МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко
підпис

«_____» _____20____ р.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ, СТВОРЕНИХ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність «Інженерія програмного забезпечення»

121 – КРМ.1 – 608м.21810826

Здобувач вищої освіти _____В. Ю. Фіник
підпис
«__» _____ 20__ р.

Керівник PhD, ст. викладач _____ І. О. Кандиба
підпис
«__» _____ 20__ р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко
«____»_____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

Видано студенту групи 608м факультету комп'ютерних наук
Фінику Владиславу Юрійовичу
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи
«Програмне забезпечення пошуку дефектів на зображеннях, створених засобами штучного інтелекту
Затверджена наказом по ЧНУ від «10» листопада 2023 р. № 234
2. Строк представлення кваліфікаційної роботи «____» _____ 2022 р.
3. Очікуваний результат роботи та початкові дані, якщо такі потрібні:
Очікуваний результат роботи: програмне забезпечення пошуку дефектів на зображеннях, створених засобами штучного інтелекту.
Початкові дані: зображення з дефектами та без, створені за допомогою нейронних мереж.
4. Перелік питань, що підлягають розробці
- огляд прикладної сфери;
- огляд вже існуючих аналогів систем, призначених для пошуку дефектів;
- дослідження алгоритмів обробки зображень та пошуку дефектів;
- вибір інструментів для реалізації застосунку;
- програмна реалізація застосунку.

5. Перелік графічних матеріалів
Презентація
6. Завдання до спеціальної частини
_____

_____

_____

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
.		

_____

Керівник роботи PhD, ст. викладач Кандиба Ігор Олександрович
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання
Фіник Владислав Юрійович_____
(прізвище, ім'я, по батькові студента)

_____
(підпис)

Дата видачі завдання «____» _____ 2023 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Програмне забезпечення пошуку дефектів на зображеннях, створених засобами штучного інтелекту»

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Затвердження завдання на виконання КРМ	08.10.2023	10.11.2023	Виконано
2.	Огляд літератури за темою роботи	20.11.2023	05.12.2023	Виконано
3.	Складання календарного плану КРМ	06.12.2023	07.12.2023	Виконано
4.	Аналіз існуючих аналогів	09.12.2023	14.12.2023	Виконано
5.	Вибір інструментів для реалізації застосунку	05.01.2024	07.01.2024	Виконано
6.	Дослідження алгоритмів пошуку дефектів	09.01.2024	18.01.2024	Виконано
7.	Пошук/генерація масиву зображень	10.01.2024	24.01.2024	Виконано
8.	Застосування алгоритмів для пошуку та оцінки дефектів	26.01.2024	01.02.2024	Виконано
9.	Тестування та формування документації	02.02.2024	05.02.2024	Виконано
10.	Попередній захист	08.02.2024	08.02.2024	Виконано
11.	Відгук керівника КРМ	10.02.2024	12.02.2024	Виконано
12.	Рецензія	13.02.2024	15.02.2024	Виконано
13.	Оформлення КРМ та презентації	17.02.2024	20.02.2024	Виконано
14.	Захист кваліфікаційної роботи	26.02.2024	28.02.2024	

Розробив студент Фіник Владислав Юрійович_____ _____
(*прізвище, ім'я, по батькові студента)* *(підпис)*

«____»_____ 2023 р.

Керівник роботи: PhD, ст. викладач Кандиба Ігор Олександрович_____
(*прізвище, ім'я, по батькові)* (*підпис)*

«____»_____ 2023 р.

АНОТАЦІЯ

до кваліфікаційної роботи магістра

«Програмне забезпечення пошуку дефектів на зображеннях, створених засобами штучного інтелекту»

Студент 608м гр.: Фіник Владислав Юрійович

Керівник: PhD, ст. викладач Кандиба І. О.

Кваліфікаційна робота магістра присвячена дослідженню алгоритмів та методів для пошуку дефектів на зображеннях, створених засобами штучного інтелекту.

Об'єкт дослідження – процес пошуку дефектів на зображеннях.

Предмет дослідження – методи та засоби пошуку дефектів на зображеннях з використанням комп'ютерного зору.

Мета – вдосконалення процесу пошуку дефектів на зображеннях створених штучним інтелектом шляхом розробки програмного забезпечення на базі інструментарію комп'ютерного зору.

Для досягнення мети було виконано наступні завдання:

1) проведено аналіз предметної області та існуючих аналогів;
2) обрано інструменти для виконання роботи;
3) згенеровано вибірки даних для подальшого тренування;
4) реалізовано відображення знайдених дефектів.

У першому розділі було розглянуто ймовірні дефекти на зображеннях та аналоги. У другому розділі проведено математичний аналіз моделей обробки зображень. У третьому розділі наводяться необхідні інструменти та описується архітектура застосунку. В четвертому розділі наведено кроки реалізації застосунку, тестування, а також реалізований інтерфейс застосунку.

В результаті виконаної роботи було проаналізовано методи аналізу дефектів та створено ПЗ для пошуку їх на зображеннях, створених засобами штучного інтелекту.

КРМ викладена на 62 с. (без додатків), містить 4 розділи, 26 рис., 1 додаток, 20 джерел в переліку посилань.

***Ключові слова**: штучний інтелект, комп'ютерне бачення, великі дані, кластеризація, нормалізація зображень, матричні операції.*.

ABSTRACT

of the Master’s Thesis

"Software for defects finding in images created by artificial intelligence"

Student of group 608m: Finyk Vladyslav Yuriiovych

Supervisor: PhD, senior lecturer Kandyba I. O.

The Master's Thesis is devoted to the study of algorithms and methods for defects finding in images created by artificial intelligence,.

The object of research is the process of finding defects in images.

The subject of research is methods and tools for finding defects in images using computer vision.

The goal is to study computer vision methods for creating software for finding defects in images created by artificial intelligence.

To achieve the goal, the following tasks were performed:

1) analyzed the subject area and existing analogues;
2) tools for performing the work were selected;
3) generated data samples for further training;
4) displaying the detected defects.

The first section considered probable defects in images and software analogues. The second section provides a mathematical analysis of image processing models. The third section presents the necessary tools and describes the application architecture. The fourth section describes the steps of application implementation, testing, and the implemented application interface.

As a result of the work performed the methods of defect analysis were analysed and created software for finding defects in images created by artificial intelligence.

The Master’s Thesis contains 62 pages (without appendices), 26 images, 1 appendix, 20 sources in the reference list.

Keywords: artificial intelligence, computer vision, big data, clustering, image normalisation, matrix operations.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

ШІ – Штучний інтелект

AI – Artificial intelligence

BSON – Binary JSON

CNN – Convolutional Neural Networks

JSON – JavaScript Object Notation

R-CNN – Region-based Convolutional Neural Networks

RoI – Region of Interest

YOLO – You Look Only Once

ВСТУП

2023 рік став роком стрімкого розвитку та впровадження різноманітних застосунків і сервісів побудованих на засобах штучного інтелекту та моделях глибокого машинного навчання. Їх використання стало не лише доступним і дешевим, а й необхідним для подальшого розвитку деяких галузей. Будучи інженером програмного забезпечення, використання таких інструментів як GitHub Copilot починає ставати необхідністю, оскільки це збільшує швидкість роботи в рази та дозволяє використовувати кращі практики одразу без подальшого рефакторингу.

В цій роботі будуть розглядатись генеративні інструменти ШІ, що створюють цифрові зображення з описів природною мовою. Існують декілька актуальних інструментів на сучасному ринку, а саме: Stable Diffusion, Midjourney та DALL-E. В усіх них є певні специфіки й відмінності у використанні та головне, що всі вони доволі стрімко розвиваються. Тому вивчення механізмів їх роботи для подальшого покращення кінцевих результатів є актуальним.

Об’єкт дослідження – процес пошуку дефектів на зображеннях.

Предмет дослідження – методи та засоби пошуку дефектів на зображеннях з використанням комп’ютерного зору.

Мета – вдосконалення процесу пошуку дефектів на зображеннях створених штучним інтелектом шляхом розробки програмного забезпечення на базі інструментарію комп’ютерного зору.

Для досягнення мети необхідно виконати наступні завдання

– провести аналіз предметної області та існуючих аналогів;
– обрати інструменти для подальшої роботи;
– використати вибірки даних для подальшого тренування;
– реалізувати відображення знайдених дефектів.

Ніколи раніше не було так просто творити щось як зараз: необхідно лише мати телефон чи персональний комп’ютер і доступ до інтернету. Проте результати

часто бувають незадовільні: погана композиція, проблеми з частинами тіл або ж оточення, нереалістичність чи зображення просто не співпадає з написаним запитом. Саме тому збільшення позитивних кінцевих результатів є актуальним, оскільки за генерацією великої кількості зображень, які можуть мати дефекти, стоїть декілька проблем:

1) час отримання результату – оскільки необхідно буде переглянути часто великі масиви зображень, збільшуються затримка між початком роботи та її опублікуванням чи використанням, що негативно впливає на прибутки, а також ризик буквально не помітити присутню помилку через людський фактор;

2) обчислювальні потужності – дана проблема має сильну кореляцію з часом, оскільки при слабкому апаратному забезпеченні затримка збільшується в декілька разів;

3) фінанси – більшість сервісів працюють по платній підписці, надаючи певний час обчислювальної роботи або ж кількість кінцевих зображень, тому зменшення дефектних результатів зменшить витрати в цілому;

Зображення, що створені за допомогою ШІ уже зараз використовуються в багатьох галузях, основними з яких є:

1) 3д-моделювання – створення ескізів моделей або ж їх текстурування стає значно простішим і швидшим;

2) анімація – створення проміжних кадрів дозволяє значно збільшити плавність мультфільмів або ж ігор [1].

Тому в даній роботі планується розроблення ПЗ з пошуку дефектів на зображеннях та дослідження можливості його використання для пошуку дефектів на зображеннях, створених засобами штучного інтелекту.

Робота пройшла апробацію під час Всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів “Методи і засоби програмної інженерії”.

1 ДОСЛІДЖЕННЯ ПРОЦЕСУ ПОШУКУ ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ

1.1 Процес створення зображення за допомогою ШІ

Задля кращого розуміння можливих дефектів та концентрації на певних з них необхідним є розбір самого процесу створення зображень з використанням наявних ШІ. Midjourney та DALL-E можна об’єднати в одну категорію, оскільки єдине що вони надають це кінцеве зображення за запитом. Єдине як можна з ними працювати це додавати вагу в ключові слова, наприклад, *(curly_hair:1.2)* – це вказує нейронній мережі, що дану величину вона має інтегрувати більше, таким чином можна регулювати рівень кучерявості тощо. За замовчуванням вони видають набір з 4-х зображень серед яких можна вибрати ті які є більш підходящими. Приклад можна побачити на рис. 1.1 де зображено версії логотипу для музичного фестивалю:

Рисунок 1.1 – Варіації логотипу

Тому для подальшого розбору буде використано Stable Diffusion, оскільки він надає значно більший вибір налаштувань. Основними з них є:

1) checkpoint – попередньо підготовлені вагові показники моделі SD, що необхідні для створення загальних або специфічних зображень (реалістичних, анімаційних, об’єднаних певною тематикою тощо);

2) sampling method – метод зменшення шумів на зображенні, пов’язаний зі специфікою їх створення [2];

3) sampling steps – кількість кроків за які модель трансформує початковий шум в зображення;

4) restore faces – відновлення обличь, спотворених шумом;

5) CFG (Classifier Free Guidance) scale – контроль відповідності запиту (менші значення дають більшу свободу, більші змушують дотримуватись його);

6) seed – зерно, або ж число що використовується для ініціації генерації.

Це основні налаштування які впливають на кінцевий результат зображення. Всі вони доступні в різних інтерфейсах для SD, таких як ComfyUI, що зображено на рис. 1.2, чи AUTOMATIC1111, який можна переглянути на рис. 1.3.

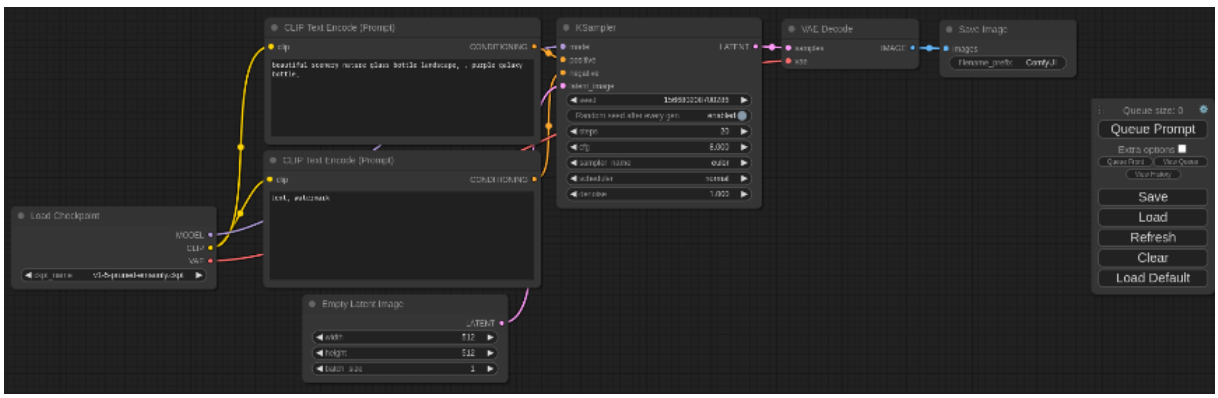


Рисунок 1.2 – Інтерфейс ComfyUI

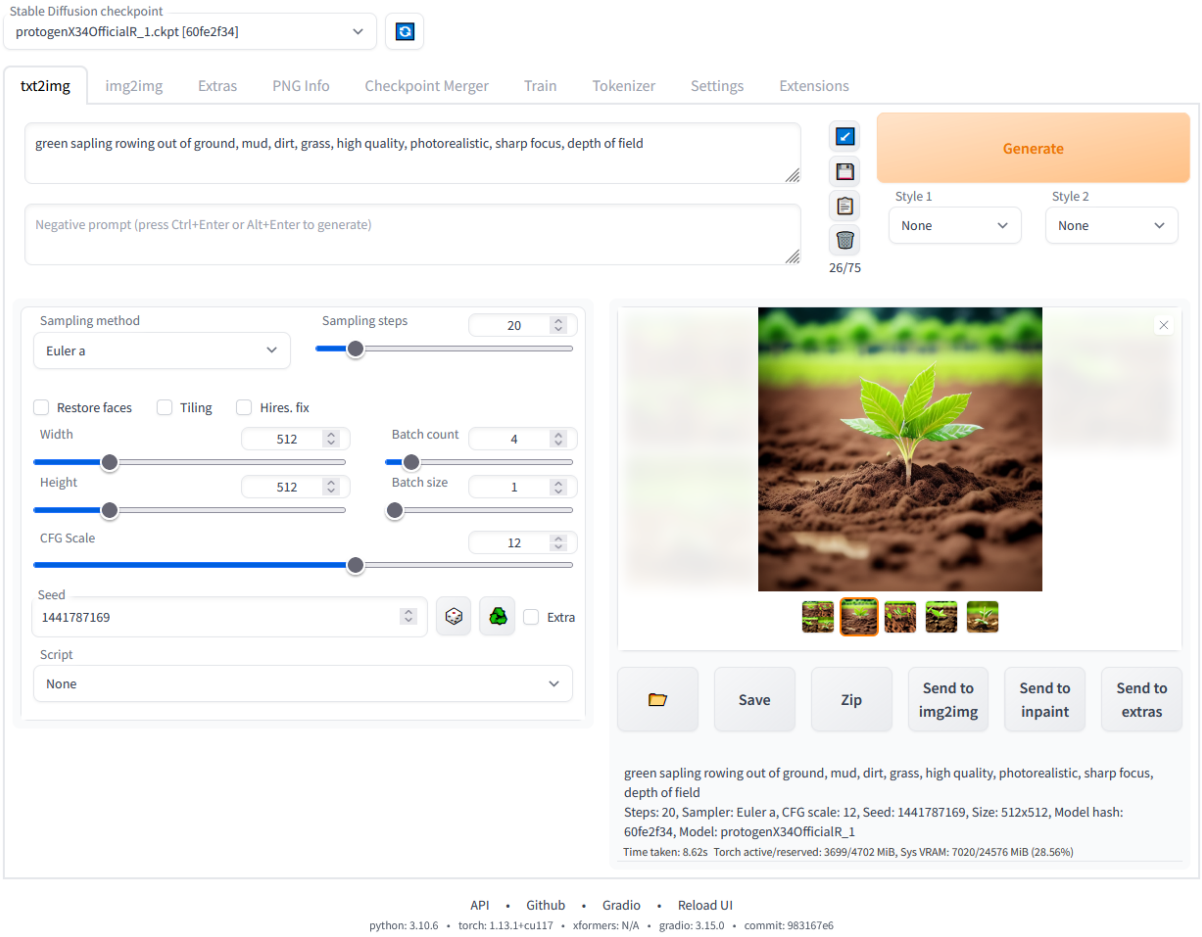


Рисунок 1.3 – Інтерфейс AUTOMATIC1111

Також важливим аспектом є запит, в який необхідно записати текст або ключові аспекти зображення, та негативний запит, який виконує зворотну функцію, а саме відкидає від генерації вказане. Використання даних механізмів разом з вагами може значно покращити кінцевий результат, але необхідно дотримуватись також і оптимальних значень налаштувань, які було вказано вище. Їх підбір є доволі індивідуальною справою, оскільки залежить від моделі, запиту та зерна.

