МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження алгоритмів швидкого пошуку фотографій у Data Lake сховищах даних

Виконав:
Студент(ка) 2 курсу, групи ІПЗМ-20-1
Олександр Прокопенко
(прізвище, ініціали)

Спеціальність	121 – Інженерія програмного забезпечення
Тип програми	освітньо – наукова
Керівник	проф. Смеляков К.С.

Допускається до захисту
Зав. Кафедри _____ З.В. Дудар

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_ _» _____ 202_ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Прокопенка Олександра Сергійовича

(прізвище, ім'я, по-батькові)

1. Тема роботи «Дослідження алгоритмів швидкого пошуку фотографій у Data Lake сховищах даних»

затверджена наказом університету від «___» _____ 202_ р. №___

2. Термін подання студентом роботи до екзаменаційної комісії «__» _____ 202__ р.

3. Вихідні дані до роботи Датасет cocoval, EXIF метадані зображення, модель машинного навчання YOLOv5, мова програмування Python, Data Lake архітектура.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної галузі та постановка задачі, опис ідеї і алгоритму, проектування і розробка програмного модуля, проектування і проведення експериментів, аналіз результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Пошук інформації та аналогічних рішень	25.09.2021	виконано
2	Розробка плану дослідження	18.10.2021	виконано
3	Вибір датасету для дослідження	1.11.2021	виконано
4	Вибір моделі для дослідження	6.11.2021	виконано
5	Аналіз предметної галузі та постановка задачі	20.11.2021	виконано
6	Опис ідеї та алгоритму	22.12.2021	виконано
7	Проектування програмного модуля	14.01.2022	виконано
8	Розробка програмної системи	25.01.2022	виконано
9	Проектування і проведення експериментів	22.02.2022	виконано
10	Аналіз отриманих результатів	25.02.2022	виконано
11	Підготовка пояснювальної записки	20.04.2022	виконано
12	Захист роботи	21.05.2022	

Дата видачі завдання _____ 202 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Смеляков К.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 69 стор., 46 рис., 10 джер.

ДЕТЕКЦІЯ ОБ’ЄКТІВ, НЕЙРОННІ МЕРЕЖІ, YOLOV5, МЕТАДАНІ, JPEG, ПОШУК ЗА ФОТОГРАФІЯМИ, DATA LAKE АРХІТЕКТУРА.

Об’єктом дослідження даної роботи є швидкий пошук за фотографіями з детекцією об’єктів на зображеннях з використанням нейронних мереж та метаданих файлу, що можуть бути використані у великих сховищах даних.

Метою роботи є забезпечення ефективності порівняння та пошуку зображень за їх змістом (за критеріями трудомісткості та точності) у сховищах великих даних.

Результатом роботи має стати програмний модуль, що використовуючи попередню обробку зображень за допомогою нейронних мереж для детекції об’єктів на знімках, зможе виконувати швидкий пошук за описом об’єктів, а також пошук найбільш подібних фотографій за змістом у великих сховищах даних.

OBJECT DETECTION, NEURAL NETWORKS, YOLOV5, METADATA, JPEG, IMAGE SEARCH, DATA LAKE ARCHITECTURE.

The object of this study is to quickly search for photographs with the detection of objects in images using neural networks and file metadata that can be used in a big data storages.

The aim of the work is to ensure the efficiency of comparison and search for images by their content (according to the criteria of labor intensity and accuracy) in big data storages for such conditions.

The result should be a software module that uses pre-processing of images using neural networks to detect objects in images, will be able to search quickly by depicted objects description, as well as to compare and find the most similar photos in a big data storages.

Я, Прокопенко Олександр Сергійович
(прізвище, ім’я, по-батькові)
студент(ка) групи ІПЗм-20-1 здобувач вищої освіти на другому (магістерському) рівні

кафедра програмної інженерії,
(повна назва кафедри)

заявляю: моя кваліфікаційна робота на тему
Дослідження алгоритмів швидкого пошуку фотографій у Data Lake сховищах даних,

що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Кожного дня у світі збільшується поток інформації, щодня породжуються петабайти нових даних від користувачів у вигляді текстових повідомлень, мільйонів фото, відео та інших різноманітних та різнопланових даних. В умовах такого швидкого наповнення інформацією світових програмних систем дуже важливою задачею є збереження можливості швидкого орієнтування у виникаючих величезних масивах даних та пошуку необхідної інформації в них за прийнятний час. Тренд останніх років вказує на те, що успішна робота з обробкою великих даних тісно пов’язана з використанням технологій машинного навчання та штучного інтелекту, бо саме вони дозволяють виконувати необхідну обробку в умовах надходження гігабайтів інформації за секунду.

Метою кваліфікаційної роботи є створення програмного модуля, що з використанням технологій машинного навчання, а саме нейронних мереж дозволить виконувати попередню обробку зображень при надходженні до системи зі знаходженням об’єктів на знімку, їхнього положення та розмірів і дозволить записувати цю інформацію до метаданих зображення, що зможуть бути використані пізніше на етапі коли користувач захоче виконати пошук за зображеннями, тоді надане зображення так само обробляється мережею і за знайденими об’єктами виконуватимемо порівняння зображення з зображеннями у системі. Порівняння зображень проходитиме двоетапно, спочатку відсіюватимемо фотографії за співпадінням типів і кількості об’єктів певного типу на двох фотографіях. Якщо результати пошуку будуть вичерпні, зможемо зупинитися на цьому етапі, в іншому ж випадку виконаємо другий етап порівняння для фото, що залишилися – порівняння за розташуванням об’єктів на знімку та перетином площ цих об’єктів видаючи схожість за шкалою від 0 до 1. Такий програмний модуль можна буде використовувати з будь-яким хмарним провайдером.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Сучасні додатки на сьогоднішній час оперують з великими потоками інформації, що кожного дня надходять до програмних систем. Часто важко сказати чи є певне фото, або схоже на нього у базах даних додатку. Для вирішення цієї задачі використовують пошук за зображеннями, що виконує порівняння зображень і видає найбільш схожі результати, такі операції виконуються зазвичай з використанням згорткових нейронних мереж та індексації картинок, як це наприклад зроблено для сервісу Google Images.

Сьогодні значна частина додатків має розподілену хмарну або гібридну хмарну архітектуру. Кожен із великих хмарних провайдерів надає своє власне API та свою власну реалізацію роботи зі збереженням зображень та управлінням ними, в таких умовах непростою задачею створення модуля, що може бути використано незважаючи на провайдера та особливості реалізації кожного конкретного продукту.

Перспективним напрямком організації простору збереження даних є організація у вигляді Data Lake, коли дані різних типів від повністю «сирих», як скажімо файли зображень чи документів, до напівструктурованих, як файли json, або xml і навіть повністю структурованих, як таблиці SQL зберігаються разом у єдиному репозиторії, де знаходяться усі дані додатку, або певної компанії. Такий підхід спрощує роботу з попередньою обробкою і дозволяє швидше та ефективніше додавати і змінювати дані у системі не виконуючи купи супутніх задач, що пов’язані зі структурованим зберіганням даних. Реалізація пошуку за зображеннями для таких систем допоможе швидко знаходити у системі схожі фото та у випадку наявності повного співпадіння видавати їх, дозволяючи приймати рішення щодо доцільності збереження поточного фото.

В даний час спостерігається експоненційна тенденція зростання кількості та обсягу персональних, корпоративних і комерційних сховищ зображень. Багато таких сховищ, наприклад фотобанки, містять мільйони зображень.

Ефективність системи управління такими сховищами безпосередньо залежить від ефективності порівняння та пошуку зображень під час пошукових запитів. З кожним днем користувачів все більше цікавить зміст зображення, а не його формальні параметри. Користувачі намагаються включати цю інформацію у свої пошукові запити. У такому випадку алгоритми пошуку повинні ефективно шукати зображення за вказаною інформацією про вміст зображень.

І така обробка зображень повинна здійснюватися ефективно (за критеріями трудомісткості та точності) за наявності мільйонів зображень у сховищі. У зв’язку з цим основна проблема пов’язана із забезпеченням ефективності порівняння та пошуку зображень за їх змістом за наявності на зображенні великої кількості однотипних об’єктів, області локалізації яких багаторазово перекриваються.

Зробимо огляд наявних аналогів.

1.2 Огляд наявних аналогів

До наявних аналогів одразу можна віднести сучасні пошукові системи за фотографіями великих корпорацій.

Google Images – спеціальний сервіс Google для пошуку картинок в Інтернеті. Googlebot-Image, пошуковий робот, що сканує сторінки для індексу картинок, здійснює пошук зображень різних форматів (JPEG, GIF, PNG, BMP, SVG, WebP, ICO).

Сервіс було відкрито у липні 2001 року. Тоді пошуковий робот проіндексував 250 мільйонів картинок. До лютого 2004 року робот проіндексував 880 мільйонів картинок, до лютого 2005 року - 1,1 мільярд картинок. 31 липня 2009 року до Google Images додано можливість пошуку схожих зображень[1].

Алгоритм використовує машинне навчання, в якому Google Images вчиться пов'язувати певні зображення одне з одним для створення кластерів, щоб забезпечити функцію зворотного пошуку зображень. Після надсилання запиту служба повертає набір мініатюрних зображень, які відповідають вашому ключовому слову.

Можливо, найпотужніша функція Google Images — зворотний пошук зображень, який використовує зображення як пошуковий термін.

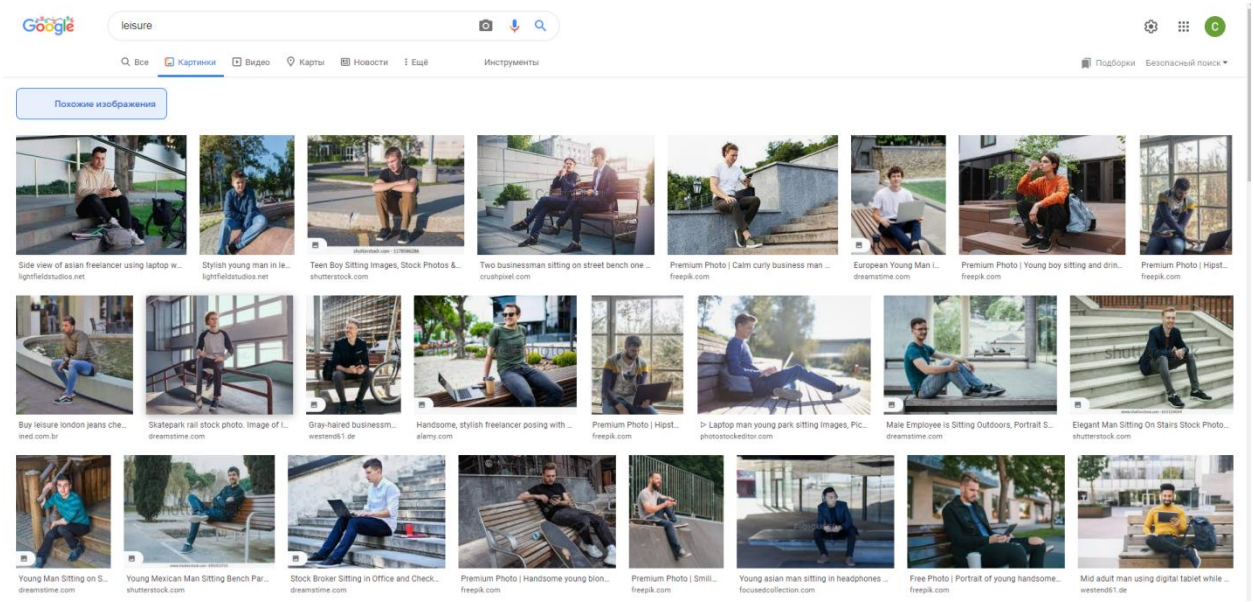

Рисунок 1 – Google Images UI

Зворотний пошук зображень, подібний до цього, може повернути два різні набори результатів:

– початковий веб-сайт: може повертати вихідні веб-сайти, на яких можна знайти зображення, а також будь-які імена або описи, пов'язані із зображенням. Це корисно, якщо у вас є зображення, але хочете знати, звідки воно;
– схожі зображення: Зворотній пошук може також показувати візуально схожі зображення. Наприклад, ви можете виконати зворотний пошук за зображенням гори, щоб побачити інші схожі шпалери гори.

Яндекс.Картинки – це аналогічний сервіс від Яндекс, що дозволяє окрім аналогічного функціоналу, що й Google Images має додаткові функції, що використовують для пошуку додаткову інформацію, що розташована на знімку.

Наприклад, можна виконати пошук товарів, що зображено на фотографії. Для зображених на фото людей можна знайти предмети одягу, які вони вдягають

або схожі на них на інтернет-майданчиках. Для зображених квартир або будівель можна знайти де купити предмети інтер’єру.

