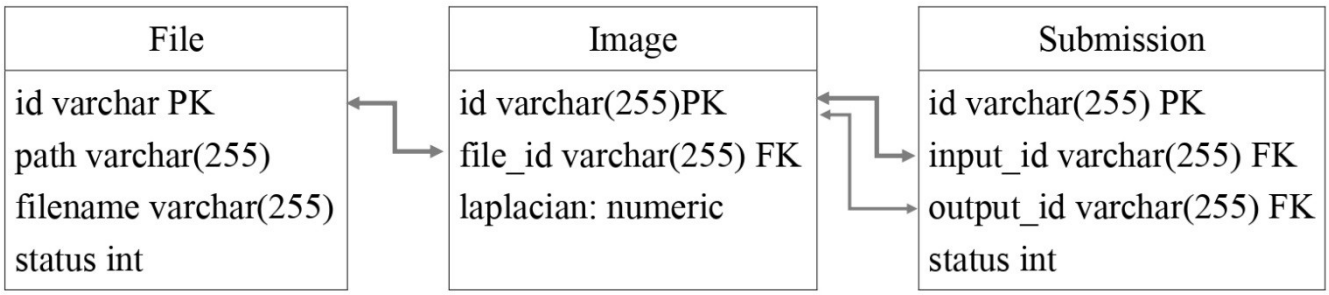


Рисунок 3.2 - Схема бази даних для Web сервісу

На рис 3.3 зображено UML-діаграму моделей даних.

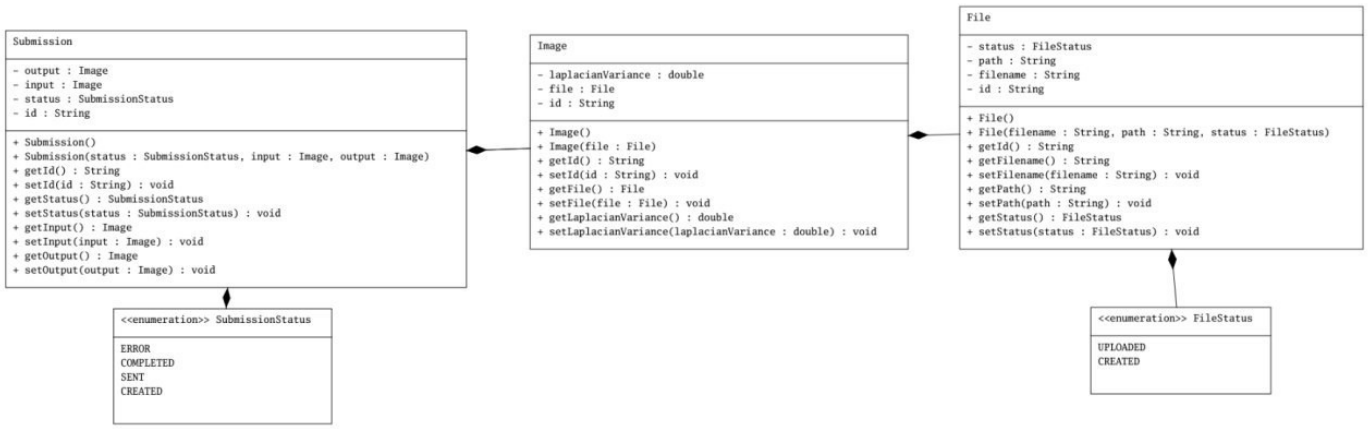


Рисунок 3.3 - UML діаграма моделей даних Web-сервісу

2. Рівень сервісів.

   На цьому рівні відбувається уся бізнес-логіка та управління даними. Розроблений додаток має наступні сервіси:

   – FileService - сервіс роботи з файлами.

     Поєднує в собі логіку роботи з FileRepository та клієнтом Minio. Має методи для збереження файлу, створення запису для вже існуючого в Minio файлу, отримання запису про файл з бази даних та для отримання тіла файлу з Minio;

   – ImageService - сервіс роботи з зображеннями.

     Дає можливість створити запис про зображення з файлу та отримати записи зображень;

   – SubmissionService - сервіс роботи з запитами до модуля НМ.

     Дозволяє сформувати запит на підвищення різкості для зображення та відправити його до модуля НМ. Також надає доступ до перегляду запитів та їх статусів;

   – MessageService - сервіс відправки повідомлень до черги RabbitMQ;

3. Рівень контролерів.

На цьому рівні відбувається обмін даних з користувачем та з модулем НМ.

В Web-сервісі є 2 контролери:

- SharpeningController - контролер, відповідаючий за взаємодію з користувачем засобами HTTP REST API.

  Він має низку методів для взаємодії з ІС:

  - ✓ [GET /test] - Тестовий метод для перевірки статусу сервера при розгортанні в Docker контейнері.

    Повертає строку-привітання та поточну дату;
  - ✓ [GET /submission] - Метод отримання усіх запитів на обробку зображень.

    Повертає список об'єктів Submission;
  - ✓ [GET /submission/{submissionId}] - Метод отримання стану зареєстрованого запиту на обробку.

    Приймає ідентифікатор запиту у вигляді змінної URL.

    Повертає об'єкт Submission;
  - ✓ [POST /submission] - Метод завантаження зображень до ІС.

    Приймає файл у форматі multipart.

    Повертає об'єкт Image із ідентифікатором;
  - ✓ [POST /submit] - Метод для формування запиту на обробку зображення.

    Приймає JSON body об'єкт з полем id зображення.

    Повертає об'єкт Submission;
  - ✓ [GET /image] - Метод отримання даних усіх зображень.

    Повертає список об'єктів Image;
  - ✓ [GET /image/{imageId}] - Метод для завантаження зображення з ІС до диску користувача.

    Приймає id зображення у якості параметра url.

На рис. 3.4 зображено UML-діаграму зв'язку компонентів усіх трьох рівнів.

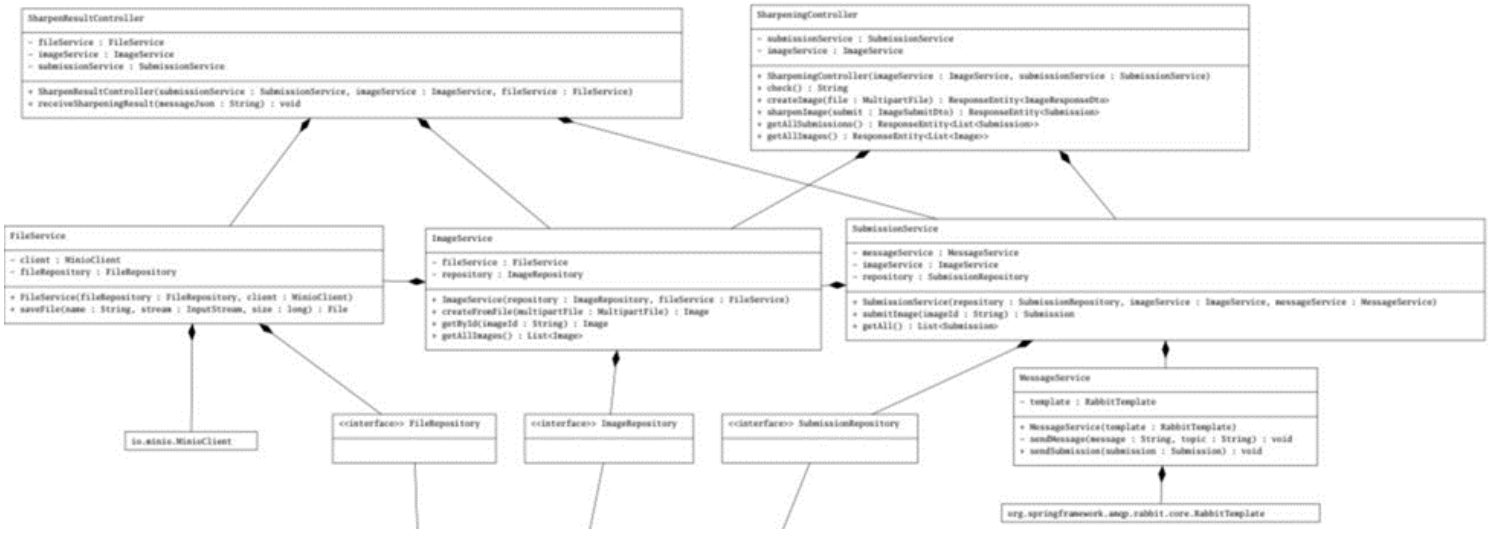


Рисунок 3.4 - Діаграма компонентів Web додатку

3.2 Структура скрипту-обгортки моделі нейронної мережі

Рисунок 3.4 - UML діаграма компонентів Web-сервісу

Модуль з НМ для підвищення різкості зображень складається із основного скрипта-клієнта та декількох скриптів з утилітами.

Основний скрипт після запуску виконує наступні дії:

- отримує конфігурацію із змінних оточення;
- ініціалізує ваги навченої моделі;
- ініціалізує черги повідомлень RabbitMQ;
- ініціалізує клієнт Minio для отримання та відправки зображень;
- за допомогою функції callback підписується на повідомлення із черги запитів.

3.2.1 Цикл роботи

Після ініціалізації скрипт буде обробляти запити, поки не буде зупинений. Цей процес умовно названо циклом роботи.

При отриманні запиту скрипт виконує наступні дії:

- Завантажує файл зображення з minio до локальної тимчасової директорії;
- За допомогою скрипта-утиліти масштабує та обрізає зображення до розміру 512x512 пікселів.

Це необхідно через особливість архітектури побудованої НМ.

На кожному з етапів декодування відбувається зменшення роздільної здатності вдвічі. Якщо розмір сторони не буде ступенем двійки, на певному етапі кодування кратність буде порушена, і розмірність даних не буде відповідати розмірності шару НМ, що призведе до помилки.

