МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНФОРМАТИКИ ТА КОМП'ЮТЕРНОЇ ТЕХНІКИ

Рівень вищої освіти	Перший (бакалаврський)
Спеціальність	Інформаційні системи та технології
Освітня програма	Інформаційні системи та технології
Група	6.04.126.010.18.1

ДИПЛОМНИЙ ПРОЕКТ

на тему: «Розроблення модуля інформаційної системи для виявлення зображень близького змісту»

Виконав: студент Марк ПАТЕР

Керівник: к.т.н., доцент Олексій ГОРОХОВАТСЬКИЙ

Консультант: -

Рецензент: к.т.н, доц. кафедри Інформатики
Харківського національного університету радіоелектроніки
доц. Валентин ЛЮБЧЕНКО

Харків – 2022 рік

РЕФЕРАТ

Дипломний проект містить: 56 сторінок, 23 рисунки, 2 таблиці, 36 джерел.

Об'єктом дослідження є методи пошуку ключових точок на зображенні, методи створення дескрипторів на їх основі і методи подальшого порівняння цих дескрипторів.

Метою роботи є імплементація модуля інформаційної системи у вигляді застосунку, який буде приймати у якості вхідних даних два зображення, давати користувачеві можливість обрати метод порівняння цих зображень на основі ключових точок, а у якості результату відображати показувати міру їх схожості у вигляді відсоткового значення.

Методами розроблення обрано методи порівняння зображень на основі ключових точок, а саме ORB, AKAZE та BRISK. Проведено дослідження якості відомих методів та їх спроможності класифікувати пару зображень близького змісту.

У результаті роботи проведено дослідження, здійснена програмна реалізація методу порівняння зображень із використанням алгоритмів на основі порівняння ключових точок, моделювання роботи алгоритмів на великій кількості даних.

ЗОБРАЖЕННЯ БЛИЗЬКОГО ЗМІСТУ, КЛЮЧОВА ТОЧКА, ДЕТЕКТОР, ДЕСКРИПТОР, ORB, AKAZE, BRISK.

ABSTRACT

Diploma project contains: 56 pages, 23 pictures, 2 tables, 36 sources.

The object of the research is the methods of features detection in the image, methods of descriptors creation based on them and methods of further comparison of these descriptors.

The goal of the research is to develop an information system module in the form of an application that will accept two images as input, allow the user to choose the method of comparing these images based on features, and as a result return the degree of similarity in percentage.

Methods of images comparison based on features, namely ORB, AKAZE and BRISK are applied. An analysis was made of the quality of known methods and their ability to classify a pair of images of similar content.

As a result, a study was conducted, the software implementation of the method of images comparison using algorithms based on the comparison of features carried out, simulation of algorithms on a large amount of data.

NEAR DUPLICATES, FEATURE, DETECTOR, DESCRIPTOR, ORB, AKAZE, BRISK.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів

КТ – ключові точки

AKAZE – Accelerated KAZE

BRIEF – Binary Robust Independent Elementary Features

BRISK – Binary Robust Invariant Scalable Keypoints

FAST – Features from Accelerated Segment Test

FLANN – Fast Library for Approximate Nearest Neighbors

FREAK – Fast Retina Keypoint

LSH – Locality Sensitive Hashing

MSE – Mean Squared Error

ND – Near-Duplicates

OpenCV – Open Source Computer Vision Library

QML – Qt Meta Language.

ORB – Oriented FAST and Rotated BRIEF

SSIM – Structural Similarity Index Measure

SNN – Siamese Neural Network

SURF – Speeded Up Robust Features

SIFT – Scale Invariant Feature Transform Keypoint

ВСТУП

У наш час цифровий контент є широко поширеним і в той самий час таким, що неймовірно легко створити. Більшість людей мають принаймні одну цифрову камеру, насамперед через широке використання смартфонів. При цьому якість камер з кожним роком зростає, як і кількість пам'яті на телефоні, де зберігається візуальна інформація.

Ці фактори сприяють тому, що люди все частіше роблять безліч знімків однієї і тієї ж сцени, щоб згодом вибрати найкращий кадр. Це у свою чергу призводить до постійного збільшення розміру бібліотеки і створює важливу проблему: пристрої та комп’ютери захаращені фотографіями, які лише трохи відрізняються і зображають майже одну й ту саму сцену.

При цьому наявність великої кількості фотографій, особливо гарної якості, значно збільшує розмір бібліотеки користувача. Нові камери дозволяють створювати зображення розміром десятки мегабайт при тому, що зображення можуть бути майже однаковими.

Такі зображення із близькім змістом називаються near duplicates (ND) і вони негативно впливають не тільки на розмір бібліотек фотографій, але і в цілому на якість та швидкість управління фотографіями та їх перегляду. Користувачам доводиться вручну переглядати набір зображень, виявляти випадки ND, а потім зазвичай залишати лише одне зображення, яке подобається найбільше. Часто трапляється так, що власники фотоколекцій взагалі не відбирають найкращі знімки, залишаючи безліч схожих зображень.

Проблема пошуку зображень близького змісту також є актуальною в системах автоматичної обробки зображень, наприклад, при аналізі відео чи послідовностей кадрів, які надходять для аналізу в режимі онлайн (один за одним), при реалізації систем пошуку зображень за запитом.

Метою дипломної роботи є розроблення модуля інформаційної системи, який реалізує метод виявлення зображень близького змісту на основі виявлення і зіставлення відомих ознак.

РОЗДІЛ 1
МЕТОДИ ВИЯВЛЕННЯ ЗОБРАЖЕНЬ БЛИЗЬКОГО ЗМІСТУ

1.1 Зображення близького змісту

З розвитком обчислювальної техніки спостерігається і стрімке покращення камер у ній. При цьому зростання якості зображень настільки велике, що сьогоднішні фотокамери на останніх смартфонах можуть видавати картинку не гірше за професійні кінокамери 10 років тому.

Гарна якість зображень обумовлена багато в чому тому, що висока роздільна здатність камери дозволяє записувати більше інформації про зафіксований кадр. Таким чином можна пояснити і високий об’єм пам’яті, що займає зображення.

З урахуванням збільшення кількості зображень та медіа-контенту останніми роками, виникла потреба в тому, щоб навчитися відповідати на запитання: "чи схожі два вибрані зображення між собою?".

Порівняння зображень - це проблема класу задач комп'ютерного зору, яка фокусується на пошуку множинних подібностей/розбіжностей між зображеннями, щоб зрештою відповісти на запитання: наскільки ці зображення схожі, чи можна їх вважати близькими за змістом. Ця задача відрізняється від багатьох інших задач класифікації зображень чи розпізнавання образів завдяки точності та суб’єктивності рішення. Якщо порівняти її, наприклад, із задачею розпізнавання осіб людей по фото обличчя, то можна відзначити принципові відмінності. Результат порівняння обличь для більшості людей є об’єктивним – майже напевно консенсус щодо того, чи є обличчя однаковими буде досягнуто для більшості людей. Задача пошуку подібних за змістом зображень є більш загальною – більшість людей буде вважати просто два зображення з різними обличчями однаковими в загальному сенсі, якщо вони створені, наприклад, на одному і тому ж фоні. Суб’єктивність полягає в тому, що кожна людина має свою точку зору щодо

того, чи є зміни між двома зображеннями суттєвими, щоб вважати зображення різними.

Отже, подібні за змістом зображення звуться near duplicates (ND), традиційно до відмінностей, які можуть бути присутні на одному з зображень відносять наявність шуму, артефактів стиснення, зміни колірного балансу, яскравості, контрасту, геометричних перетворень (масштабування, повороту, обрізання) та більш складні зміни сцени (пересування камери, зміна умов освітлення). Приклад ND відображено на рис. 1.1. Відповідно, зображення, які не є схожими ми будемо називати not near duplicates (NND).

Рисунок 1.1 – Приклад ND зображень

Існує декілька відомих методів зіставлення зображень, найбільш простими з яких є піксельні методи, та найбільш популярними – методи на основі зіставлення ознак.

1.2 Методи піксельного порівняння зображень

Найбільше наївним та інтуїтивним підходом у порівняння пари зображень є обчислення попіксельної різниці. Загальна ідея дуже проста – виконується порівняння яскравостей кожного пікселя на обох зображеннях. Механізм порівняння отримує кольори пікселів, які мають однакові

координати всередині зображення, і порівнює ці кольори.

Найбільш простою та відомою метрикою є значення MSE (Mean Squared Error), яке представляє собою середнє значення квадратів різниць між фактичними та очікуваними значеннями. Це значення є похибкою, і чим воно є меншим, тим більш подібними є зображення. Знаходження MSE можна записати у вигляді:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2, \quad (1.1)$$

де n – кількість точок даних (пікселів);

y – фактичне значення точки даних;

ŷ – очікуване значення точки даних.

Таким чином, одне зображення розглядається як еталонне, а друге – як зображення, значення пікселів якого ми хочемо порівняти з першим. Варто зазначити, що зображення повинні мати однакові розміри.

Щоб порівнювати зображення один з одним, нам потрібно якимось чином перетворити їх у порівнянні дані, які зможе розуміти комп'ютер (такими є числові дані). Зображення можна подати у вигляді тривимірного тензора, де кожному виміру відповідатиме матриця одного з кольорів RGB.

Виходячи з цих даних, концептуально алгоритм обчислення MSE для кольорових зображень виглядає наступним чином:

- розрахувати квадрат різниці між значеннями яскравостей пікселів в окремих RGB-каналах зображення, піксель за пікселем;

- обчислити суму квадратів різниці для всіх пікселів у кожному з RGB-каналів;

- отриманий результат поділити на 3 та на площу зображення.

Іншим показником є SSIM (Structural Similarity Index Measure), який кількісно визначає погіршення якості зображення, викликане обробкою, наприклад стисненням даних.

SSIM фактично вимірює різницю сприйняття між двома схожими

зображеннями, при цьому за допомогою нього не можна визначити, яке з двох зображень краще, якщо ми заздалегідь цього не знаємо.

На відміну від MSE, при визначенні SSIM порівнюються два «вікна» (тобто невеликі підвибірки), а не все зображення. При цьому ми обчислюємо не різницю між пікселями, а подібність у їх щільності. Ще однією важливою особливістю є те, що метод враховує «сприйняття помилки» завдяки врахуванню структурної зміни інформації. Ідея полягає в тому, що пікселі мають взаємозв'язок, який можна використовувати як інформацію про структуру об'єктів та про «вікно» в цілому.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + \mathrm{c}_1)(\sigma_x^2 + \sigma_y^2 + \mathrm{c}_2)}, \quad (1.2)$$

де x, y – порівнювані вікна, які мають однаковий розмір $N \times N$; μ_x, μ_y – середнє значення інтенсивності пікселів вікон; σ_x^2, σ_y^2 – дисперсії вікон; σ_{xy} – коваріація вікон.