Можемо виконати постановку задачі.

1.3 Постановка задачі

Спробуємо підсумувати проведений аналіз предметної галузі та огляд наявних на ринку аналогів і сформуємо в результаті специфікацію задачі, що має бути реалізована у даній роботі.

Розроблюваний у цій роботі програмний продукт має відповідати таким вимогам.

Створений програмний модуль повинен використовувати технології машинного навчання, а саме нейронні мережі родини YOLO для пошуку об’єктів на фотографії. YOLO це акронім для 'You only look once', в перекладі «ти дивишся лише один раз», це алгоритм знаходження об’єктів, що розділяє зображення на сіткову. Кожна клітина в сітці, відповідає за знаходження об’єктів всередині себе. YOLO це один з найбільш відомих алгоритмів детекції об’єктів, відомий своєю швидкістю та точністю.

Виданий нейронною мережею результат у вигляді декількох значень, що позначають кожен знайдений на картинці об’єкт складатиметься з мітки класу, тобто тип знайденого об’єкта, наприклад людина, собака, тощо. 4 числа, що позначають координати x та y знайденого об’єкта та його розміри: висоту та ширину, додатково можна зберігати впевненість мережі в тому, що знайдений об’єкт відповідає певному класу, тобто значення від 0 до 1, для складніших варіантів алгоритму можна задіяти для більш точного пошуку.

Для зберігання даних про знайдені об’єкти система використовуватиме метадані файлів зображення. Більшість сучасних форматів зображень відповідають стандарту Exif (Exchangeable Image File Format), що дозволяє додавати до зображень та інших медіафайлів додаткову інформацію (метадані), коментує цей файл, що описує умови та способи його отримання, авторство тощо[2].

Цей стандарт специфікує зокрема декілька полів, які можуть бути використані для збереження будь-якої довільної інформації, що визначена користувачем, тому може бути використана в тому числі для зберігання інформації про зображені на фото об’єкти.

Модуль зможе виконувати пошук по фотографіям, що зберігаються у системі. Для цього фотографія, що є предметом пошуку пройде процедуру детекції об’єктів так само як і збережені у системі фото, після цього порівняння з фотографіями, що зберігаються у системі за алгоритмом пошуку ранжуватиме наявні фотографії за коефіцієнтом схожості у проміжку від 0 до 1.

Алгоритм пошуку потрібно реалізувати у декількох формах, що дозволить виконувати простіший та більш складний пошук, також у якості певної евристики можна використовувати простіший алгоритм для попередньої фільтрації і застосовувати складніший алгоритм до результатів простішого.

В якості простішого алгоритму треба взяти порівняння за кількістю та типом об’єктів на знімку, такий алгоритм навряд чи видасть корисні результати для користувача, оскільки усі зображення де є лише одна людина будуть однаково схожі між собою, проте як зазначено раніше таке відсіювання дозволить прибрати усі точно несхожі варіанти.

Складніший алгоритм включає до себе порівняння перетинів площ об’єктів. Тобто об’єкти з різних фотографій однакового типу порівнюватимуться між собою за коефіцієнтом c_o, що розраховується за наступною формулою:

$$c_o = \frac{S_{\text{пер}}}{S_{\text{заг}}},$$

де c_o – площа перетину двох об’єктів;

$S_{\text{пер}}$ – площа перетину двох об’єктів;

$S_{\text{заг}}$ – загальна площа двох об’єктів.

Тоді коефіцієнт схожості зображення дорівнюватиме:

$$c = \frac{\sum_1^n c_i}{n},$$

де c – коефіцієнт схожості двох зображень;

n – кількість об’єктів на знімку з більшим числом об’єктів;

c_i – коефіцієнт схожості і-го об’єкту з аналогом з іншого зображення.

Таким чином отриманий коефіцієнт матиме перестановочну властивість, тобто порівняння зображення a з зображенням b матиме такий самий результат, як і порівняння b з a. З отриманих із застосування алгоритму коефіцієнтів відсортовуватимемо наявні зображення за схожістю і обиратимемо за деяким порогом найбільш схожі варіанти. Найбільш схожі результати повертатимемо користувачу.

2 ОПИС ІДЕЇ ТА АЛГОРИТМУ

2.1 Опис проблеми яку вирішує програмний продукт

В сучасних програмних системах дуже часто виникає ситуація, коли в процесі роботи виникає потреба опрацьовувати гігабайти вхідної інформації та часто орієнтуватися у великих обсягах є непростою задачею. Наприклад для Data Lake архітектур зберігання усіх даних додатку відбувається у єдиному репозиторії, де разом зберігаються дані різних форматів, це можуть бути сирі дані, накшталт фото або відео файлів, напівструктуровані дані, що зазвичай зберігаються у додатках у вигляді JSON або XML документів, а також повністю структуровані дані, такі як SQL-таблиці баз даних.

В якості сховища для Data Lake систем можуть бути використані різноманітні хмарні провайдери, найбільші з яких є Microsoft Azure, Amazon S3, Google Cloud. У якості сховищ можуть бути використані storage account у Microsoft Azure, для яких налаштовуються канали отримання даних (зазвичай їх декілька, наприклад показники певних датчиків, пошта до розсилання, дані, що користувачі хочуть зберігати у системі). За схожою схемою для Data Lake сховища можна налаштувати бакети S3, для того, щоб зберігати в сирому вигляді дані системи, а також сховище Google Cloud. Кожен з провайдерів пропонує власні інструменти для зберігання та зручного управління даними їх надходженням, зберіганням, обробкою та архівуванням, або видаленням з системи.

Для вищеописаних хмарних систем дуже часто корисною є можливість дізнатися, чи зберігає система певне фото, чи маємо ми додавати її повторно, або виконати пошук за фотографіями та швидко отримати схожі варіанти, для того, щоб переглянути які користувачі, в який час та за яких обставин додавали чи використовували схожі зображення.

За такого різноманіття провайдерів та інструментів зручним буде модуль, що можна використовувати незважаючи на конкретного провайдера. Цей модуль лише потрібно буде застосувати при попередній обробці зображень, оскільки ми зможемо використати метадані зображення для збереження даних про об’єкти, що

знайдені на зображенні. Це дозволить виконати пошук за зображеннями, що зберігаються у системі в будь-який час за необхідності[3].

Опишемо загальний алгоритм.

2.2 Опис алгоритму роботи

В даній роботі використовуються зображення у форматі JPEG, одному з основних форматів зберігання та передачі даних для зображень на сьогоднішній день.

Цей формат використовує як власні метадані специфіковані своїм форматом, так і підтримує стандарт метаданих Exif, який реалізують також інші відомі формати зберігання зображень, наприклад PNG. Цей стандарт специфікує метадані, що можуть бути використані для опису зображення, зокрема він описує поле для метаданих “UserComment” – воно може бути використане для будь-якого довільного опису змісту файлу, його розмір сягає 64 кілобайт, тобто доволі простору для запису будь-якої описової інформації про зображення.

У цій роботі планується запис у метадані даних про об’єкти, що знаходяться на зображенні у вигляді декількох ліній, кожна з яких описує один об’єкт на знімку.

Опис об’єкта матиме наступний формат:

cls x y w h,

де ***cls*** – мітка класу, що позначає тип знайденого на знімку об’єкта;

x – координата x об’єкта на знімку, що позначає центр об’єкта;

y – координата y об’єкта на знімку, що позначає центр об’єкта;

w – ширина об’єкта на знімку;

h – висота об’єкта на знімку.

Ми будемо вирішувати проблеми порівняння зображень за допомогою метаданих зображень і трьох різних алгоритмів порівняння.

Перш за все, варто зазначити, що після того, як зображення розмічено та позначено даними зображуваних об’єктів, стає відносно легко відфільтрувати велику кількість зображень, залишаючи лише підмножину, яка наприклад зображує собаку та людину.

Усі алгоритми порівняння зображень використовують каскадний підхід і дають нам оцінку подібності двох зображень, як коефіцієнт між 0 і 1 включно, де 0 - це порівняння зображень, які зображують набори об'єктів, які не перетинаються, а 1 - результат порівняння зображення з самим собою.

Почнемо з найпростішого тривіального алгоритму, за допомогою якого ми будемо порівнювати дві фотографії. Ми маємо фотографію A, яка має набір виявлених об’єктів A = { O_{11}, O_{12}, … O_{1N} } і фотографію B, яка має набір виявлених об’єктів B = { O_{21}, O_{22}, … O_{2M} }.

Перший крок – згрупувати об’єкти за міткою класу та підрахувати кількість об’єктів кожного класу, які містять фотографії. Взагалі кажучи, у нас буде два словники, де ключі — це позначки класів, а значення — кількість об’єктів на зображенні.

Ми хочемо, щоб результат порівняння A з B був таким же, як порівняння B з A.

Щоб досягти цього, ми візьмемо список різних міток класів, які можна знайти на двох зображеннях.

Після цього ми можемо обчислити подібність двох зображень за міткою кожного класу.

Розрахунок настільки простий, наскільки це можливо, одне зображення буде містити меншу або рівну кількість об’єктів з певною міткою класу, ніж інше, тому ми поділимо це значення на кількість об’єктів з цією міткою класу на іншому зображенні.

Ми припускаємо, що всі мітки класів мають однакову вагу, тому для обчислення остаточного коефіцієнта подібності нам потрібно підсумувати всі подібності за мітками і розділити цю суму на кількість різних міток класів.

Це виражається у вигляді наступної формули:

$$S = \sum_{i=1}^{n} w * \frac{min(O_1^i, O_2^i)}{max(O_1^i, O_2^i)},$$

де S – коефіцієнт подібності двох зображень;

n – кількість різних типів об’єктів на двох зображеннях;

w – ваговий коефіцієнт типу (розраховується як 1 / n);

O_1^i – кількість об’єктів і-го типу на першому зображенні;

O_2^i – кількість об’єктів і-го типу на другому зображенні.

Цей алгоритм також буде використовуватися як попередній фільтр до двох інших алгоритмів, оскільки він не містить важких обчислень і може видаляти всі зображення, які не мають жодного перетину з нашим цільовим зображенням з точки зору об’єктів, тому не має жодного сенсу порівнювати розташування об’єктів на цих зображеннях.

Коли ми говоримо про сотні тисяч зображень у базі даних, це скоротить набір зображень, який слід порівняти з кількома відсотками від початкової кількості зображень залежно від порогу, який використовується для фільтрації.

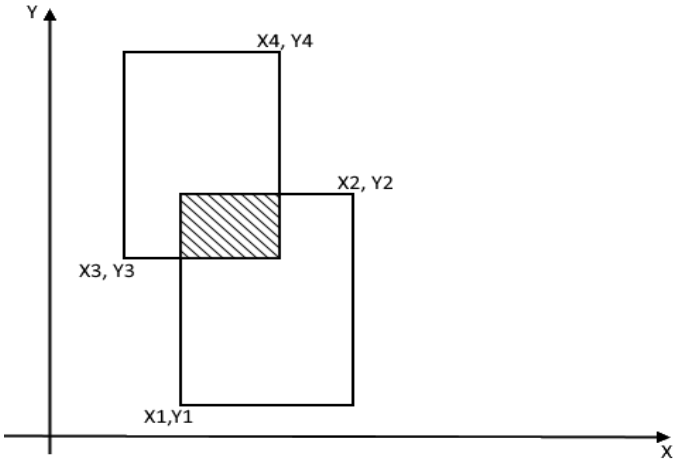

Рисунок 2 – Площа перетину двох прямокутників

Цей алгоритм також буде використовуватися як попередній фільтр до двох інших алгоритмів, оскільки він не містить важких обчислень і може видаляти всі зображення, які не мають жодного перетину з нашим цільовим зображенням з

точки зору об’єктів, тому не має жодного сенсу порівнювати розташування об’єктів на цих зображеннях.

Коли ми говоримо про сотні тисяч зображень у базі даних, це скоротить набір зображень, який слід порівняти з кількома відсотками від початкової кількості зображень залежно від порогу, який використовується для фільтрації.

Другий алгоритм фокусується на порівнянні двох зображень залежно від розташування об'єктів. Так само, як і в першому алгоритмі, ми будемо порівнювати об’єкти одного типу, але тепер порівняємо не тільки їх кількість на обох зображеннях, а й площу перетину двох об’єктів.

Щоб обчислити коефіцієнт між двома об’єктами, нам потрібно обчислити площу їх перетину, що є тривіальним завданням, враховуючи, що обмежувальні прямокутники є прямокутниками, а отриманий перетин також є прямокутником.