Сучасні генератори зображень представляють з себе дифузійні моделі, на відміну від перших генеративних загальних мереж (Generative Adversarial Networks), які використовували декілька нейронних мереж одночасно: одна виробляла зображення, а інша оцінювала точність результату. Кожна з них навчалась за допомогою змагального навчання, під час якого завданням генератора

було переконати дискримінатора в тому, що зображення, які були вироблені, є реальними. Дифузійні моделі працюють трохи інакше: вони навчаються на мільярдних наборах даних, де кожне зображення має підпис і опис, завдяки чому модель може витягнути контекстні знання про те, як ці слова пов’язані безпосередньо з зображенням. Одним з основних аспектів є те, що дифузійна модель не копіює існуючі елементи чи компоненти зображень, а натомість розробляє кожен елемент з нуля, хоча доволі цікавим є прецедент, що відбувається зараз з Midjourney V6 [3], де зображення майже точно співпадають з кадрами фільмів. Це може бути спричинено навчанням на тих же наборах даних для збільшення продуктивності, але сама ситуація є доволі цікавою і може призвести до судових позовів, а сама сфера ШІ нині є погано визначеною в правовому полі.

1.2 Дефекти на зображеннях, створених ШІ

Через те що зображення створюються з шуму і послідовно перетворюються в кінцевий результат на проміжних етапах часто можуть виникати помилки, які призводять до певних артефактів. Такі дефекти можна класифікувати за різними типами.

Візуальні дефекти

Непропорційність – будівлі чи частини тіла можуть бути з сильно помітною асиметрією або ж спотвореними пропорціями. Наприклад, занадто довга шия, невідповідність розміру рук відносно тіла або ж будівлі з кривими каркасними конструкціями.

Злиття композиції – при об’єднанні різних елементів в межах зображення, особливо якщо їх чимало, ШІ може зливати ці об’єкти між собою, створюючи таким чином химерні переходи.

Позиції рук – пальці теж є доволі складним об’єктом для створення, оскільки необхідно враховувати фізику та біологічні можливості їх рухів, тому часто на

зображеннях можна побачити пальці викривленими або ж в неприродньому для себе жесті.

Аномалії волосся – оскільки зачіски мають багато деталей, при чому часто дрібних, їх генерація стає великим викликом для ШІ, через що регіонах з волоссям часто присутні дефекти на кшталт відокремлених пасм чи дивних форм.

Текст – найбільш помітним і поширеним дефектом є нечитабельний текст, оскільки він часто являє собою набір символів які не схожі на сучасну мову і сильно вирізняється з усього контексту.

Сюрреалістичні елементи

Вигадані об’єкти – в дану класифікацію можна віднести багато чого: від синіх собак до екранів моніторів з зубами, проте більш важливими є непомітні аспекти, наприклад колір дзьоба у пташки, яка виглядає реалістичною, але при цьому не має нічого спільного з реальністю.

Ефект порожніх очей – при зміщенні погляду людиною на зображенні часто можна помітити його штучність, оскільки в погляді відчувається певна порожнеча, що при перегляді реалістичної композиції може викликати ефект «Незвичайної долини».

Малоймовірні комбінації – наприклад, фіолетові або червоні очі хоч і є результатом певних генетичних спадковостей альбінізму, проте зустрічаються надзвичайно рідко.

Композиція

Освітлення – часто певні частини зображення є занадто або недостатньо освітленими, відображення працюють не так як повинні згідно законів фізики або ж розсіювання світла майже немає, що викликає відчуття штучності.

Текстури – зображення шкіри чи волосся часто може бути занадто гладким або ж навпаки нерівномірним (морщини), мати недостатньо недоліків чи навпаки бути надзвичайно бездоганним.

Вказаний вище список недоліків може бути неповним, оскільки існує дуже багато різних артефактів які необхідно класифікувати інакше, проте він є вичерпним. Те, що може не здаватись дефектом насправді може вказувати на штучність зображення, що в певних випадках може бути корисним, проте тенденція створення зображень прямує до зменшення розриву між якість створеного контенту людиною та за допомогою ШІ.

1.3 Огляд та аналіз наявних аналогів

Існує певний прошарок застосунків і систем що надають функціонал з пошуку дефектів. Більшість з них пристосовані для виробництва і шукають різні мікро тріщини, нерівності тощо в матеріалах, наприклад, металевих плитах. У сфері генерованого ШІ контенту подібних аналогів знайдено не було, проте близькими за змістом є різні застосунки з розпізнавання та сегментації зображень.

oPRO.ai Image defect recognition

oPRO.ai – це компанія з розробки різних ШІ для впровадження їх в промислових масштабах так звана Industry 4.0. Серед доступних модулів наявний модуль для розпізнавання дефектів на зображеннях [4].

Переваги:

1) точність – за ствердженнями компанії вони змогли підвищити точність успішних знаходжень проблемних ділянок на 30%;

2) швидкість – відносно людських спостережень продукт використовує значно менше часу;

3) персоналізація – можливість імплементації власних ШІ/МН рішень розроблена таким чином, щоб навіть користувачі з обмеженими технічними знаннями могли розробити та розгорнути моделі для власних сфер діяльності.

Недоліки:

1) закритість – задля попереднього перегляду можливостей необхідно контактувати безпосередньо з командою;

2) обмеженість – враховуючи сферу виробництва, існує небагато рішень саме з пошуку дефектів, тому користувачам доведеться власноруч розробляти певні аспекти.

Google Lens

Google Lens – це технологія розпізнавання зображень, призначена для отримання відповідної інформації, пов’язаної з об’єктами, які ідентифікуються за допомогою візуального аналізу на основі нейронної мережі.

Переваги:

1) багатозадачність – в залежності від контексту можуть виконуватись різні задачі: розв’язок рівнянь, переклад тексту чи пошук схожих товарів в магазинах [5];

2) швидкість;

3) простота використання – все що необхідно, встановити застосунок і не виконувати ніяких налаштувань тощо.

Недоліки:

1) точність – часто результати є непідходящими чи взагалі знаходитись поза контекстом, наприклад, при пошукові схожого одягу з фото на ринку;

2) необхідний постійний доступ до інтернету.

Ximilar: Image Recognition

Ximilar – програмна компанія, яка створює рішення для інтелектуальної обробки даних на основі комп’ютерного зору та машинного навчання [6]. Серед них найбільш підходящим в контексті даної роботи є рішення Custom Visual Search: для звичайних типів зображень, на кшталт декору або стокових фотографій уже наявні певні моделі, проте якщо необхідно щось більш унікальне, то вони реалізують модель для специфічних вимог.

Переваги:

1) універсальність;

2) швидкість;

3) пошук та рейтинг подібних продуктів– розроблені рішення чудово підходять для пошуку продуктів, наприклад одягу, посуду тощо.

Серед недоліків можна виділити те, що необхідний постійний доступ до інтернету.

Як можна побачити з наданої вище інформації, на ринку присутні рішення як з пошуку дефектів, так і з розпізнавання, класифікації та сегментації зображень, проте жодне з них не працює з контентом, створеним ШІ. При роботі з ним необхідно враховувати різні аспекти, наприклад, розпізнавання дефектів частин тіла людини є складним для реалізації на звичайних системах розпізнавання зображень. Для цього необхідно враховувати зовнішні аспекти, такі як запит, модель тощо.

1.4 Специфікація вимог до створюваного програмного забезпечення

Призначення та межі проєкту

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Система призначена для обробки зображень, створених засобами штучного інтелекту, задля подальшої їх сегментації і пошуку дефектів, які могли виникнути при генерації.

Погодження, що ухвалені в програмній документації

Було погоджено, що для роботи з даними будуть використовуватись бібліотеки OpenCV, PyTorch та Scikit-Image, оскільки вони містять в собі інструменти для спрощеної роботи з зображеннями.

Межі проєкту ПЗ

Проєкт має реалізовувати функції, що були узгоджені при проєктуванні ПЗ та вирішувати головні недоліки застосунків-аналогів, а саме: відкритість вихідного коду, можливість використання без доступу до інтернету або з ним, але на віддалених серверах та специфікація під особливості дефектів.

Загальний опис

Сфера застосування

Створюваний проєкт має використовуватись медіа-творцями, пересічними користувачами та різного роду медіа.

Загальна структура і склад системи

Система має складатись з прошарку для роботи з зображеннями, інтерфейсу користувача, та, в майбутньому, прошарку переконструювання запитів дефектних зображень в більш оптимізовані з меншою кількістю вихідних похибок. Вся система має мати відкритий вихідний код для спрощення подальшої розробки та інтеграції розширень.

Вимоги до інформаційного забезпечення

Джерела і зміст вхідної інформації (даних)

Джерелами є відкриті масиви зображень, а також згенеровані зображення з дефектами.

Вимоги до способів організації, збереження та ведення інформації

Має бути проведено дослідження з вибору оптимального способу збереження великої кількості зображень, наприклад HDF5 чи нереляційної бази даних.

Вимоги до програмного забезпечення

Архітектура програмної системи

Програмна система має складатися з двох ланок: клієнтський застосунок та прошарок роботи з зображеннями.

Мова і технологія розробки ПЗ

Мовою розробки було обрано Python (механізми роботи з зображеннями), а для клієнтського інтерфейсу React та Node.js.

Вимоги до зовнішніх інтерфейсів

Інтерфейс користувача

Користувацький інтерфейс має задовольняти стандартні UI та UX вимоги.

Властивості програмного забезпечення

Доступність

Від ПЗ вимагається доступність для будь-якого користувача, в якого є наявні бібліотеки та дані, що можуть оброблятись системою.

Супроводжуваність

Комплекс застосунків має бути легко супроводжуваним та придатним для подальшої модифікації чи доповнення.

Переносимість

Програмне забезпечення має бути кросплатформним. Клієнтські застосунки можуть бути використані будь-яким ПК.

Надійність

Створюваний проєкт має належним чином оброблювати помилки та припиняти свою роботу у разі виникнення нефатальних помилок та демонструвати їх у сприятливій для людського читання формі. Потенційно небезпечні для системи помилки не мають бути показані користувачам.

Безпека

Оскільки даним програмним забезпеченням на момент першого етапу розробки не передбачаються елементи реєстрації користувачів або ж надсилання їх даних на зовнішній сервер, безпекових характеристик визначено не було. При наступних ітераціях розробки даний пункт може бути змінено.

Висновки до розділу 1

В результаті написання першого розділу кваліфікаційної роботи магістра було наведено основну інформацію, що необхідна для більш повного та поглибленого розуміння зазначеної теми.

Надано детальну інформацію щодо створення зображення за допомогою ШІ, а саме параметри, які є необхідними або важливими для використання, властивості, за які вони відповідають, а також загальні рекомендації щодо їх використання.

Проведено аналіз можливих дефектів та детально розписано їх ключові характеристики. Серед них було обрано ключові, з якими буде проводитись робота в подальшому, а саме дефекти рук та очей.

Проаналізовано основні аналоги що в певній мірі реалізовують функціонал розроблюваної системи або ж є подібними, але при цьому мають низку критично важливих недоліків, вирішення яких є однією з задач дипломної роботи.

Сформовано специфікацію вимог до розроблюваного програмного забезпечення задля більш чіткого представлення та опису його поведінки. В ній наведено функціональні і нефункціональні вимоги.

2 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПОШУКУ ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ, СТВОРЕНИХ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Математичні моделі обробки зображень

Сингулярний розклад матриці

Оскільки для тренування нейронної мережі, яка б змогла визначити що є дефектом, а що ні необхідно мати величезні масиви зображень, доцільним є зменшення їх розміру без втрати якості. В дослідженні коледжу LaGrange [7] на прикладі хоч і доволі примітивного, але кольорового зображення, було продемонстровано, що зменшення кількості сингулярних значень з 574 до 100 для кожного шару кольору призвело до зменшення кількості займаної пам’яті вдвічі, при цьому сама ілюстрація залишилась доволі якісною. Тому можна провести дослідження з тренування декількох моделей: першу – на оригінальних зображеннях, другу – на стиснених і третю – на комбінованих. Після цього перевірити якість кінцевого результату розпізнавання, і якщо він буде відрізнятись в межах 2-3%, то використовувати оброблені за допомогою СРМ.

Афінні інваріантні області

Афінні інваріанти – це властивості, які залишаються незмінними під час афінного перетворення. Афінне перетворення – це відображення площини або простору в собі, при якому паралельні прямі переходять в паралельні прямі, пересічні – в пересічні, мимобіжні – в мимобіжні. В комп’ютерній графіці вони часто використовуються для перетворення об’єктів, наприклад, повороту, стиснення, розтягування тощо. Різні зображення одного і того ж об’єкта можуть відрізнятись між собою і проходити різні перетворення в залежності від змін у камері та її налаштуваннях, освітленні тощо, що в контексті обраної теми визначається композицією зображення. Розроблено новий підхід, заснований на афінних інваріантних областях, що дозволило одночасно сегментувати та

розпізнавати об’єкти на зображеннях [8]. Для пошуку дефектів це є критично важливим, оскільки зменшення кількості обробок зображення для кожного виду вад ілюстрацій значно збільшить продуктивність.

Поступовий Байєсівський підхід

Через складність реалізації поступового тренування, існує лише невелика кількість моделей які дотримуються даного підходу. Це є обмеженням для нейронної мережі, оскільки тренування для декількох категорій стає значно складнішим. При використанні генеративної ймовірнісної моделі, що представляє форму та зовнішній вигляд сукупності ознак, параметри моделі вивчаються поступово Байєсівським способом [9]. Це дозволяє використовувати меншу кількість вхідних масивів даних та завдяки інкрементному навчанню значно пришвидшує процес.

Використання наведених математичних моделей необхідно для формування якісної вибірки даних, яка буде оптимізована за розміром та якістю, що має позитивно вплинути на подальший процес навчання нейронної мережі. Якщо ж результати будуть відрізнятись від очікуваних, це стане приводом для більш глибокого дослідження даних моделей і аналізу першопричин.

2.2 Математичні моделі кластеризації зображень

Метод k–середніх

Це популярний алгоритм машинного навчання, що використовується для розбиття набору точок даних на K окремих підгруп чи кластерів, що не перетинаються [10]. Його метою є групування схожих точок і виявлення основних закономірностей в даних. Алгоритм складається з двох різних фаз: в першій розраховуються k центроїдів, а в другій відносить кожну точку до кластера, який має найближчий центроїд до відповідної точки даних.

Серед його сильних переваг можна визначити те, що він є доволі простий в реалізації, проте існують також і певні недоліки: кінцевий результат є сильно

залежним від обраних початкових центроїдів, тому якщо вони обираються випадковим чином, то результати можуть відрізнятись в залежності від їх розміщення. Тому в такому складному завданні як сегментація зображень він є неоптимальним, оскільки необхідно враховувати велику кількість контекстів та об’єктів.