Для обробки зображень інших розмірів їх необхідно розбивати та доповнювати до розмірів 512x512 пікселів:

- Переформатує завантажене зображення в формат моделі, виконує процес підвищення різкості;
- Переформатує результат роботи НМ назад до формату зображення;
- За допомогою скрипта-утиліти розраховує Лапласівська дисперсія для вхідного та вихідного зображень;
- Завантажує вихідне зображення до Minio;
- Формує відповідь для Web-сервісу, включаючи туди id запиту, шлях до вихідного зображення, показники дисперсії для обох зображень;
- У разі виникнення помилки, формує відповідь для Web-сервісу з вказанням помилки.

3.2.2 Скрипти-утиліти

В процесі створення та тренування НМ були розроблені допоміжні скрипти, які знадобились у процесі роботи створеної ІС.

Серед них:

- функція підготовки даних, яка конвертує зображення до розмірів 512x512 пікселів;
- функція розрахунку Laplacian variance засобами opencv.

Додатково була розроблена утиліта командного рядка для підготовки розмитих зображень накладанням Гаусівського розмиття з параметрами, що регулюються.

3.3 Конфігурація мікросервісів

Для забезпечення роботи ІС в мікросервісному режимі було створено декілька мікросервісів:

- sharp-app: контейнер Web-сервісу.
  Включає в себе .jar файл з бекенд частиною ІС. Має можливість конфігурування за допомогою змінних оточення. Побудований на базі контейнеру alpine з достановленням пакету openjdk17-jre. Завдяки тому, що усі залежності окрім JVM в складі .jar файлу, дуже легок в створені та розгортанні. Для автоматизації процесу збірки .jar файлу та образу Docker був розроблений невеликий bash скрипт;
- sharp-worker: контейнер скрипта-обгортки НМ, або воркера.
  Включає в себе основний скріпт main.py, кілька скриптів-утиліт, ваги для розробленої НМ. Образ побудовано на базі python:3.10-slim, також має повну конфігурабельність через змінні оточення. Для коректної роботи мусить встановити бібліотеки pytorch-cpu, python-opencv, pika, numpy, а також декілька нативних пакетів з репозиторія дистрибутиву, щоб забезпечити роботу opencv;
- sharp-db: контейнер бази даних, до якої звертается Web-сервіс.

Використовується образ postgres:latest;

- sharp-minio: контейнер об'єктного сховища minio, у якому зберігають файли запитів та відповідей Web-сервіс та воркер. Використовує образ minio/minio. В сервісі створена одна корзина для обміну зображеннями. Для коректної роботи ІС після запуску контейнеру необхідно зайти до панелі адміністратора minio, створити корзину для файлів та згенерувати ключ доступа. Оскільки неможливо автоматично створити ключі доступів, що унеможливлює автоматичну конфігурацію при старті, процес створення корзини не був автоматизований;
- sharp-rabbitmq: контейнер черги повідомлень, через яку комунікують Web-сервіс та воркер.
  Через нестабільну поведінку останнього релізу, був використаний образ 3.12.1-management-alpine. Після запуску хоча б одного з розроблених контейнерів створить дві черги: чергу запитів (за замовченням request) та відповідей (за замовченням response).

Усі бібліотечні образи отримують свою конфігурацію через змінні середовища, які передаються через docker-compose.
Для спрощення розгортання мікросервісів був створений файл docker-compose.yaml, який надає конфігурацію за замовченням для розгортки на локальній машині.

У скрипті описані необхідні шляхи, порти та локальні томи для коректної роботи ІС. Також до скрипта були додані відкриті порти до хост-машини, які відрізняються від типових портів зазначених сервісів задля запобігання конфліктів. Код docker-compose.yaml, скрипти-утиліти для розгортки та користування ІС, а також код Dockerfile обох розроблених образів див. у додатку Б.

ВИСНОВКИ

При виконанні кваліфікаційної роботи була створена ІС для підвищення різкості зображень на основі мікросервісної архітектури, в основі якої лежить НМ, побудована на ефективній архітектурі U-net.

Швидкість роботи при завантажені моделі на відеокарту в теоретичному наближені достатня для роботи з відео в реальному часі.

Час обробки одного зображення в середньому дорівнює 4 мс, в той час максимальним часом обробки одного кадру в відео при 60 кадрах на секунду є 16 мс.

На першому етапі виконання роботи було проведено тренування великої кількості моделей з метою виявлення оптимальних параметрів моделі та її навчання.

Запропонований метод навчання на базі оператора Canny для виявлення границь зображення виявився менш ефективним при тренуванні моделі на великому датасеті в 500 зображень, що стало неочікуваним результатом, враховуючи кращі показники тренування на малому датасеті.

Як показали порівняльні випробування, найбільш ефективною виявилась функція тренування на базі НМ VGG16.

В процесі випробувань була виявлена схильність моделі до "пікселізації" вихідного зображення, яку вдалось компенсувати введенням нормалізації параметрів в проміжкових шарах НМ.

Збільшення етапу навчання до 40 епох дозволило майже ідентично відтворити показники Лапласової дисперсії у вихідних зображеннях при близьких до максимальних показниках SSIM серед усіх натренованих моделей.

Під час тестування натренованої моделі були помічені ледь помітні артефакти та "пікселізація" на зображеннях з високою кількістю деталей.

Була висунута гіпотеза, що введення перемінного розміра згорткового ядра на перших декількох енкодерах дозволять нейтралізувати цей ефект завдяки захопленю більшої області зображення, що має теоретично можливість покращити результати на зображеннях з великою кількістю щільно згрупованих границь зображення, на яких НМ повинна фокусуватись.

Код динамічно конфігуруемої моделі показав себе дуже ефективним в процесі багаторазового інтенсивного навчання декількох моделей, дозволяючи швидко вносити зміни.

Подальший розвиток цього інструменту може призвести до створення ефективного інструменту для вивчення можливостей архітектури U-net, яка є найбільш ефективною у вирішені задач регресії зображень.

Розгортання цієї моделі на базі гнучкої у розгортанні ІС з мікросервісною архітектурою з доступом по API суттєво розширює коло практичних задач, які можуть вирішувати нейронні мережі для підвищення різкості зображень.

На відміну від закритих для модифікацій веб-сайтів та вузького кола професійного ПЗ, такий підхід дозволяє автоматизувати обробку зображень і використовувати їх для широкого кола задач.

ПЕРЕЛІК ПОСИЛАНЬ

1. Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation URL: https://arxiv.org/abs/1505.04597 (дата звернення:)
2. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition URL: https://arxiv.org/abs/1512.03385 (дата звернення: 18.12.2024)
3. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative Adversarial Networks URL:https://arxiv.org/abs/1406.2661 (дата звернення: 18.12.2024)
4. Federico Sörenson, AMQP vs HTTP URL: https://dev.to/fedejsoren/amqp-vs-http (дата звернення: 19.12.2024)
5. Jim Nilsson, Tomas Akenine-Möller, Understanding SSIM URL: https://arxiv.org/abs/2006.13846 (дата звернення: 19.12.2024)
6. dalle-mini/open-images URL:https://huggingface.co/datasets/dalle-mini/open-images (дата звернення: 18.12.2024)
7. Аврахов М.А. Метод тренування нейромережі, що враховує границі, для підвищення різкості зображень. «Молодь: наука та інновації» 2024: матеріали XII Міжнародної науково-технічної конференції студентів, аспірантів та молодих вчених, Дніпро, 13–15 листопада 2024 року (у 3-х томах) / Національний технічний університет «Дніпровська політехніка» – Дніпро : НТУ «ДП», 2024. Том 2. С. 89-90. URL:https://rmv.nmu.org.ua/ua/arkhiv-zbirok-konferentsiy/molod-nauka-ta-innovatsii-2024/molod-2024-vol2.pdf
8. Unsupervised learning of hierarchical representations with convolutional deep belief networks URL: https://dl.acm.org/doi/10.1145/2001269.2001295 (дата звернення: 19.12.2024)

9. Zhou Wang; A.C. Bovik; H.R. Sheikh; E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity URL: https://ieeexplore.ieee.org/document/1284395 (дата звернення: 19.12.2024)

10. Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization URL: https://arxiv.org/abs/1412.6980 (дата звернення: 19.12.2024)

11. Udi Nachmany, Kubernetes: Evolution Of An IT Revolution URL: https://www.forbes.com/sites/udinachmany/2018/11/01/kubernetes-evolution-of-an-it-revolution/ (дата звернення: 19.12.2024)

1.

ДОДАТОК А
Код динамічної НМ архітектури U-net та її компоненти

```
import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.functional import relu
import torchvision.models as models
import torchvision.transforms as transforms

class DoubleConv(nn.Module):
    def init(self, in_channels, out_channels, use_bn=False):
        super().init()
        self.conv1 = nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels) if use_bn else nn.Identity()
        self.conv2 = nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels) if use_bn else nn.Identity()

    def forward(self, x):
        x = self.bn1(relu(self.conv1(x)))
        x = self.bn2(relu(self.conv2(x)))
        return x

class DownSample(nn.Module):
    def init(self, in_channels, out_channels, use_bn=False):
        super().init()
        self.conv = DoubleConv(in_channels, out_channels, use_bn=use_bn)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        skip = self.conv(x)
        pool = self.pool(skip)
        return skip, pool

class UpSample(nn.Module):
    def init(self, in_channels, out_channels, use_bn=False):
        super().init()
        self.up = nn.ConvTranspose2d(in_channels, in_channels//2, kernel_size=2, stride=2)
        # self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=False)
        self.conv = DoubleConv(in_channels, out_channels, use_bn=use_bn)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        x = torch.cat([x1, x2], 1)
        return self.conv(x)

class SharpenerModelDynamic(nn.Module):
    def init(self, num_layers=5, start_in_channels=64, use_bn=False):
        super().init()
        self.encoder_layers = nn.ModuleList()
        in_channels = 3
        out_channels = start_in_channels
        for i in range(num_layers):
            self.encoder_layers.append(DownSample(in_channels, out_channels, use_bn=use_bn if i != 0 else False))
            in_channels = out_channels
            out_channels *= 2
```

```
        self.bottleneck = DoubleConv(in_channels, out_channels, use_bn=use_bn)

        self.decoder_layers = nn.ModuleList()
        for i in range(num_layers):
            in_channels = out_channels
            out_channels = in_channels // 2
             self.decoder_layers.append(UpSample(in_channels, out_channels, use_bn=use_bn if i != num_layers-1 else
False))

        self.out = nn.Conv2d(in_channels=out_channels, out_channels=3, kernel_size=1)

    def forward(self, x):
        skips = []
        for i, encoder in enumerate(self.encoder_layers):
            skip, pool = encoder(x)
            skips.append(skip)
            x = pool
        skips = skips[::-1]


        x = self.bottleneck(x)
        x = self.decoder_layers[0](x, skips[0])

        for i, decoder in enumerate(self.decoder_layers):
            if i == 0:
                continue
            skip = skips[i]
            x = decoder(x, skip)

        x = self.out(x)
        return x
```