Наведемо формули для розрахунку площі перетину та загальної площі двох прямокутників, адже саме такого вигляду набувають бокси об’єктів на зображенні. Спочатку обрахуємо координати лівої верхньої точки та правої нижньої точки для обох прямокутників. Вони відповідно дорівнюють:

$$P_1 = \left(x - \frac{w}{2}, y - \frac{h}{2}\right), P_2 = (x + \frac{w}{2}, y + \frac{h}{2})$$

Тоді для площі перетину вираховуємо, як

$$S_{\text{пер}} = \left(MIN(P_2^1.x,\ P_2^2.x) - MAX(P_1^1.x,\ P_1^2.x)\right) * \left(MIN(P_2^1.y,\ P_2^2.y) - MAX(P_1^1.y,\ P_1^2.y)\right)$$

І звідси просто знаходимо загальну площу двох прямокутників, як:

$$S_{\text{заг}} = S_1 + S_2 - S_{\text{пер}}$$

Введемо коефіцієнт схожості двох об’єктів.

$$Is(O_1, O_2) = \frac{S_{пер}}{S_{заг}}$$

Тоді розраховуємо схожість двох зображень за цим алгоритмом наступним чином.

$$S = max\left(\sum_{i=1}^{n} w * Is\left(O_i, O_j^2\right)\right),$$

де S – коефіцієнт подібності двох зображень;

n – кількість об’єктів на зображенні, що містить більшу кіл-ть об’єктів;

w – ваговий коефіцієнт типу (розраховується як 1 / n);

O_i – і-ий об’єкт на порівнюваному зображенні;

O_j^2 – об’єкт з другого зображення того самого типу.

Ідея цього алгоритму полягає в тому, щоб знайти пару одного типу для кожного виявленого об'єкта на зображенні та знайти їх коефіцієнт перетину (0, якщо парний об'єкт не існує або об'єкти не перетинаються). У випадку, якщо є кілька об’єктів одного типу, ми будемо обчислювати коефіцієнти попарно та вибирати пари, щоб максимізувати загальний коефіцієнт.

Один раз обчислюємо коефіцієнт для кожної пари об’єктів певного типу. Тоді ми повинні вибрати пари об’єктів, які максимізують суму коефіцієнтів. Ця процедура повторюється для кожної мітки класу, і загальна сума коефіцієнтів ділиться на кількість об’єктів на зображенні. Цей алгоритм дозволяє отримати більш точні результати, ніж перший, оскільки враховує розташування об’єктів, крім їх класів і номерів на зображенні.

Цей алгоритм містить деякі істотні недоліки, які можуть призвести до не дуже точних результатів і повільного виконання. Цей алгоритм найкраще працює, коли на зображенні не так багато об’єктів одного типу. У випадку, якщо у нас є дві групові фотографії, точні зони перетину пар людей на двох фотографіях можуть бути невеликими, в той час як загальна зона зайнята сфотографованими

людьми значна. Друга проблема - це питання часу виконання, оскільки якщо на обох зображеннях є десятки об'єктів одного типу, то пошук максимальної суми коефіцієнтів може зайняти значний час, оскільки це рішення задачі присвоєння, яке має часову складність $O(n^3)$.

Третій алгоритм також враховує розташування об'єктів і усуває недоліки, які має алгоритм порівняння боксів. Замість того, щоб порівнювати конкретні об’єкти, ми будемо порівнювати ділянки, покриті об’єктами певного типу на обох зображеннях. Для обчислення площ перетинання об’єктами на двох зображеннях будемо використовувати матрицю з розміром зображення, де значення кожного пікселя в матриці дорівнює кількості зображень, які містять об’єкт цього типу в розташуванні пікселя. Приклад наведено на рисунку 3.

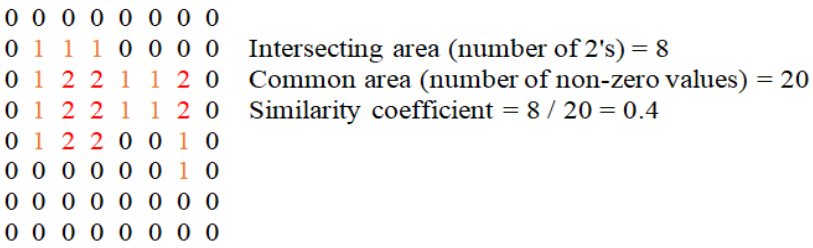

Рисунок 3 – Приклад підрахунку матричного коефіцієнта схожості

Щоб розрахувати коефіцієнт подібності за типом об’єкта за допомогою цієї матриці, нам потрібно обчислити кількість двійок в матриці, яка представляє області перетинання та кількість ненульових значень, які будуть представляти загальну площу об’єктів на обох зображеннях, тоді ми отримаємо коефіцієнт шляхом поділу площі перетинання на загальну площу.

Основною складністю цього алгоритму є правильна реалізація, яка дозволить розраховувати коефіцієнти за прийнятний час. Першою проблемою є порівняння зображень з високою роздільною здатністю, якщо матриця містить кілька тисяч або навіть мільйонів значень, її сканування займає багато часу.

Ми можемо вирішити цю проблему за допомогою масштабування. Зображення містять інформацію про виявлені об’єкти у своїх метаданих як

відносні значення. Таким чином, ми можемо стиснути матрицю до певного невеликого розміру, який буде досить швидким для обчислення. Стиснення супроводжується втратою точності, але, як показує практика, різниця в результатах обчислення для стиснутих зображень і повнорозмірних зображень становить лише кілька відсотків. Далі в експериментальній частині цієї статті ми покажемо, що зміною результатів можна знехтувати, а точність досить висока, навіть коли коефіцієнт стиснення становить всього 0,2 від початкового розміру.

$$S = \sum_{i=1}^{n} w * \frac{S_{int}}{S_1^i + S_2^i - S_{int}},$$

де S – коефіцієнт подібності;

n – кількість різних типів об’єктів на обох зображеннях;

w – ваговий коефіцієнт типу (розраховується як 1 / n);

S_1^i – площа яку об’єкти i-го типу займають на першому зображенні;

S_2^i – площа яку об’єкти i-го типу займають на другому зображенні;

S_{int} – площа перетину об’єктів i-го типу.

Ідея цього алгоритму полягає в тому, щоб обробляти зображення з великою кількістю об’єктів одного типу, де перекриття окремих об’єктів не настільки важливе, як перекриття кластерів об’єктів загалом. Потім подібність розраховується як середнє подібностей за різними типами.

Час виконання цього алгоритму можна ще більше покращити. Алгоритми матричного порівняння можна паралельно використовувати з великим покращенням часу виконання.

Перш за все, одночасне виконання може бути реалізовано на кількох рівнях обчислень.

Найвищий рівень - це рівень типу об'єкта, порівняння більшої частини зображень дозволить виявити кілька типів об'єктів, і всі коефіцієнти подібності за типом об'єкта можна зробити незалежно, що означає, що час обчислення в ідеальних умовах скорочується від суми разів до обчислення подібності на всі

типи об'єктів до часу, який займає одне найдовше обчислення коефіцієнта подібності за типом об'єкта.

Наступним можливим покращенням є паралельне сканування матриці. Оскільки матриця має прямокутну форму, ми можемо розбити її на менші сегменти та делегувати обчислення площ перетину та загальної площі окремим процесам.

Перейдемо до вибору моделі.

2.3 Опис використаної моделі для детекції об’єктів

Першою та основною частиною роботи є попередня обробка зображень з детекцією об’єктів на ньому та записом результатів до метаданих поданого на обробку зображення.

У цій роботі пропонується використовувати нейронну мережу родини YOLO останньої версії YOLOv5.

YOLO, абревіатура від «Ти дивишся лише один раз», — це алгоритм виявлення об’єктів, який поділяє зображення на сітку. Кожна клітинка сітки відповідає за виявлення об'єктів всередині себе. YOLO є одним з найвідоміших алгоритмів виявлення об'єктів завдяки своїй швидкості та точності[4].

YOLOv5 використовує фреймворк для машинного навчання PyTorch і відповідно мову програмування Python.

Архітектура мережі Yolov5. Вона складається з трьох частин: хребет: CSPDarknet, шия: PANet і голова: шар Yolo. Дані спочатку вводяться в CSPDarknet для отримення фіч, а потім передаються в PANet для перемішування фіч. Нарешті, Yolo Layer виводить результати виявлення (клас, оцінка, розташування, розмір). Архітектуру моделі YOLOv5 наведено на рисунку 4.

Для моделі YOLOv5 є попередньо треновані варіанти, що вільно можуть використовуватися будь-ким у своєму програмному забезпеченні, оскільки вона розповсюджується за ліцензією GPL-3.0.

Варіанти відрізняються кількістю рівнів мережі, кількістю нейронів в них та відповідно потрібними для роботи обсягами пам’яті та часу.

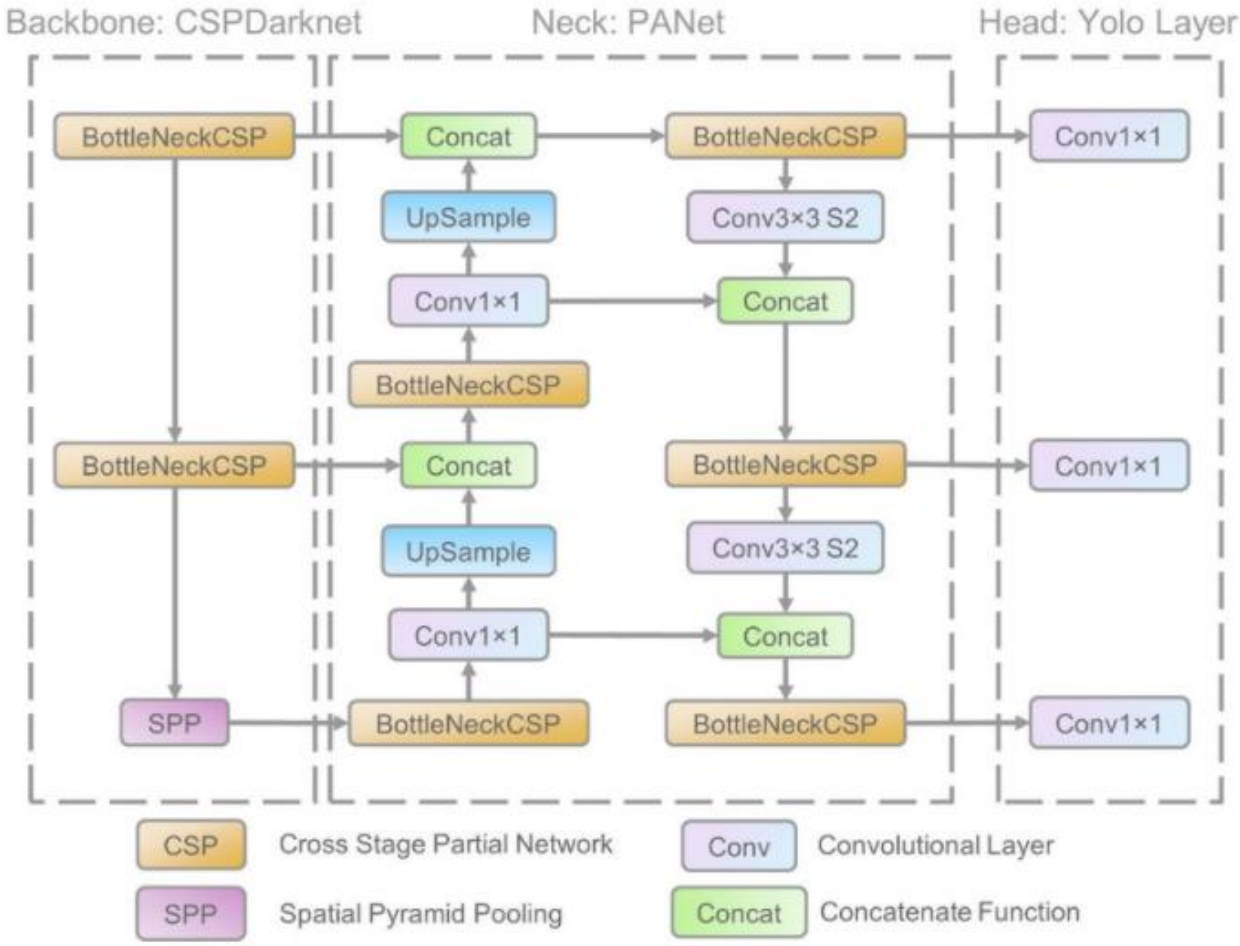

Рисунок 4 – Архітектура моделі YOLOv5

В рамках кваліфікаційної роботи пропонується використати модифікацію YOLOv5s, що надає потрібний баланс у швидкості роботи точності знаходження об’єктів та витратах пам’яті під час роботи.