Спектральна кластеризація

Даний метод побудований на концепції зв’язності графів, що дозволяє поєднувати їх в більш складні мережі і, на відміну від методу k-середніх, знаходити кластери довільної форми, а не опуклої [11]. Також він є значно гнучкішим у представленні даних, оскільки проєктуючи їх на підпростір, спектральна кластеризація гарантує, що схожі точки знаходяться близько одна до одної, а несхожі – далеко. Цей метод можна використовувати в різних областях: від сегментації зображень, що є необхідним згідно обраної теми, до категоризації документів чи виявлення аномалій, що також є великою перевагою.

Проте в нього існують й певні недоліки, основними з яких є:

1) продуктивність – обчислення власних векторів матриці Лапласіана може вимагати значної потужності, особливо з великими наборами даних;

2) чутливість до параметрів – вибір кількості кластерів може сильно вплинути на якість результатів;

3) обмежена масштабованість – алгоритм може бути складно адаптувати для роботи з дуже великими наборами даних, що нівелює його практичну корисність.

Алгоритм DBSCAN

DBSCAN (Density-based spatial clustering of application with noise) – алгоритм кластеризації, що заснований на щільності: для заданої множини точок у деякому просторі він відносить в одну групу точки, що розташовані найбільш щільно, а ті, які розташовані з малою щільністю або далеко, як викиди [12]. Він є найбільш

поширеним серед усіх алгоритмів кластеризації. Серед переваг можна виділити такі:

1) здатність ідентифікувати кластери довільної форми;
2) стійкість до шуму – оскільки не відносить викиди до жодного кластеру;
3) автоматичне визначення кількості кластерів;
4) ефективність на великих базах даних.

Серед недоліків же можна визначити такі:

1) чутливість до параметрів – продуктивність може сильно залежати від двох вхідних параметрів, а саме епсилон (ε) та мінімальної кількості точок, необхідних для формування щільної області;
2) труднощі зі змінною щільністю – цей пункт випливає з попереднього, оскільки алгоритм покладається на єдиний глобальний поріг щільності для масиву даних;
3) високі вимоги до пам’яті та обчислювальних ресурсів при роботі з дуже великими даними.

Більш наочна демонстрація відмінностей між алгоритмами кластеризації зображена на рис 2.1.

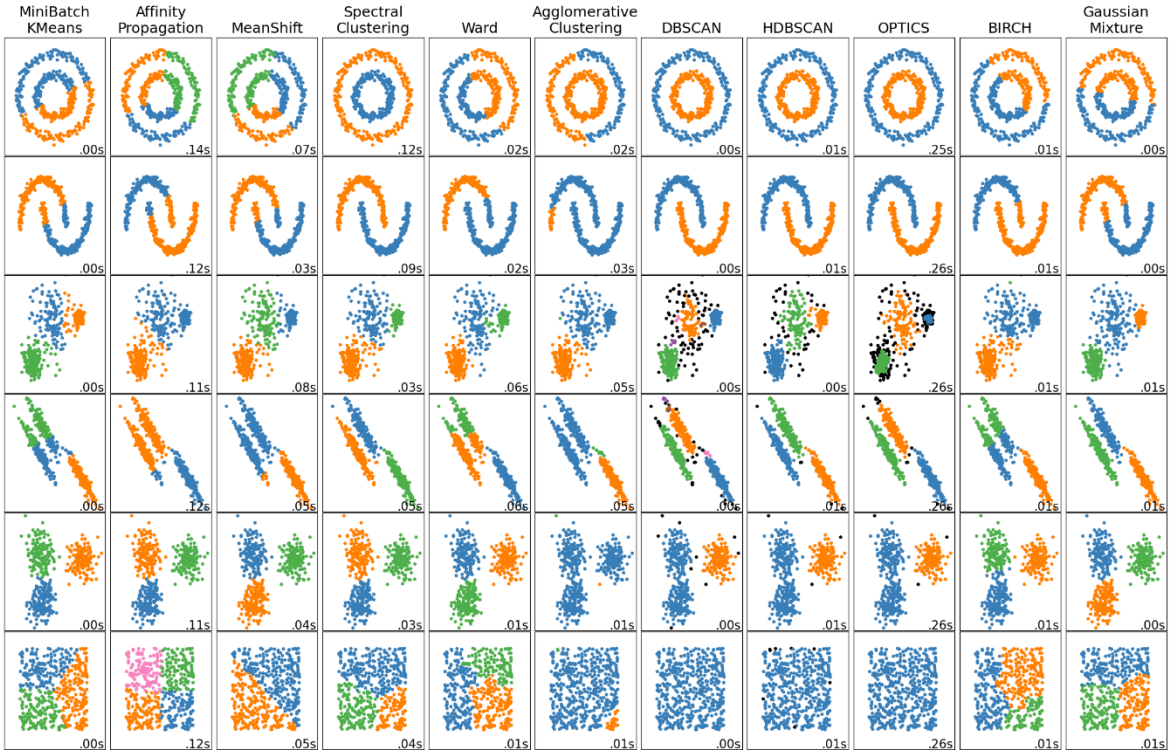


Рисунок 2.1 – Порівняння алгоритмів кластеризації (Scikit-Learn)

Звідси можна зробити висновок, що найбільш підходящим є алгоритм DBSCAN.

2.3 Використання нейронних мереж при роботі з зображеннями

Комп'ютерний зір є підмножиною машинного навчання, що використовує відео та цифрові зображення для того, щоб допомогти комп'ютерам високий рівень концепцій. Він фокусується на аналізі зображень для автоматизації завдань, що включають візуальні входи. Для цього необхідно створити потужну цілісну систему з різноманітним функціоналом, що математичні моделі якого було описано вище. Проте це лише базова інформація, яка пояснює принципи роботи й обробки зображень, а їх аналіз і знаходження зв'язків пов'язано з тренуванням нейронних мереж. Існують різні їх типи, які можна використовувати для обробки зображень.

Нейронні мережі прямого зв'язку

Вони є основним типом нейронних мереж і складаються з вхідного шару, одного або кількох прихованих шарів та вихідного шару. Дані проходять через мережу в прямому напрямку: від вхідного рівня до вихідного.

Нейронні мережі прямого зв’язку можуть використовуватись для різноманітних завдань, включаючи розпізнавання мови, зображень чи прогнозного моделювання. У ній вхідні дані передаються через мережу і кожен нейрон в прихованому шарі (чи шарах) виконує зважену суму вхідних даних, застосовує функцію активації та передає вихід на наступний рівень. Ваги та зміщення регулюються під час навчання щоб мінімізувати помилку між прогнозованим і фактичним результатом.

Багатошаровий перцептрон (MLP)

Це тип нейронної мережі, що містить в собі декілька шарів перцептронів [13]. Перцептрон – це один з найперших типів нейронних мереж, який частіше за все використовується для завдань класифікації, при цьому має великий недолік: він може вирішувати лише ті проблеми, у яких дані можна поділити на дві категорії за допомогою прямої лінії. Багатошаровий перцептрон вирішує цю проблему, оскільки дині подаються між шарами в конвеєрному режимі.

Згорткові нейронні мережі (CNN)

Це тип нейронних мереж, призначений для роботи з сітковими даними, якими якраз і являються зображення [14]. Вони складаються з кількох рівнів, у тому числі згорткових шарів, шарів об’єднання та повністю зв’язаних шарів, кожен з яких відіграє окрему роль в обробці даних і спрощенні виходів. Частіше за все саме їх використовують для класифікації зображень чи розпізнавання об’єктів на них. У цих нейронних мережах дані обробляються за допомогою кількох згорткових шарів, потім передаються через рівні об’єднання, які зменшують дискретизацію даних і їх розмірність, приклад чого можна побачити на рис. 2.2. Нарешті, вихідні дані проходять через повністю розв’язані рівні, які виконують остаточну класифікацію чи прогноз.

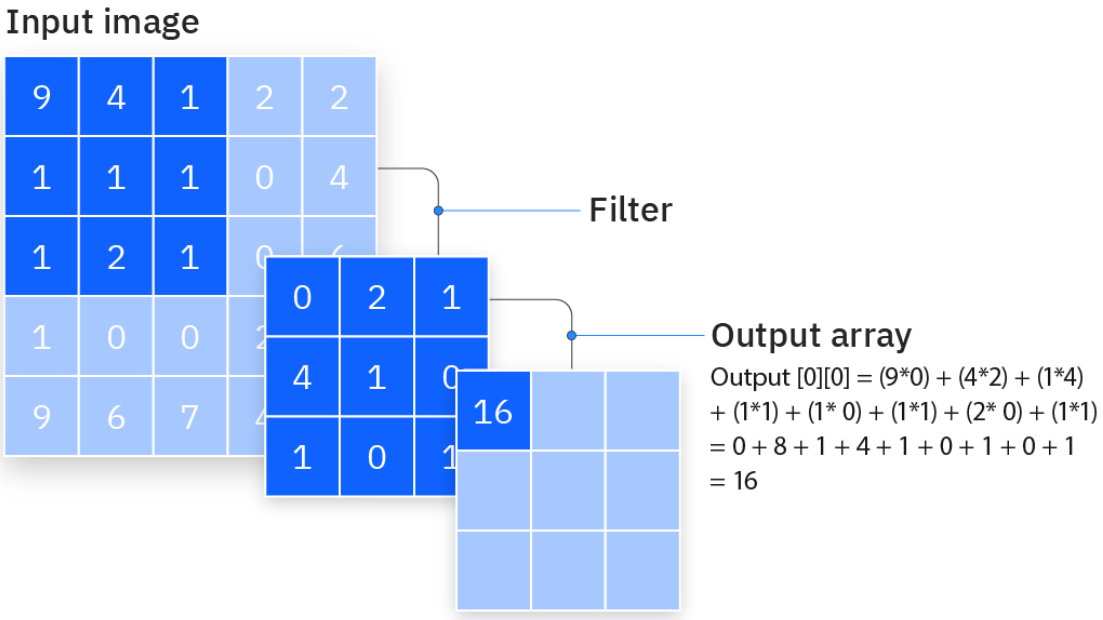


Рисунок 2.2 – Спрощення зображення за допомогою фільтрів

Швидкі згорткові нейронні мережі, що базуються на регіонах (Fast R-CNN)

Перед безпосереднього переходу до даного методу, необхідно спочатку розглянути його попередника R-CNN (згорткові нейронні мережі, що базуються на регіонах). Основна проблема зі звичайними CNN полягає в тому, що при виявленні необхідних об’єктів передчасно невідомо про їх кількість, через що довжина вихідного шару буде змінною, а не постійною. Можна було б поділити зображення на області і використовувати звичайні згорткові мережі для класифікації об’єктів в них, проте об’єкти можуть мати різні просторові розташування на зображенні та різні співвідношення сторін. Через це, ймовірно, доведеться обрати величезну кількість регіонів, що в подальшому може призвести до проблем з обчисленнями. Саме для цього було розроблено R-CNN.

Щоб обійти проблему з вибору величезної кількості регіонів було запропоновано метод, за допомогою якого використовується вибірковий пошук, щоб виділити лише 2000 регіонів з зображення [15; 16]. Приклад алгоритму роботи подібної нейронної мережі можна побачити на рис. 2.3.

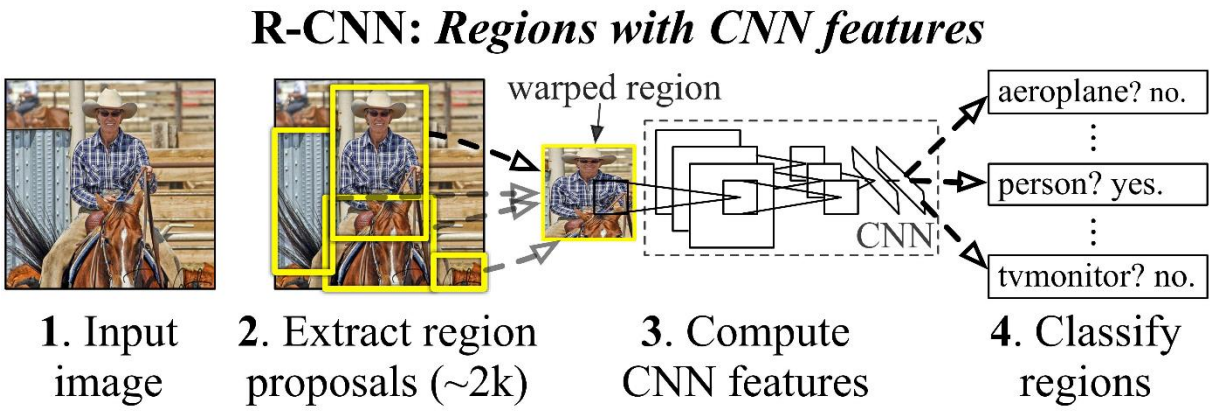


Рисунок 2.3 – Алгоритм R-CNN

Проте в даного алгоритму теж є декілька недоліків, основними з яких є [17]:

1) час навчання – класифікація 2000 регіонів це складний і витратний процес;

2) для обробки кожного зображення необхідно близько 47 секунд;

3) алгоритм вибіркового пошуку є фіксованим, тому на цьому етапі не відбувається навчання, що може призвести до поганих пропозицій регіонів-кандидатів.

Тут на заміну йому автор попередньої статті про R-CNN запропонував швидший алгоритм виявлення об’єктів, який назвали Fast R-CNN. Підхід подібний до минулого, але замість передавання пропозицій щодо регіону в CNN, передається вхідне зображення для створення згорткової карти особливостей. Схему даного процесу продемонстровано на рис. 2.4.

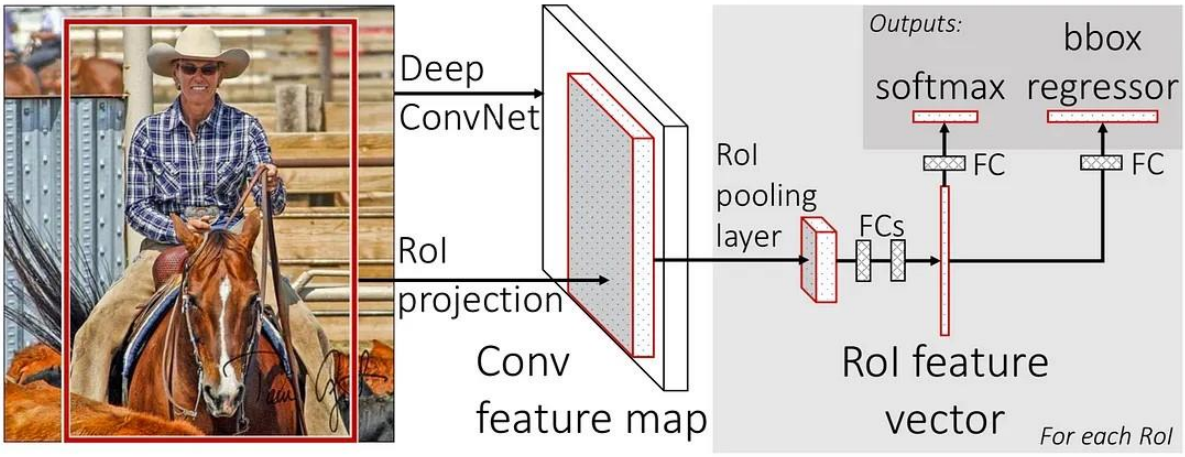


Рисунок 2.4 – Архітектура Fast R-CNN

На цій карті визначається область пропозицій та деформується у квадрати, а за допомогою шару об’єднання регіонів інтересів (RoI) змінюються до фіксованого розміру задля введення в повністю зв’язаний шар. В цьому й полягає суть покращеної швидкості даного алгоритму: замість того щоб щоразу передавати 2000 пропозицій регіонів до згорткової нейронної мережі, операція згортки виконується лише раз для зображення і з цього генерується карта особливостей. Динаміку зміни швидкості можна побачити на рис. 2.5 [17].