ДОДАТОК Б
скрипти розгортання та експлуатації ІС

Б1 - docker-compose.yaml

```
services:
  app:
    image: sharp-app:latest
    ports:
      # - "8080:8080"
      - "7090:8080"
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://sharp-db:5432/postgres
      - SPRING_DATASOURCE_USERNAME=postgres
      - SPRING_DATASOURCE_PASSWORD=cherepashka123
      - SPRING_JPA_HIBERNATE_DDL_AUTO=update
      - SHARPENER_MINIO_URL=http://sharp-minio:9000/
      - SPRING_RABBITMQ_HOST=sharp-rabbitmq
      - SPRING_RABBITMQ_PORT=5672
  sharp-worker:
    image: "sharp-worker:latest"
    environment:
      - MINIO_HOST=sharp-minio:9000
      - RABBITMQ_HOST=sharp-rabbitmq
      - RABBITMQ_PORT=5672
  sharp-db:
    image: "postgres:latest"
    restart: unless-stopped
    environment:
      - POSTGRES_PASSWORD=cherepashka123
    volumes:
      - volume-pg:/var/lib/postgresql/data
    ports:
      - "5222:5432"
  sharp-minio:
    image: minio/minio
    ports:
      - "9100:9000"
      - "9101:9001"
    volumes:
      - ./minio_storage:/data
    environment:
      - MINIO_ROOT_USER=${MINIO_ROOT_USER:-admin}
      - MINIO_ROOT_PASSWORD=${MINIO_ROOT_PASSWORD:-adminpass}
    command: server --console-address ":9001" /data
  sharp-rabbitmq:
    image: rabbitmq:3.12.1-management-alpine
    hostname: rabbitmq
    restart: unless-stopped
    environment:
      - RABBITMQ_DEFAULT_USER=rmuser
      - RABBITMQ_DEFAULT_PASS=rmpassword
          #       - RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS=-rabbit log_levels [{connection,error},
{default,error}] disk_free_limit 2147483648
    volumes:
      - ./rabbitmq:/var/lib/rabbitmq
    ports:
      - "5572:5672"
      - "15572:15672"
```

```
volumes:
  volume-pg:
    external: false

Б2 - Web Dockerfile
FROM alpine
RUN apk add openjdk17-jre
ARG JAR_FILE=*.jar
COPY ${JAR_FILE} application.jar
ENTRYPOINT ["java", "-jar", "application.jar"]

Б3 - Web build.sh
mvn clean package -DskipTests
cp ./target/*.jar .
docker build . -t sharp-app

Б4 - Worker Dockerfile
FROM python:3.10-slim

WORKDIR /app

RUN apt-get update && apt-get install -y \
    build-essential \
    libatlas-base-dev \
    libopenblas-dev \
    liblapack-dev \
    libx11-dev \
    libgtk-3-dev \
    && rm -rf /var/lib/apt/lists/*

copy main.py /app
copy sharp_model.py /app
copy sharp_utils.py /app
copy sharpener_l5_e40_vgg_bn_100_images_open_data.pth /app
RUN pip install --upgrade pip
RUN pip install numpy
RUN pip install minio
RUN pip install pika
RUN pip install opencv-python

RUN pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu

CMD ["python", "main.py"]


Б5 - blur_image.py
import argparse
from PIL import Image
import numpy as np
from sharp_utils import SharpUtils
import cv2

def apply_gaussian_blur(image_path, output_path, kernel_size, sigma):
    image = SharpUtils.load_image(image_path)
    image_array = np.array(image)
    blurred_image_array = cv2.GaussianBlur(image_array, (kernel_size, kernel_size), sigma)


    cv2.imwrite(output_path, cv2.cvtColor(blurred_image_array, cv2.COLOR_RGB2BGR))
```

```
    print(f"Blurred image saved to {output_path}. Shape: {blurred_image_array.shape}")

def main():
    parser = argparse.ArgumentParser(description="Apply Gaussian blur to an image.")
    parser.add_argument("image_path", help="Path to the input image.")
    parser.add_argument("output_path", help="Path where the output image should be saved.")
    parser.add_argument("--kernel_size", type=int, default=5, help="Kernel size for Gaussian blur.")
    parser.add_argument("--sigma", type=float, default=1.0, help="Sigma value for Gaussian blur.")

    args = parser.parse_args()

    apply_gaussian_blur(args.image_path, args.output_path, args.kernel_size, args.sigma)

if name == "main":
    main()

# python blur_image.py .tmp/25591848-d9dc-481d-a75d-d1f44ecf01f0.jpg ./blur1.jpg --kernel_size=7 --sigma=5


Б6 -
```