Порівняння різних варіацій моделі YOLOv5 наведено на рисунку 5.

Ця мережа спеціально натренована на детекцію об’єктів на знімках, що вона бачить вперше і показує кращі показники швидкості та затрат пам’яті ніж її прямі конкуренти на цьому полі, такі як EfficientDet, CenterNet, Faster R-CNN, SSD, ResNet.

Порівняння ефективності різних моделей для детекції об’єктів показано на рисунку 6.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.4	46.0	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.2	56.0	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.2	63.9	224	8.2	1.7	21.2	49.0
YOLOv5l	640	48.8	67.2	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	34.0	50.7	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.5	63.0	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.0	69.0	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.6	71.6	1784	15.8	10.5	76.7	111.4
YOLOv5x6 + TTA	1280 1536	54.7 **55.4**	**72.4** 72.3	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Рисунок 5 – Порівняння роботи різних моделей YOLOv5

Враховуючи дані бенчмарку можна зробити висновок, що один з найлегших варіантів YOLOv5 – YOLOv5s демонструє найкращі швидкісні познаки серед усіх учасників порівняння[5].

Отримана швидкість обробки у понад 270 кадрів щосекунди дозволяє використовувати її для обробки відеоданих у реальному часі навіть враховуючи час необхідний на перетворення зображення з камери на використовуваний моделлю формат[6].

До того ж показники Mean Average Precision (MAP) є такими ж або навіть кращими ніж у аналогів цієї моделі за умови демонстрації найліпшої швидкості роботи.

Враховуючи беззаперечне лідерство у швидкості детекції об’єктів на зображенні при показнику точності, що є вищим за середній, робимо вибір у бік моделі YOLOv5 у підсумку вищезгаданих факторів.

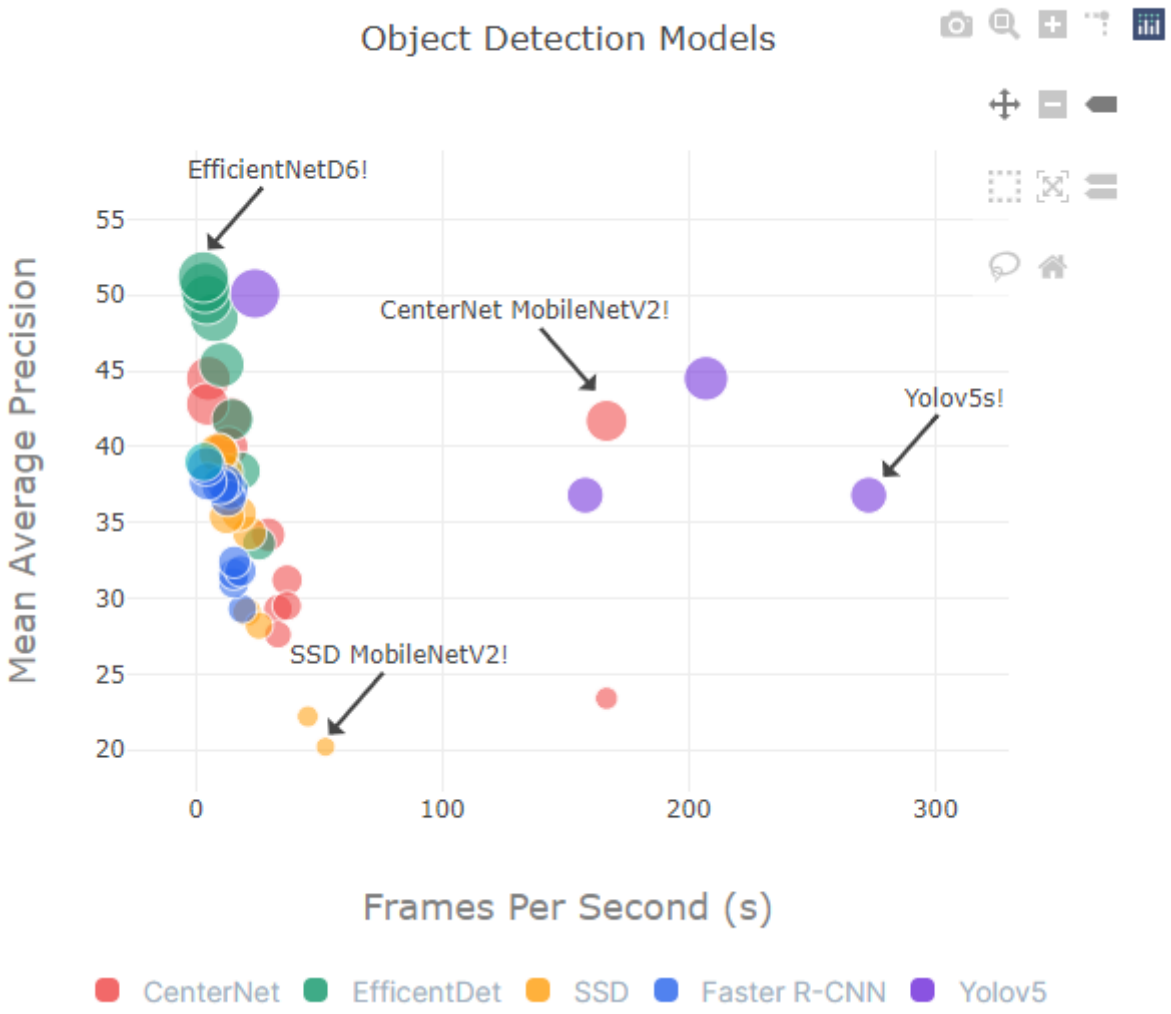

Рисунок 6 – Порівняння швидкості і точності роботи різних моделей для детекції об’єктів

Враховуючи проведений аналіз точності та швидкості роботи різних нейронних мереж було обрано саме YOLOv5 і було прийняте рішення про використання саме цієї нейронної мережі у даній роботі[7].

2.4 Формат метаданих

Як вже зазначалося вище для даної роботи використано формат зберігання зображень JPEG та стандарт, що він реалізує Exif.

EXIF (Exchangeable Image File Format) — стандарт, що дозволяє додавати до зображень та інших медіафайлів додаткову інформацію (метадані), коментує цей файл, що описує умови та способи його отримання, авторство тощо. Набув широкого поширення у зв'язку з появою цифрових фотокамер. Інформація, записана у цьому форматі, може використовуватись як користувачем, так і різними пристроями, наприклад, принтером.

Стандарт EXIF є надзвичайно гнучким (наприклад, дозволяє зберегти отримані з приймача GPS координати місця зйомки) і припускає широкий розвиток - як правило, фотоапарати додають до файлу інформацію, специфічну лише для конкретної камери. Правильно інтерпретувати таку інформацію можуть лише програми від виробника фотоапарата.

Розробник формату - Japan Electronics and Information Technology Industries Association (JEITA). EXIF є частиною ширшого стандарту DCF. Версія Exif 2.2 (відома також Exif Print) введена в 2002 році. Найбільш суттєві доповнення стосуються даних, що описують умови зйомки, потрібні для коректного друку таких зображень. Ці дані можуть знадобитися, наприклад, при друкуванні нічних знімків, для яких велика кількість темряви не є помилкою фотографа, автоматика принтера може намагатися «врятувати» такі знімки, але не повинна цього робити. Остання версія - 2.31.

Exif специфікує більше ніж 100 різних тегів у які може бути записана описуюча зображення інформація.

Нас в першу чергу цікавить інформація, що може бути записана довільним чином для описання об’єктного складу зображення. Виявляється, що існує два різних теги, які призначені для зберігання подібної інформації, наведемо їх назви та опис з офіціального сайту exiv2.org, що описує стандарт, а також пов’язані з ним специфікації:

- тег Exif.Image.ImageDescription: Рядок, що дає назву зображення. Це може бути коментар на кшталт «компанія на пікніку 1988 рік» або подібне. Двобайтові коди символів не можна використовувати. Якщо потрібен 2-байтовий код, слід використовувати приватний тег Exif <UserComment>;
- тег Exif.Photo.UserComment: Тег для користувачів Exif для запису ключових слів або коментарів до зображення, крім тих, що містяться в <ImageDescription>, і без обмежень коду символів тегу <ImageDescription>.

Обираючи серед двох наданих опцій, обираємо тег UserComment, виходячи з міркувань подальшого розвитку та зручності. Наприклад, може постати потреба записувати локалізовану додаткову інформацію щодо об’єкта, для мов, що використовують двобаштові символи, нам не вдасться це зробити у випадку з тегом ImageDescription, проте ми зможемо просто досягти використовуючи тег UserComment.

Наступною задачею, що постає є задача читання та запису метаданих зображення, у даній роботі пропонується використовувати для цього такий інструмент як Exiftool.

ExifTool — це безкоштовна програма з відкритим вихідним кодом для читання, запису та керування зображеннями, аудіо, відео та метаданими PDF.

Він незалежний від платформи, доступний як бібліотека Perl (Image::ExifTool), так і програма командного рядка. ExifTool зазвичай включається в різні типи цифрових робочих процесів і підтримує багато типів метаданих, включаючи Exif, IPTC, XMP, JFIF, GeoTIFF, ICC Profile, Photoshop IRB, FlashPix, AFCP і ID3, а також специфічні для виробника формати метаданих багатьох цифрових камер[8].

Для нашої реалізації мова програмування Perl не використовуватиметься, тому візьмемо крос-платформену реалізацію у вигляді програми командного рядка, до того ж більшість мов програмування містить інструменти для виклику

та отримання результатів cli-програм, тому маємо також перевагу простого запровадження до будь-якого программного кода.

На рисунку 7 навдено короткий синопсис команди програмного рядка exiftool.

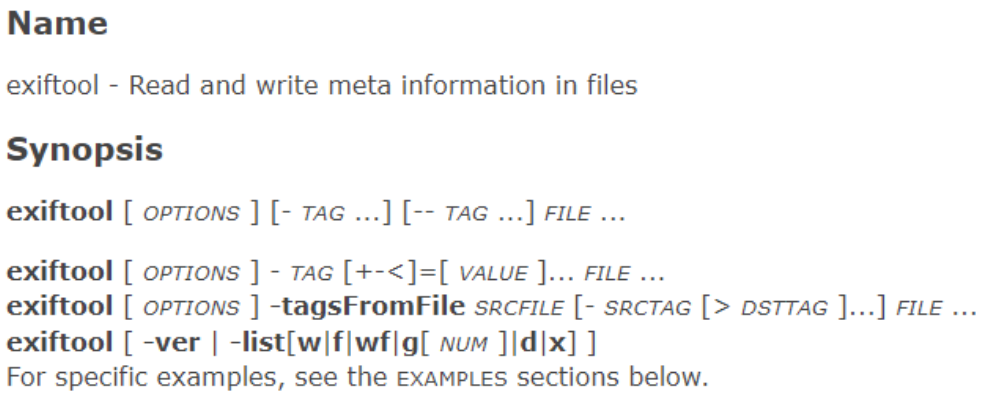

Рисунок 7 – Використання програми командного рядка Exiftool

Наведемо приклад команди, що використовуватиметься у нашому додатку:

```
.\exiftool -Exif:UserComment="comment" "D:\SomePath\image.jpg" -overwrite_original
```

Для запису у метадані файлу замінимо усі переноси рядків на символ розділювач, наприклад вертикальну риску.

Далі розглянемо використаний датасет.

2.5 Датасет

Датасетом для проведення дослідження було обрано COCO 2017, що розшифровується як «Common objects in context», що в перекладі означає «Звичайні об’єкти в контексті».

COCO — це широкомасштабний набір даних для детекції об’єктів, сегментації та підписування. COCO має такі переваги[9]:

– сегментація об’єктів;

– розпізнавання в контексті;

– суперпіксельна сегментація матеріалу;

– 330 тис. зображень (позначених >200 тис.);

– 1,5 мільйона екземплярів об’єктів;

– 80 категорій об’єктів;

– 91 категорія матеріалів;

– 5 підписів на зображення;

– 250 000 людей із ключовими точками.

Для проведення роботи було обрано тренувальний підсет, що складається з 5000 знімків з різним складом та контекстом. Перевагами цього датасету є безкоштовність, доступність, а також різноманітний контекст зображень, що на відміну від багатьох специфікованих дата сетів на певній тематиці дозволяє отримати різні приклади фотографій із зображеними на них об’єктами.

Це моделює ситуацію реального світу коли користувачі системи можуть завантажувати досить різні за наповненням та сенсом зображення, тож є можливість випробувати систему та провести дослідження за різних умов.

Це моделює ситуацію реального світу коли користувачі системи можуть завантажувати досить різні за наповненням та сенсом зображення, тож є можливість випробувати систему та провести дослідження за різних умов.