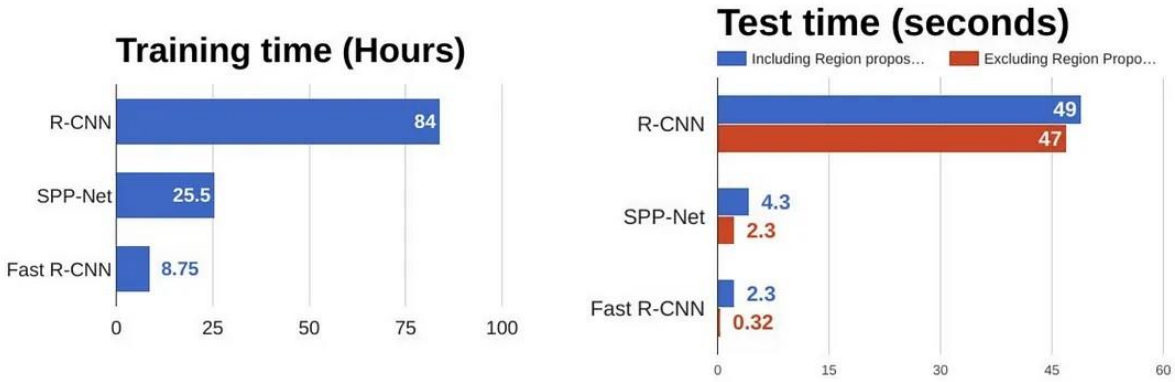


Рисунок 2.5 – Порівняння часу тренування та тестування

З наведеного вище графіку чітко видно, що різниця в швидкості є колосальною.

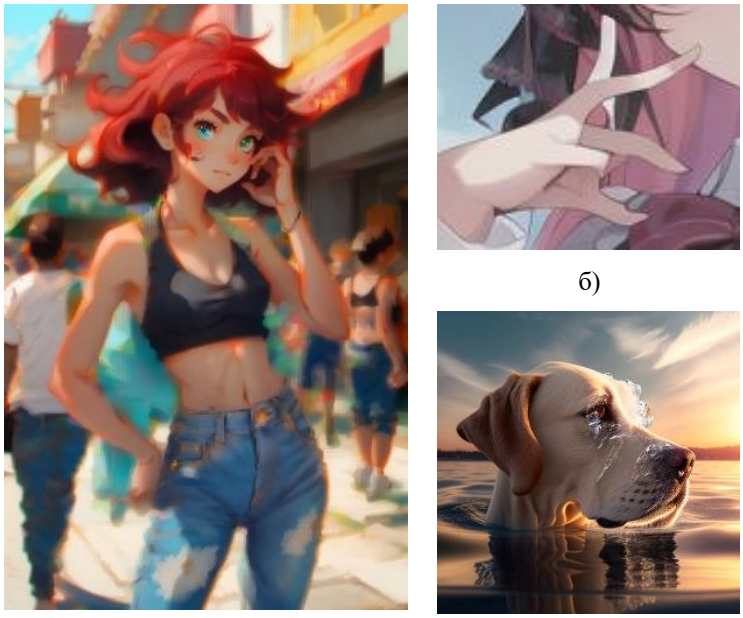
Алгоритм YOLO

Цей алгоритм відрізняється від попередніх, де виявлення об’єктів було побудоване на використанні областей для локалізації цих їх на зображенні. Він представляє з себе єдину згорткову мережу, що передбачає обмежувальні рамки та класи ймовірностей для кожної з них. Принцип полягає в розбитті зображення на сітку NxN, у межах кожної з них обираються m обмежувальних рамок. Для кожної з цих рамок мережа виводить значення класу ймовірності та зміщення. Ті ж обмежувальні рамки, імовірність класу яких перевищує порогове значення, вибираються та використовуються для визначення та місцезнаходження об’єкта на зображенні.

Його основною й доволі великою перевагою є те, що він на порядки швидший (до 45 кадрів на секунду), що дозволяє обробляти величезні масиви даних або ж потокове відео. Проте існують й певні обмеження, оскільки при роботі з дрібними об’єктами, наприклад, зграєю тварин, можуть виникати труднощі.

2.4 Проблематика пошуку дефектів на зображеннях

Більшість наявних на ринку рішень з пошуку дефектів зазвичай стосуються матеріалів або ж об’єктів, які мають спільні характеристики й вигляд (томографія, флюорографія тощо). Проте в даному випадку підходи, що використовуються, є нерелевантними, оскільки визначення відмінностей на рівномірних суцільних об’єктах є відносно нескладною задачею. Що ж стосується зображень, створених штучним інтелектом, дефекти, які можуть утворюватися є значно складнішими для розпізнавання. Наприклад, неправильна поза пальців або ж відсутня певна їх кількість. В залежності від пози, це часто може трактуватись як поворот кисті, через що не буде визначатись як помилка. Приклади можливих дефектів серед обраних можна переглянути на рис. 2.6.



а) б) в)

Рисунок 2.6 – Дефекти зображень: (а) сильне розмиття, при відсутності його в запиті; (б) аномалія пальців; (в) стан води на голові собаки

Тому необхідно провести симбіоз 2 моделей з різними задачами: одна повинна розпізнавати ті частини тіла, на яких будуть проводитись пошуки дефектів, та інша, яка буде визначати що є дефектом, а що ні. Проте це є складним в реалізації, оскільки ці задачі не можна розділяти і вони мають відбуватись одночасно.

Також необхідно дослідити ефект «неприродньої долини», коли об’єкт на зображенні виглядає звичним, але певні нюанси змушують людей почуватись дискомфортно. Дана проблема часто зустрічається на згенерованих штучним інтелектом зображеннях [18]. Подібні дефекти значно складніше відслідкувати, оскільки часто важко інтерпретувати відчуття і емоції в цифрові значення, тому з високою долею ймовірності для штучного інтелекту зрозуміти, чому ж зображення виглядає неправильно, буде великою проблемою.

2.5 Розробка діаграми використання

Після розгляду поставлених задач та проведених оцінок застосунків-аналогів, було сформовано можливості використання застосунку користувачем, що зазначені на рис. 2.7

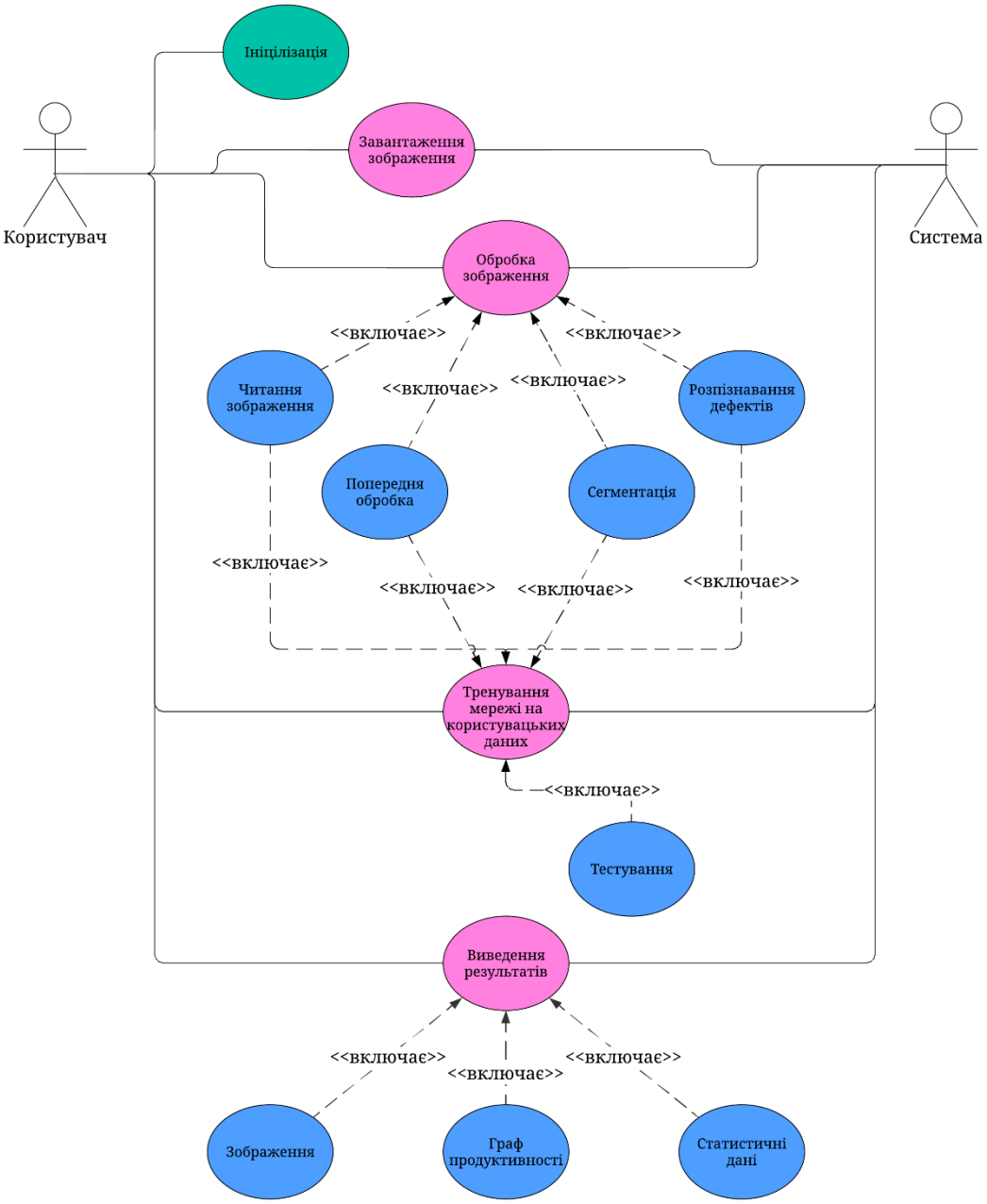


Рисунок 2.7 – Діаграма використання застосунку

Тренування мережі виділено як окрему функцію, оскільки існує велика кількість моделей та LoRA (Low-Rank Adaptation) які використовуються для встановлення додаткових налаштувань для використання різних стилів. Наприклад, реалістичних, подібних до аніме, 3D моделей тощо. Оскільки вони можуть сильно відрізнятись між собою, а навчання на всіх можливих варіаціях є майже неможливим, було обрано саме таке вирішення проблеми.

2.6 Моделювання інтерфейсу застосунку

Існує велика кількість застосунків для створення дизайну. Основними з них є Adobe Photoshop, Figma та Canva. Кожен з них має свої переваги і недоліки, але найкращим для відображення декількох екранів є Figma, оскільки вона дозволяє доволі якісно керувати фреймами, має шаблони основних екранів з підписами (наприклад, розширення iPhone 14) та має інструменти для створення власних компонентів, завдяки чому можна легко й швидко стандартизувати інтерфейси.

Продуманий дизайн застосунку є доволі важливим, оскільки саме через нього відбувається взаємодія користувача з ним і якщо він буде незручним чи незрозумілим, то користуватись ним не будуть, не зважаючи на функціонал. Розробка інтерфейсів не є новітнім напрямком в інженерії ПЗ, тому найкращі практики уже було узагальнено. Основними з них є:

1) KISS (Keep it simple, stupid) – якщо розглядати середовище розробки, то за цим принципом більшість систем працюють краще, якщо вони є простими і не ускладненими;
2) кольорова палітра не має бути тьмяною: текст повинен бути чітко видимим і вирізнятись, елементи не мають зливатись між собою, проте і занадто контрастною палітра теж не має бути;
3) елементи мають бути згруповані за своєю сутністю;
4) елементи повинні виконувати завдання, які від них очікуються;
5) якщо щось має натискатись, це повинно бути виділено стилістично;

6) використовуючи термінологію, необхідно враховувати усереднений словниковий запас пересічного користувача, аби уникнути ризику нерозуміння певних функцій;

7) шрифти мають бути легкими для читання на всіх типах екранів.

Взявши до уваги зазначені принципи, було розроблено основні мінімалістичні інтерфейси, що зображені на рис. 2.8–2.10

На рис. 2.8 зображено головну сторінку застосунку, де наявна основна інформація про актуальність розробки програмного забезпечення, наведено базову покрокову інструкцію з того як застосунок працює та в кінці кнопка для того щоб спробувати його функціонал самостійно. Дана кнопка відповідає за вибір файлу для подальшого пошуку дефектів і хоча не є стандартною за виглядом, проте має очікуваний функціонал відповідно до теми сайту.

Оскільки макети зазвичай роблять відповідно до всього контенту, зображення є повздовжнім і відрізнятиметься від кінцевого результату, де буде доступна прокрутка. Для попереднього перегляду в Figma є функція попереднього перегляду, де є можливість адаптувати до ширини екрану, що дозволяє попередньо переглядати ймовірний дизайн, через що значно легшим є підбір розміру шрифтів, розташувань елементі тощо.

Рисунок 2.8 – Макет головної сторінки застосунку

Наступною сторінкою є відображення усіх завантажених зображень, що відображено на рис. 2.9. Основний її функціонал це виступати списком для переходу до сторінки конкретного зображення, де буде відображатись більш детальна інформація.

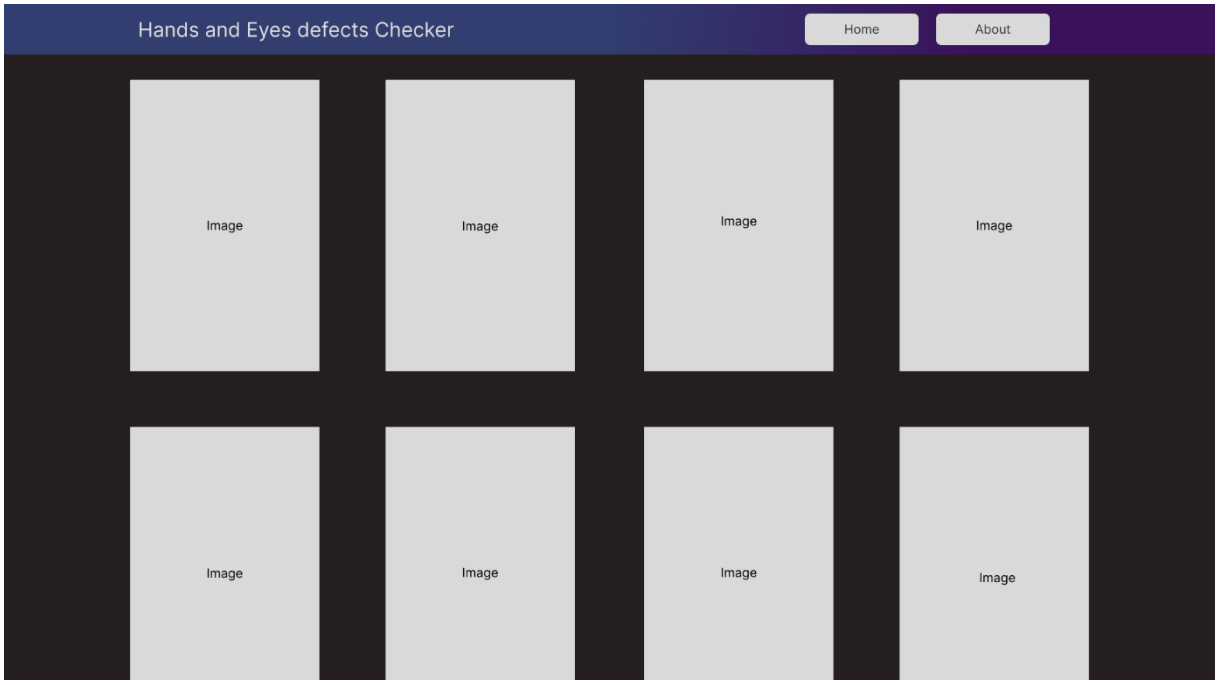


Рисунок 2.9 – Макет сторінки відображення зображень

На рис. 2.10 зображено макет сторінки конкретного зображення, де наведена інформація про запит, оригінальне зображення та, за наявності дефектів, зображення зі знайденими помилками.