Значення SSIM є нормованим і знаходиться в інтервалі від -1 до 1 (1 означає максимальну схожість, -1 означає максимальну різницю).

Обидва наведених методи є поширеними, але їх недоліком є те, що вони слабко враховують природній спосіб сприйняття і порівняння зображень людиною. Окрім цього, вони вимагають встановлення додаткових параметрів, які знаходяться емпіричним шляхом. Доведено також, що SSIM та MSE складаються з одних і тих самих значень, згрупованих різним чином та мають проблеми із обробкою зображень із геометричними перетвореннями, з низькою освітленістю та контрастом [3].

1.3 Методи порівняння зображень на основі зіставлення відомих ознак

Це сімейство методів є класичним для вирішення задачі пошуку

відомого об’єкта на зображенні. Кожне зображення ідентифікується за допомогою набору так званих ключових точок (feature keypoints, КТ) [19-23]. Саме ці точки при обробці зображень вважаються найцікавішими та найважливішими. Унікальність набору цих точок (ознак) для зображення вважається достатньо унікальною для їх співставлення між декількома зображеннями. Зазвичай порівняння зображень із використанням ознак виконується в три етапи:

- знаходження КТ;
- побудова векторів-дескрипторів для кожної точки;
- зіставлення наборів дескрипторів.

Знаходження КТ – це перший, фундаментальний етап в алгоритмі порівняння зображень, так само, як і етап створення дескрипторів на основі цих ознак, і їх подальше порівняння цих дескрипторів залежить від знайдених КТ [4-5].

КТ формуються за допомогою детектора ключових точок (feature detector), який знаходить точки на зображенні, які відрізняються, наприклад, за кольором, яскравістю або напрямком. Результат пошуку ознаки виражається у її розташуванні, тобто на виході ми матимемо координату (x, y). Прикладом такого детектора ознак є детектор кутів Харріса [7-8].

Окрім розташування особливих ознак (точок) для подальшого порівняння необхідно мати інформацію про властивості кожної ознаки, щоб їх можна було розрізняти не тільки за місцезнаходженням, але й за особливостями зовнішньої області навколо точки. Для цього існує дескриптор ознак (feature descriptor). Вхідним значенням для дескриптора ознак є розташування (x, y) ознаки на зображенні, а вихідним є вектор чисел (підпис), який і вважається дескриптором.

Щоб ідентифікувати один і той самий об’єкт на двох різних зображеннях, треба знайти дескриптори на двох зображеннях та порівняти їх за допомогою пошуку відповідних дескрипторів (feature matching). Алгоритм повного перебору (brute force) використовується в тому випадку, коли

кількість зображень, над якими нам потрібно виконати відповідність функцій, відносно невелика. Такий алгоритм передбачає, що ми будемо зіставляти кожен дескриптор одного зображення з кожним дескриптором іншого для знаходження найкращого збігу.

Окрім повного перебору, існує також метод співставлення ознак FLANN (Fast Library for Approximate Nearest Neighbours) [9]. FLANN – це бібліотека для швидкого пошуку найближчих сусідів у багатовимірних просторах. Система автоматично вибирає найкращий алгоритм та оптимальні параметри залежно від набору даних. Серед алгоритмів, які використовує FLANN для зіставлення дескрипторів, можна зазначити ієрархічне дерево k-means та рандомізоване k-d-дерево [10].

Існує декілька відомих методів пошуку ознак та побудови дескрипторів. Наприклад SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features), ORB (Oriented FAST and Rotated BRIEF), AKAZE (Accelerated KAZE), BRISK (Binary Robust Invariant Scalable Keypoints) – це комбінації дескриптора та детектора, в той час як FAST (Features from Accelerated Segment Test) – це лише детектор, а BRIEF (Binary Robust Independent Elementary Features) і FREAK (Fast Retina Keypoint) – це лише дескриптори. Іноді може бути необхідно використати комбінацію окремого детектора та дескриптора, наприклад FAST та BRIEF [11].

Дескриптори можуть бути з плаваючою комою (SIFT і SURF) або бінарними (BRIEF, ORB, BRISK і FREAK). Дескриптори з плаваючою комою мають високу точність, але не підходять для застосування в режимі реального часу через повільну швидкість роботи. Бінарні дескриптори розроблені з меншими вимогами до обчислень, тому працюють швидше.

У кожного з методів є як переваги, так і недоліки, тому той чи інший метод краще вибирати залежно від вхідних даних та задачі, що вирішується. Так, наприклад, детектор FAST не буде працювати добре для зображень з цифровим шумом або розмиттям, так як він не врахує масштаб, а дескриптор BRIEF не враховує поворот зображення в процесі побудови вектора опису

[12].

Порівняння найбільш відомих методів, можливості їх використання для роботи зі змінами масштабу, повороту та освітлення на зображенні представлено на рис. 1.2.

Detectors	Scale space	Invariance to rotation	Invariance to illumination	Descriptor	Invariance to scale	Invariance to rotation	Invariance to illumination
Harris	—	✓	✓	—	—	—	—
FAST	—	—	✓	—	—	—	—
BRISK	✓	—	✓	BRISK	✓	✓	✓
ORB	—	✓	✓	ORB	×	✓	✓
—	—	—	—	BRIEF	×	×	✓
—	—	—	—	FREAK	×	×	✓
KAZE	✓	✓	✓	KAZE	✓	✓	✓
SIFT	✓	✓	✓	SIFT	✓	✓	✓
SURF	✓	✓	✓	SURF	✓	✓	✓
DetNet	×	×	×	Pre-Net	✓	×	✓
TILDE	×	×	✓	Siamese-Net	×	✓	✓
LIFT	×	✓	✓	Triplet-Net	×	✓	✓
Multiscale	✓	✓	✓	LIFT	×	✓	✓
SuperPointer	✓	✓	✓	SuperPointer	✓	✓	✓

Рисунок 1.2 – Порівняння детекторів и дескрипторів [13]

1.4 Пошук із використанням хешування

Хешування зображень та порівняння хешів також є одним зі способів швидкого зіставлення зображень. Хешування в цьому випадку відрізняється від криптографічного, оскільки тут хеші для схожих частин зображення повинні також бути схожими. Криптографічне хешування (SHA1, MD5 та ін.) генерує зовсім різні значення хешу навіть для дуже близьких, але не однакових послідовностей. Ця властивість є зручною та ефективною для пошуку точних дублікатів, але не підходить для виявлення схожих зображень, що не є дублікатами.

Locality-sensitive hashing (LSH) – ймовірнісний спосіб зниження

розмірності багатовимірних даних. Цей метод дозволяє будувати структуру для швидкого наближеного (імовірнісного) пошуку n-мірних векторів, схожих на вхідний шаблон. Він часто використовується, коли є дуже великі обсяги даних, які необхідно порівняти між собою. LSH має велику кількість застосувань. Він використовується в таких областях, як розпізнавання образів, у класифікації та кластеризації, у пошуках плагіату, у роботі з великими системами документообігу, а також для пошуку ND.

Говорячи про пошук ND, алгоритм LSH складається з наступної послідовності дій:

Крок 1 – стиснення зображень та створення хешу. Хешування зображення – це такий процес дослідження вмісту зображення, який передбачає створення значення (хеш), яке однозначно ідентифікує зображення на основі цього вмісту. Таким чином ми можемо застосувати хеш-функцію та обчислити «хеш зображення» на основі його візуального вигляду. Зображення, які є ND, також повинні мати хеші, які є «подібними».

Крок 2 – створення пар-кандидатів. В ідеалі потрібно порівнювати лише потенційно відповідні хеші.

Крок 3 – на цьому етапі знайдені раніше пари кандидатів порівнюються одна з одною для пошуку відповідностей. Часто для порівняння значень хешів використовується відстань Хеммінга, тобто кількість бітів, у яких обидві сигнатури різні. Якщо це число, поділене на довжину сигнатури (відстань Хеммінга / k^2) є нижчим за якийсь наперед визначений поріг t, можна вважати зображення, на яких було згенеровано ці хеші, близькими (ND). Наприклад, порогове значення t, що дорівнює 0.8, встановлює 80% рівень відповідності двох зображень.

У LSH немає єдиного підходу до хешування. Дійсно, усі методи мають однакову логіку «зберігати подібні зразки за допомогою хеш-функції», але вони можуть значно відрізнятися. При цьому хоча хеш-функції як результат повинні повернути дані зменшеного фіксованого розміру, але роблять вони це по-різному.

Алгоритм середнього хешування спочатку перетворює вхідне зображення у відтінки сірого, а потім зменшує його. Наприклад, якщо ми хочемо створити 64-бітний хеш, зображення зменшується до 8×8 пікселів. Далі обчислюється середнє значення яскравості на основі всіх сірих пікселів зображення, а потім пікселі розглядаються один за одним, і якщо значення сірого кольору більше за середнє, до хеша додається 1, інакше 0 (рис. 1.3).

Рисунок 1.3 – Середнє хешування зображення [36]

Алгоритм хешування блоків ділить зображення на блоки і генерує значення для кожного блоку, 1 або 0. Наприклад, якщо нам потрібен 64-бітовий хеш, зображення ділиться на 64 блоки.

Диференційний хеш, подібно до середнього хеш-алгоритму, спочатку генерує зменшене зображення у відтінках сірого з вхідного, а потім з кожного рядка перші 8 пікселів розглядаються послідовно зліва направо і порівнюються з сусідніми справа (рис. 1.4).