Також цей датасет надає значне 80 різних категорій, що може здатися невеликим у порівнянні з тисячами класів відомого ImageNet, проте у свою чергу він надає необхідне для пошуку за фотографіями узагальнення.

Таке узагальнення ми були б вимушені робити працюючи наприклад з датасетом ImageNet, тому що для користувача часто досить того, що на двох зображеннях показано кота, порода кота є частіше за все не настільки важливою для того, щоб сказати, що два фото схожі між собою.

У якості бонуса обрана нами модель YOLOv5 є натренованою саме на цьому датасеті, а це означає, що ми можемо розраховувати на якісні результати порівнянь[10].

2.6. Приклади роботи і використання

Наведемо приклад роботи проведеної над одним із зображень, що знаходиться у датасеті.

Рисунок 8 – Вихідне фото

На рисунку 8 наведено вихідне фото, що ми візьмемо для прикладу. Можемо побачити на ньому відвідувача зоопарка, що годує слона.

До помітних неозброєним оком об’єктів на цьому знімку можна віднести якраз відвідувача зоопарка та слона. Проведемо наступні маніпуляції з цим зображенням:

– проведемо розпізнавання об’єктів на ньому з позначенням положення, меж і впевненості у тому, що об’єкт належить певному класу;
– позначимо знайдені об’єкти на фото і проілюструємо наочно, як виглядає оброблене фото;

– винесемо інформацію про знайдені об’єкти компактно в окремий файл;

– покажемо усі метадані, що наявні у вхідного фото.

Розмічене фото можна побачити на рисунку 9.

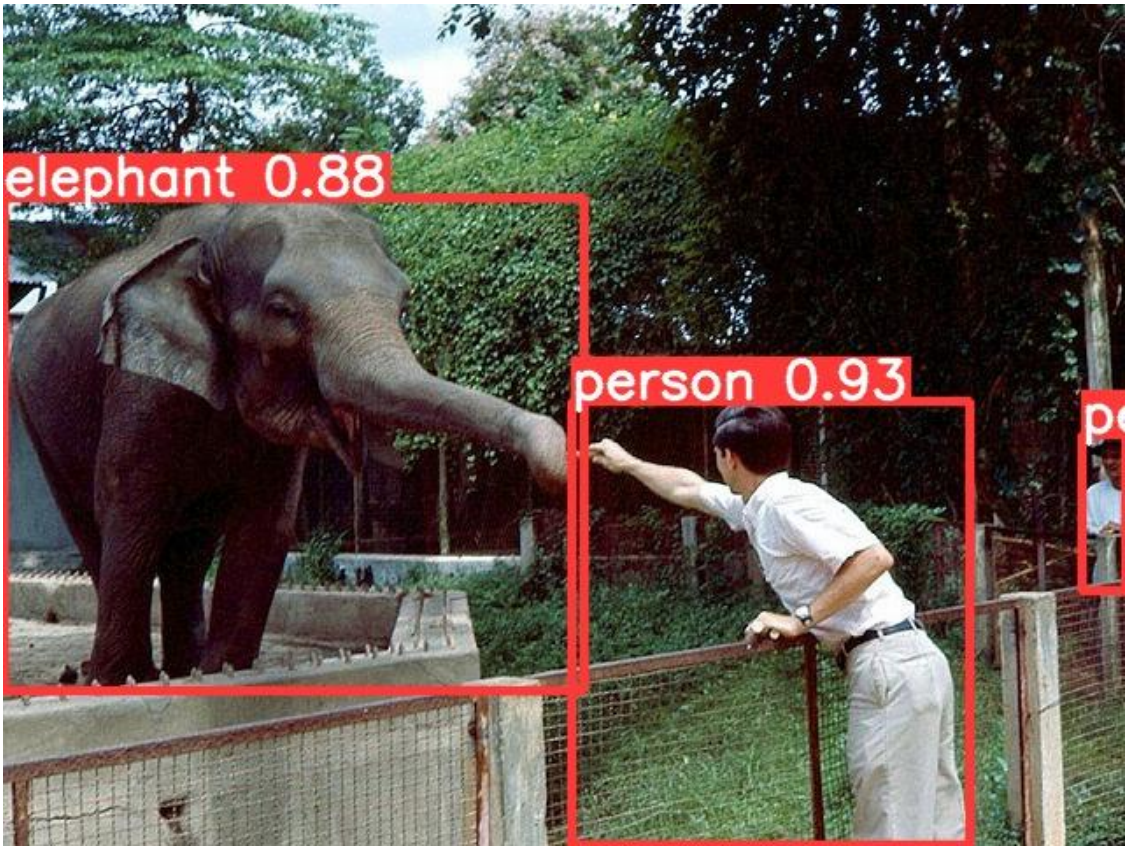

Рисунок 9 – Розмічене YOLOv5 фото

Бачимо, що на фото було знайдено три різних об’єкти.

На даному фото було знайдено дві людини та слон. Знайдені об’єкти, детектовано з наступними упевненостями:

– слона ідентифіковано з впевненістю у 88%;

– людину в центрі з впевненістю 93%;

– людину, що у правій частині фотографії зі впевненістю 45%.

Проводячи аналіз отриманих результатів бачимо, що людина в центрі та слон мають доволі типовий для свого класу вигляд, а розміри цих об’єктів на фото дозволяють впевнено їх ідентифікувати.

В той самий час людина з правого боку знімку зображена лише верхньою частиною тулуба і має маленький розмір, тому впевненість моделі в детекції цього об’єкта нижча за 50 відсотків.

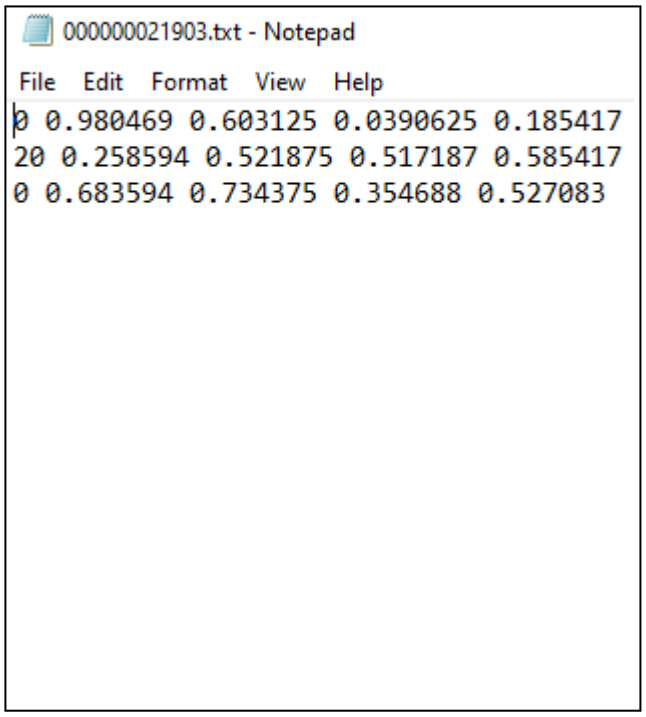

Рисунок 10 – Приклад текстового файлу з даними про типи і розташування об’єктів на знімку

Зберігаємо інформацію про знайдені на зображенні об’єкти у форматі, що було зазначено раніше. Приклад текстового файлу в якому збережено дані про об’єкти наведено на рисунку 10.

Запишемо дані утилітою Exiftool до тегу UserComment метаданих оброблюваного зображення. Поглянемо, які ще метадані зберігаються в цьому зображенні. У розділі Composite зберігаються розміри знімку та параметри камери на яку було зроблено фото. Наступним розділом є метадані задані стандартом Exif, саме в нього і додаються наші дані, можемо побачити також, що Exiftool залишає власні метадані для подальших обробок фото з вказівкою на використану версію.

Бачимо, що в них вказано роздільну здатність простір кольорів та версію Exif стандарту, також бачимо частково наші дані записані до тегу UserComment.


```
"Composite:ImageSize": "640 480",
"Composite:Megapixels": 0.3072,
```


Рисунок 11 – Розділ Composite метаданих знімку


```
"EXIF:ColorSpace": 65535,
"EXIF:ComponentsConfiguration": "1 2 3 0",
"EXIF:ExifVersion": "0232",
"EXIF:FlashpixVersion": "0100",
"EXIF:ResolutionUnit": 2,
"EXIF:UserComment": "0 0.980469 0.603125 0.0390625 0.185417 |
"EXIF:XResolution": 72,
"EXIF:YCbCrPositioning": 1,
"EXIF:YResolution": 72,
"ExifTool:ExifToolVersion": 12.36,
```


Рисунок 12 – Метадані специфіковані стандартом EXIF


```
"File:BitsPerSample": 8,
"File:ColorComponents": 3,
"File:Directory": "D:/Univer/Master Diploma/coco2017val/coco/images/val2017",
"File:EncodingProcess": 0,
"File:ExifByteOrder": "MM",
"File:FileAccessDate": "2021:11:29 17:56:24+02:00",
"File:FileCreateDate": "2021:11:01 08:25:31+02:00",
"File:FileModifyDate": "2021:11:29 17:56:24+02:00",
"File:FileName": "000000021903.jpg",
"File:FilePermissions": 100666,
"File:FileSize": 227403,
"File:FileType": "JPEG",
"File:FileTypeExtension": "JPG",
"File:ImageHeight": 480,
"File:ImageWidth": 640,
"File:MIMEType": "image/jpeg",
"File:YCbCrSubSampling": "1 1",
```


Рисунок 13 – Розділ File метаданих зображення

Наступним розділом є метадані задані стандартом Exif, саме в нього і додаються наші дані, можемо побачити також, що Exiftool залишає власні метадані для подальших обробок фото з вказівкою на використану версію.

Бачимо, що в них вказано роздільну здатність простір кольорів та версію Exif стандарту, також бачимо частково наші дані записані до тегу UserComment.

Далі в розділі File метаданих зображення наведено усю службову інформацію про файл. В тому числі кількість компонент кольору, директорію в якій знаходиться файл, назву файлу, дати доступу, створення та модифікації файлу.

Дозволи доступу до файлу, його розмір, розширення, та тип міме з зазначенням його розмірів також зберігаються там, їх наведено на рисунку 13.

І нарешті останнім за порядком розділом є JFIF. Його метадані наведені на рисунку 14.


```
"JFIF:JFIFVersion": "1 1",
"JFIF:ResolutionUnit": 1,
"JFIF:XResolution": 72,
"JFIF:YResolution": 72,
```


Рисунок 14 – Розділ JFIF метаданих зображення

Таким чином ми навели приклад роботи з зображенням і його метаданими, а також їх використання у даній роботі.

3 ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ

3.1 Проектування програмного модуля

Зважаючи на поставлені цілі розробки, головними рисами розроблюваного програмного модуля мають бути легкість та переносимість. Розроблений програмний модуль має виконуватися швидко і на більшості сучасних операційних систем.

Оскільки ми беремо за основу наших алгоритмів використання моделі машинного навчання YOLOv5, що виконується за допомогою PyTorch фреймворку, має сенс оформити роботу наших алгоритмів у вигляді python скрипта.

Для виконання задач детекції об’єктів на зображенні YOLOv5 містить detect.py скрипт.

Цей скрипт виконує декілька важливих функцій: виконує детекцію об’єктів на переданому зображенні за допомогою переданої моделі та інших параметрів, за потреби зберігає зображення з відміченими боксами знайдених об’єктів, за потреби зберігає дані про знайдені об’єкти у вигляді текстового файлу.

Нам потрібно модифікувати цей скрипт таким чином, щоб застосовуючи додаткові аргументи командного рядка ми могли для обробленого зображення зберігати знайдені об’єкти у метаданих, знаходити схожі зображення у сховищі за одним з трьох розроблених алгоритмів, а також розробити скрипт, що дозволить шукати фотографії за описом зображених об’єктів для демонстраційної програми.

Завдання запису знайдених об’єктів до метаданих зображення покладемо на утиліту командного рядка exiftool як це показано у другому розділі цієї роботи.

Перейдімо до розробки.

3.2. Розробка програмного модуля

Для виконання боксового та матричного алгоритму має сенс створити простий об’єкт зображення, що описуватиме зображення та зберігатиме дані про об’єкти, що знаходяться на ньому.

Після попередньої фільтрації простим алгоритмом матимемо завантажену в пам’ять модель зображень, що зможемо перевикористати. Додатково створимо простий скрипт, що дозволить знаходити зображення на яких є об’єкти зі вказаними типами для демонстраційної програми.