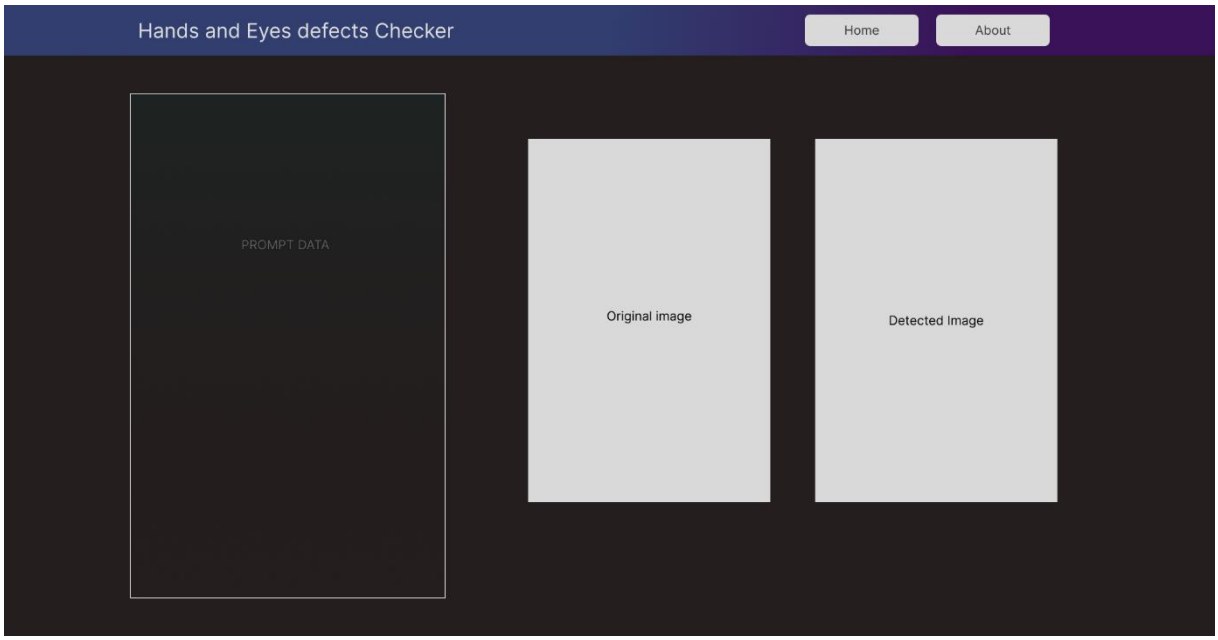


Рисунок 2.10 – Макет сторінки зображення

Висновки до розділу 2

В цьому розділі було розглянуто різноманітні математичні моделі, які будуть корисними при розробці програмного забезпечення. Їх було поділено на категорії відповідно до задач, для яких вони використовуються.

Щодо збереження і обробки даних наведено 2 математичні моделі, що можуть бути використані для оптимізації сховища/бази даних та покращення загальної тренувальної вибірки.

Також наведено моделі, що використовуються при кластеризації зображень, оцінено кожну з них та обрано найбільш оптимальну згідно з задачами, які можуть виникнути під час виконання даної роботи.

Проведено аналіз різних типів нейронних мереж, що використовуються при роботі з задачами комп’ютерного зору, наведено основні переваги та недоліки кожної та принципи їх роботи.

Також було вказано на проблематику пошуку дефектів на зображеннях, створених за допомогою ШІ і наведено певні приклади та можливі успішні рішення.

Створено діаграму використання, яка детально демонструє можливості використання застосунку, а саме розпізнавання дефектів та можливість завантаження користувацького набору даних для подальшого тренування. А також розроблено макети інтерфейсу застосунку.

3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПОШУКУ ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ, СТВОРЕНИМИ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Пошук даних для подальшої обробки та підготовка зображень

Пошук дефектів на згенерованих зображеннях є доволі специфічною і саме наборів даних, що містили б в собі дефекти немає. Існує велика кількість *cursed* (або ж проклятих) зображень та відео, на яких наявна велика кількість різноманітних дефектів. Але хоч вони й доступні в відкритих джерелах, та через певні причини вони не підходять: при навчанні нейронної мережі краще обрати якісні зображення, але з певними дефектами, оскільки інакше це може призвести до небажаного результату, коли, власне, дефектами буде вважатись не те що потрібно. Тих же зображень, що є доволі хорошими, але містять в собі обрані дефекти, у відкритому доступі надзвичайно мало, оскільки користувачі зазвичай пропускають дані результати і не зберігають, а тим більше не викладають в мережу.

Існують сайти з різними наборами даних для навчання, основним з яких серед генеративних ШІ є Hugging Face та CIVITAI, проте там не було знайдено підходящих для даної роботи. Тому було прийняте рішення для ручного створення масиву зображень. Для цього буде використано Stable Diffusion та різні моделі, такі як: stable-diffusion-xl-base-1.0, PicX_real, RealCartoon-Anime.

Можна надати зображення мережі в сирому вигляді, тобто ті, що були отримані на виході іншої text-to-image нейронної мережі, проте результат буде часто не передбачуваним та кількість часу, що необхідна для тренування зросте в рази. Для уникнення цього існують застосунки, що надають змогу маркувати зображення: додавати анотації, метадані тощо.

Анотація за допомогою CVAT

CVAT [19] (Computer Vision Annotation Tool) – це безкоштовний, відкритий, веб-базований інструмент для анотації зображень та відео, який використовується

для позначення даних для алгоритмів комп’ютерного зору. Він був розроблений Intel і зараз підтримується OpenCV. CVAT надає функції анотації для завдань з виявлення об’єктів, класифікації, відстеження та сегментації. Особливим є можливість розпізнавання об’єктів через LIDAR, створення 3D кубоїдів на зображеннях, та «скелети» об’єктів.

Серед інших переваг та особливостей, можна також виділити:

1) безкоштовність;
2) відкритий код;
3) інтеграції з іншими сервісами;
4) напівавтоматичні анотації;
5) можливість використання як на власному ПК, так і в браузері;
6) підтримка інтерполяції.

Проте наявний також ряд певних недоліків:

1) часті проблеми з продуктивністю;
2) відсутність версифікації даних;
3) нестача розширеного фільтрування і сортування.

Приклад використання даного застосунку можна побачити на рис. 3.1.

Рисунок 3.1 – Ручне виділення руки з дефектом (6 пальців)

Анотації на завантажених зображеннях можна експортувати і використати при навчанні нейронної мережі.

Набір даних COCO

COCO [20] (Common Objects in Context) – це великий набір даних для розпізнавання зображень, який використовується для завдань виявлення об’єктів, сегментації та опису. Він містить понад 330 000 зображень, кожне з яких анотовано 80 категоріями об’єктів та 5 підписами, що описують сцену. Набір даних COCO широко використовується в дослідженнях комп’ютерного зору і був використаний для навчання та оцінки багатьох передових моделей виявлення та сегментації об’єктів.

Зображення організовані в ієрархію каталогів, де каталог верхнього рівня містить підкаталоги для наборів тренування, перевірки та тестування1. Анотації надаються у форматі JSON, де кожен файл відповідає одному зображенню. Кожна анотація в наборі даних включає наступну інформацію: ім’я файлу зображення,

розмір зображення (ширина та висота), список об’єктів з наступною інформацією: клас об’єкта (наприклад, «людина», «автомобіль»); координати обмежувального прямокутника (x, y, ширина, висота); маска сегментації (формат полігону або RLE); ключові точки та їх позиції (якщо доступні) та п’ять підписів, що описують сцену. Набори даних звідси можна використовувати для початкового тренування з пошуку персон, з подальшим покращенням до пошуку окремих частин тіл.

Основними перевагами є:

1) велика кількість зображень та анотацій;

2) різноманітність сцен;

3) додаткові анотації (ключові точки, сегменти тощо).

Проте є також і певні недоліки, а саме:

1) недостатнє представлення певних категорій (наприклад, відсутні руки, обличчя, тощо);

2) прогалини в якості наборів, приклад чого можна побачити на рис. 3.2.

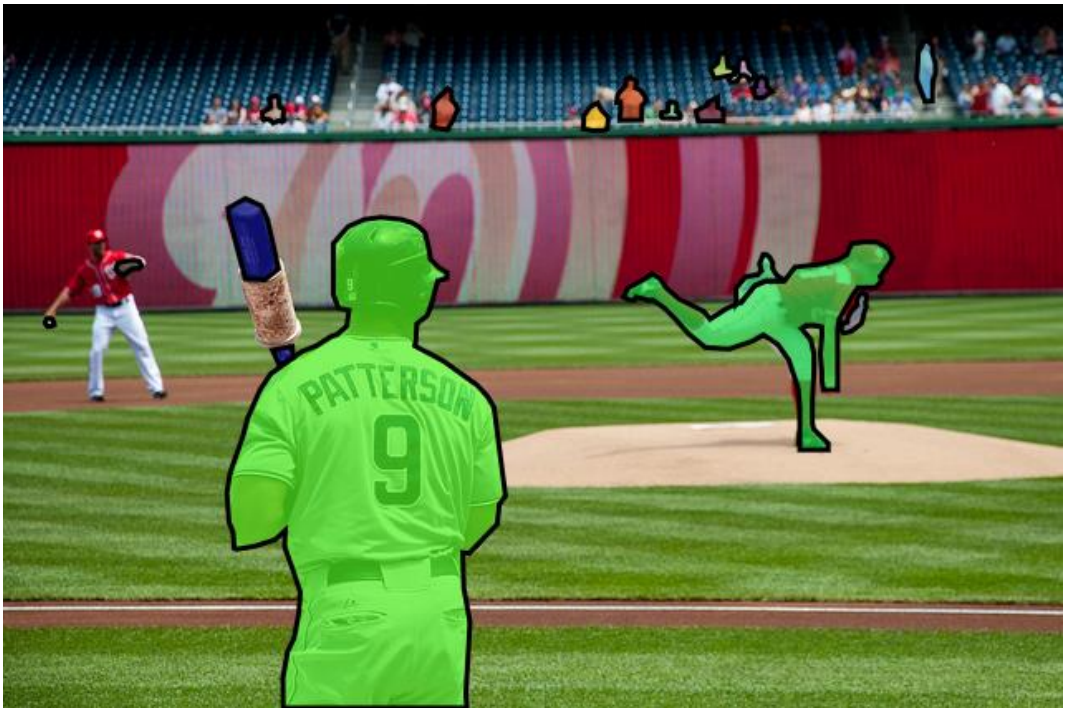


Рисунок 3.2 – Низькоякісне зображення для тегу «person»

3.2 Розробка архітектури застосунку

При розробці застосунків, що пов’язані з машинним навчанням основним є утримання балансу між зрозумілістю коду, зручністю його використання та гнучкістю архітектури. Оскільки самостійно усвідомлювати це може зайняти певний час, уже було узагальнено набір принципів SOLID, що допомагає приймати правильні архітектурні рішення. Для кращого розуміння, необхідно розшифрувати абревіатуру:

а) Single responsibility principle (Принцип єдиної відповідальності) – клас має відповідати тільки за одну частину функціональності програми, причому вона повинна бути повністю інкапсульована в ній.

б) Open/closed principle (Принцип відкритості/закритості) – класи мають бути відкритими для розширення, але закритими для зміни, щоб не ламати існуючий код при внесенні змін.

в) Liskov substitution principle (Принцип підстановки Лісков) – підкласи повинні доповнювати, а не заміщати поведінку базового класу, щоб у випадку чого об’єкти могли бути заміненими нащадками без зміни коду програми.

г) Interface segregation principle (Принцип поділу інтерфейсу) – краще мати багато спеціалізованих інтерфейсів ніж один універсальний задля майбутньої гнучкості компонентів.

д) Dependency inversion principle (Принцип інверсії залежностей) – класи верхніх рівнів не мають залежати від класів нижніх рівнів; обидва повинні залежати від абстракцій.

Згідно з наведеними принципами було розроблено UML діаграму класів для системи тренування нейронних мереж, що зображена на рис. 3.3.

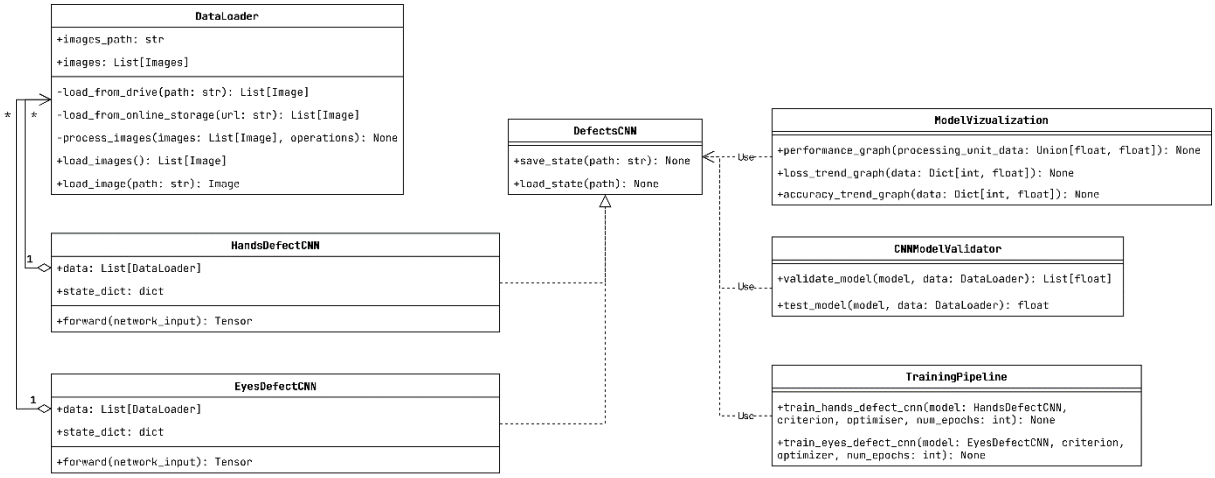


Рисунок 3.3 – Діаграма класів шару тренування нейронної мережі

Було прийнято рішення по розділенню функцій по різних класах, аби не наповнювати класи нейронних мереж зайвими функціями і властивостями. Для цього створено окремі класи для тренування, валідації, візуалізації моделей. Це дозволить легше відслідковувати весь процес, за необхідності розбивати задачі та легше проводити пошук помилок чи змінювати конфігурації моделей.

Для поєднання різних частин коду буде використано Docker контейнери, окремо для роботи з тренуванням мереж, їх використанням та поєднанням цього з frontend частиною.

3.3 Використання контейнерів Docker

Docker - це відкрита платформа для розробки, доставки та запуску застосунків. Вона дозволяє відокремити застосунки від інфраструктури, завдяки чому можна швидко доставляти програмне забезпечення до кінцевого користувача. Використовуючи Docker можна пакувати та запускати застосунок в слабко ізольованому середовищі, що називається контейнером. Контейнери є легкими і містять все необхідне для запуску застосунку, через що не потрібно перейматись про те, що встановлено на хості. Їх можна поширювати під час роботи, через що можна бути певним, що кожен розробник чи користувач отримає такий самий контейнер, що буде працювати ідентично.

Використовуючи Docker можна також спростити процеси безперервної інтеграції (CI/CD) та тестування. Завдяки його архітектурі, яку можна переглянути на рис. 3.4, процес віддаленої або ізольованої розробки є доволі простим та легким в налаштуванні.