ДОДАТОК В
Вихідні коди розробленого Web сервісу

```
package com.master.diploma.sharpener.configuration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.minio.MinioClient;

/**
 * MinioConfiguration
 */
@Configuration
public class MinioConfiguration {

  @Bean
  public MinioClient minioClient(
      @Value("${sharpener.minio.url}") String url,
      @Value("${sharpener.minio.public.key}") String publicKey,
      @Value("${sharpener.minio.private.key}") String privateKey) {
    return MinioClient.builder().endpoint(url)
        .credentials(publicKey, privateKey).build();
  }

}
package com.master.diploma.sharpener.configuration;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQConfiguration {

  private String exchangeName;
  private String requestQueueName;
  private String responseQueueName;
  private String requestRoutingKey;
  private String responseRoutingKey;

  public RabbitMQConfiguration(
      @Value("${sharpener.exchange.name:exchange}") String exchangeName,
      @Value("${sharpener.request.queue.name:request}") String requestQueueName,
      @Value("${sharpener.response.queue.name:response}") String responseQueueName,
      @Value("${sharpener.request.routing.key:request-key}") String requestRoutingKey,
      @Value("${sharpener.response.routing.key:response-key}") String responseRoutingKey) {
    this.exchangeName = exchangeName;
    this.requestQueueName = requestQueueName;
    this.responseQueueName = responseQueueName;
    this.requestRoutingKey = requestRoutingKey;
    this.responseRoutingKey = responseRoutingKey;
```

```
  }

  @Bean
  public Queue requestQueue() {
    return new Queue(requestQueueName, true);
  }

  @Bean
  public Queue responseQueue() {
    return new Queue(responseQueueName, true);
  }

  @Bean
  public DirectExchange exchange() {
    return new DirectExchange(exchangeName);
  }

  @Bean
  public Binding requestBinding(@Qualifier("requestQueue") Queue requestQueue, DirectExchange exchange) {
    return BindingBuilder.bind(requestQueue).to(exchange).with(requestRoutingKey);
  }

  @Bean
  public Binding responseBinding(@Qualifier("responseQueue") Queue responseQueue, DirectExchange exchange)
{
    return BindingBuilder.bind(responseQueue).to(exchange).with(responseRoutingKey);
  }

}
package com.master.diploma.sharpener.controller;

import java.io.IOException;
import java.io.InputStream;
import java.util.Date;
import java.util.List;

import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestPart;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;

import com.master.diploma.sharpener.entity.image.Image;
import com.master.diploma.sharpener.entity.image.ImageResponseDto;
import com.master.diploma.sharpener.entity.image.ImageSubmitDto;
import com.master.diploma.sharpener.entity.submission.Submission;
import com.master.diploma.sharpener.service.FileService;
import com.master.diploma.sharpener.service.ImageService;
import com.master.diploma.sharpener.service.SubmissionService;

/**
 * SharpeningController
 */
```

```
@RestController
public class SharpeningController {

  private ImageService imageService;
  private SubmissionService submissionService;
  private FileService fileService;

  public SharpeningController(ImageService imageService, SubmissionService submissionService,
      FileService fileService) {
    this.imageService = imageService;
    this.submissionService = submissionService;
    this.fileService = fileService;
  }

  @GetMapping(path = "/test")
  public String check() {
    return "Hi! Sharpening backend is running! " + new Date();
  }

  @PostMapping("/image")
      public ResponseEntity<ImageResponseDto> createImage(@RequestPart("file") MultipartFile file) throws
IOException {
    Image image = imageService.createFromFile(file);
    return ResponseEntity.ok(new ImageResponseDto(image));
  }

  @PostMapping("/submit")
  public ResponseEntity<Submission> sharpenImage(@RequestBody ImageSubmitDto submit) {
    return ResponseEntity.ok(submissionService.submitImage(submit.getId()));
  }

  @GetMapping("/submission")
  public ResponseEntity<List<Submission>> getAllSubmissions() {
    return ResponseEntity.ok(submissionService.getAll());
  }

  @GetMapping("/submission/{submissionId}")
  public ResponseEntity<Submission> getSubmission(@PathVariable String submissionId) {
    System.out.println("IN");
    return ResponseEntity.ok(submissionService.getById(submissionId));
  }

  @GetMapping("/image")
  public ResponseEntity<List<Image>> getAllImages() {
    return ResponseEntity.ok(imageService.getAllImages());
  }

  @GetMapping("/image/{imageId}")
  public ResponseEntity<byte[]> downloadImage(@PathVariable String imageId) throws IOException {
    Image image = imageService.getById(imageId);
    InputStream fileStream = fileService.downloadImage(image.getFile().getPath());
    byte[] fileBytes = fileStream.readAllBytes();
    return ResponseEntity.ok()
        .contentType(MediaType.APPLICATION_OCTET_STREAM)
        .header(HttpHeaders.CONTENT_DISPOSITION,
            "attachment; filename=\"" + image.getFile().getFilename() + "\"")
        .body(fileBytes);
  }
```

```
}
package com.master.diploma.sharpener.controller;

import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.master.diploma.sharpener.entity.file.File;
import com.master.diploma.sharpener.entity.file.FileStatus;
import com.master.diploma.sharpener.entity.image.Image;
import com.master.diploma.sharpener.entity.submission.Submission;
import com.master.diploma.sharpener.entity.submission.SubmissionResult;
import com.master.diploma.sharpener.entity.submission.SubmissionStatus;
import com.master.diploma.sharpener.service.FileService;
import com.master.diploma.sharpener.service.ImageService;
import com.master.diploma.sharpener.service.SubmissionService;

@Component
public class SharpenResultController {

 private SubmissionService submissionService;
 private ImageService imageService;
 private FileService fileService;

 public SharpenResultController(SubmissionService submissionService, ImageService imageService,
    FileService fileService) {
  this.submissionService = submissionService;
  this.imageService = imageService;
  this.fileService = fileService;
 }

 @RabbitListener(queues = "response")
 public void receiveSharpeningResult(String messageJson) {
  ObjectMapper objectMapper = new ObjectMapper();
  SubmissionResult result;
  try {
   System.out.println(messageJson);
   result = objectMapper.readValue(messageJson, SubmissionResult.class);
  } catch (JsonProcessingException e) {
   System.out.println("An Exception has been thrown!");
   return;
  }
  Submission submission = submissionService.findById(result.getSubmissionId());
  if (submission == null) {
   System.out.println("Looks like submission clould not be found! Aborting handler.");
   return;
  }
  if (result.getIsError()) {
   submission.setStatus(SubmissionStatus.ERROR);
   submissionService.updateSubmission(submission);
  }

  File file = new File(result.getOutputPath(), result.getOutputPath(), FileStatus.UPLOADED);
  file = fileService.createExisting(file);
  Image image = new Image(file);
  image = imageService.createFromExistingFile(image);
  submission.setOutput(image);
```

```
    submission.setStatus(SubmissionStatus.COMPLETED);
    submissionService.updateSubmission(submission);
  }

}
package com.master.diploma.sharpener.entity.file;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

/**
 * File
 */
@Entity
public class File {

  @Id
  @GeneratedValue(strategy = GenerationType.UUID)
  private String id;
  @Column
  private String filename;
  @Column
  private String path;
  @Column
  private FileStatus status;

  public File() {
  }

  public File(String filename, String path, FileStatus status) {
    this.filename = filename;
    this.path = path;
    this.status = status;
  }

  public String getId() {
    return id;
  }

  public String getFilename() {
    return filename;
  }

  public void setFilename(String filename) {
    this.filename = filename;
  }

  public String getPath() {
    return path;
  }

  public void setPath(String path) {
    this.path = path;
  }
```

```
  public FileStatus getStatus() {
    return status;
  }

  public void setStatus(FileStatus status) {
    this.status = status;
  }
}
package com.master.diploma.sharpener.entity.file;

/**
 * FileStatus
 */
public enum FileStatus {
  CREATED,
  UPLOADED
}
package com.master.diploma.sharpener.entity.image;

import com.master.diploma.sharpener.entity.file.File;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;

/**
 * Image
 */
@Entity
public class Image {

  @Id
  @GeneratedValue(strategy = GenerationType.UUID)
  private String id;
  @OneToOne
  @JoinColumn(name = "file_id")
  private File file;
  @Column
  private double laplacianVariance;

  public Image() {
  }

  public Image(File file) {
    this.file = file;
  }

  public String getId() {
    return id;
  }

  public void setId(String id) {
    this.id = id;
  }
```

```
  public File getFile() {
    return file;
  }

  public void setFile(File file) {
    this.file = file;
  }

  public double getLaplacianVariance() {
    return laplacianVariance;
  }

  public void setLaplacianVariance(double laplacianVariance) {
    this.laplacianVariance = laplacianVariance;
  }
}
package com.master.diploma.sharpener.entity.image;

/**
 * ImageSubmitDto
 */
public class ImageSubmitDto {

  private String id;

  public ImageSubmitDto() {
  }

  public String getId() {
    return id;
  }

  public void setId(String id) {
    this.id = id;
  }
}
package com.master.diploma.sharpener.entity.submission;

import com.master.diploma.sharpener.entity.image.Image;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;

@Entity
public class Submission {

  @Id
  @GeneratedValue(strategy = GenerationType.UUID)
  private String id;
  @Column
  private SubmissionStatus status;
  @ManyToOne
```

```
  @JoinColumn(name = "input_id", unique = false)
  private Image input;
  @ManyToOne
  @JoinColumn(name = "output_id", unique = false)
  private Image output;

  public Submission() {
  }

  public Submission(SubmissionStatus status, Image input, Image output) {
    this.status = status;
    this.input = input;
    this.output = output;
  }

  public String getId() {
    return id;
  }

  public void setId(String id) {
    this.id = id;
  }

  public SubmissionStatus getStatus() {
    return status;
  }

  public void setStatus(SubmissionStatus status) {
    this.status = status;
  }

  public Image getInput() {
    return input;
  }

  public void setInput(Image input) {
    this.input = input;
  }

  public Image getOutput() {
    return output;
  }

  public void setOutput(Image output) {
    this.output = output;
  }

}
package com.master.diploma.sharpener.entity.submission;

public class SubmissionResult {

  private String submissionId;
  private String outputPath;
  private boolean isError;

  public SubmissionResult() {
  }
```

```
  public String getSubmissionId() {
    return submissionId;
  }

  public void setSubmissionId(String submissionId) {
    this.submissionId = submissionId;
  }

  public String getOutputPath() {
    return outputPath;
  }

  public void setOutputPath(String minioPath) {
    this.outputPath = minioPath;
  }

  public boolean getIsError() {
    return isError;
  }

  public void setIsError(boolean isError) {
    this.isError = isError;
  }

  @Override
  public String toString() {
      return "SubmissionResult [submissionId=" + submissionId + ", outputPath=" + outputPath + ", isError=" +
isError
        + "]";
  }

}
package com.master.diploma.sharpener.entity.submission;

public enum SubmissionStatus {
  CREATED,
  SENT,
  COMPLETED,
  ERROR
}
package com.master.diploma.sharpener.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.master.diploma.sharpener.entity.file.File;

/**
 * FileRepository
 */
@Repository
public interface FileRepository extends CrudRepository<File, String> {
}
package com.master.diploma.sharpener.repository;

import org.springframework.data.repository.CrudRepository;
```

```
import com.master.diploma.sharpener.entity.image.Image;

/**
 * ImageRepository
 */
public interface ImageRepository extends CrudRepository<Image, String> {



}
package com.master.diploma.sharpener.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.master.diploma.sharpener.entity.submission.Submission;

@Repository
public interface SubmissionRepository extends CrudRepository<Submission, String>{
}
package com.master.diploma.sharpener.service;

import java.io.IOException;
import java.io.InputStream;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import org.springframework.stereotype.Service;

import com.master.diploma.sharpener.entity.file.File;
import com.master.diploma.sharpener.entity.file.FileStatus;
import com.master.diploma.sharpener.repository.FileRepository;

import io.minio.GetObjectArgs;
import io.minio.MinioClient;
import io.minio.PutObjectArgs;
import io.minio.errors.ErrorResponseException;
import io.minio.errors.InsufficientDataException;
import io.minio.errors.InternalException;
import io.minio.errors.InvalidResponseException;
import io.minio.errors.ServerException;
import io.minio.errors.XmlParserException;

/**
 * FileService
 */
@Service
public class FileService {

  private FileRepository repository;
  private MinioClient client;

  public FileService(FileRepository fileRepository, MinioClient client) {
    this.repository = fileRepository;
    this.client = client;
  }

  public File saveFile(String name, InputStream stream, long size) {
    File file = new File(name, null, FileStatus.CREATED);
```

```
    file = repository.save(file);
    System.out.println(file.getFilename());
    System.out.println(file.getFilename().lastIndexOf('.'));
    String objectName = file.getId() + file.getFilename().substring(file.getFilename().lastIndexOf('.'));
    try {
      client.putObject(
          PutObjectArgs.builder().object(objectName).bucket("images").stream(stream, size, -1).build());
    } catch (Exception e) {
      e.printStackTrace();
      throw new RuntimeException(e);
    }
    file.setPath(objectName);
    file.setStatus(FileStatus.UPLOADED);
    file = repository.save(file);
    return file;
  }

  public File createExisting(File file) {
    return repository.save(file);
  }

  public InputStream downloadImage(String path) {
    InputStream fileStream;
    try {
      fileStream = client.getObject(
          GetObjectArgs.builder()
              .bucket("images")
              .object(path)
              .build());
    } catch (InvalidKeyException | ErrorResponseException | InsufficientDataException | InternalException
        | InvalidResponseException | NoSuchAlgorithmException | ServerException | XmlParserException
        | IllegalArgumentException | IOException e) {
      e.printStackTrace();
      return null;
    }
    return fileStream;
  }
}
package com.master.diploma.sharpener.service;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.server.ResponseStatusException;

import com.master.diploma.sharpener.entity.file.File;
import com.master.diploma.sharpener.entity.image.Image;
import com.master.diploma.sharpener.repository.ImageRepository;

/**
 * ImageService
 */
@Service
```

```
public class ImageService {

  private ImageRepository repository;
  private FileService fileService;

  public ImageService(ImageRepository repository, FileService fileService) {
    this.repository = repository;
    this.fileService = fileService;
  }

  public Image createFromFile(MultipartFile multipartFile) throws IOException {
    File file = fileService.saveFile(multipartFile.getOriginalFilename(), multipartFile.getInputStream(),
        multipartFile.getSize());
    Image image = new Image(file);
    image = repository.save(image);
    return image;
  }

  public Image createFromExistingFile(Image image) {
    return repository.save(image)  ;
  }

  public Image getById(String imageId) {
    Optional<Image> image = repository.findById(imageId);
    if (image.isEmpty()) {
      throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Image with id " + imageId + "not found!");
    }
    return image.get();
  }

  public List<Image> getAllImages() {
    List<Image> all = new ArrayList<>();
    repository.findAll().iterator().forEachRemaining(all::add);
    return all;
  }

}
package com.master.diploma.sharpener.service;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Service;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.master.diploma.sharpener.entity.submission.Submission;

/**
 * MessageService
 */
@Service
public class MessageService {

  private RabbitTemplate template;

  public MessageService(RabbitTemplate template) {
    this.template = template;
  }
```

```
  private void sendMessage(String message, String topic) {
    System.out.println("Sending to" + topic);
    template.convertAndSend("exchange", topic, message);
  }

  public void sendSubmission(Submission submission) {
    ObjectMapper mapper = new ObjectMapper();
    String message;
    try {
      message = mapper.writeValueAsString(submission);
    } catch (JsonProcessingException e) {
      throw new RuntimeException(e);
    }
    sendMessage(message, "request-key");
  }

}
package com.master.diploma.sharpener.service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import com.master.diploma.sharpener.entity.image.Image;
import com.master.diploma.sharpener.entity.submission.Submission;
import com.master.diploma.sharpener.entity.submission.SubmissionStatus;
import com.master.diploma.sharpener.repository.SubmissionRepository;

@Service
public class SubmissionService {

  private SubmissionRepository repository;
  private ImageService imageService;
  private MessageService messageService;

  public SubmissionService(SubmissionRepository repository, ImageService imageService,
      MessageService messageService) {
    this.repository = repository;
    this.imageService = imageService;
    this.messageService = messageService;
  }

  public Submission submitImage(String imageId) {
    Image image = imageService.getById(imageId);
    Submission submission = new Submission(SubmissionStatus.CREATED, image, null);
    submission = repository.save(submission);
    messageService.sendSubmission(submission);
    submission.setStatus(SubmissionStatus.SENT);
    submission = repository.save(submission);
    return submission;
  }

  public List<Submission> getAll() {
    List<Submission> listAll = new ArrayList<>();
```

```
    repository.findAll().iterator().forEachRemaining(listAll::add);
    return listAll;
  }

  public Submission findById(String submissionId) {
    Optional<Submission> submission = repository.findById(submissionId);
    if (submission.isPresent()) {
      return submission.get();
    } else {
      return null;
    }
  }

  public Submission getById(String submissionId) {
    Submission submission = findById(submissionId);
    if (submission == null) {
      throw new ResponseStatusException(HttpStatus.NOT_FOUND);
    }
    return submission;
  }

  public Submission updateSubmission(Submission submission){
    Submission current = getById(submission.getId());
    current = repository.save(submission);
    return current;
  }

}
package com.master.diploma.sharpener;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SharpenerApplication {

  public static void main(String[] args) {
    SpringApplication.run(SharpenerApplication.class, args);
  }
}
application.properties.yaml

spring.application.name=Sharpener
spring.datasource.url=jdbc:postgresql://127.0.0.1:5222/postgres
spring.datasource.username=postgres
spring.datasource.password=cherepashka123
spring.jpa.hibernate.ddl-auto=update

sharpener.minio.url=http://localhost:9100/
sharpener.minio.bucket=images
sharpener.minio.public.key=1jBO51odWa9ctWqufolO
sharpener.minio.private.key=98fKb8JjBJVRh4ie32AmItFQAAQuFVXl4Qzkkckh

sharpener.exchange.name=exchange
sharpener.request.queue.name=request
sharpener.response.queue.name=response
sharpener.request.routing.key=request-key
sharpener.response.routing.key=response-key
```