Наведемо розроблені код розроблених python-скриптів.

```
class Image:
    def __init__(self, path, objects):
        img = PImage.open(path)
        self.path = path
        self.objects = []
        self.imw = img.width
        self.imh = img.height
        for object in objects:
            self.objects.append(ImageObject(object.split(" "), img.width, img.height))
        self.classes = [ob.cls for ob in self.objects]
        self.classes_set = set(self.classes)
        self.class_counter = Counter(self.classes)

    def compare(self, other):
        similarity = 0.0
        if not (self.classes_set & other.classes_set):
            return 0.0
        classes = self.classes.copy()
        classes.extend(other.classes)
        class_list = Counter(classes)
        multiplier = 1 / len(class_list)
        for cls in class_list.keys():
            more, less = self.class_counter[cls], other.class_counter[cls]
            if less > more:
                more, less = less, more
            similarity += multiplier * less / more

        return similarity

    def compare_boxes(self, other):
        longer = self if len(self.objects) >= len(other.objects) else other
        shorter = other if len(self.objects) >= len(other.objects) else self
        longer_classes = [ob.cls for ob in longer.objects]
        counter = Counter(longer_classes)
        sum = 0.0
        for key in counter:
            longer_objects = [ob for ob in longer.objects if ob.cls == key]
            shorter_objects = [ob for ob in shorter.objects if ob.cls == key]
            if len(longer_objects) == 0 or len(shorter_objects) == 0:
                continue
            ob_scores = []
            for ob in longer_objects:
                scores = []
                for secob in shorter_objects:
                    scores.append(ob.interscore(secob))
                ob_scores.append(scores)
            while len(ob_scores) and max([len(sc) for sc in ob_scores]):
                maxval = ob_scores[0][0]
                maxi = 0
```

```
                maxj = 0
                for i in range(len(ob_scores)):
                    for j in range(len(ob_scores[i])):
                        if ob_scores[i][j] > maxval:
                            maxval = ob_scores[i][j]
                            maxi = i
                            maxj = j
                sum += maxval
                del ob_scores[maxi]
                for sc in ob_scores:
                    del sc[maxj]

        coef = sum / len(longer.objects)
        return coef

    def compare_matrix(self, other, comp):
        self_classes = [ob.cls for ob in self.objects]
        self_classes.extend([ob.cls for ob in other.objects])
        class_list = Counter(self_classes)
        classes_number = len(class_list)
        w, h = max(self.imw, other.imw), max(self.imh, other.imh)
        results = {}
        def func(cls):
            matrix = [[0 for y in range(int(h * comp))] for x in range(int(w * comp))]
            self_class_objects = [ob for ob in self.objects if ob.cls == cls]
            other_class_objects = [ob for ob in other.objects if ob.cls == cls]
            inter = 0
            common = 0
            for ob in self_class_objects:
                for x in range(int(ob.x1 * comp), int(ob.x2 * comp)):
                    for y in range(int(ob.y1 * comp), int(ob.y2 * comp)):
                        if matrix[x][y] == 0:
                            matrix[x][y] += 1
                            common += 1
            for ob in other_class_objects:
                for x in range(int(ob.x1 * comp), int(ob.x2 * comp)):
                    for y in range(int(ob.y1 * comp), int(ob.y2 * comp)):
                        if matrix[x][y] == 1:
                            matrix[x][y] += 1
                            inter += 1
                        elif matrix[x][y] == 0:
                            common += 1
            cls_coef = inter / common
            results[cls] = cls_coef
        threads = []
        for cls in class_list:
            th = Thread(target=func(cls))
            th.start()
            threads.append(th)
        for th in threads:
            th.join()
        coef = sum(list(results.values())) / classes_number
        return coef



class ImageObject:
    def __init__(self, info, imw, imh):
        self.cls = int(info[0])
```

```
        self.w = int(float(info[3]) * imw)
        self.h = int(float(info[4]) * imh)
        x = int(float(info[1]) * imw)
        y = int(float(info[2]) * imh)
        self.x1 = x - self.w / 2
        self.y1 = y - self.h / 2
        self.x2 = x + self.w / 2
        self.y2 = y + self.h / 2

    def interscore(self, other):
        if self.cls != other.cls:
            return 0.0
        width = min(self.x2, other.x2) - max(self.x1, other.x1)
        height = min(self.y2, other.y2) - max(self.y1, other.y1)

        if width <= 0 or height <= 0:
            return 0.0

        intersquare = width * height
        commonsquare = (self.x2 - self.x1) * (self.y2 - self.y1) + (other.x2 - other.x1) * (other.y2
- other.y1) - intersquare

        return intersquare / commonsquare

import argparse
import os
import sys
from pathlib import Path
from collections import Counter
from tkinter.messagebox import YES
from PIL import Image as PImage
import numpy as np
import torch
import torch.backends.cudnn as cudnn
import json
import exiftool
FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative
from models.experimental import attempt_load
from utils.general import check_requirements, increment_path, print_args, set_logging
from utils.torch_utils import select_device

class Image:
    def __init__(self, path, objects):
        img = PImage.open(path)
        self.path = path
        self.objects = []
        self.imw = img.width
        self.imh = img.height
        for object in objects:
            self.objects.append(ImageObject(object.split(" "), img.width, img.height))
        self.classes = [ob.cls for ob in self.objects]
        self.classes_set = set(self.classes)
        self.class_counter = Counter(self.classes)

    def is_match(self, classes):
```

```
        for obj in self.objects:
            if len(classes) == 0:
                return True
            if obj.cls in classes:
                classes.remove(obj.cls)

        if len(classes) == 0:
            return True
        return False


class ImageObject:
    def __init__(self, info, imw, imh):
        self.cls = int(info[0])
        self.w = int(float(info[3]) * imw)
        self.h = int(float(info[4]) * imh)
        x = int(float(info[1]) * imw)
        y = int(float(info[2]) * imh)
        self.x1 = x - self.w / 2
        self.y1 = y - self.h / 2
        self.x2 = x + self.w / 2
        self.y2 = y + self.h / 2


@torch.no_grad()
def run(weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
        tags='',  # tags to search images by, if empty all images are matched
        separator=',', # separator to split tags
        limit=20, # limit of returned results
        project=ROOT / 'search',  # save results to project/name
        name='results',  # save results to project/name
        ):
    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=True)  # increment run
    save_dir.mkdir(parents=True, exist_ok=True)  # make dir
    set_logging()
    device = select_device()
    # Load model
    w = str(weights[0] if isinstance(weights, list) else weights)
    names = [f'class{i}' for i in range(1000)]
    model = torch.jit.load(w) if 'torchscript' in w else attempt_load(weights, map_location=device)
    names = model.module.names if hasattr(model, 'module') else model.names  # get class names
    tags = tags.split(separator)
    classes = [names.index(tag) for tag in tags]
    directory = "C:\\Users\\sasha\\images"
    files = list(Path(directory).glob('*'))
    matches = []
    for file in files:
        full_path = f'{directory}\{file.name}'
        objects = []
        pimage = PImage.open(full_path)
        exif = pimage._getexif()
        if exif is not None:
            objects = exif[37510].decode('UTF-8')[8:].split(" | ")
        image = Image(full_path, objects)
        if image.is_match(classes.copy()):
            matches.append(file.name)
            if len(matches) == limit:
                break
```

```
    txt_path = str(save_dir / 'matches.txt')
    with open(txt_path + '', 'a') as sf:
        sf.write(json.dumps(matches, indent = 4))

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model
path(s)')
    parser.add_argument('--tags', type=str, default='', help='tags to search images by, if empty all
images are matched')
    parser.add_argument('--separator', type=str, default=',', help='separator to split tags')
    parser.add_argument('--limit', type=int, default=20, help='limit of returned results')
    parser.add_argument('--project', default=ROOT / 'search', help='save results to project/name')
    parser.add_argument('--name', default='results', help='save results to project/name')
    opt = parser.parse_args()
    print_args(FILE.stem, opt)
    return opt


def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))


if __name__ == "__main__":
    opt = parse_opt()
    main(opt)
```

Демонстраційну програму створимо за звичайною архітектурою сервер-клієнт, де зможемо шукати зображення за списком зображених об’єктів, а також виконувати пошук схожих зображень за одним з трьох розроблених алгоритмів.

Серверна частина розроблена на .Net, клієнтська частина на Angular, вибір у більшій мірі обумовлено особистими вподобаннями автора роботи, оскільки для демонстрації роботи алгоритмів згодяться абсолютно будь-які технології та інструменти.

Для створення UI застосунку використано сучасну бібліотеку Material, що надає програмістові досить широкий набір керівних елементів, які можна використати при побудові сторінки. Запити до серверної частини виконуються асинхронно, тому можна виконувати декілька запитів одночасно з можливістю паралельного виконання декількох пошукових та порівняльних задач. Після виконання скриптів, результат зчитується з текстового файлу.

Демонстраційна програма наводить знайдені знімки схожі на розшукуваний, а також відсоток схожості між фотографіями. UI демонстраційної програми наведено на рисунку 15.

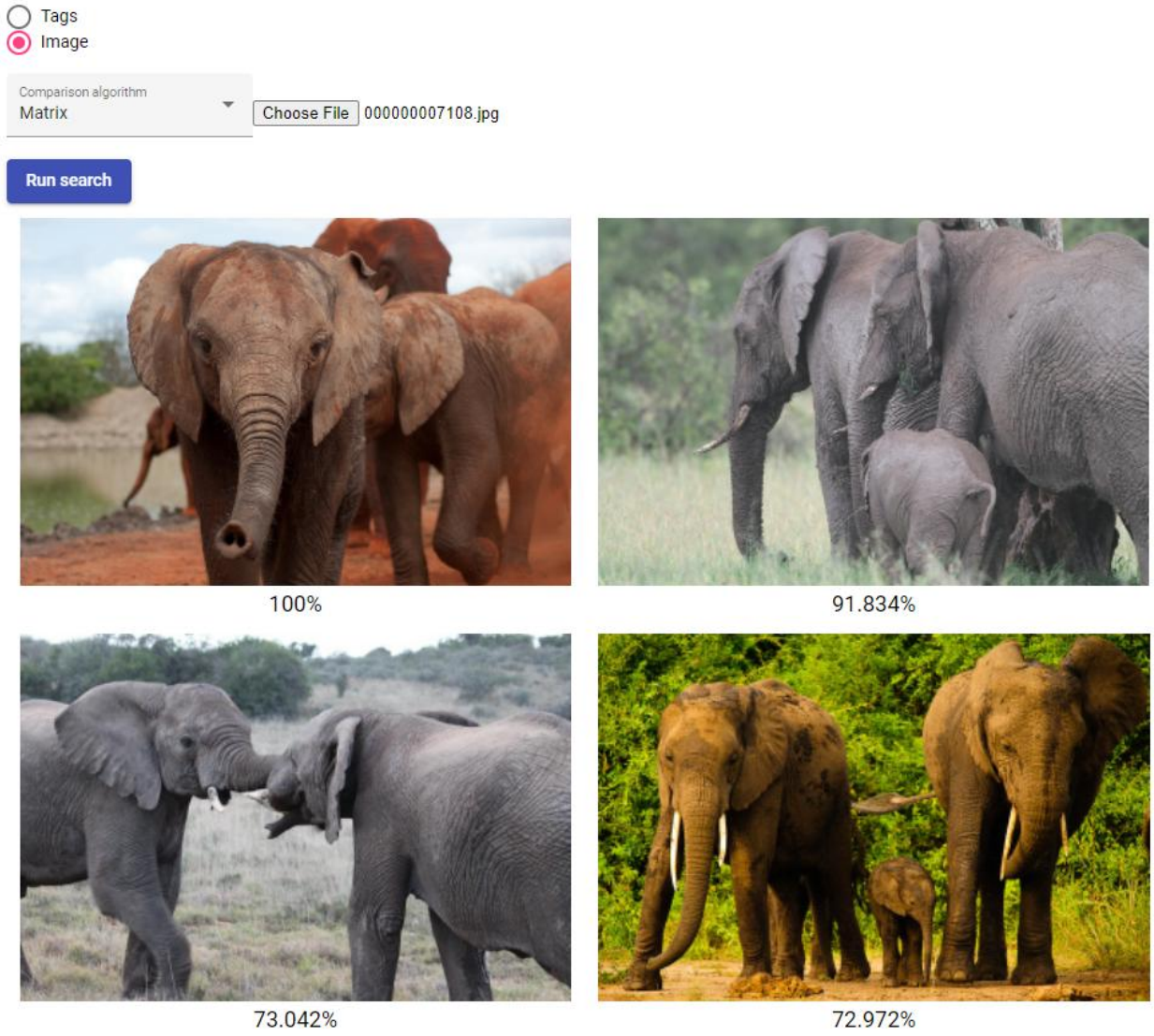

Рисунок 15 – UI демонстраційної програми

З використанням демонстраційної програми можна переконатися у доцільності підходу з метаданими та збереженням об’єктів для подальшого виконання контекстних запитів та управління великими сховищами зображень.