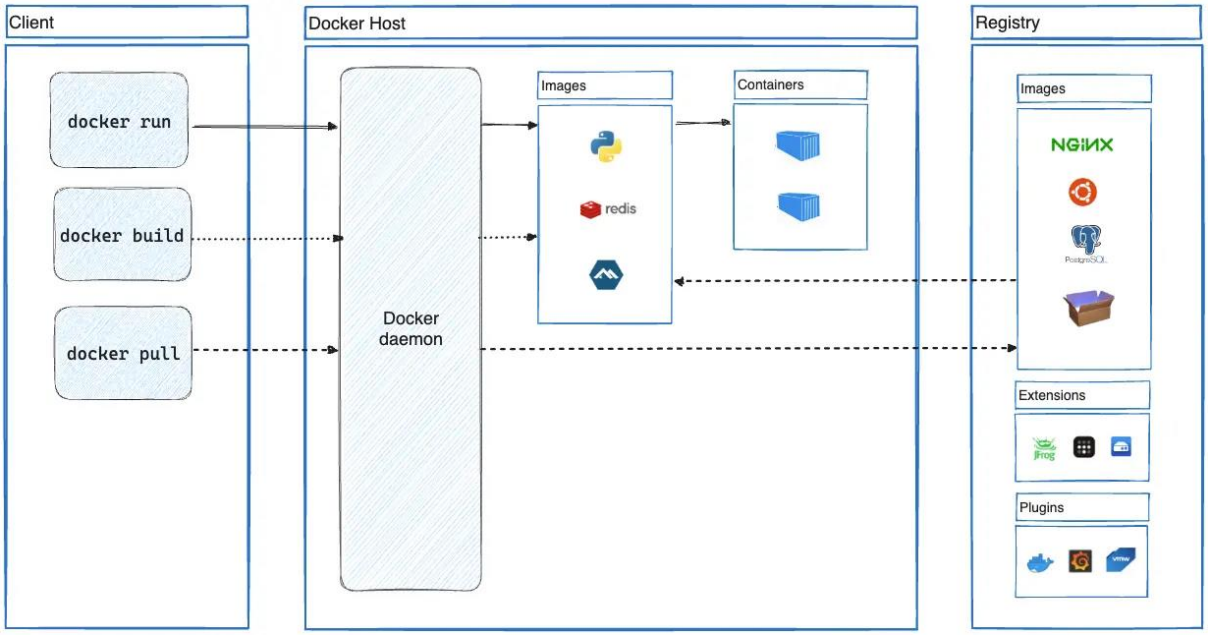


Рисунок 3.4 – Архітектура застосунку з використанням Docker

Перед використанням контейнерів необхідно створити зображення – шаблон, що доступний лише для читання, з інструкціями щодо створення контейнеру. Частіше за все зображення базуються на інших, але з додатковими елементами. Для створення власного, необхідно написати Dockerfile з покроковою інструкцією того, що має встановлюватись, запускатись тощо.

Використання Docker на віддаленому сервері з більшими потужностями CPU та GPU дозволить зменшити навантаження на локальні машини, делегувати завдання з тренування та валідації, що дозволить зосередитись безпосередньо на розробці.

3.4 Інструменти для виконання роботи

Мова програмування

Однозначним фаворитом в темі створення та обробки зображень є Python. Існує декілька ключових особливостей, що зумовили саме такий вибір, а саме:

1) для неї створено велику кількість модулів, що спрощують процес розробки та тестування;

2) основні модулі для роботи з зображеннями, наприклад, PyTorch (що оснований на Torch), OpenCV тощо розробляються саме для цієї мови;

3) завдяки використанню існуючих бібліотек можна реалізувати різні види застосунків, наприклад, вебзастосунок або застосунок пристосований безпосередньо до певної операційної системи;

4) швидкість вивчення базового функціоналу бібліотек є вищою відносно інших мов, тому інтегрування їх в код стає більш швидким, від чого сама розробка прототипу робочого програмного забезпечення потребує значно менше часу;

5) динамічна типізація, що дозволяє реалізовувати ті конструкції, що є неможливими або складними для написання в мовах зі статичною типізацією. Також за необхідністю можна типізувати необхідні ділянки коду самостійно, що може зменшити кількість проблем із використанням реалізованих функцій;

6) простота розгортання є великою, оскільки в більшості дистрибутивів, що основані на операційній системі Linux Python наявний за замовчуванням, а в Windows його можна легко встановити за допомогою вбудованого магазину застосунків.

Із недоліків Python можна виділити те, що він є повільнішим в роботі з певними розрахунковими задачами у порівнянні з деякими іншими мовами. І хоча зазвичай це не помітно, при роботі з великою кількістю розрахунків або опрацюванню значних масивів даних це може призвести до певних проблем. Але ці проблеми вирішуються різними модулями, що можна встановити і використовувати в коді: вони дозволяють значно збільшити швидкість роботи Python завдяки інтеграції C++, C.

Середовище розробки

Серед великої кількості IDE, в яких є змога працювати з Python, основними можна визначити PyCharm та Visual Studio Code, який є редактором коду, подібним до IDE. Вибір залежить від багатьох факторів, але основним є те, що VS Code поставляється без попередньо встановлених пакетів і, хоча й є можливість додавання великої кількості функціоналу за допомогою цих плагінів, проте в PyCharm це все за замовчуванням. Єдиною проблемою може бути те, що він є платним, проте JetBrains видають безкоштовні ліцензії студентам, що ще більше схиляє до вибору даного продукту.

В ньому є велика кількість різноманітного функціоналу: візуалізація даних при роботі з бібліотеками, спрощена робота з базами даних, простота налаштування різних інтерпретаторів, вбудована інтеграція з GIT та ще багато чого іншого. Також автодоповнення коду є доволі хорошим, хоча щодо більш сучасного GitHub Copilot є слабшим.

Бібліотека Pandas

Pandas – це бібліотека для аналізу, обробки і маніпуляції з даними. Основними перевагами даної бібліотеки є:

1) швидкий та ефективний клас DataFrame для роботи з даними;
2) розумне вирівнювання даних та інтегрована обробка відсутніх даних;
3) гнучка зміна та поворот даних;
4) легкі вставка та видалення стовбців із структури даних;
5) високопродуктивне **злиття** та **поєднання** наборів даних;

Окремо необхідно відмітити високу оптимізацію, що зумовлена написанням критичних частин коду на Cython або C.

Враховуючи отримані набори даних, дана бібліотека є ідеальним варіантом для зручної, швидкої та простої роботи з ними не тільки через методи обробітку таблиць, а й можливість читання й запису файлів безпосередньо з використанням даної бібліотеки.

Бібліотека NumPy

NumPy – це бібліотека, що дозволяє розширити стандартні структури даних завдяки доданню багатовимірних масивів та матриць, а також великої колекції математичних функцій для роботи з цими масивами.

Більшість переваг даної бібліотеки збігається з описаною вище бібліотекою Pandas, оскільки вони добре сумісні між собою і часто використовуються разом для максимальної ефективності і швидкості. Можна легко конвертувати структури даних, надані однією бібліотекою, в структуру даних іншої, що робить код дуже гнучким. Також найбільш потужна частина NumPy, а саме ***ndarray***, що представляє з себе багатовимірний масив, часто використовується при тій чи іншій роботі з DataFrame. Використання вбудованих в NumPy математичних функцій дозволяє спростити роботу з таблицями і матрицями, оскільки в Pandas реалізований режим сумісності, завдяки чому можна легко та швидко обраховувати необхідні показники та застосовувати перетворення до рядків чи стовпчиків таблиць.

Оскільки зображення за своєю суттю є матрицями, використання даних бібліотек для роботи з зображеннями є необхідним.

Бібліотека PyTorch

PyTorch – це відкрита бібліотека машинного навчання, розроблена на основі бібліотеки Torch для програм на Python. Він підтримує загальні перетворення комп’ютерного зору в модулі torchvision.transforms. Ці перетворення можна використовувати для перетворення або доповнення даних для навчання або виведення різних завдань (класифікація зображень, виявлення, сегментація, класифікація відео)

Серед його основних переваг можна виділити такі:

1) інтуїтивний синтаксис;
2) виконання за бажанням;
3) потужна GPU оптимізація;
4) гнучкість щодо завдань.

Проте, він має й недоліки, основним з яких є те, що в нього, на відміну від TensorFlow, відсутні інтерфейси для моніторингу та візуалізації, що може ускладнити процес тестування та пошуку багів.

Бібліотека TensorFlow

TesnorFlow – це відкрита бібліотека для машинного навчання, яку розробила Google Brain. Вона пропонує високорівневий та абстрактний підхід до організації низькорівневого числового програмування. TensorFlow може бути використаний у різних сценаріях, але особливу увагу приділяється тренуванню та виведенню глибоких нейронних мереж. Він підтримує розподілене навчання, негайну ітерацію моделі та легке налагодження з Keras, і багато іншого.

TensorFlow підтримує загальні перетворення комп’ютерного зору в модулях tf.keras.utils.image_dataset_from_directory та tf.keras.layers.Rescaling. Ці перетворення можна використовувати для завантаження та попередньої обробки зображень.

Серед переваг можна виділити наступні:

1) масштабованість;
2) відмінні можливості візуалізації даних;
3) зручність налагодження;
4) паралелізм;
5) підтримка технології Tensor Processing Unit (TPU).

Проте з іншого боку, він підтримує лише відеокарти NVIDIA, є відносно повільним та дозволяє використовувати TPU лише для *виконання* моделей, а не їх навчання.

Її було досліджено як пошук бібліотек, але надано перевагу використанню PyTorch та Scikit-learn.

Бібліотека Scikit-learn

Scikit-learn – це ще одна відкрита бібліотека для машинного навчання, розроблена командою Google Brain. Вона використовує концепцію

обчислювального графа, автоматичне диференціювання та адаптивну структуру API Python, що спрощує вирішення реальних проблем.

При роботі з зображеннями, Scikit-learn надає високорівневі утиліти та шари для читання директорії зображень на диску. Можна використовувати такі утиліти, як skimage.data.camera() та шари, такі як skimage.filters.sobel(image), для завантаження та попередньої обробки зображень.

Його переваги можна продемонструвати наступним списком:

1) безкоштовне використання;
2) універсальність;
3) простота використання;
4) серйозна підтримка спільноти.

Проте наявний і ряд мінусів, таких як відсутність вбудованої підтримки обчислень на GPU та глибокого навчання.

Бібліотека OpenCV

OpenCV (Open Source Computer Vision Library) - це відкрита бібліотека програмних функцій, яка призначена переважно для реального комп’ютерного зору. Вона надає доступ до більш ніж 2500 оптимізованих алгоритмів, які включають в себе всеосяжний набір як класичних, так і сучасних алгоритмів комп’ютерного зору та машинного навчання. Ці алгоритми можуть бути використані для виявлення та розпізнавання облич, ідентифікації об’єктів, класифікації людських дій у відео, відстеження рухів камери, відстеження рухомих об’єктів, витягування 3D-моделей об’єктів, створення 3D-хмар точок зі стереокамер, зшивання зображень разом для створення зображення всієї сцени з високим розширенням, пошуку подібних зображень у базі даних зображень, видалення червоних очей з зображень, зроблених за допомогою спалаху, відстеження рухів очей, розпізнавання пейзажів та встановлення маркерів для нашарування його з доповненою реальністю та ще багато всього іншого. Як можна помітити, функціоналу для роботи з зображеннями тут доволі багато.

При роботі з зображеннями, OpenCV надає високорівневі утиліти та шари для читання директорії зображень на диску. Можна використовувати різні утиліти також для попередньої обробки зображень.

Основною перевагою даної бібліотеки є велика кількість і доступність алгоритмів, які є високоефективними. Проте він теж не має вбудованої підтримки обчислень на GPU та глибокого навчання.

Висновки до розділу 3

В цьому розділі було проведено пошук даних, що могли б підійти для подальшої обробки і навчання нейронних мереж. Таких даних знайдено не було, через специфіку характеристик, які необхідні для успішного виконання роботи. Тому було прийнято рішення в ручній генерації зображень, та подальшій їх класифікації за допомогою CVAT.

Було розроблено архітектуру застосунку, яка поділена на декілька шарів. Обрано клієнт-серверну архітектуру, де на серверній частині буде Docker, в якому будуть різні зображення і контейнери, що відповідатимуть за хостинг застосунку, впровадження CI/CD, та навчання нейронних мереж.

Створено діаграму класів для відображення шару роботи з нейронними мережами. Класи було розділено за принципами SOLID. Було прийнято рішення про розділення класу нейронної мережі на різні, при тому, що логіку можна було б інкапсулювати в одному класі, проте такий підхід дозволяє краще зосереджуватись на окремих аспектах розробки.

Також було наведено інструменти, що плануються для використання у роботі, а саме проведено аналіз мов програмування, серед яких обрано Python, різних IDE, серед яких обрано PyCharm, та досліджено різні бібліотеки, що необхідні для розробки нейронних мереж.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Контейнеризація системи

Існує велика кількість зображень, доступних для завантаження і подальшої роботи. Одним з них є зображення для фреймворку Flask, що використовується для розробки вебзастосунків. Є також для використання PyTorch, TensorFlow та інших, проте вони не оновлювались доволі давно або ж існують певні проблеми з версифікацією, наприклад, того ж NumPy, тому необхідно докласти великих зусиль для того щоб вони запрацювали, через що краще створити власні з нуля під обрану версію Python.

В нових версіях Docker стало доступно використання оточення розробника (Dev Environment), яка знаходиться в беті, проте уже зараз дозволяє редагувати код, вмонтований в контейнер, що дозволяє мати декілька робочих версій поруч. Одним із хороших прикладів є розгортання Flask застосунку разом з MongoDB. Це буде використовуватись для клієнт-серверної частини застосунку.

Щодо шару з нейронною мережею, було ухвалене рішення помістити її в той же контейнер. Це зумовлено декількома причинами:

1) всі компоненти в одному контейнері – це спрощує можливість спільного використання елементів та коду;

2) простота налаштування і розгортання;

3) зручність налагодження – таким чином простіше спостерігати на взаємодію між Flask та нейронною мережею;

4) менші витрати – використання одного контейнера зменшує витрати на пам’ять та процесорний час.

І хоча є певні недоліки в цьому підході, наприклад, залежність між компонентами, проте вони не перекривають основні переваги.

Таким чином було створено середовище розробника та контейнери, що зображені на рис. 4.1. Контейнер з ім’ям «diploma» відповідає за середовище розробки, а з ім’ям «nginx-flask-mongo» є безпосереднім проєктом.

Рисунок 4.1 – Створені контейнери

4.2 Створення масиву даних

Для створення масиву даних для навчання було обрано використання сервісу CVAT. Проте, оскільки експорт зображень разом з анотаціями є платним, було проведено також співставлення зображень з їх анотаціями. Вихідним файлом є файл формату *.xml*, що зображено на рис. 4.2, в якому описано знайдені елементи та їх розташування. Зображення обирались на основі різних стилів та запитів, що може призвести до різних результатів. Планується навчити нейронну мережу визначати дефекти на різних зображеннях, проте це може мати зворотній ефект, при якому загальний результат буде недостатнім, хоча це й відносно малоймовірно.

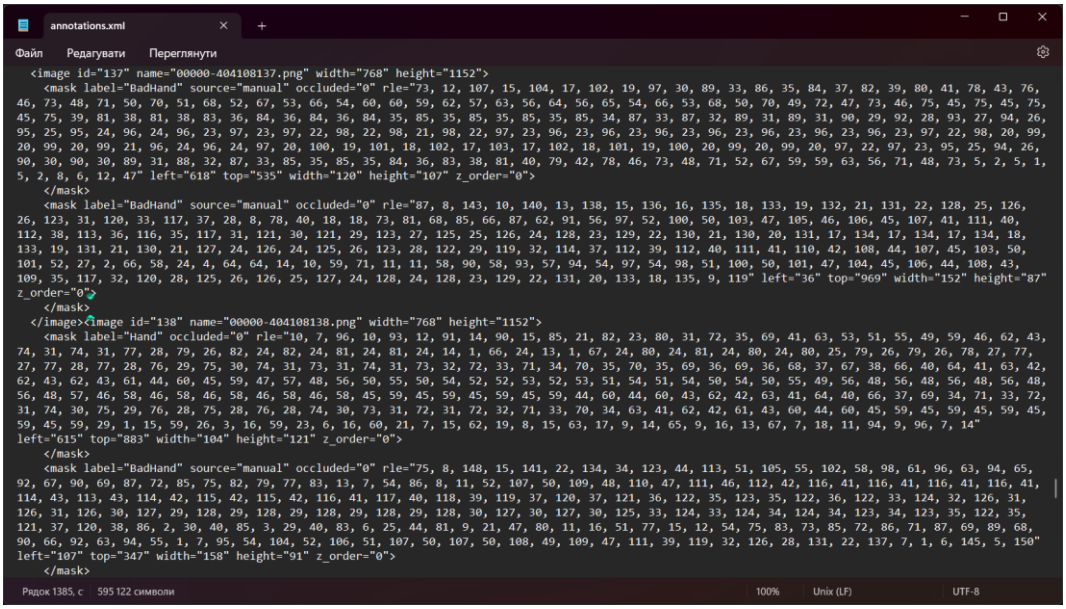


Рисунок 4.2 – Файл з анотаціями зображень

Було використано як автоматичні анотації, що створені за допомогою моделі «Human pose estimation», яка визначала руки, так і в ручному режимі, оскільки необхідно було позначити саме дефектні елементи.