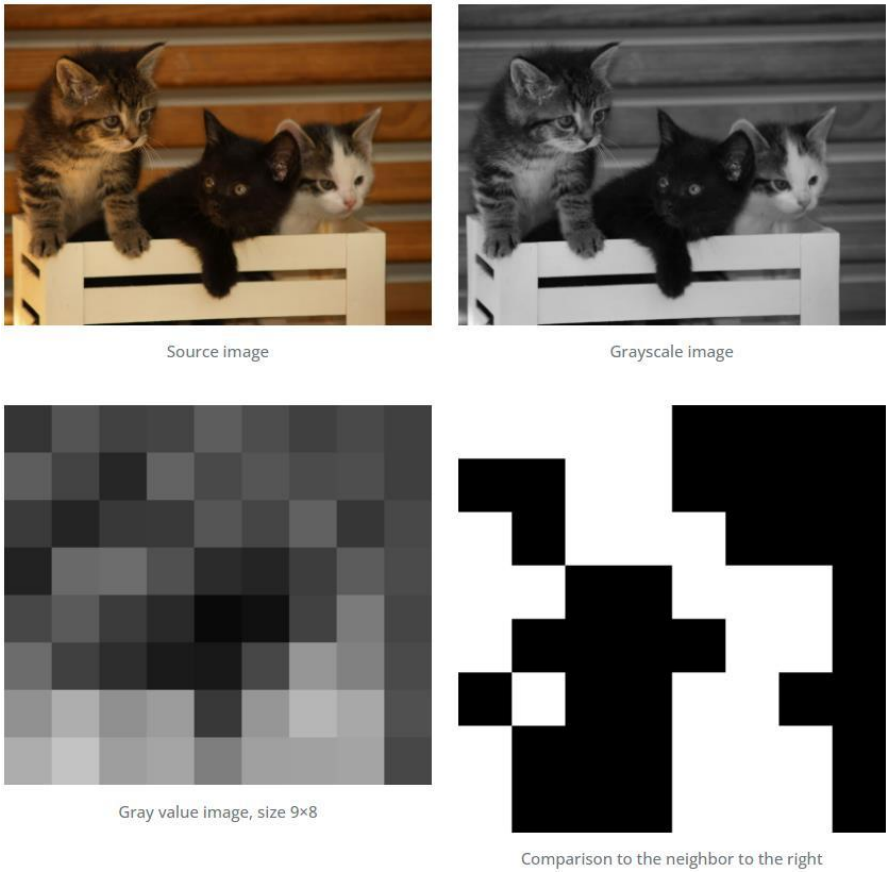

Рисунок 1.4 – Диференційне хешування [36]

Алгоритм медіанного хешування працює подібно до середнього хеш-алгоритму, за винятком того, що він порівнюється не із середнім значенням сірого кольору, а з медіаною.

Алгоритм перцептивного хешування також спочатку обчислює зображення сірого значення, потім до отриманого зображення застосовується дискретне косинусне перетворення спочатку на рядок, а потім на стовпець. Після цього пікселі з високими частотами будуть розташовані у верхньому лівому куті, для отримання хеш-значення треба підрахувати медіану сірих значень на цьому сегменті та порівняти їх аналогічну медіанному хешуванню.

Недолік LSH полягає в точності одержуваного результату. Через те, що розмір хешу часто дуже обмежений, це може впливати на запам'ятовування важливої інформації, яка може допомогти визначити, чи є зображення ND.

1.5 Нейромережеві методи порівняння

Використання нейронних мереж набуло надзвичайної популярності останніми рокам для вирішення багатьох практичних задач, в тому числі, і для пошуку зображень близького змісту. Оскільки в цьому випадку треба обробляти пару зображень, зручно використовувати так звані сіамські нейронні мережі (SNN).

SNN – це клас нейронних мереж, які містять дві або більше однакових підмереж. Вперше вони були представлені на початку 1990-х років для вирішення проблеми перевірки підписів як задачі зіставлення зображень [17].

Ці мережі є перцептронами з прямим зв'язком, які працюють паралельно, а для отримання остаточного результату порівнюють свої вихідні дані, використовуючи показник подібності, який представляє взаємозв'язок між двома вхідними даними. Вони також використовують однакові ваги під час роботи з різними вхідними даними.

Підмережі SNN з’єднані разом через так званий повнозв’язний шар (dense), який потрібен для визначення подібності між вхідними даними. SNN приймає вхідні дані x1 та x2 (рис. 1.5), які можуть бути зображеннями, текстом або навіть записаним мовленням, а потім перетворює їх залежно від типу вхідних даних в необхідний формат.

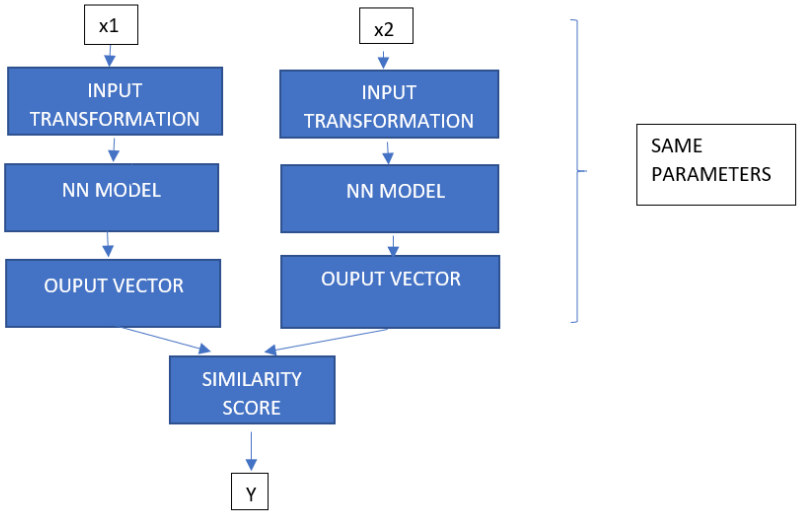

Рисунок 1.5 – Архітектура роботи SNN [32]

Після паралельного навчання мереж з однаковими вагами будуть сгенеровані вихідні дані в обох підмережах. На основі цих вихідних даних виконується визначення показника подібності через косинусну відстань між ними. Таким чином вимірюється подібність за допомогою косинуса кута між двома векторами в багатовимірному просторі:

$$similarity(x, y) = \cos(\theta) = \frac{x * y}{|x||y|}, \quad (1.4)$$

де x та y – вихідні дані, представлені у якості векторів.

Ця оцінка знаходиться в діапазоні від -1 до 1. Якщо оцінка близька до -1, то об'єкти порівняння відрізняються, а якщо оцінка близька до 1, то об'єкти можна віднести до ND.

Особливість такої нейронної мережі у тому, що вона здатна вчитися на дуже малій кількості даних. Але така перевага зумовлено тим недоліком, що їй необхідно більше часу на навчання, ніж звичайним (pointwise) мережам, оскільки сіамські мережі включають квадратичні пари для навчання.

1.6 Постановка задачі

Порівняння зображень та знаходження схожих за змістом серед них є актуальним завданням в сфері компьютерного зору.

Об’єктом дослідження в роботі є методи пошуку ключових точок на зображенні, методи створення дескрипторів на їх основі і методи подальшого порівняння цих дескрипторів.

Метою виконання дипломного проекту є імплементація модуля інформаційної системи у вигляді застосунку, який буде приймати у якості вхідних даних два зображення, давати користувачеві можливість обрати метод порівняння цих зображень на основі ключових точок, а у якості результату відображати міру їх схожості у вигляді відсоткового значення.

Логічне рішення про те, чи є пара зображень схожими, приймається

особисто людиною. Для того, щоб зробити цей процес більш ефективним та автоматичним, доцільним є дослідження якості відомих методів та їх спроможності класифікувати пару зображень.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ ВИЯВЛЕННЯ І ЗІСТАВЛЕННЯ ОЗНАК

2.1 Детектори ключових точок

Методи порівняння зображень, засновані на пошуку відповідностей між характерними точками, завжди мають на початку етап пошуку координат цих особливих точок. Для виявлення ознак використовуються три методи: виявлення країв (edge detection), виявлення кутів (corner detection) і виявлення краплі (blob detection).

Виявлення країв (границь) - це метод, що використовується для виявлення ознак завдяки розривам, просто кажучи, різким змінам яскравості зображення. В цілому, основна мета більшості детекторів країв полягає в обчисленні градієнта яскравості на зображенні. Таким чином вони виділяють області з високою просторовою частотою, які часто відповідають краям. Як вхідні, так і вихідні дані – це зображення в градаціях сірого. Зазвичай їх перетворюють у відтінки сірого для того, щоб зменшити обчислювальні витрати.

Найпростішим і раннім методом виявлення країв є перехресний оператор Робертса, який обчислює суму квадратів різниць між діагонально суміжними пікселями. Це може бути виконано згорткою зображення з двома ядрами:

$$\begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix} \text{ і } \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}, \tag{2.1}$$

Таким чином в методі Робертса використовується сумарний вектор із двох діагональних векторів перепаду.

Покращеною версією оператора Робертса є оператор Собеля, який використовує значення інтенсивності тільки в околиці 3×3 кожного пікселя

для отримання наближення відповідного градієнта зображення, і використовує тільки цілі значення вагових коефіцієнтів яскравості для оцінки градієнта.

У кожній точці зображення наближене значення величини градієнта можна обчислити за наступною формулою:

$$G = \sqrt{G_x^2 + G_y^2}, \tag{2.2}$$

де G_x і G_y – два зображення, на яких кожна точка містить наближені похідні за x і за y. Вони обчислюються наступним чином:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathrm{A} \text{ і } G_y = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathrm{A}, \tag{2.3}$$

де A – вхідне зображення.

Цей тип пошуку КТ доволі простий, але має ряд вагомих недоліків. Так, якщо ми говоримо про прості методи по типу Робертса та Собеля, то вони не можуть бути використані для зображень з цифровим шумом. Детектор Кені [14] за рахунок фільтра Гауса дозволяє прибрати будь-які шуми на зображенні, але це призводить до проблем з використанням такого детектора у реальному часі через складні обчислення.

Виявлення кутів було запропоновано в 1977 році, коли був створений перший кутовий детектор [6]. З тих пір було розроблено багато альтернатив, найпопулярнішим з яких є детектор кутів Харріса, але жодному з них не вдається виявляти виключно кути. Кутами називають точки, утворені від перетину двох або більше ребер (рис. 2.1). Краї зазвичай описують межі різних частин об’єкта або двох об’єктів і, дивлячись на напрямок цих країв, можна виявити кути.

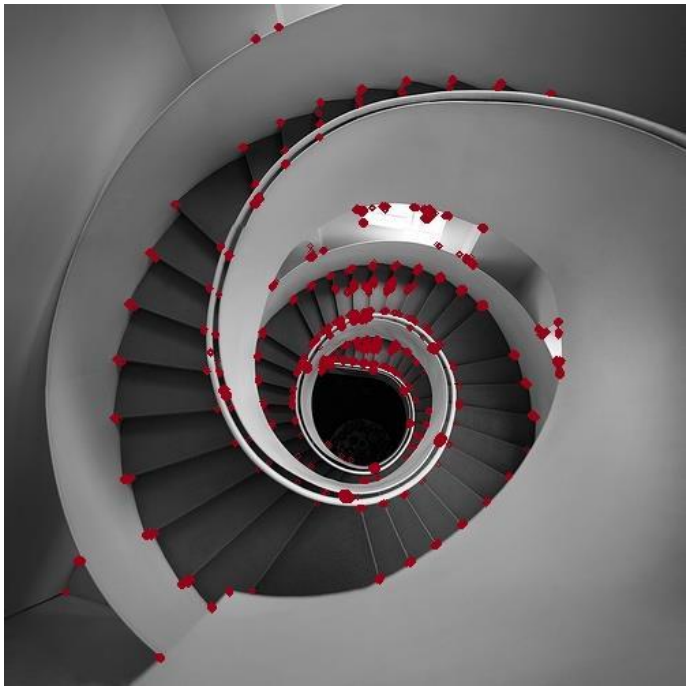

Рисунок 2.1 – Ключові точки

Кутовий детектор Харріса визначає точку-кандидат як кутову точку шляхом обчислення градієнтів. Він є інваріантним до повороту зображення, афінних перетворень, інтенсивності та зміни точки зору у відповідних ознаках [8].