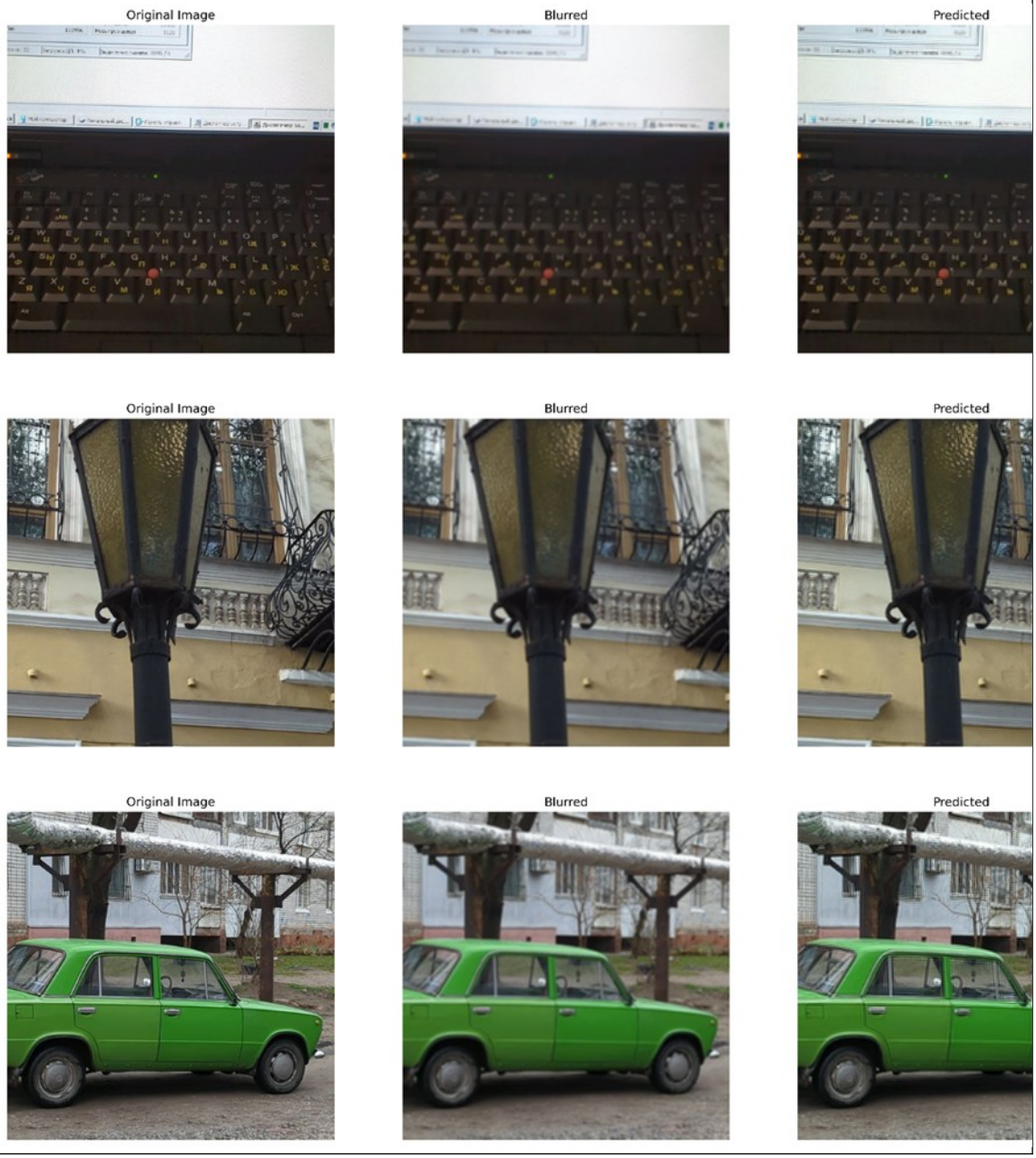
```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5572
spring.rabbitmq.username=rmuser
spring.rabbitmq.password=rmpassword
pom.xml

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.4.0</version>
  <relativePath/> <!-- lookup parent from repository -->
 </parent>
 <groupId>com.master.diploma</groupId>
 <artifactId>Sharpener</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <name>Sharpener</name>
 <description>Backend for image sharpening diploma project</description>
 <url/>
 <licenses>
  <license/>
 </licenses>
 <developers>
  <developer/>
 </developers>
 <scm>
  <connection/>
  <developerConnection/>
  <tag/>
  <url/>
 </scm>
 <properties>
  <java.version>17</java.version>
 </properties>
 <dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-amqp</artifactId>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <scope>runtime</scope>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.amqp</groupId>
    <artifactId>spring-rabbit-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
      <groupId>io.minio</groupId>
      <artifactId>minio</artifactId>
      <version>8.5.2</version>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
  </plugins>
 </build>

</project>
```