4 ПРОЕКТУВАННЯ І ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ

4.1. Вихідні дані

Ми проведемо експерименти з продуктивністю та порівняємо алгоритми на частині датасету COCO 2017.

Ця частина включає п’ять тисяч зображень і зображує 80 типів об’єктів. Ми оберемо п’ять із п’яти тисяч зображень для виконання наших тестів і порівняємо результати алгоритмів за часом виконання та отриманими результатами порівняння.

Рисунок 16 – Зображення обрані для експериментів

На рисунку 16 наведено зображення обрані для експериментів, що демонструють такі групи об’єктів: ведмідь, стіл з їжею, натовп, сім’я з фрисбі та тенісист.

4.2. План експериментів

Перш за все, ми запустимо порівняння зображень для всіх п'яти вибраних зображень із простим порівнянням, порівнянням боксів і порівнянням матриці.

Потім порівняємо їх: час виконання; найкращий результат; подібність; діапазон коефіцієнтів п’яти найбільш схожих результатів. Ми порівняємо отримані результати зі значеннями отриманими, під час порівняння гістограм двох зображень за допомогою метрики кореляції, наданої бібліотекою OpenCV.

Таке порівняння має дати нам уявлення про те, наскільки ефективні та точні наші результати в порівнянні з класичним широко використовуваним алгоритмом.

Тести будуть виконуватися з порогом довіри на рівні 0,5, щоб відфільтрувати всі об’єкти, в яких YOLOv5 не впевнена, оскільки велика кількість об’єктів з низькою впевненістю можуть різко зменшити коефіцієнти подібності, навіть коли зображення виглядають схоже.

Для матричного алгоритму спробуємо підібрати оптимальний коефіцієнт стиснення зображення для того, щоб отримати прийнятний час виконання для великої кількості зображень, що зберігається у сховищі даних в той самий час не втративши точність у значній мірі.

Оберемо також зображення, на прикладі якого порівняємо топ-5 результати кожного алгоритму відмічаючи переваги та недоліки кожного алгоритму.

Попередньо ми запустимо виявлення об’єктів на всіх тестових зображеннях, щоб мати візуальну підказку щодо знайдених об’єктів, вони показані на рисунку 17.

Робочою станцією для проведення експериментів є ноутбук HP Pavilion Gaming Laptop 17-cd0xxx. Цей ноутбук оснащений процесором Intel(R) Core(TM) i5-9300H @ 2,40 ГГц, має 24 гігабайти оперативної пам’яті, 128 ГБ SSD та 1 ТБ HDD. На робочій станції встановлена відеокарта GeForce GTX 1650 від Nvidia, побудована на архітектурі Turing з 4 ГБ GDDR5.

Графічна карта містить 896 ядер CUDA. NVIDIA надає набір програмних програм машинного навчання та аналітики для прискорення наскрізних конвеєрів даних на графічних процесорах.

Цю роботу забезпечили понад 15 років розробки CUDA. Бібліотеки з GPU-прискоренням абстрагують сильні сторони низькорівневих примітивів CUDA.

Численні бібліотеки, такі як лінійна алгебра, розширена математика та алгоритми розпаралелювання, закладають основу для екосистеми додатків із інтенсивними обчисленнями.

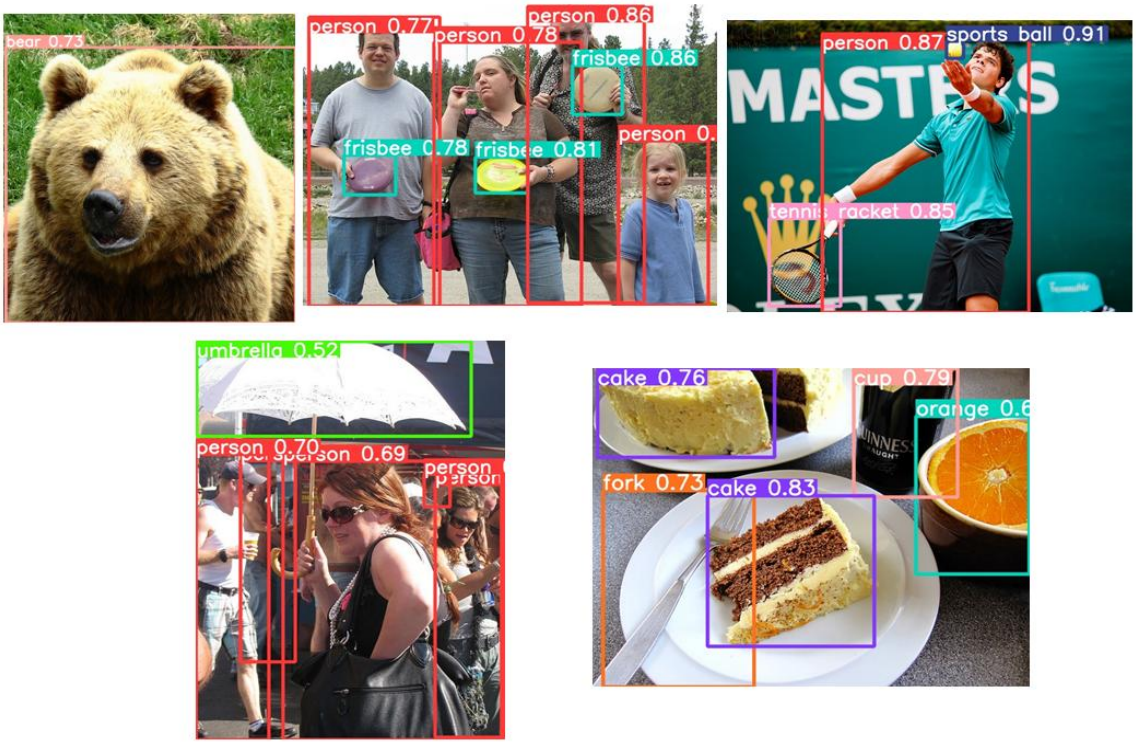

Рисунок 17 – Знайдені об’єкти на зображеннях

Операційна система Windows 10 Professional, що дозволяє нам проводити наші експерименти з усіма останніми оновленнями та найновішими драйверами для обладнання. Для запуску наших скриптів ми використовуємо версію Python 3.9 з фреймворком PyTorch.

4.3 Результати експериментів

У таблицях 1, 2 і 3 наведено загальні результати для середнього часу виконання, вибраних зображень і найвищого знайденого коефіцієнта подібності для тестованих зображень для розроблених алгоритмів.

У таблиці 4 наведено отримані результати для порівняння гістограми за кореляційною метрикою.

Таблиця 1. Результати виконання простого алгоритму

	Ведмідь	Родина з фрисбі	Гравець в теніс	Натовп	Стіл з їжею
Час виконання (мс)	718 мс	843 мс	812 мс	796 мс	796 мс
Найвищий коефіцієнт схожості	1	0.75	1	0.916667	0.75
Діапазон коефіцієнтів топ-5 виборів	1.0 - 1.0	0.75 - 0.708	1.0 - 1.0	0.917 - 0.9	0.75 - 0.5

Час виконання в таблиці не включає час на читання та розбір метаданих для трьох запропонованих алгоритмів, також боксовий та матричний алгоритми показують час виконання без попередньої фільтрації.

Час виконання для порівняння гістограми враховує час для побудови, нормалізації та порівняння гістограм для 5000 зображень у наборі даних.

Час на зчитування даних зображення зі сховища на машині не враховується, так само, як і час на читання метаданих для перших алгоритмів.

Таблиця 2. Результати виконання боксового алгоритму

	Ведмідь	Родина з фрисбі	Гравець в теніс	Натовп	Стіл з їжею
Час виконання (мс)	0 мс	135 мс	109 мс	126 мс	31 м
Найвищий коефіцієнт схожості	0.907722	0.293879	0.30171	0.318123	0.145482
Діапазон коефіцієнтів топ-5 виборів	0.907 - 0.45	0.293 - 0.264	0.301 - 0.268	0.318 - 0.243	0.145 - 0.123

Таблиця 3. Результати виконання матричного алгоритму

	Ведмідь	Родина з фрисбі	Гравець в теніс	Натовп	Стіл з їжею
Час виконання (мс)	141 мс	303 мс	309 мс	366 мс	455 мс
Найвищий коефіцієнт схожості	0.910818	0.42502	0.262802	0.588056	0.159144
Діапазон коефіцієнтів топ-5 виборів	0.91 - 0.491	0.425 - 0.331	0.262 - 0.23	0.588 - 0.426	0.159 - 0.089

Оскільки цей час сильно залежить від апаратного забезпечення і може бути значно покращений за допомогою кешування та інших підходів.

Нашею задачею стоїть перевірка саме часу виконання алгоритмів у поставлених умовах.

Таблиця 4. Результати виконання алгоритму порівняння гістограм

	Ведмідь	Родина з фрисбі	Гравець в теніс	Натовп	Стіл з їжею
Час виконання (мс)	1786 мс	1754 мс	1609 мс	1617 мс	1724 мс
Найвищий коефіцієнт кореляції	0.834534	0.962971	0.937198	0.980637	0.799085
Діапазон коефіцієнтів топ-5 виборів	0.834 - 0.772	0.963 - 0.959	0.937 - 0.328	0.980 - 0.971	0.799 - 0.773

Як і очікувалося, простий алгоритм дає багато подібних результатів для зображень, які зображують точну або дуже близьку конфігурацію об’єктів. Порівняння зображень ведмедя дало дюжину зображень з одним виявленим ведмедем на них. Один із найкращих результатів був не дуже близьким, оскільки два ведмеді були помилково виявлені як один ведмідь. Для зображення сім'ї з фрісбі ми можемо побачити зображення команди фрісбі як найкраще співпадання і коефіцієнт, менший за одиницю, оскільки кількість виявлених людей і фрісбі різна. Образ іншого тенісиста вибирається найкращим для зображеного тенісиста з тенісною ракеткою і м'ячем.

Зображення натовпу з жінкою з парасолькою спереду має схоже зображення, як найкраще, єдина різниця - це розташування та ціла картинка, тому що тут ми бачимо лише ноги, тоді як на перевіреному зображенні ми бачимо

людей на повний зріст, це зображення, ймовірно, не буде вважатися найкращим відповідним алгоритмами, що залежать від розташування.

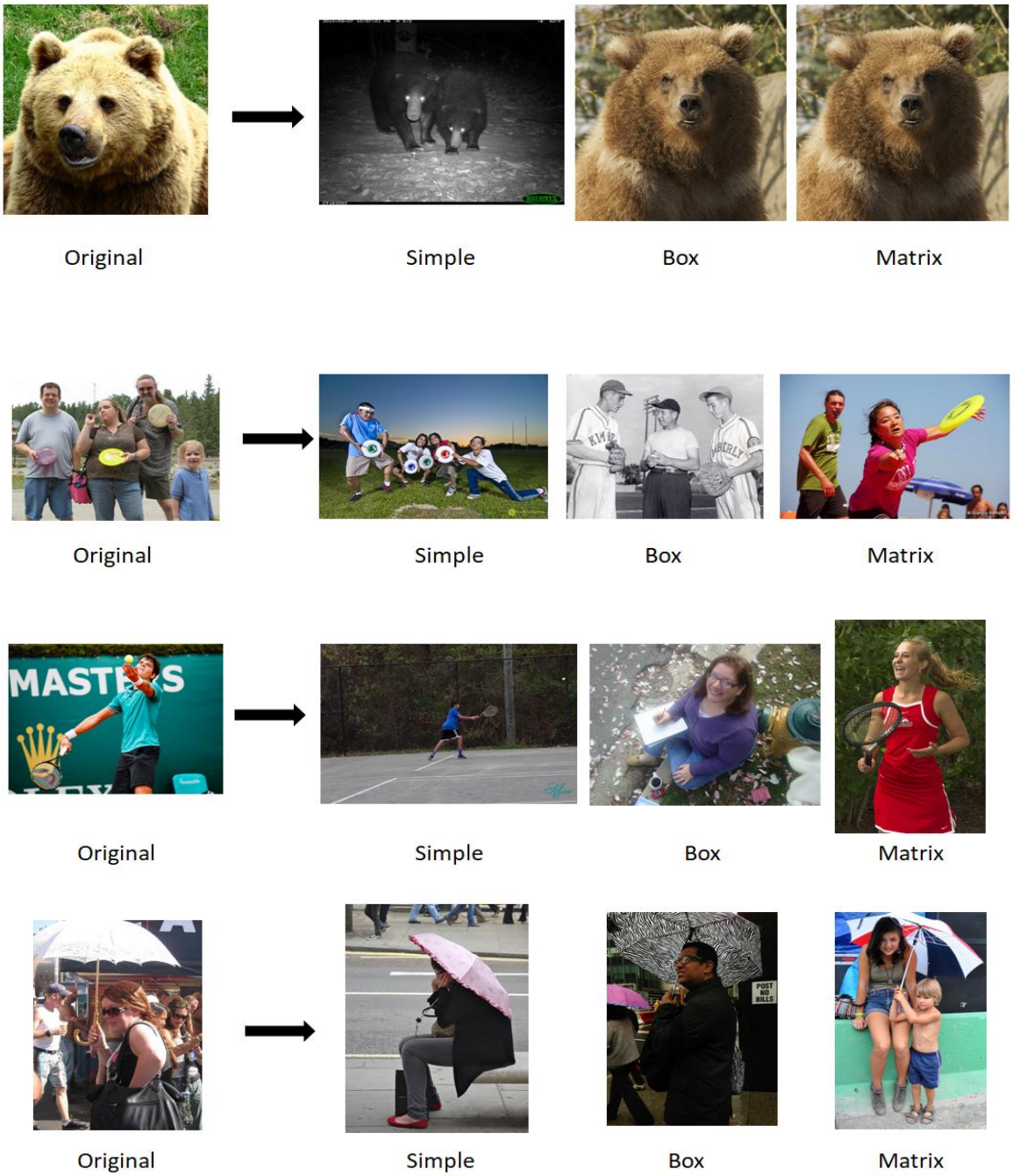

Рисунок 18 – Найкращі вибори алгоритмів для зображень ведмедя, родини з фрисбі, тенісиста та натовпу