Інформація, наведена в анотаціях допоможе нейронній мережі значно краще розуміти елементи, необхідні для розпізнавання, що дозволить пришвидшити процес та зменшити кількість необхідних зображень для навчання.

В підсумку було створено масив розміром 200 файлів, що є відносно середнім за величиною, необхідною для базового навчання нейронної мережі. Для створення більшого набору даних необхідно залучати інших користувачів та дослідників, що на даному етапі відкидається.

4.3 Розробка шару роботи з зображеннями

Під час виконання роботи була заново розглянута концепція роботи з файлами, відмінна від тієї, яка пропонувалась спочатку. Замість використання зберігання зображень як на диску, було прийнято рішення використовувати MongoDB. На це є декілька причин:

1) гнучкість схеми: можна легко додавати теги, опис, метадані тощо до документів;

2) масштабованість: дана база даних дозволяє легко горизонтально масштабуватись, що дуже актуально при роботі з великою кількістю зображень;

3) зберігання великих файлів: завдяки специфікації GridFS можна зберігати файли, що перевищують ліміт файлів BSON в 16мбайт.

За завантаження даних до бази відповідає роут «/upload», який зберігає файли в базу даних, а клас DataLoader відповідає за читання з різних хмарних джерел та бази даних. Основні функції даного класу відображені на лістингу нижче:

```
class DataLoader:
    def __init__(self, mongo_uri):
        self.mongo_client = MongoClient(mongo_uri)
        self.db = self.mongo_client['Diploma']


    def get_all_images_from_mongodb(self):
        images = []
        for image_doc in self.collection.find({}):
            images.append(image_doc)
        return images


    def get_image_by_id(self, image_id):
        image_doc = self.collection.find_one({'_id': image_id})
        if image_doc:
            return image_doc['image']
        else:
            return None
```

Завдяки даному класу можна одночасно отримувати зображення для відображення у веб інтерфейсі, а також для тренування нейронної мережі.

4.4 Реалізація класів для пошуку дефектів

Згідно зі структурою класів, розробленою раніше, було створено класи для тренування нейронних мереж, що шукатимуть дефекти. Один з них можна побачити на лістингу нижче:

```
    class HandsDefectCNN(nn.Module):
        def __init__(self):
            super(HandsDefectCNN, self).__init__()
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=16,
kernel_size=3, padding=1)
            self.conv2 = nn.Conv2d(in_channels=16, out_channels=32,
kernel_size=3, padding=1)
            self.pool = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(32 * 50 * 50, 512)
            self.fc2 = nn.Linear(512, 1)

        def forward(self, x):
            x = self.pool(F.relu(self.conv1(x)))
            x = self.pool(F.relu(self.conv2(x)))
            x = x.view(-1, 32 * 50 * 50)
            x = F.relu(self.fc1(x))
            x = torch.sigmoid(self.fc2(x))
            return x.squeeze()
```

Він містить в собі 2 згорткові шари, перший з 16 вихідними каналами та ядром розміром 3x3, та другий, з 32 вихідними каналами і ядром 3x3. Максимальний пулінг визначено ядром розміру 2x2, оскільки це зменшує розмір зображення вдвічі при цьому зберігаючи важливі ознаки.

Метод forward виконує прямий прохід мережі, застосовуючи при цьому згорткові шари та активуючи функцію ReLU, яка активує нейрон, якщо вхідне значення більше за нуль. Після пулінгу тензор розгортається в одновимірний вектор та проходить через пов’язаний шар. Після цього використовується сигмоїда для отримання оцінки і на виході повертається вектор оцінок. Таким чином це дозволяє оцінити як наявність дефектів, так і їх відсутність, не використовуючи вхідне зображення двічі.

Клас для пошуку дефектів очей був реалізований подібним чином.

4.5 Тестування та створення документації

Першим важливим кроком є отримання зображень з бази даних, та змінення розміру вхідного зображення шляхом відкидання непотрібної інформації. Використання фільтрів було вирішено не використовувати, наприклад, grayscale, оскільки завдяки зменшенню зображення та оптимізацій в класі нейронної мережі швидкодія є оптимальною. Завантаження зображень та розподілення необхідної інформації можна побачити на лістингу нижче:

```
    def train_hands_defect_cnn(model: HandsDefectCNN, criterion, optimizer,
num_epochs: int):
        transform = transforms.Compose([
            transforms.Resize((100, 100)),
            transforms.ToTensor(),
        ])

        data_loader = DataLoader('localhost:27017')

        train_images = []
        train_labels = []
        for image_doc in data_loader.get_all_images_from_mongo_db():
            image_readed = Image.open(image_doc['image'])
            image = transform(image_readed)

            train_images.append(image)
            train_labels.append(image_doc['labels'])
```

Тут проводиться отримання бінарних файлів і відбувається їх приведення до типу зображення, а також заповнення списків з, власне, зображеннями і мітками.

Тренування мережі нерозривно пов’язане з розрахунком втрати (англ. Loss) або ж похибки, що є різницею між прогнозованими значеннями моделі, та фактичними. Під час навчання оптимізатор використовує функцію втрати для визначення кроків, які потрібно зробити для зменшення похибки.

Задля вимірювання помилки було обрано використання BCELoss, оскільки задля класифікації використовуються лише 2 класи які позначають чи є щось

дефектним, чи ні. Хоча можна було і використати підхід з CrossEntropyLoss, який має певні переваги в тому, що йому не потрібен окремий шар активації, проте згідно з поставленою задачею перший метод є більш відповідним. Також було використано Adam оптимізатор, що оновлює ваги моделі задля зменшення кількості негативних результатів.

Також оскільки мітки є чисельно-символьними (“BadHand” і числові значення маски), необхідно було провести кодування їх до числового формату.

Результати проведеної роботи можна побачити на наступному лістингу, де також наведено цикл з епохами навчання нейронної мережі, кількість яких було встановлено числом 14, задля зменшення ймовірності недостатнього і занадто сильного навчання:

```
    images = torch.stack(train_images)

    encoder = LabelEncoder()
    encoder.fit(train_labels)
    labels = torch.Tensor(encoder.transform(train_labels))

    dataset = torch.utils.data.TensorDataset(images, labels)
    loader = torch.utils.data.DataLoader(dataset, batch_size=32,
shuffle=True)

    for epoch in range(num_epochs):
        for i, data in enumerate(loader, 0):
            inputs, labels_new = data
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            print(f"Epoch {epoch+1}, Loss: {loss.item()}")
```

Результат виконання даного коду можна побачити на рис. 4.3:

```
Epoch 1, Loss: 0.9072780754213702
Epoch 2, Loss: 0.8762236842932027
Epoch 3, Loss: 0.8740879857220563
Epoch 4, Loss: 0.7576107854098598
Epoch 5, Loss: 0.6991745568256209
Epoch 6, Loss: 0.6870743785877738
Epoch 7, Loss: 0.8917508798572864
Epoch 8, Loss: 0.5484779943999007
Epoch 9, Loss: 0.5449983185642225
Epoch 10, Loss: 0.5345221256520969
Epoch 11, Loss: 0.5304217164375512
Epoch 12, Loss: 0.5161331254904915
Epoch 13, Loss: 0.454196515691177
Epoch 14, Loss: 0.3514747684972642
```

Рисунок 4.3 – Втрати в залежності від покоління (HandsDefectCNN)

Як можна помітити, з кожним наступним поколінням кількість негативних результатів зменшувалась, хоча і був стрибок майже на екваторі. Загалом кількість втрат становить 0.35, що якщо конвертувати у відсотки, буде позначати ймовірність успіху у 65%. Цей результат може здатись незначним, проте враховуючи величину вибірки та складність навчання через елементи з дефектами і без, відсоток успіху є дійсно великим.

Щодо проведення такого ж тестування щодо класу пошуку дефектів серед очей, результат виявився кращим, а саме втрата становила 0.22, що можна побачити на рис. 4.4:

```
Epoch 1, Loss: 0.7092763527196793
Epoch 2, Loss: 0.682739118576904
Epoch 3, Loss: 0.6780129096590071
Epoch 4, Loss: 0.6633315329531316
Epoch 5, Loss: 0.6027238826550799
Epoch 6, Loss: 0.5704427947295392
Epoch 7, Loss: 0.5394526049276267
Epoch 8, Loss: 0.5367897697520103
Epoch 9, Loss: 0.4784090959932669
Epoch 10, Loss: 0.46768751236877526
Epoch 11, Loss: 0.46611465052051737
Epoch 12, Loss: 0.4295937981881569
Epoch 13, Loss: 0.34034725207306793
Epoch 14, Loss: 0.22900237187376737
```

Рисунок 4.4 – Втрата в залежності від покоління (EyesDefectCNN)

Це скоріше зумовлено тим, що пошук дефектів на очах є дещо простішим, оскільки існує менша кількість можливих артефактів.

4.6 Реалізація інтерфейсу

Після навчання нейронної мережі та її тестування, було розроблено графічний інтерфейс за споєктованими раніше макетами. Головну сторінку, на якій відображена стисла інформація про проєкт, а також скорочено те як його використовувати, зображено на рис. 4.5.

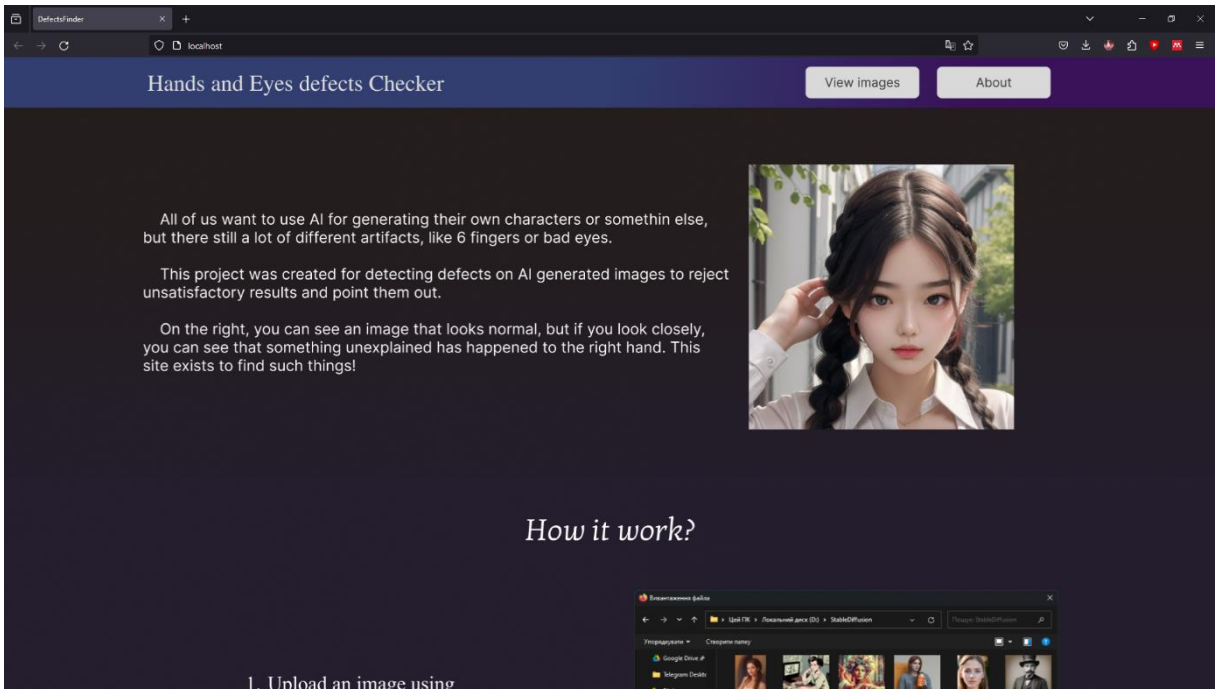


Рисунок 4.5 – Головна сторінка

Відображення ж усіх зображень, що використовувались для тренування, а також тих, які завантажує користувач, можна побачити на рис. 4.6 Серед них присутні зображення різних стилів, з різним контентом, а також з дефектами або ж ні, що зменшило ймовірність перебільшеного тренування, коли нейронна мережа завчасно знає які зображення дефектні, а які ні.

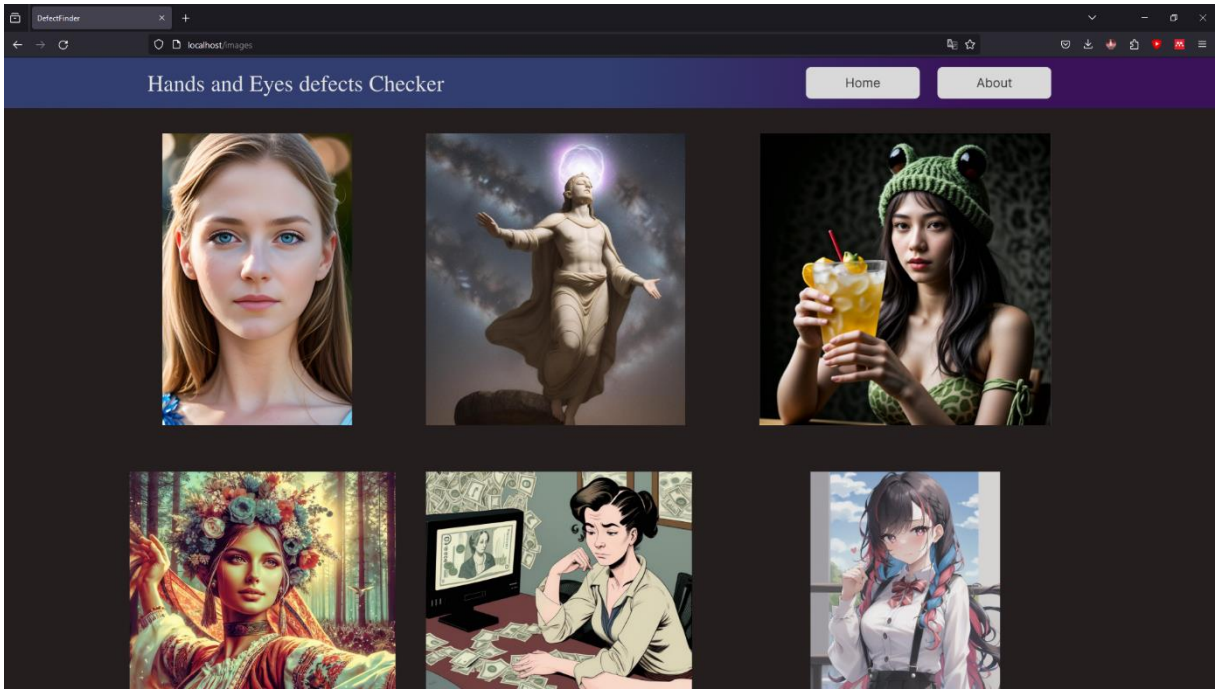


Рисунок 4.6 – Сторінка відображення усіх зображень

З цієї сторінки можна перейти безпосередньо до сторінки опису зображення, де присутні метадані, оригінальне зображення, та зображення зі знайденими елементами, дефектними або ж ні. Відображення даного інтерфейсу можна переглянути на рис. 4.7.