Метод складається з трьох кроків. Перший – розрахунок відповідного градієнту за математичним визначенням. Другий крок полягає у виборі кутової точки-кандидата з урахуванням заданого порогу. Третій крок полягає у підборі оптимальних координат для позначення кутів, для цього знаходять локальні максимуми як кути у вікні, яке є фільтром розміром 3x3.

Формально детектор кутів Харріса визначається як:

$$M_{Harris} = \begin{bmatrix} A & C \\ C & B \end{bmatrix}, A = (I_x)^2 \oplus w, B = (I_y)^2 \oplus w, C = (I_x I_y)^2 \oplus w, \quad (2.4)$$

де M – квадрат різниці матриці; $\oplus$ – операція згортки; w – ядро згортки, яке називається вікном фільтра; I_x та I_y – градієнтні представлення вхідного зображення.

Для отримання означення R (2.5), яке і буде визначати, чи був знайдений саме кут, використовується наступна формула:

$$R = AB - C^2 - k(A + B)^2, \quad (2.5)$$

де k – коефіцієнт чутливості та константа, яка, як правило, дорівнює 0,04.

Якщо отримане значення R вище за вказаний поріг, то можна вважати, що кут знайдений.

Оскільки детектор кутів виявляє не тільки кути, але й інші ознаки, такі як кінці ліній, необхідно використовувати інші методи, щоб переконатися, що виявлені точки є реальними кутами. З іншого боку, кутові детектори мають можливість знаходити один і той же кут на різних зображеннях, що забезпечує високу повторюваність ознаки.

Кутовий детектор FAST, запропонований Е. Ростеном та Т. Драммондом [33], є удосконаленим детектором Харріса за рахунок бінарного розрахунку та машинного навчання [12].

Нехай піксель l_p розглядається як кандидат кута (рис. 2.2). Цей піксель можна ідентифікувати як кутову точку, якщо існує набір суміжних пікселів на шаблоні кола Брезенхема [34], і всі вони яскравіші за інтенсивність пікселя-кандидата $l_p + t$, або навпаки, всі темніші, ніж $l_p - t$, де t – порогове значення інтенсивності.

Теоретично радіус шаблону кола може бути будь-якого розміру або масштабу. Оригінальний алгоритм використовував 3 пікселя у якості радіусу, перевіряючи 12 пікселів в отриманій окружності. Для FAST перевірки виконується тільки для пікселів на місцях 1, 5, 9 і 13, як показано на рис. 2.2.

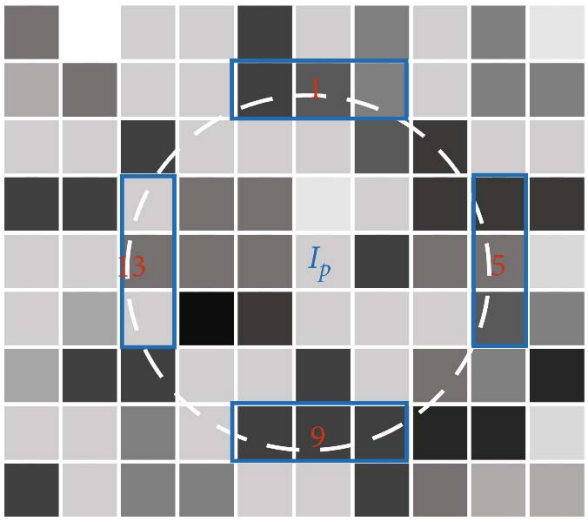

Рисунок 2.2 – Виявлення кута через алгоритм FAST [13]

Основу для методу пошуку кутів у вигляді «краплі» було закладено приблизно в 1990-х роках, коли були зроблені спроби виявити одні й ті ж кути на зображеннях об’єкта в різних масштабах [8]. У 1993 році професор Тоні Ліндеберг з KTH (Kungliga Tekniska Högskolan) опублікував дослідження [16] з використанням математичного методу для обробки варіації ознак у різних масштабах. Ця робота стала основою детектора, який би міг працювати з багатомасштабними зображеннями. Кутові детектори не могли виявити з точністю ознаки в багатомасштабних зображеннях, тому довелося створити новий метод для боротьби зі змінами масштабу зображення. Це було важливо подолати, оскільки погане виявлення ознак може призвести до неточного виявлення об’єкта.

Краплями називають ділянки, в яких група пікселів має однакові властивості. Кожна ділянка має різні властивості в порівнянні з сусідніми ділянками, що робить кожну краплю відмінною від усіх інших. Детектори крапель класифікуються на основі математичних методів, які вони використовують. Математично крапля представлена у вигляді пари, що складається з сідлової точки та однієї точки екстремуму, завдяки чому вона виглядає як пік у частотній області (показано на рисунку 2.3).

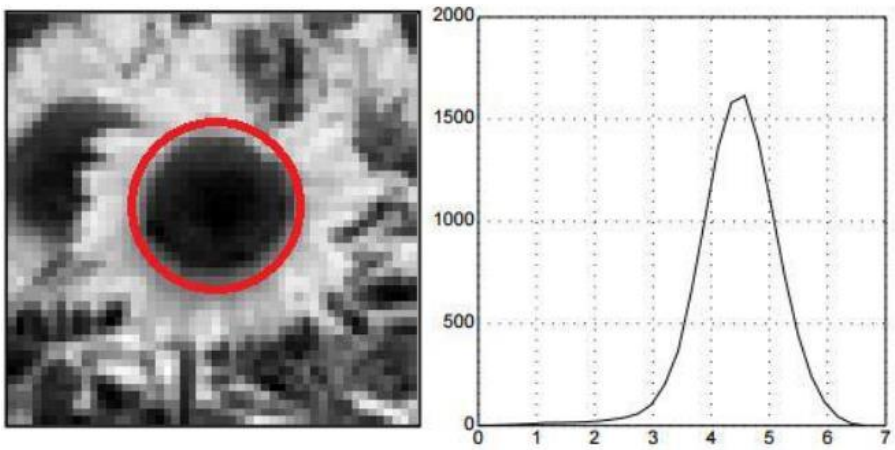

Рисунок 2.3 – Виявлення крапель [16]

2.2 Дескриптори ознак

Окрім розташування особливих точок, для подальшого їх зіставлення між двома зображеннями необхідно отримати дескриптори. Вхідним значенням для дескриптора ознак є розташування (x, y) ознаки на зображенні, а вихідним є вектор чисел певних параметрів (підпис). Дескриптор ознак кодує цікаву інформацію в ряд чисел і діє як свого роду числовий «відбиток пальця», який можна використовувати, щоб відрізнити одну ознаку від іншої. Цей ряд чисел описується таким чином, який (в ідеалі) є інваріантним щодо змін освітлення, трансляції, масштабу та обертання в площині.

Сучасні дескриптори, як, наприклад, BRIEF, FREAK, A-KAZE та ORB, описують точку у вигляді бінарного вектора.

Пошук ключових точок через алгоритм A-KAZE базується на нелінійному масштабуванні зображення за схемою FED (Fast Explicit Diffusion). Як бінарний дескриптор використовується M-LDB (Modified-Local Difference Binary).

Виявлення КТ здійснюється шляхом пошуку точок локального максимуму Гессе після нормалізації в різних масштабах. Матриця Гессе розраховується як:

$$B_{Hessian} = \sigma^2(B_{xx}B_{yy} - B_{xy}^2), \quad (2.6)$$

де B_{xx} – горизонтальна похідна другого порядку, B_{yy} – вертикальна похідна другого порядку, B_{xy} – крос-похідна другого порядку.

Дескриптор M-LDB використовує інформацію про градієнт та інтенсивність з нелінійного масштабного простору. На відміну від дескриптора LDB, дескриптор M-LDB отримує інваріант обертання шляхом оцінки основної орієнтації ключової точки і повороту сітки LDB. M-LDB використовує похідні, обчислені на етапі виявлення ознак, що скорочує кількість операцій, необхідних для побудови дескриптора.

ORB – це комбінація детектора FAST і вдосконаленого дескриптора BRIEF [26], який розглядається як заміна SIFT, так як досягає кращої ефективності за допомогою бінарного опису, що дозволяє використовувати його в задачах реального часу. Його перевагою також є інваріантність орієнтації в дескрипторах.

Якщо розбити роботу алгоритму покроково, отримаємо наступну послідовність дій:

Крок 1 – знаходження КТ за допомогою детектора FAST. Використовуючи деревовидний швидкий алгоритм FAST на оригінальному документі і на кількох зображеннях з піраміди зменшених зображень ми отримуємо КТ [27].

Крок 2 – вибір КТ з урахуванням обчисленої міри Харріса, кандидати з низьким значенням міри відкидаються [28].

Крок 3 – Розрахунок кута орієнтації КТ. ORB використовує центроїд інтенсивності для вимірювання кутової орієнтації. Центроїд ділянки Ip ключової точки обчислюється за наступними формулами:

$$\forall p, 1 \in \{0,1\} : m_{pq} = \sum_{x,y} x^p y^q I\,(x, y), \tag{2.7}$$

$$\theta = arctan2(m_{01}, m_{10}) \tag{2.8}$$

де x, y – піксельні координати, I – яскравість.

Рівняння 2.6 знаходить так звані моменти зображення, тобто деякі середньозважені інтенсивності пікселів зображення, а рівняння 2.8 – центроїд ознаки, тобто кут орієнтації КТ.

Крок 4 – зміна напряму КТ за отриманим кутом. Нові положення точок описуються за формулою:

$$\begin{pmatrix} x_i' \\ y_i' \end{pmatrix} = R(\theta) \begin{pmatrix} x_i \\ y_i \end{pmatrix} \tag{2.9}$$

Крок 5 – проведення розрахунку бінарного дескриптора BRIEF за точками, отриманими після зміни напряму.

BRISK будує багатомасштабний простір для виявлення та проектування шаблонів вибірки для орієнтації, що забезпечує інваріантність до масштабу та обертання [31].