Зауважимо, що час виконання запроваджених алгоритмів порівняння зображень розраховувався чисто, а це означає, що час на читання метаданих або

читання даних зображення для побудови гістограми не було включено в результати.

Рисунок 19 – Найкращі вибори алгоритмів для столу з їжею

Це важливий момент, оскільки час для читання інформації з метаданих зображення набагато швидше, ніж для побудови та нормалізації гістограми зображення.

Рисунок 20 – Найбільш схожі зображення з тенісистом (простий алгоритм)

Читання та розбір метаданих займає в середньому 1,4 секунди для 5000 зображень, тоді як читання та побудова гістограм для одного набору даних займає в середньому 22,5 секунди, якщо зображення спочатку зменшено, однак ми

можемо покращити цей час, якщо гістограми будуть попередньо обчислюватися та зберігатися в метаданих або кеші.

Рисунок 21 – Найбільш схожі зображення з тенісистом (боксовий алгоритм)

Рисунок 22 - Найбільш схожі зображення з тенісистом (матричний алгоритм)

На рисунках 18 – 22 представлені тестові зображення з найкращими виборами, щоб побачити найкращу відповідність, знайдену кожним алгоритмом.

Також для зображення з тенісистом показано 5 кращих варіантів кожного алгоритму, оскільки отримані результати найкраще ілюструють різницю між трьома запропонованими алгоритмами.

Тож ми отримали вичерпні результати для аналізу.

4.4. Аналіз отриманих результатів

Перше, що потрібно відзначити в результатах, це час виконання. Час виконання для простого алгоритму в середньому становить 793 мс для 5000 зображень. Як боксовий, так і матричний алгоритми працюють із зображеннями, вже відфільтрованими за простим алгоритмом.

Боксовий алгоритм приймає всі зображення, які перевищили поріг 0,1 для простого алгоритму, оскільки він швидкий для невеликої кількості об’єктів на зображенні, його час виконання в середньому становить 80 мс у наших тестах для до 500 зображень. Матричний алгоритм значно повільніший через сканування зображень, тому ми обмежуємо кількість зображень для обробки до топ-100, коли виконуємо простий алгоритм. Він становить у середньому 314 мс, кількість оброблених зображень коливалася від 50 до 100 вибірок. У реальному додатку ми можемо використовувати цей підхід для пошуку схожих зображень, він показав себе як достатньо ефективний для часу виконання запиту. Для реальних сценаріїв ми можемо запровадити подальшу оптимізацію з кешуванням нещодавно отриманих метаданих файлів, модифікаціями сховища файлів тощо.

Загалом простий алгоритм показав найвищі коефіцієнти, як і очікувалося, тому що він не обчислює розташування об’єктів на додаток до їх типів і кількостей, звичайно, він не ранжує зображення з однаковою кількістю об’єктів за їх схожістю, вони всі рівні. Ми бачили, що в 4 з 5 тестів матричний алгоритм показав вищі коефіцієнти подібності, ніж боксовий алгоритм, також зробили висновок про те, що результати для матричного алгоритму ближчі до цільового контексту зображення, тоді як боксів алгоритм іноді повертав зображення, які містять певні об'єкти в тих самих місцях, але не мають жодного об’єкта іншого типу.

Під час експериментів усі три алгоритми показали, що залежно від змісту зображення, кількості та різноманітності виявлених об’єктів їх підхід може бути кращим, щоб отримати зображення, схожі з точки зору людини. Наприклад, зображення столу з їжею, де було виявлено багато різних об’єктів, таких як виделка, чашка, торт, апельсин, найкраще відповідає простому алгоритму, з нашої власної точки зору, він давав результати ближчі до оригінальних, ніж алгоритми, що залежать від розташування зображення. У той же час, боксовий і матричний алгоритм показали домінування для зображень з одним виявленим об'єктом. Вибір найкращих варіантів набагато ближчий до оригіналу, ніж для простого алгоритму.

Варто зазначити, що боксів алгоритм корисний для пошуку зображень з тією ж конфігурацією, як приклад ми можемо навести сім'ю з фрисбі, у топ-5 вибраних результатів боксовим алгоритмом немає фрисбі, але замість цього у нас є зображення, які показують нам групу з трьох людей, що стоять, як на оригінальному зображенні.

У певному сенсі цей алгоритм є більш простим, ніж матричний алгоритм, часто він не фіксує контекст, а повертає зображення з найбільшим перекриттям між деякими з об’єктів, також цей алгоритм багато в чому залежить від кожного окремого об’єкта на відміну від інших алгоритмів, які більше залежать від типів об’єктів, ніж на кожному окремому об’єкті на них.

Подібність між двома зображеннями суб’єктивна для кожної людини, але, на нашу думку, якщо ми хочемо вибрати алгоритм, який повертає найближчі вибірки, то матричний алгоритм є таким, принаймні для тестованих зображень.

З нашої точки зору, це стало компромісом між простим алгоритмом, який піклується лише про контекст, але не конфігурацію, та боксовим алгоритмом, який зосереджений переважно на конфігурації об’єктів на зображенні з невеликою увагою до контексту.

Матричний алгоритм став інструментом, який дозволяє оцінювати розташування всіх об’єктів певного типу в цілому, а не окремих сутностей, це

найповільніший алгоритм з трьох, але за допомогою кількох оптимізацій та хитрощів він може виконати своє завдання у прийнятний час.

Ми ввели алгоритм порівняння гістограм як еталон для порівняння деяких класичних алгоритмів, які використовуються для оцінки подібності зображень. Результати показують, що читання метаданих зображення відбувається в 16 разів швидше, ніж читання даних зображення та створення його гістограми.

Коли ми порівнюємо час, витрачений на виконання алгоритмів, ми бачимо, що простий алгоритм в 2 рази швидше, ніж порівняння гістограм, а боксовий та матричний алгоритми в 1,5-1,7 рази швидше. Експерименти з алгоритмами дали нам деякі з бажаних результатів, залишаючи при цьому багато місця для вдосконалення. Перший момент, який може стати метою майбутніх досліджень, — це надання кожному типу об’єктів певної ваги, наприклад люди важливіші за фрисбі чи тенісні ракетки, тому ми можемо збалансувати вплив не дуже важливих об’єктів на порівняння.

Також коефіцієнти можуть надаватися об'єктам кожного зображення окремо в залежності від площі, яку вони охоплюють. Як можливе підвищення точності порівняння, ми можемо порівняти не тільки розташування об’єктів на зображенні, а й впевненість моделі, що сектор містить об’єкти певного типу. Як зазначалося раніше, продуктивність матричного алгоритму є одним із ключових моментів удосконалення.

Реалізація порівняння на відеокарті може збільшити швидкість обчислення коефіцієнта подібності, що дозволить їй працювати на більшому обсязі вхідних зображень. Результати експериментів цієї статті показали нам можливість використовувати моделі машинного навчання з метаданими файлів для ефективного управління великими обсягами даних.

Правильне використання метаданих і моделей машинного навчання дає нам інструмент для реалізації деяких нетривіальних функціональних можливостей для додатків, які працюють з великою кількістю даних, як-от програми з архітектурою data lake, де всі дані просто зберігаються в необробленому форматі.

ВИСНОВКИ

У ході кваліфікаційної роботи було спроектовано програмну систему, що може працювати в умовах постійно зростаючого потоку інформації, де кожного дня з’являються сотні і навіть тисячі нових фото, було дослідно встановлено, що можливим є впровадження моделей для детекції об’єктів на зображенні для того, щоб використати цю інформацію, як вхідні дані при порівнянні під час пошуку за фотографіями.

Результатом кваліфікаційної роботи стало створення програмного модуля, що з використанням технологій штучного інтелекту та машинного навчання, а саме нейронних мереж дозволяє виконувати попередню обробку зображень при надходженні фотографій до системи зі знаходженням об’єктів на знімку, їхнього положення та розмірів і дозволяє записувати цю інформацію до метаданих зображення, що використовується для виконання пошуку за зображеннями, надане зображення так само обробляється мережею і за знайденими об’єктами виконується порівняння зображення з зображеннями у системі і відбувається сортування у спадаючому порядку фотографій, що знаходяться у системі.

У порівнянні з відомим алгоритмом, запропонований підхід показав у 1,5-2 рази швидше виконання, що означає, що наші алгоритми можуть бути виконані в системах з великими сховищами даних зображення за прийнятний час.

Запропоновані алгоритми показали, що кожен алгоритм можна використовувати для отримання найкращих результатів пошуку та порівняння залежно від об’єктів та конфігурації зображення.

Експерименти також підтвердили надзвичайну ефективність паралельних обчислень для підвищення продуктивності алгоритмів порівняння зображень, оскільки ці алгоритми здебільшого можна легко розподілити для паралельного виконання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Гонсалес Р. С., Вудс Р. Е. Цифровая обработка изображений : 3-тє вид., перероб. та доп. Москва : Техносфера, 2012. 1104 с.

2. X. Liu, G. Cheung, C. W. Lin, D. Zhao, W. Gao. Prior-Based Quantization Bin Matching for Cloud Storage of JPEG Images//IEEE Transactions on Image Processing. №7(27). 2018. с. 3222-3235.

3. K. Smelyakov, A. Chupryna, D. Sandrkin, M. Kolisnyk. Search by Image Engine for Big Data Warehouse//2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream). 2020. P. 1-4.

4. Холл Г. М. Адаптивный код на C#. Проектирование классов и интерфейсов, шаблоны и принципы SOLID. Москва : Вільямс, 2016. 432 с.

5. Албахарі Д., Албахарі Б. C# 6.0. Справочник. Полное описание языка. Москва : Вільямс, 2016. 1040 с.

6. Osterwalder A., Pigneur Y. Business model generation: навч. посіб. – Амстердам: Moddermann Druckwerk, 2010. – 273 с.

7. K. Smelyakov, M. Shupyliuk, V. Martovytskyi, D. Tovchyrechko, O. Ponomarenko. Efficiency of image convolution//2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL). 2019. P. 578-583.

8. Дейт К. Дж., Введення у системи баз Даних : 7е видання. Москва : Вільямс, 2001. 846 с.

9. Гайдаржи В. І., Ізварін І. В. Бази даних в інформаційних системах. Київ : Університет «Україна», 2018. 418 с.

10. K. Smelyakov, A. Chupryna, O. Bohomolov and I. Ruban. The Neural Network Technologies Effectiveness for Face Detection//2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP). 2020. P. 201-205.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. K. Smelyakov, A. Chupryna, D. Sandrkin, M. Kolisnyk. Search by Image Engine for Big Data Warehouse//2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream). 2020. P. 1-4.

7. K. Smelyakov, M. Shupyliuk, V. Martovytskyi, D. Tovchyrechko, O. Ponomarenko. Efficiency of image convolution//2019 IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL). 2019. P. 578-583.

10. K. Smelyakov, A. Chupryna, O. Bohomolov and I. Ruban. The Neural Network Technologies Effectiveness for Face Detection//2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP). 2020. P. 201-205.