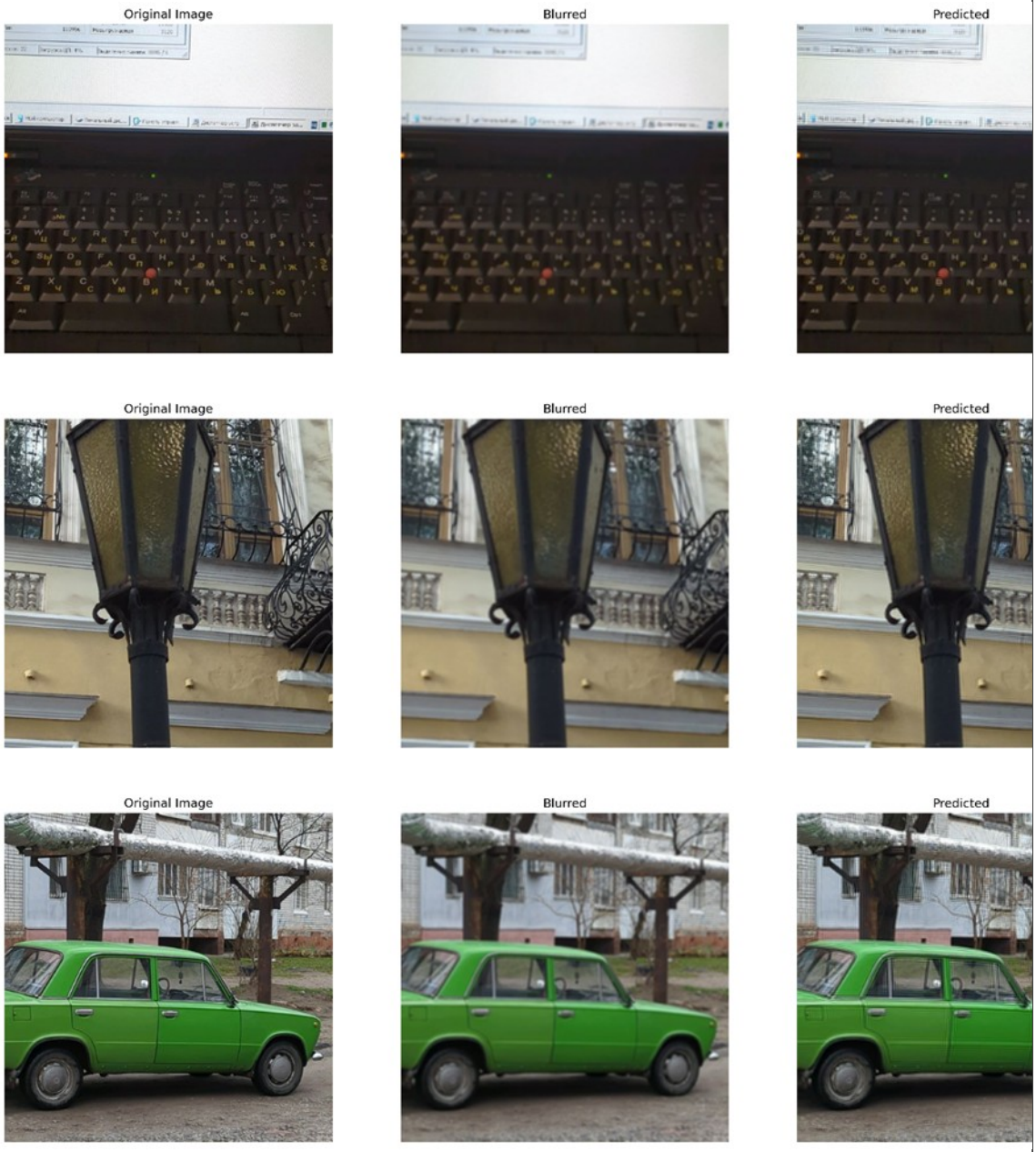
ДОДАТОК Г

результати роботи навчених НМ на зображеннях різної складності

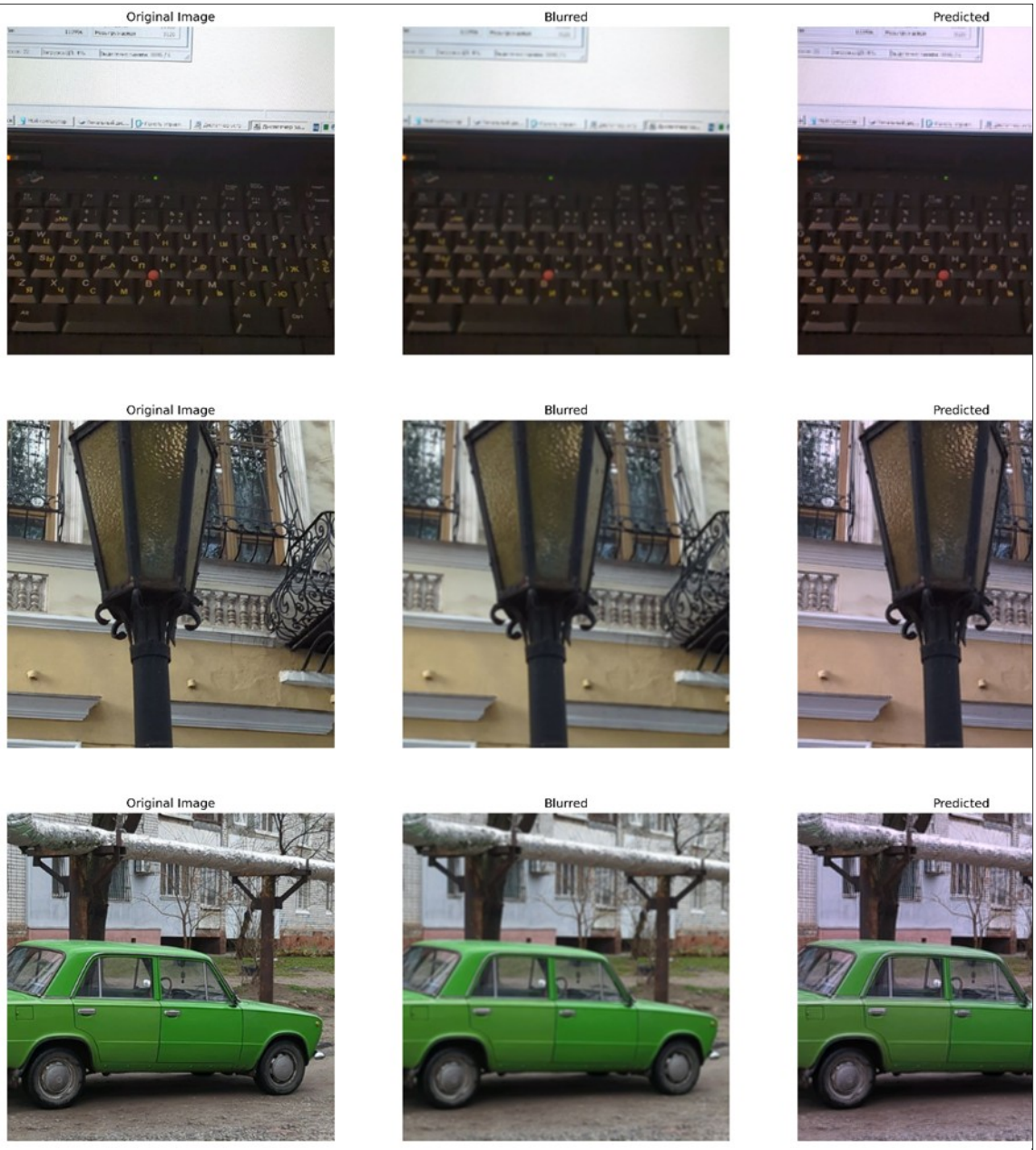
Результат роботи НМ MSE, 3 шари, 20 епох



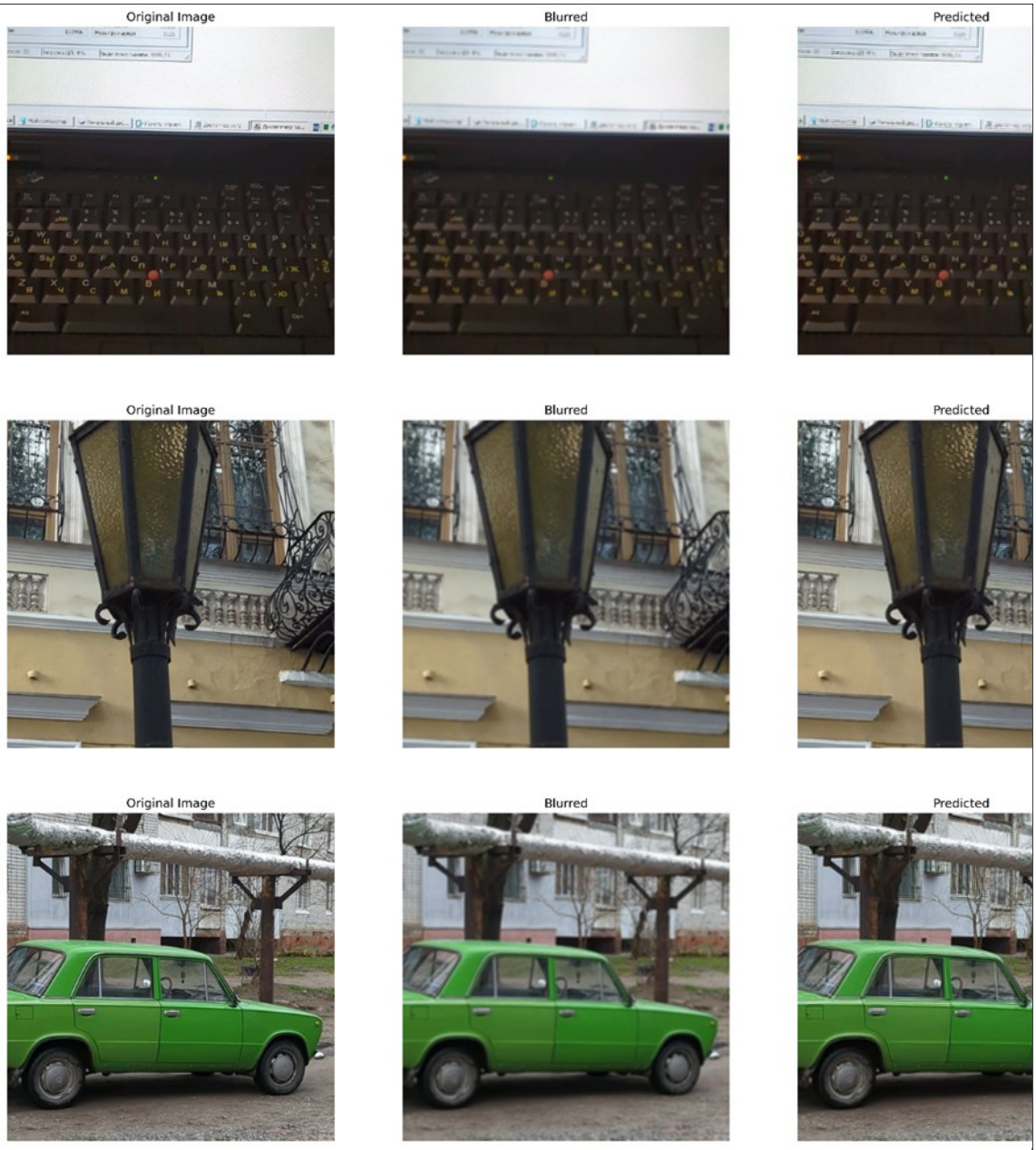
Результат роботи НМ MSE, 4 шари, 20 епох