По-перше, масштабно-просторова піраміда будується шляхом понижувальної дискретизації вихідного зображення c_0 з 4 октавами c_i та 4 інтраоктавами d_i, кожна з яких знаходиться між шарами c_i та c_{i+1} (рис. 2.4).

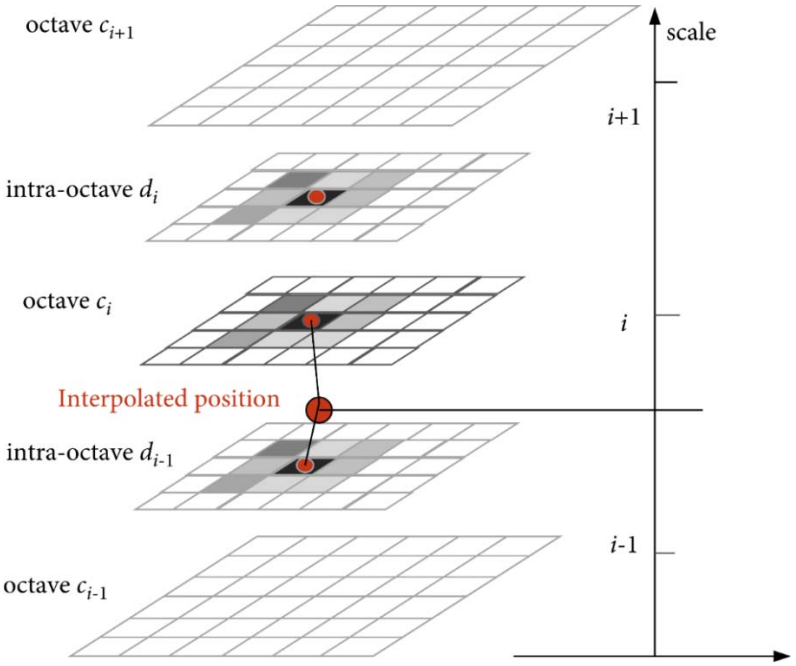

Рисунок 2.4 – Побудова багатомасштабного простору [13]

По-друге, шаблон вибірки відрізняється від BRIEF, де пари точок вибірки розташовані однаково на колах, концентричних з ключовою точкою, і згладжуються ядром Гауса, щоб уникнути ефектів накладання. На рис. 2.5 показано положення 60 точок відбору зразків кутової точки.

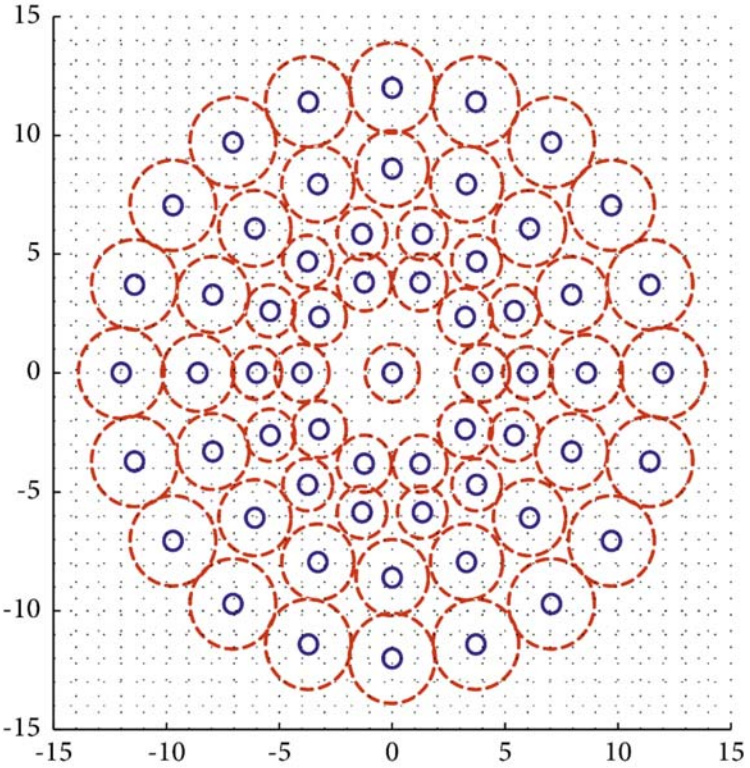

Рисунок 2.5 – Шаблон вибірки BRISK з 60 точок [13]

Червоні пунктирні кола – це ядра Гаусса, які використовуються для згладжування околиць. Сині кола – розташування вибірок.

Крім того, для досягнення інваріантності до обертання напрямок візерунка кутової точки визначається наступним чином:

$$g = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \cdot \sum_{(p_i, p_j) \epsilon L} g(p_i, p_j), \tag{2.10}$$

де $g(p_i, p_j)$ – локальний градієнт, а L – підмножина пар точок на великій відстані.

2.3 Співставлення ознак

Найпростішим співставленням ознак є алгоритм повного перебору (brute force matching). Він бере дескриптор однієї ознаки в першому наборі та порівнює з усіма іншими ознаками у другому наборі за допомогою обчислення відстані. Та відстань, яка буде найменша, і буде вважатися парою. При цьому для різних детекторів використовуються різні міри розрахунку відстані. Так, для ORB, BRIEF і BRISK використовується розрахунок відстані Хеммінга, а для SIFT і SURF - манхеттенська та евклідова відстані. Недоліком цього методу в контексті порівняння дескрипторів є те, що він завжди знайде найближчий дескриптор, навіть якщо він дуже відрізняється від шуканого, а інші дескриптори відрізняються ще більше.

Алгоритми співставлення можуть повертати не тільки найкращий збіг, але й певну кількість кращих збігів, що може бути корисним у випадку проведення теста Девіда Лоу [24]. Він запропонував простий метод фільтрації збігів за ключовими точками, виключаючи збіги, коли другий найкращий збіг майже такий самий.

Припустимо, що L1 - це набір ключових точок зображення 1, а L2 - це набір ключових точок для зображення 2. Спочатку порівнюються ключові точки в L1 з двома ключовими точками в L2, які є найкращими збігами для кожної ключової точки. Виходячи з припущення, що ключова точка на зображенні 1 не може мати більше одного еквіваленту на зображенні 2, ми робимо висновок, що обидва ці збіги не можуть бути правильними: принаймні один з них неправильний. Відповідно до міркувань Лоу, збіг з найменшою дистанцією є «хорошим», а збіг із другою найменшою відстанню є еквівалентом випадкового шуму. Якщо «хороший» збіг не можна відрізнити від шуму, то його слід відхилити, оскільки він не приносить нічого інформаційного. Отже, загальний принцип полягає в тому, що між найкращим і другим кращим матчами має бути достатня різниця, за замовчуванням це 25%.

Недоліком методу повного перебору в контексті вирішення задачі пошуку зображень близького змісту є те, що до будь-якого дескриптора на першому зображенні завжди знайдеться відповідний дескриптор на другому, навіть якщо відстань між ними є великою. Тож, додатково можна розглянути можливість пошуку тільки таких відповідностей дескрипторів, відстань мі якими не перевищує деяке порогове значення.

РОЗДІЛ 3
РОЗРОБЛЕННЯ МОДУЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ВИЯВЛЕННЯ ЗОБРАЖЕНЬ БЛИЗЬКОГО ЗМІСТУ

3.1 Обґрунтування вибору технічних засобів розробки

Для реалізації модулю інформаційної системи для виявлення зображень близького змісту було застосовано мови програмування Python, QML, а також бібліотеку комп'ютерного зору OpenCV [29].

Python – високорівнева скриптова мова програмування. Його відмінна риса – універсальність, адже він чудово підходить для вирішення різноманітних завдань. Сьогодні він застосовується в багатьох сферах, таких як Machine Learning, розробка програмного забезпечення, WEB, обробка зображень, парсинг файлів та багато чого ще.

Python є інтерпретованою мовою програмування, програмний код якої не компілюється. Таким чином, до момента запуску він є звичайним текстовим файлом. Відповідно, програмувати можна майже на всіх платформах, а сама мова є логічною і добре спроектованою. При цьому обсяг коду часто виходить меншим, ніж при використанні інших мов програмування, тому розробка здійснюється швидше.

Головною перевагою на користь Python є велика кількість модулів, що існують та підтримуються комп'ютерною спільнотою для вирішення різних задач, наприклад, для досліджень, пов'язаних із виконанням роботи застосовано SciPy, NumPy, Matplotlib і OpenCV. Завдяки наявності спеціалізованих бібліотек, а також через простоту освоєння, багато вчених (фізики, математики, біологи) вибирають саме цю мову.

OpenCV – це відкрита бібліотека для роботи з алгоритмами комп'ютерного зору, машинним навчанням та обробкою зображень. Написана на C++, але існує також для Python, JavaScript, Ruby та інших мов програмування. Працює на Windows, Linux та MacOS, iOS та Android.

Бібліотека має понад 2500 оптимізованих алгоритмів, які включають повний набір алгоритмів комп'ютерного зору та машинного навчання. Ці алгоритми можуть використовуватися для виявлення та розпізнавання осіб, ідентифікації об'єктів, класифікації дій об’єкта на відео, відстеження об'єктів, що рухаються, вилучення тривимірних моделей об'єктів, пошук схожих зображень у базі даних зображень тощо.

Спільнота налічує десятки тисяч людей, а кількість завантажень перевищує 18 мільйонів. При цьому OpenCV широко використовується як у великих компаніях, таких як Google, Microsoft, Apple, Intel, IBM, Sony, Toyota, так і в малих дослідницьких групах, державних органах, стартапах: Applied Minds, VideoSurf, Zeitera та інші.

Крім кросплатформенності та підтримки багатьох мов програмування, які дозволяють використовувати програми на різних системах, бібліотека OpenCV дуже ефективна (у порівнянні з іншими схожими бібліотеками, як наприклад MatLAB) з точки зору обчислень, оскільки майже всі функції та оператори в ній векторизовані.

QML (Qt Meta Language) – це декларативна мова програмування, яка дозволяє описувати інтерфейси користувача в термінах їх візуальних компонентів і того, як вони взаємодіють і співвідносяться один з одним. QML була розроблена для забезпечення можливості динамічного з'єднання компонентів, тому вона дозволяє легко повторно використовувати і настроювати компоненти в інтерфейсі користувача.

3.2 Розроблення інтерфейсу модуля застосунку

Інтерфейс застосунку надає наступний функціонал:

- вибір методу пошуку ключових точок та створення дескрипторів. Надається 4 варіанти: SIFT, ORB, AKAZE та BRISK.
- вибір зображень для порівняння. Можна обрати зображення через окремі кнопки, або просто перетягнути їх безпосередньо на форму застосунку

через drag&drop.