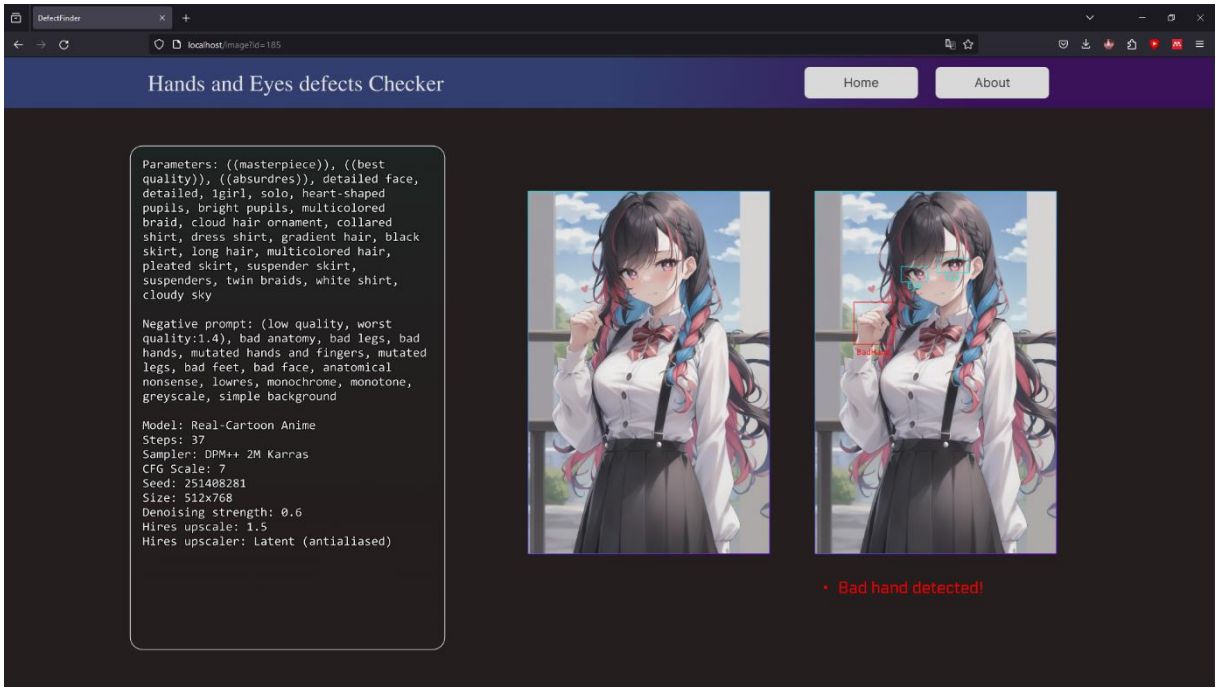


Рисунок 4.7 – Сторінка інформації про зображення

Висновки до розділу 4

В цьому розділу було розглянуто контейнеризацію системи, а саме створення окремого середовища розробника з необхідними пакетами та модулями, що дозволило спростити розгортання застосунку. Було прийнято рішення об’єднати шари з нейронними мережами і бекенд частиною, для спрощеної та швидшої взаємодії між частинами.

Було створено масив зображень в кількості 200 одиниць, кожен з яких було в автоматично ручному режимі анотовано, що дозволило в подальшому значно покращити вихідні результати навчання нейронних мереж.

Також було прийняте рішення частково відхилитись від попередньо розробленої архітектури оскільки в процесі було досліджено та виявлено значно кращі альтернативи для взаємодії та розробки, а саме MongoDB разом з GridFS. Відповідно до цього були певним чином перероблені та оновлені класи.

Основним завданням було реалізувати нейронні мережі для пошуку дефектів на зображеннях, чому й присвячена основна частина розділу. Детально наведено інформацію щодо обраних рішень, проведено пояснення основних функцій та обґрунтовано їх використання відносно інших можливих аналогів.

Було протестовано розроблене програмне забезпечення, результати чого виявились кращими за очікувані. На виході кількість успішних розпізнавань становить більше половини, а саме 65% в випадку з руками, та 78% у випадку з очима.

Останнім етапом було реалізовано інтерфейс застосунку згідно з розробленими раніше макетами, при цьому в певних аспектах покращено початковий дизайн.

ВИСНОВКИ

Результати проведеного аналізу показали, що навіть при використанні інструментів для зменшення кількості дефектів результат часто буває незадовільним, саме тому є актуальним дослідження артефактів та причин їх виникнення.

Для досягнення поставленої мети виконано наступні завдання:

– проведено аналіз предметної області та існуючих аналогів;
– обрано інструменти для подальшої роботи;
– згенеровано вибірку даних для подальшого тренування;
– реалізовано відображення знайдених дефектів;

В даній кваліфікаційній роботі магістра було розроблено програмне забезпечення для пошуку основних дефектів, які є на зображеннях, створених штучним інтелектом, а саме дефекти рук та очей. Під час виконання було проведено аналітику більшості ймовірних артефактів, оглянуто основні застосунки-аналоги та визначено основні їх переваги та недоліки. Було сформовано специфікацію вимог до розроблюваного ПЗ, згідно з якою проводилась робота.

Також було детально розглянуто та проведено порівняння різних математичних моделей, моделей кластеризації зображень, видів нейронних мереж які б найкраще підійшли до поставлених задач.

Проведено попереднє моделювання та розробка діаграм використання для застосунку, що було орієнтиром в подальшій розробці програмного забезпечення, але всі рішення, які виявлялись кращими в процесі роботи впроваджувались одразу.

Також було спроєктовано ПЗ з технічної точки зору, а саме підготовку зображень, архітектуру застосунку, та інструменти, які є основними в виконанні подібних завдань.

Проведено контейнеризацію системи, що дозволило легше й швидше розгортати застосунок, а також спростило процес розробки через наявність усього в одному місці. Було також сформовано масив зображень з анотаціями, що дозволило значно покращити процес навчання нейронної мережі та збільшити її ефективність.

Також проведено тестування та розроблена інструкція користувача, яка була інтегрована в вебзастосунок, який було реалізовано згідно з попередньо розробленими макетами.

Наведено кінцеві результати всього дослідження та розробки програмного забезпечення.

Робота пройшла апробацію під час Всеукраїнської науково-практичної конференції молодих вчених, аспірантів і студентів “Методи і засоби програмної інженерії” (Миколаїв, 31 січня – 2 лютого 2024 р.).

За результатами кваліфікаційної роботи магістра опубліковано 1 друковану працю.

Робота складається з вступу, чотирьох фахових розділів, висновків, переліку джерел посилання з 20 джерел, 1 додатку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Deep Learning Super Sampling (DLSS) Technology | NVIDIA. URL: https://www.nvidia.com/en-us/geforce/technologies/dlss/ (date of access: 15.01.2024.

2. Stable Diffusion Samplers: A Comprehensive Guide - Stable Diffusion Art. URL: https://stable-diffusion-art.com/samplers/ (date of access: 16.01.2024).

3. Matthias Bastian: Midjourney will «find you and collect that money» if you infringe any IP with v6. *The Decoder*. URL: https://the-decoder.com/midjourney-will-find-you-and-collect-that-money-if-you-infringe-any-ip-with-v6/ (date of access: 28.12.2023)

4. Image Defect Detection - oPRO.ai. URL: https://opro.ai/ai-modules/image-defect-detection/ (date of access: 10.12.2023).

5. Google Lens - Search What You See. URL: https://lens.google/ (date of access: 11.12.2023).

6. Ximilar: Image Recognition & Visual Search. URL: https://www.ximilar.com/ (date of access: 12.12.2023).

7. Compton E. A., Ernstberger S. L. Singular Value Decomposition: Applications to Image Processing. *Citations Journal of Undergraduate Research*. 2020. Vol. 17.

8. Ferrari V., Tuytelaars T., Gool L. Van. Simultaneous object recognition and segmentation from single or multiple model views. *International Journal of Computer Vision*. 2006. Vol. 67, № 2. P. 159–188.

9. Fei-Fei L., Fergus R., Perona P. “Learning generative visual models from few training examples”: An incremental bayesian approach tested on 101 object

categories. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2004.

10. Dhanachandra N., Manglem K., Chanu Y. J. Image Segmentation Using K -means Clustering Algorithm and Subtractive Clustering Algorithm. *Procedia Computer Science*. 2015. Vol. 54. P. 764–771.

11. Najman L., Challa A., Danda S., Sagar D. Power Spectral Clustering. *Journal of Mathematical Imaging and Vision*. 2018.

12. Ester M., Kriegel H.-P., Sander J., Xu X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *KDD-96: The Second International Conference on Knowledge Discovery and Data Mining*. 1996.

13. Bento C. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. *Towards Data Science.* 2021.

14. What are Convolutional Neural Networks? | IBM. URL: https://www.ibm.com/topics/convolutional-neural-networks (date of access: 17.01.2024).

15. Girshick R., Donahue J., Darrell T., Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5). *IEEE Conference on Computer Vision and Pattern Recognition.* 2014.

16. Uijlings J. R. R., Sande K. E. A. Van De, Gevers T., Smeulders A. W. M. Selective Search for Object Recognition. *International Journal of Computer Vision*. 2013.

17. Gandhi R. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithm. *Towards Data Science.* 2018.

18. Stone Z. Know Thy Selfie: A Journey Into the Uncanny Valley of AI-Generated Avatars. *The Information*. 2023.

19. CVAT. URL: https://www.cvat.ai/ (date of access: 14.01.2024).

20. COCO - Common Objects in Context. URL: https://cocodataset.org/#home (date of access: 16.01.2024).

ДОДАТОК А

Матеріали апробації роботи

Міністерство освіти і науки України
Чорноморський національний університет
імені Петра Могили

«Інформаційні технології та інженерія»

Всеукраїнська науково-практична конференція молодих вчених, аспірантів і студентів

ТЕЗИ

31 січня – 2 лютого 2024 року

Миколаїв – 2024

Методи і засоби програмної інженерії

Вебтехнології та вебдизайн

Інформаційні технології у навчальному процесі

Отже розпізнавання та сегментація об'єктів на зображеннях за допомогою комп'ютерного зору є важливими завданнями в області комп'ютерної обробки зображень та штучного інтелекту. Розвиток технології включає в себе подальше вдосконалення алгоритмів для роботи з різноманітними умовами та розширення його в нових галузях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Segmentation and object recognition using edge detection techniques / Y. Ramadevi, T. Sridevi, B. Poornima, B .Kalyani, international Journal of Computer Science & Information Technology (IJCSIT). Vol 2, No 6. 2010. P. 153-161.

2. Tetiana M., Kondratenko Y., Sidenko I. Computer Vision Mobile System for Education Using Augmented Reality Technology. Journal of Mobile Multimedia. 2021. P. 555–576.

3. Ozturk O., Sariturk B., Seker D. Z. Comparison of fully convolutional networks (FCN) and U-Net for road segmentation from high resolution imageries. International journal of environment and geoinformatics. Vol. 7, Issue 3. P. 272–279.

УДК 004.8

Фіник В. Ю., Кандиба І. О.
Чорноморський національний університет імені Петра Могили
м. Миколаїв, Україна

ПОШУК ДЕФЕКТІВ НА ЗОБРАЖЕННЯХ, СТВОРЕНИХ ЗАСОБАМИ ШТУЧНОГО ІНТЕЛЕКТУ

Нейронні мережі стають доволі популярнішими, оскільки вони значно пришвидшують людську взаємодію з оточенням та інформацією. Їх можна класифікувати за різними критеріями та функціоналом і в цій роботі будуть розглянуті генеративні штучним інтелектом, а саме для створення різного роду медіа контенту. Їх можна використовувати для ілюстративних зображень, наприклад, коли використання справжніх людей може поставити їх під загрозу, для реклами та просування контенту, для відображення свого аватару в мережі та різного роду комерції. Це доволі перспективна галузь, оскільки хороших Prompt Engineer зараз не так багато.

Якщо ж говорити власне за нейромережі, то є декілька якими можна користуватись без складних маніпуляцій, а саме Midjourney (платна, підписка), DALL-E (інтегрована в ChatGPT-4, платна, система купівлі кредитів).

Окремо треба виділити Stable Diffusion оскільки вона безкоштовна, але при цьому вимагає певних навичок для початку роботи, а саме її запуску, налаштувань і встановленні доповнень, проте через це вона є більш гнучкою, оскільки можна краще контролювати процеси. Всі ШІ доволі непогано справляються зі своїми задачами і з кожним оновленням рівень похибки стає меншим, але це все за умови узагальнених запитів. Якщо ж є бажання створити доволі конкретне зображення, то не завжди результат може бути задовільним. Зазвичай це погана композиція, не реалістичність або ж зображення не співпадає з уявою автора. Наприклад, існує ComfyUI – що є одночасно і графічним інтерфейсом і бекендом для Stable Diffusion. Завдяки йому можна значно простіше коригувати свої запити і переглядати результати.

Для відстеження даних дефектів людського бачення буває замало, тому було прийнято рішення розробити програмне забезпечення, яке б допомогло спростити взаємодію користувача з штучним інтелектом і покращити кінцевий результат шляхом зменшення загального відсотку дефектів.

Розробники Stable Diffusion надали певний функціонал для можливості зменшення дефектів, наприклад, можна використовувати негативні запити, на кшталт bad_hands_v3. В даному дослідженні планується використання комп'ютерного зору для відстеження дефектів з подальшим проведенням декомпозиції процесу створення зображення для підбору оптимальних аргументів і їх ваги задля отримання бажаного результату.

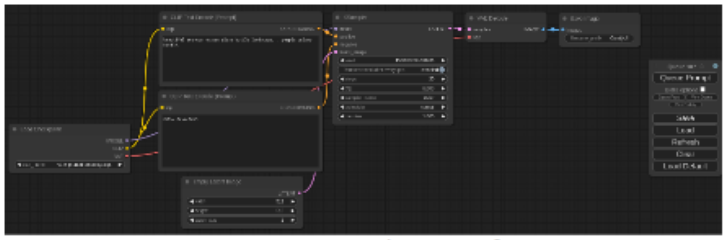


Рисунок 1 – Інтерфейс ComfyUI

Оскільки дефекти можуть бути доволі різними, було обрано рішення сконцентрувати увагу на таких як: аномалії частин тіл (пальці, руки, ноги) та аномалії обличчя.

Існують деякі дослідження які описують алгоритми для оцінки жестів рук [1]. Для цього є доцільним використання глибоких згорткових нейронних мереж, оскільки вони здатні автоматично навчатись та виділяти ієрархічні ознаки вхідних даних. Зокрема в

даному випадку вони можуть розрізняти складні деталі та варіації жестів при зміні орієнтації рук. Дану концепцію можна розширити для використання і з іншими частинами тіла.

Існують дослідження, які пропонують використання Кругового Перетворення Хафа для пошуку очей, але в даній роботі його використання потребує вдосконалення, оскільки обличчя може мати різні кути і самі зіниці можуть мати пошкодження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ben-Ari L., Ben-Ari A., Hermon C., Puad Ismail A., Athirah Abd Aziz F., Mohamat Kasim N., Daud K. Hand gesture recognition on python and opencv. *IOP Conference Series: Materials Science and Engineering*. 2021. Vol. 1045, № 1. P. 012043.

Вебтехнології та вебдизайн

УДК 004.4

Ільчишина Ю. В., Швайко В. К.
Хмельницький національний університет,
м. Хмельницький, Україна

РОЗРОБКА ДИЗАЙНУ ІНТЕРФЕЙСУ КОРИСТУВАЧА МОБІЛЬНОГО ЗАСТОСУНКУ СПОРТ КОНЕКТ

Ідея полягає у розробці дизайну інтерфейсу користувача мобільного застосунку Спорт Конект для імплементації результатів дослідження, представлених у [1].

Перше вікно (рис. 1а), яке зустрічає користувача і очікує дій – це стартове вікно застосунку «Спорт конект». Тут розташовані дві кнопки, перша «Реєстрація» та друга «Вхід», також наявне лого проєкту. Важливою частиною екрану є лого Хмельницького Національного Університету, адже реалізацією проєкту займаються викладачі та студенти Хмельницького Національного Університету.

Вікно реєстрації (рис. 1б), є важливим етапом у даному застосунку, саме тут користувач заповнює потрібну інформацію про себе, таку як електрону адресу, ім'я, прізвище, вік, школу, клас, стать, та вводить пароль. Для зручності користувача у кожному полі, де потрібно ввести інформацію про себе, присутні підказки сірого кольору, в якому форматі має бути заповнене поле. Не менш важливим етапом реєстрації у застосунку є ознайомлення з політикою конфіденційності.