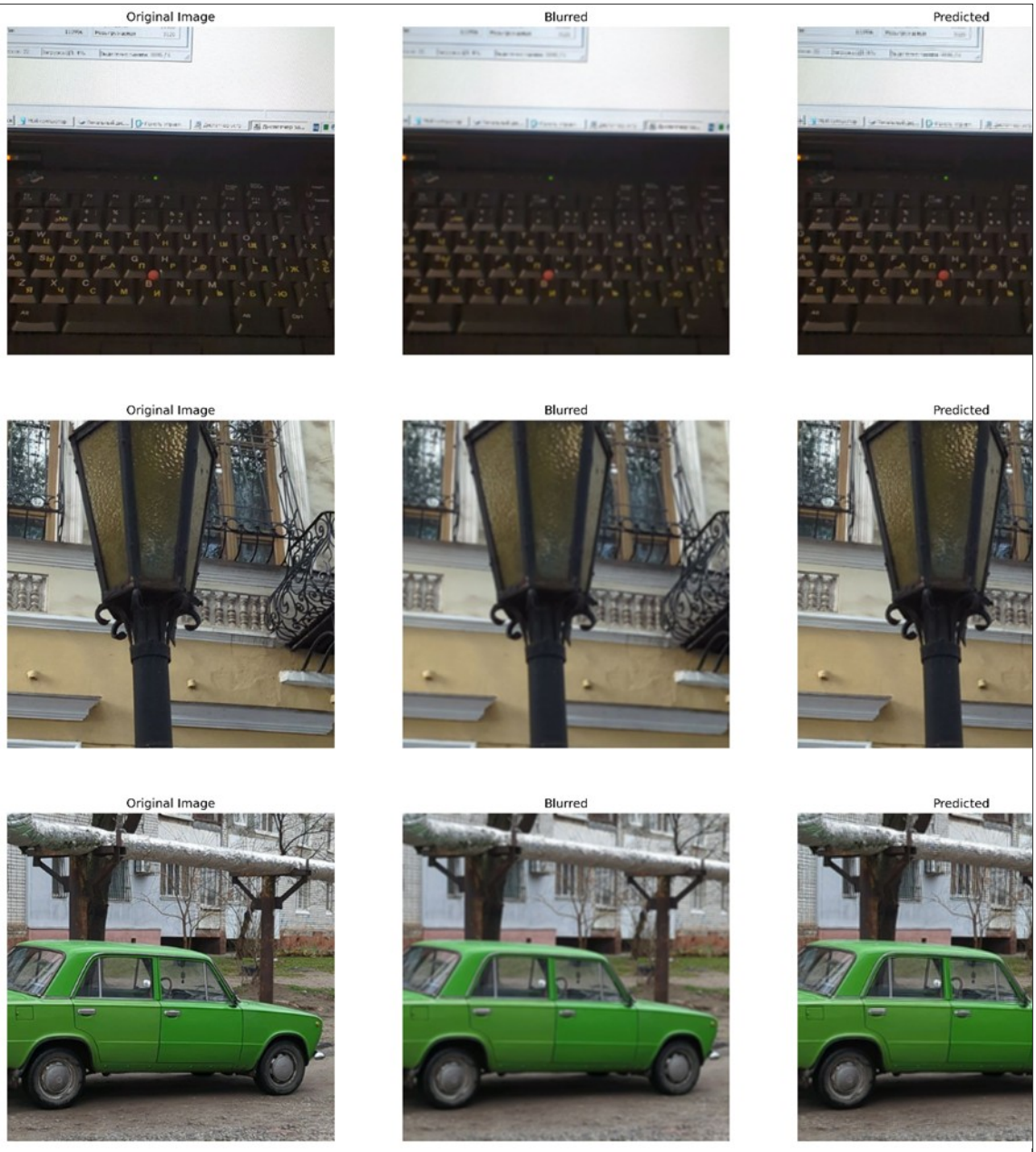
Результат роботи НМ MSE, 5 шарів, 20 епох



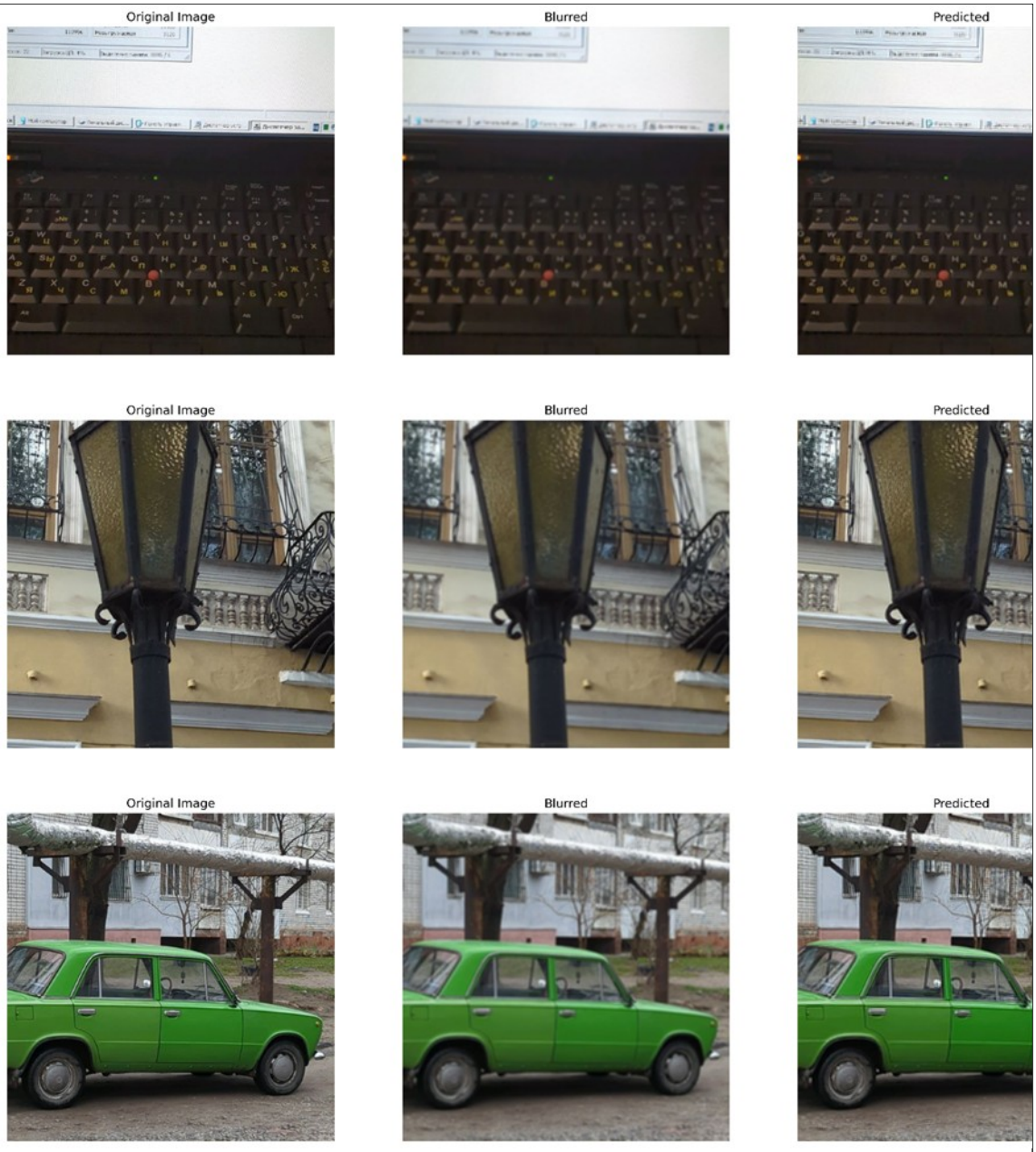
Результат роботи НМ MSE_CANNY, 5 шарів, 20 епох



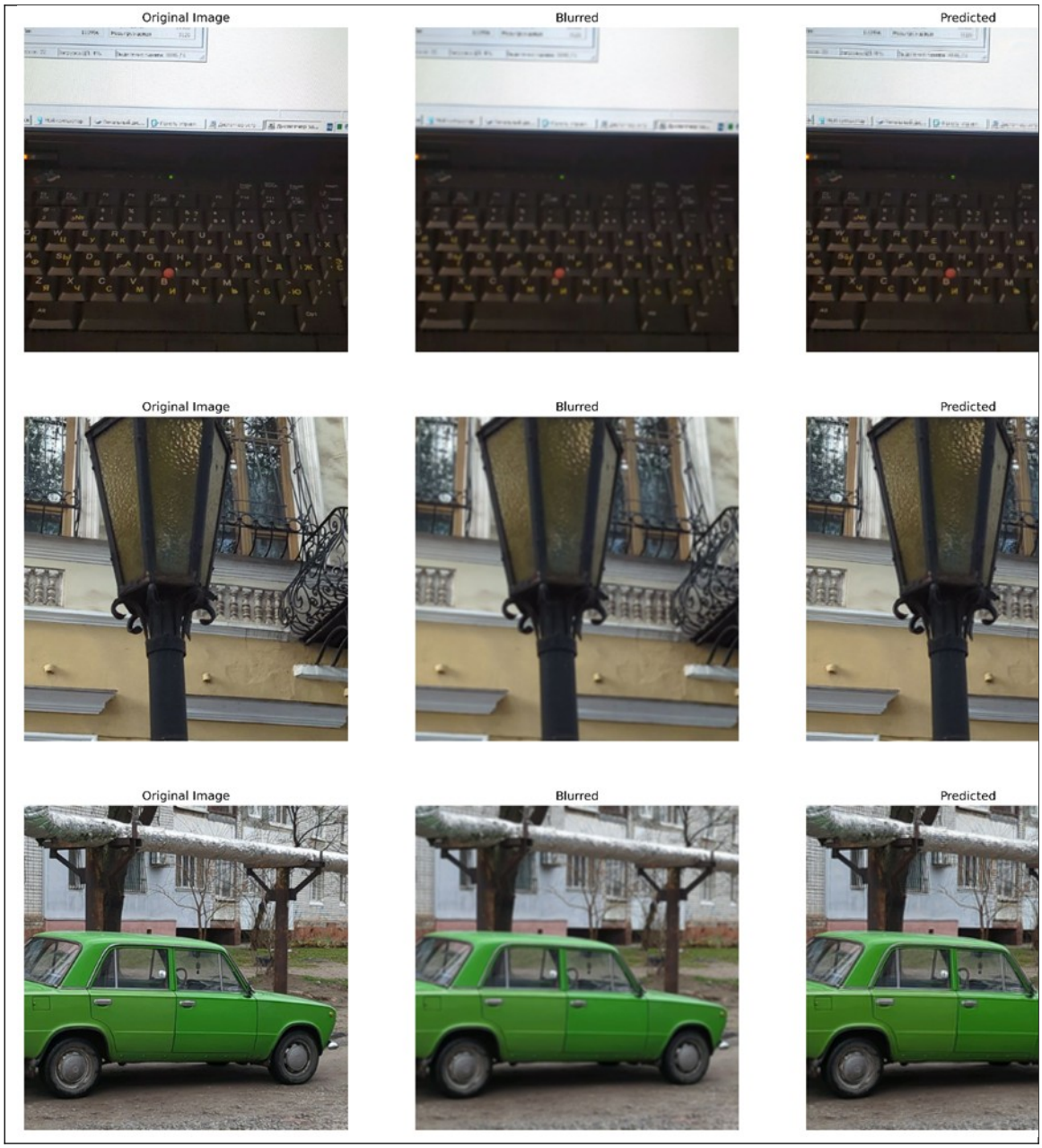
Результат роботи НМ VGG, 5 шарів, 20 епох