- можливість розпочати аналіз зображень та їх порівняння, а також можливість очистити результат порівняння.

Результат порівняння включає інформацію про обрані зображення (шляхи до відповідних файлів у файловій системі), знайдену кількість ключових точок на обох фотографіях, кількість ключових точок після проведення ratio-тесту та гомографії, і як фінальний результат – відсоток того, яка кількість дескрипторів одного зображення збіглася з дескрипторами іншого. Чим вище цей відсоток, тим більше зображення відносяться до типу ND.

Для створення обраного детектора, який у той же час є і дескриптором, було створено функцію create_detector. У якості параметру функція отримує строку, яку порівнює з детекторами, які можна використати у програмі. Результатом роботи функції є створений детектор – один із наслідників абстрактного класу Feature2D, який ми використовуватимемо для виявлення КТ і надалі для формування дескрипторів (рис. 3.1).

```
def create_detector(detector):
    if detector == 'sift':
        return cv.SIFT_create()
    elif detector == 'orb':
        return cv.ORB_create(10000)
    elif detector == 'akaze':
        return cv.AKAZE_create()
    elif detector == 'brisk':
        return cv.BRISK_create()
    return None
```

Рисунок 3.1 – Функція створення детектора

В залежності від обраного типу детектора створюється об’єкт класу порівняння дескрипторів в функції create_matcher. При цьому для всіх типів детектора алгоритм зіставлення один – алгоритм повного перебору, але параметр normType класу BFMatcher залежить від обраного типу (рис. 3.2).

```
def create_matcher(detector):
    if detector == 'sift':
        return cv.BFMatcher(cv.NORM_L2)
    return cv.BFMatcher(cv.NORM_HAMMING)
```

Рисунок 3.2 – Функція створення об’єкта для порівняння

Після порівняння дескрипторів важливо відкинути ті збіги, які не задовольняють ratio-тесту, який запропонував Девід Лоу [24]. Таким чином ми отримає тільки збіги, які з більшою долею імовірності є вірними. Для цього було створено функцію filter_matches, яка у якості параметрів приймає КТ обох зображень, а також результат зіставлення дескрипторів. У якості результату повертає вже відфільтровані колекції КТ (рис 3.3).

```
def filter_matches(kp1, kp2, matches, ratio = 0.75):
    mkp1, mkp2 = [], []
    for m in matches:
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            best_m = m[0]
            mkp1.append( kp1[best_m.queryIdx] )
            mkp2.append( kp2[best_m.trainIdx] )
    p1 = np.float32([kp.pt for kp in mkp1])
    p2 = np.float32([kp.pt for kp in mkp2])
    return p1, p2
```

Рисунок 3.3 – Ratio-тест

Функція check_errors перевіряє на валідність вхідні дані зображень та детектора (рис. 3.4). Зображення не повинні бути порожніми об’єктами після виконання функції imread, тобто файли повинні існувати. Така перевірка зумовлена тим, що між завантаженням зображень і проведенням тесту з ними може щось статися, наприклад, вони можуть бути видалені, і для запобігання помилкам тест у такому разі проводитися не повинен. Детектор також повинен існувати, інакше неможливо отримати КТ та дескриптори до них.

```
def check_errors(img1, img2, detector):
    if img1 is None:
        print('Error: failed to load first image')
        sys.exit(1)
    if img2 is None:
        print('Error: failed to load second image')
        sys.exit(1)
    if detector is None:
        print('Error: failed to create detector')
        sys.exit(1)
```

Рисунок 3.4 – Перевірка вхідних даних

Головна функція має назву compare, у якості аргументів вона приймає тип детектору та шляхи до зображень. Функція створює об'єкти обраних зображень, трансформує їх у зображення у відтінках сірого кольору, отримує детектор та об'єкт для порівняння. Після цього виконує валідацію даних, після чого створює КТ та дескриптори, зіставляє та фільтрує їх. Якщо кількість зіставлених ознак більше ніж 3, проводиться гомографія для отримання перетворення у вигляді матриці, яка співвідносить КТ одного зображення у відповідні точки на іншому зображенні (рис. 3.5).

```
def compare(detector_type, first_image_url, second_image_url):
    img1 = cv.imread(first_image_url, cv.IMREAD_GRAYSCALE)
    img2 = cv.imread(second_image_url, cv.IMREAD_GRAYSCALE)
    detector = create_detector(detector_type)
    matcher = create_matcher(detector_type)

    check_errors(img1, img2, detector)

    features1, descriptors1 = detector.detectAndCompute(img1, None)
    features2, descriptors2 = detector.detectAndCompute(img2, None)
    raw_matches = matcher.knnMatch(descriptors1, descriptors2, k = 2)
    matched_features1, matched_features2 = filter_matches(features1, features2, raw_matches)

    # It needs atleast 4 correct points to find the transformation.
    if len(matched_features1) >= 4:
        _, status = cv.findHomography(matched_features1, matched_features2, cv.RANSAC, 5.0)
        inliers = int(np.sum(status))
        matched = len(status)
        return [len(features1), len(features2), inliers, matched, "{:.2f}%".format(inliers * 100 / matched)]
    else:
        return [len(features1), len(features2), len(matched_features1), 0, 0]
```

Рисунок 3.5 – Функція порівняння зображень

Функція compare повертає колекцію чисел, яка містить інформацію про кількість знайдених КТ на обох зображеннях, кількість КТ, отриманих після гомографії і кількість тих, які вдалося зіставити між собою. Також вона містить значення точності, яке ми використовуємо для відображення відсотка зіставлених КТ. Це значення розраховується наступним чином:

$$Similarity = \frac{inliers \cdot 100\%}{matched}, \tag{3.1}$$

де matched – кількість КТ, отриманих після трансформації, а inliers – кількість КТ, які при цьому були зіставлені між собою.

Головна функція __main__ містить налаштування середовища Qt (рис. 3.6). Налаштування встановлює стиль Material та створює об'єкти головних класів для роботи з графічним інтерфейсом – QGuiApplication та QQmlApplicationEngine, які працюють з файлом view.qml, що містить код GUI.

```
if __name__ == '__main__':
    app = QGuiApplication(sys.argv)
    QQuickStyle.setStyle("Material")
    engine = QQmlApplicationEngine()

    # Get the path of the current directory, and then add the name
    # of the QML file, to load it.
    qml_file = Path(__file__).parent / "ui/view.qml"
    engine.load(qml_file)

    if not engine.rootObjects():
        sys.exit(-1)

    sys.exit(app.exec())
```

Рисунок 3.6 – Налаштування середовища Qt.

Для зручної інтеграції QML у Python та взаємодії верхнього рівня абстракції з нижнім був створений клас Bridge, який містить обробники на події зміни детектора та запуску зіставлення зображень (рис. 3.7).

```
@QmlElement
class Bridge(QObject):

    @Slot(str)
    def setDetector(self, d):
        self.detector = d.lower()

    @Slot(str, str)
    def compareImages(self, firstImageUrl, secondImageUrl):
        firstImagePath = unquote(urlparse(firstImageUrl).path)
        secondImagePath = unquote(urlparse(secondImageUrl).path)
        self.showResult.emit(firstImagePath, secondImagePath,
                             compare(self.detector, firstImagePath, secondImagePath))

    detector = "sift"
    showResult = Signal(str, str, list)
```

Рисунок 3.7 – Структура класу Bridge

Для роботи програмного застосунку були використані зображення (рис. 3.8) із набору INRIA [30]. Цей набір містить 1491 зображення, які можна розділити на 500 груп, кожна з яких буде складатися з зображень типу ND.

Рис. 3.8 – Приклади досліджуваних зображень із набору INRIA

Інтерфейс розробленої програми містить вибір алгоритму знаходження КТ та створення дескрипторів на їх основі, вибір зображень, можливість розпочати порівняння зображень та очищення результатів (рис. 3.9). Зображення відображаються у відведених для них областях, виділених пунктиром.

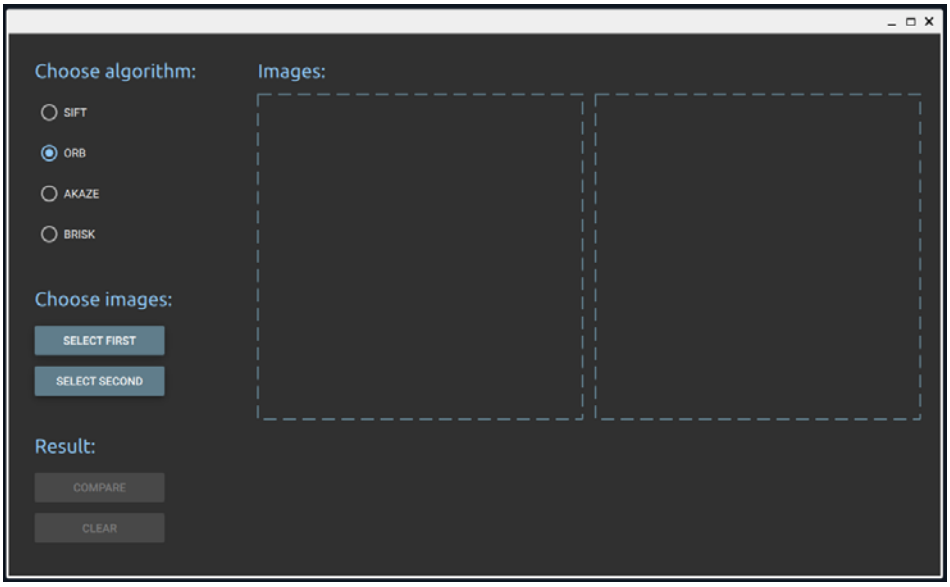

Рисунок 3.9 – Інтерфейс програми на момент запуску

При цьому вибір зображень може відбуватися як через кнопки «Select first/second», так і через функціонал Drag&Drop, який можна використати при переміщені зображень в область їх відображення (рис. 3.10).

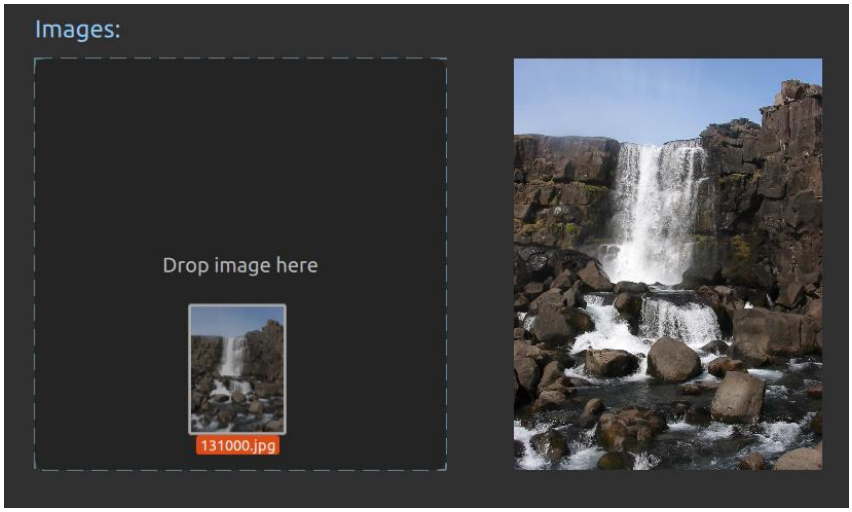

Рисунок 3.10 – Функціонал Drag&Drop

Після вибору алгоритму порівняння та зображень з'являється можливість знайти КТ, створити дескриптори та порівняти їх, щоб таким чином отримати у якості результату значення їх схожості у відсотках, яке базується на кількості зіставлених ознак. Так, наприклад, для двох зображень, які є ND, це значення дорівнює 90.1% (рис. 3.11).

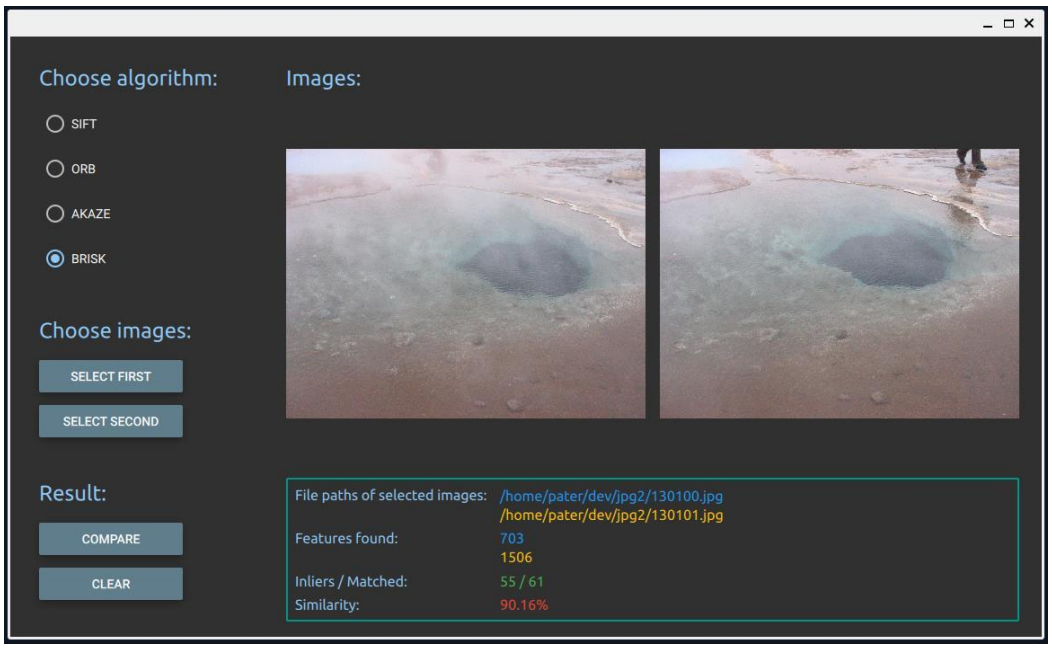

Рисунок 3.11 – Результат роботи програми для двох зображень типу ND

Також ми можемо отримати додаткову інформацію про процес порівняння: шляхи зображень у файловій системі, кількість знайдених КТ на зображеннях, а також кількість КТ, які були здобуті після гомографії і ті, котрі при цьому вдалось зіставити. Саме останні КТ впливають на те, яким буде відсоток схожості між зображеннями, що були показано у формулі 3.1.

Якщо порівняти два однакових зображення, то ми отримаємо 100% збігу. При цьому такий результат ми отримаємо при використанні будь-якого детектора чи дескриптора, а кількість знайдених КТ повинна бути однакова, у чому можна переконатися на рис. 3.12.

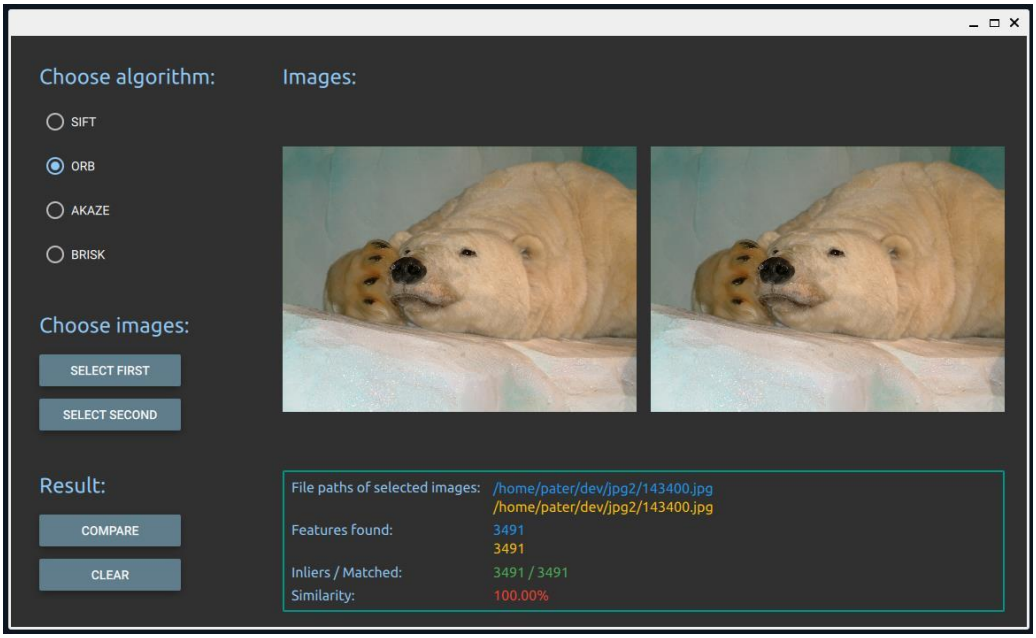

Рисунок 3.12 – Результат роботи програми для двох однакових зображень

При цьому порівняння двох абсолютно різних зображень дасть 0% збігу. Хоча КТ можуть бути знайдені у достатньо великій кількості, будь то сотні чи навіть тисячі, як це показано на рис. 3.13, проведення гомографії дозволяє усунути такі КТ, які не можна співвіднести між двома зображеннями. Тому КТ після трансформації будуть відображати лише ті, котрі дійсно можна спробувати зіставити, а не всі існуючі.

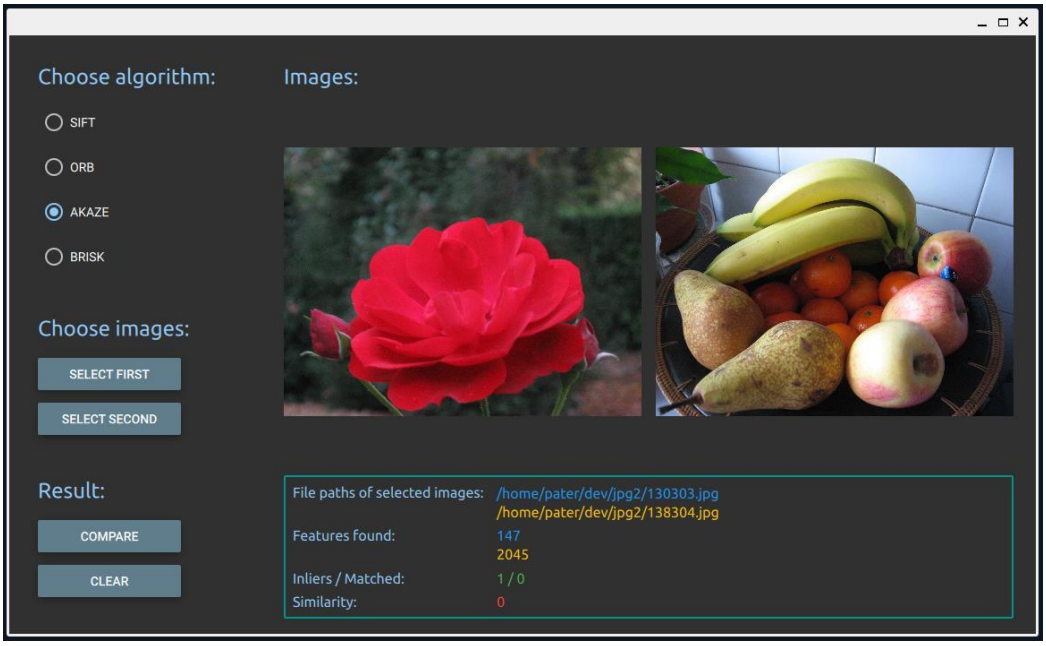

Рисунок 3.13 – Результат роботи програми для двох різних зображень

3.3 Аналіз результатів моделювання

Після реалізації застосунку, який дозволяє виконувати аналіз пари зображень та визначати, чи є вони близькими за змістом, залишається питання ефективності підходу, що використовується. Невідомо, вибір яких дескрипторів із якими налаштуваннями є найбільш вдалим. Кількісний показник точності порівнянь пар зображень також є невідомим. Врешті-решт, навіть остаточне прийняття рішення про те, чи є пара зображень схожими, є суб'єктивним і залежить від людини, яка оцінює відсоток схожості.

Метою цього підрозділу є виконати моделювання для оцінки ефективності деяких популярних дескрипторів та інших методів та проаналізувати можливість автоматично прийняти рішення про те, чи є пари зображень схожими.

Для цього дослідження знову використаємо набір даних INRIA Holidays [30], який містить 1491 зображення, що становить загалом понад 1,1 мільйона пар зображень. 2072 пари в цьому наборі є такими, що можна віднести до класу ND, а інші 1108723 відносяться до класу NND.

Моделювання було виконано в два етапи. Під час першого було використано лише 100 перших зображень набору даних, щоб оцінити попередню якість методу та обчислити границю, яка дозволяє отримати найбільш ефективне рішення. Ці 100 зображень включають 4950 пар порівнянь (105 ND і 4845 NND). Мета цього етапу полягала в тому, щоб виявити, чи можна відрізнити пари схожих зображень від несхожих, використовуючи певний поріг для різних існуючих методів.

Для методів із пошуком відповідних дескрипторів були використані обмеження максимальної відстані Хеммінга між дескрипторами, щоб залишити тільки найбільш якісні відповідності. Кожен метод має також набір власних параметрів, які можуть бути налаштовані.