Результат роботи НМ VGG_BN, 20 епох



Результат роботи НМ VGG_BN, 40 епох


Original Image
Blurred
Predicted
Original Image
Blurred
Predicted
Original Image
Blurred
Predicted

ДОДАТОК Д
код тренування НМ

```
import os
import sys
import cv2
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
import numpy as np

import torch
import torch.optim as optim
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
import torch.multiprocessing as mp

from torch.nn.functional import relu
from worker.sharp_utils import SharpUtils
from worker.sharp_model import SharpenerModelDynamic, VGG16FeatureExtractor

import requests
from PIL import Image
from io import BytesIO
import torch
from torchvision import transforms
import datasets as ds
import random
import time

from skimage.metrics import structural_similarity as ssim
import torchvision.models as models

def is_valid_url(url):
    try:
        response = requests.get(url, timeout=10)
        return response.status_code == 200
    except requests.exceptions.RequestException:
        return False

def process_image(image_bytes, target_size=(512, 512)):
    img = Image.open(BytesIO(image_bytes)).convert("RGB")
    width, height = img.size
    img.thumbnail(
        (
            width if width > height else target_size[0],
            height if height > width else target_size[1]
        )
    )

    width, height = img.size
    center_w, center_h = width // 2, height // 2

    left = center_w - target_size[0] // 2
    right = center_w + target_size[0] // 2
    top = center_h - target_size[1] // 2
    bottom = center_h + target_size[1] // 2

    img = img.crop((left, top, right, bottom))

    return img
```

```
dataset_path = './dataset1/'
ods_path = '/home/jc/hdd/diploma/dataset/'
img1_path = './dataset1/photo_18_2024-11-07_19-14-58.jpg'

# mp.set_start_method('spawn', force=True)
def load_dataset(path, device = 'cpu'):
    images = []
    edges_arr = []
    blurred_images = []

    for img in os.listdir(path):
        if img.endswith('.jpg'):
            img_path = os.path.join(path, img)
            image = SharpUtils.load_image(img_path)


            img_edges = cv2.Canny(image, 100, 200)

            img_edges = cv2.GaussianBlur(img_edges, (3,3), 1.5)
            img_edges[img_edges != 0] += 255 - np.max(img_edges)

            # _, img_edges = cv2.threshold(img_edges, .25, 1, cv2.THRESH_BINARY)


            edges_arr.append(img_edges)

            blurred = cv2.GaussianBlur(image, (5,5), 1.5)
            images.append(image)
            blurred_images.append(blurred)

    return ImageDataset(np.array(images), np.array(blurred_images), np.array(edges_arr), device)



class ImageDataset(Dataset):
    def init(self, images, blurred_images, image_edges, device='cpu'):
        self.images = torch.tensor(images.astype(np.float32) / 255.0).permute(0,3,1,2).to(device)
        self.blurred_images = torch.tensor(blurred_images.astype(np.float32) / 255.0).permute(0,3,1,2).to(device)
        self.image_edges = torch.tensor(image_edges / 255.0).to(device)
        self.device = device

    def len(self):
        return len(self.images)

    def getitem(self, idx):
        # print(f'Getting {idx}')
        # image = self.images[idx].astype(np.float32) / 255.0
        # image = torch.tensor(image).permute(2, 0, 1)


        # image_edge = self.image_edges[idx].astype(np.float32) / 255.0
        # image_edge = torch.tensor(image_edge)


        # blurred_image = self.blurred_images[idx].astype(np.float32) / 255.0
        # blurred_image = torch.tensor(blurred_image).permute(2, 0, 1)
        # return blurred_image.to(self.device), image.to(self.device), image_edge.to(self.device)
        return self.blurred_images[idx], self.images[idx], self.image_edges[idx]
batch_size = 1
```

```
# dataset = ImageDataset(images, blurred_images, edges_arr)
dataset = load_dataset(dataset_path, 'cuda')
ods = load_dataset(ods_path, 'cuda')


dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

Колюня, [20.12.2024 8:46]
# ods = OpenImageDataset(d, train_indexes_np[0:200])
openLoader = DataLoader(ods, batch_size=batch_size, shuffle=True)
# , shuffle=True, num_workers=8, pin_memory=True, prefetch_factor=4
def l2_loss_edges(prediction, target, edges, a1 = 1.0):
    loss = prediction - target
    square_loss = loss * loss
    altered_loss = square_loss * (edges.unsqueeze(1).expand(-1, 3, -1, -1) * a1 + 1)
    return torch.mean(altered_loss)

def l2_loss(prediction, target):
    return F.mse_loss(prediction, target)

vgg = VGG16FeatureExtractor(img_rows=512, img_cols=512, weights="imagenet", inference_only=True).cuda()

def l2(y_true, y_pred):
    if y_true.ndimension() == 4:
        return torch.mean(torch.abs(y_pred - y_true), dim=[1, 2, 3])
    elif y_true.ndimension() == 3:
        return torch.mean(torch.abs(y_pred - y_true), dim=[1, 2])
    else:
        raise NotImplementedError("L2 loss calculation on 1D tensors is not supported for this network.")

def loss_perceptual(pred, real):
    loss = l2(pred, real)
    return loss


layers = 5
use_bn= True
model = SharpenerModelDynamic(num_layers=layers, use_bn=use_bn).cuda()
optimizer = optim.Adam(model.parameters(), lr=0.0002)

%%time

pure = 'pure_mse'
edged = 'edged_mse'
edged_v2 = 'edged_v2_mse'
vgg = 'vgg'
loss_type = vgg

# Training loop
num_epochs = 40
loader = openLoader

for epoch in range(num_epochs):
    print(f'-------Epoch {epoch+1}-------')
    model.train()
    # running_loss = 0.0
    running_loss = torch.tensor([0.0], device='cuda')
    validation_loss_mse = torch.tensor(0.0, device='cuda')
```

```
    validation_loss_edges = torch.tensor([0.0], device='cuda')
    epoch_start = time.time()
    for i, (inputs, targets, edges) in enumerate(loader):
        pass_start = time.time()
        t = time.time()
        # targets, edges =targets.cuda(non_blocking=True), edges.cuda(non_blocking=True)
        # print(torch.max(edges))
        optimizer.zero_grad()
        outputs = model(inputs)
        outputs_time = time.time() - t
        t = time.time()
            loss = l2_loss(outputs, targets) if loss_type == pure else l2_loss_edges(outputs, targets, edges, 0.25) if loss_type == edged_v2 else loss_perceptual(outputs, targets)
        # loss = l2_loss_edges(outputs, targets, edges)
        # loss = loss_perceptual(outputs, targets)
        # print(loss.shape)
        loss_time = time.time() - t


        t = time.time()
        loss.backward()
        optimizer.step()
        opti_time = time.time() - t
        t = time.time()

        running_loss += loss.detach() # += loss.item()
        pass_end = time.time()
        if (i + 1) % 25 == 0:
                print(f'Pass {i + 1}/{len(loader)} Loss: {loss}, time: {((pass_end - pass_start)*1000):.3f} Forward Pass Time: {(outputs_time*1000):.3f}, Loss Function Time: {(loss_time*1000):.3f}, Optimizer Time: {(opti_time*1000):.3f}')
    with torch.no_grad():
        for i, (inputs, targets, edges) in enumerate(dataloader):
            # inputs, targets, edges = inputs.cuda(), targets.cuda(), edges.cuda()
            outputs = model(inputs)
            loss = loss_perceptual(outputs, targets)
            validation_loss_edges += loss
            loss = l2_loss(outputs, targets)
            validation_loss_mse += loss


    if (epoch + 1) % 5 == 0:
        checkpoint = {
            'epoch': epoch + 1,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
        }
        torch.save(checkpoint, f'/home/jc/hdd/diploma/sharpener_l{layers}_e{epoch+1}_{loss_type}{"_bn_" if use_bn else "_"}100_images_open_data.pth')
        print(f'Saved checkpoint epoch {epoch+1}')
    epoch_end = time.time()
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {(running_loss.item() * 1000)/len(loader)} V_Loss_mse: {(validation_loss_mse.item() * 1000)/len(dataloader)} V_Loss_Vgg: {(validation_loss_edges.item() * 1000)/len(dataloader)} Epoch time: {(epoch_end - epoch_start)}")
```