Таблиця 3.1 містить результати обчислень для різних методів. В таблиці вказано метод та особливості пошуку відповідних дескрипторів, значення для

прийняття рішення, поріг та результати точності класифікації пар зображень. Значення для прийняття рішення має різний сенс для різних методів, наприклад, для методів пошуку відповідних дескрипторів це просто кількість таких дескрипторів, для інших методів можуть бути значення, обчислені за допомогою (1.1) чи (1.2).

Як можна побачити, метод із простим порівнянням гістограм [35] виявився достатньо ефективним, а методи MSE, NMSE, SSIM, LSH показали не дуже хороші результати. Серед методів, заснованих на пошуку відповідних дескрипторів, всі виявилися більш-менш ефективними.

Таблиця 3.1 – Методи знаходження та детектування ознак, порогові значення та ефективність

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
BRISK Максимальна відстань 50, поріг визначення ключової точки 60, співставлення із використанням brute force (BF).	0 – 1557	0 – 16	>=1 >=2	0.81, 0.89 0.73, 0.95
BRISK Максимальна відстань 64, поріг визначення ключової точки 70, співставлення із використанням brute force (BF).	0-971	0-25	>=5	0.73, 0.94

Продовження таблиці 3.1

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
BRISK Максимальна відстань 64, поріг визначення ключової точки 50, співставлення із використанням brute force (BF).	0-4841	0-75	>=8 >=14	0.84, 0.84 0.77, 0.93
BRISK Максимальна відстань 50, поріг визначення ключової точки 50, співставлення із використанням brute force (BF).	0-3415	0-23	>=3	0.78, 0.94
BRISK Максимальна відстань 64, поріг визначення ключової точки 60, співставлення із використанням brute force (BF).	0-2145	0-39	>=4 >=8	0.84, 0.85 0.74, 0.94
BRISK Максимальна відстань 64, поріг визначення ключової точки 60, співставлення із використанням двох найближчих сусідів (KNN) з тестом Lowe 0.75	0-2133	0-65	>=2 >=3	0.82, 0.91 0.79, 0.97

Продовження таблиці 3.1

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
AKAZE Максимальна відстань 50, співставлення із використанням BF.	0 – 11206	0-102	>=4 >=5	0.88, 0.85 0.86, 0.88
AKAZE Максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	0-12993	0-73	>=16 >=23	0.86, 0.85 0.8, 0.92
ORB 500 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням BF.				
ORB 100 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням BF.	0-54	0 – 24	>=10	0.62, 0.81
ORB 200 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням BF.	6-107	0 – 42	>=26	0.63, 0.87

Продовження таблиці 3.1

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
ORB 200 ознак, детектор кутів Harris, максимальна відстань 50, співставлення із використанням BF.	0-93	0 – 23	>=4	0.74, 0.78
ORB 300 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням BF.	12-158	2– 74	>=36 >=42	0.7, 0.7 0.64, 0.84
ORB 100 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	0-54	0-54	>=4	0.4, 0.6
ORB 300 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	0-136	1-37	>=6	0.61, 0.86

Продовження таблиці 3.1

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
ORB 500 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	1-225	0-37	>=4 >=5	0.78, 0.61 0.71, 0.75
ORB 1000 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	2-467	0-44	>=8 >=10	0.76, 0.76 0.68, 0.88
ORB 1000 ознак, детектор кутів Harris, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	2-467	0-44	>=8 >=10	0.76, 0.76 0.68, 0.88
ORB 10000 ознак, детектор кутів FAST, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75.	22-4142	0-126	>=52	0.86, 0.89

Закінчення таблиці 3.1

Метод та його налаштування	Діапазон значень для ND	Діапазон значень для NND	Поріг для визначення ND	Точність для відповідного порога (ND, NND)
MSE Із масштабуванням більшого зображення до розмірів меншого.	549 - 20817	1300 – 30700	>=5400	0.32, 0.33
NMSE Із масштабуванням більшого зображення до розмірів меншого.	0.18 – 1.3	0.25 - 3	>=0.55	0.34, 0.33
SSIM Із масштабуванням більшого зображення до розмірів меншого	0.06 – 0.77	0.06 - 0.7	>=0.4	0.59, 0.76
LSH Хеш різниці (із різницею як мінімум 20 пікселів), розмір хешу 16 фрагментами (band) по 32.	0.43 – 0.96	0.36 – 0.96	>=0.7	0.53, 0.65
Histogram Порівняння нормалізованих гістограм в просторі HSV по 40 значень на канали H та S.	0.06 – 0.99	-0.11 – 0.96	>=0.35 >=0.45	0.85, 0.85 0.81, 0.9

На другому етапі був реалізований пошук схожих пар для всього набору даних. Для прийняття рішення про належність пари зображень до класу ND були використані порогові значення, знайдені в результаті попереднього

етапу.

В таблиці 3.2 наведено результати повномасштабного моделювання процесу пошуку схожих зображень на всьому датасеті для обраних методів. Як можна побачити, результати є значно гіршими порівняно із тими, що наведені в табл. 3.1, тож, знаходження порогового значення на основі пар зображень, побудованих на перших 100 зображеннях датасету, виявилось не дуже ефективним. Якщо використати більше зображень для порівняння на першому етапі, розділяючий поріг можна підібрати краще.

Також видно, що порівняння гістограм є майже в 5 разів швидшим за використання методів пошуку та співставлення ключових точок.

Покращити точність також можливо, застосувавши каскад із декількох методів. В якості першого методу можна обрати порівняння гістограм, якщо рішення по ньому не є впевненим (близьке до порогового значення), можна використати результат класифікації іншим методом (наприклад, ORB).

Результати моделювання показали, що комбінація вказаних методів дозволяє правильно класифікувати 85% пар як для класу ND, так і для NND, що є хорошим результатом.

В якості першого методу був обраний метод на основі порівняння гістограм із порогом 0.35. Таким чином вдалося правильно класифікувати 885064 (79.93%) пар, які не є схожими та 1740 (83.98%) ND. Рішення вважалось впевненим, якщо значення для його прийняття було поза межами 30% відхилення від 0.35, тобто, більшим за 0.455 або меншим за 0.245. Таких значень виявилось 908851 (82.97%). Для пар зображень, значення порівняння гістограм яких виявилось в межах від 0.245 до 0.455 було застосовано порівняння ключових точок із використанням метода ORB (табл. 3.2). Це дозволило виправити результати класифікації 76 пар близьких зображень, при цьому 45 правильно класифікованих таких пар було втрачено. Відповідний показник виправлених пар, складених з різних зображень, склав 70688, в той час як 16343 попередньо правильно класифікованих пар отримали неправильну мітку класу. Після такої корекції маємо близько 85% правильно

класифікованих пар ND (1771 з 2072) та NND (939409 з 1108723).

Таблиця 3.2 – Методи знаходження та детектування ознак, порогові значення та ефективність

Метод та його налаштування	Точність для відповідного порога (ND, NND) на перших 100 зображеннях	Точність для відповідного порога (ND, NND) на всьому датасеті	Середній час на обробку пари зображень, сек.
Histogram Порівняння нормалізованих гістограм в просторі HSV по 40 значень на канали H та S. Поріг для визначення ND >=0.35	0.8476, 0.8452	0.8397, 0.7983	0.35
ORB 10000 ознак, детектор кутів FAST, максимальна відстань 64, співставлення із використанням KNN і тестом Lowe 0.75. Поріг для визначення ND >=52	0.8571, 0.8859	0.6786, 0.8608	1.7
BRISK Максимальна відстань 64, поріг визначення ключової точки 60, співставлення із використанням brute force (BF). Поріг для визначення ND >=4	0.8381, 0.8469	0.6554, 0.8343	1.7

ВИСНОВКИ

У рамках виконання дипломного проекту було розроблено та реалізовано модуль інформаційної системи у вигляді застосунку для проведення порівняння пари зображень з метою отримати відсоток їх збігу, який можна використовувати для визначення, чи є вони схожими за змістом (near-duplicates, ND) чи ні (not near-duplicates, NND).

Були проаналізовані відомі методи для порівняння зображень. Розглянуто та порівняно основні детектори та дескриптори, були виділені їх сильні та слабкі сторони.

Створено програмний застосунок, який дозволяє обрати один з методів знаходження ключових точок, порівнює зображення та надає відсоток схожості двох зображень.

Проведено дослідження ефективності декількох методів для вирішення задачі порівняння пари зображень в пошуках схожих із метою автоматизувати процес прийняття рішення. Оцінено ефективність та швидкодію декількох методів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wang Z., Bovik A. C., Sheikh H. R. Si-moncelli E. P. Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Transactions on Image Processing. 2004. Vol. 13, no. 4. P. 600–612.
2. Loza A., Mihaylova L. Structural Similarity-Based Object Tracking in Video Sequences, In: Proceedings of the 9th International Conf. on Information Fusion, Florence, Italy, 10-13 July, 2006.
3. Pambrun J., Noumeir R. Limitations of the SSIM quality metric in the context of diagnostic imaging, in Proceedings of the International Conference on Image Processing, ICIP 2015. P. 2960–2963. IEEE, Canada.
4. Verma N. K., Kumar N., Goyal A. Object Matching Using Speeded Up Robust Features. Intelligent and Evolutionary Systems. Springer International Publishing, 2016. P. 415–427.
5. Andersson O., Marquez S. R. A comparison of object detection algorithms using unmanipulated testing images: Comparing SIFT, KAZE, AKAZE and ORB, 2016.
6. Parks D., Gravel D. P. Corner Detection, University of McGill, 2004.
7. Harris Corner Detection. OpenCV documentation. URL: https://docs.opencv.org/3.1.0/dc/d0d/tutorial_py_features_harris.html. (date of access: 29.04.2022).
8. Harris C., Stephens M. A. combined corner and edge detector, in Procedings of the Alvey Vision Conference, vol. 15, no. 50. P. 10–5244, 1988.
9. Feature Matching with FLANN. OpenCV documentation. URL: https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html. (date of access: 29.04.2022).
10. Muja M., Lowe D. G. Fast approximate nearest neighbors with automatic algorithm configuration, in Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, vol 1. 2009.
11. Jungong H., Eric P., Paul M. Z. Visible and infrared image registration in man-

made environments employing hybrid visual features, Pattern Recognition Letters, vol. 34, no. 1. P. 42–51. 2013.

12. Rosten E., Drummond T. Fusing points and lines for high performance tracking, in Proceedings of the IEEE International Conference on Computer Vision, vol 2. P. 1508–1511. 2005.
13. Liu C., Xu J., Wang F. A Review of Keypoints' Detection and Feature Description in Image Registration, Scientific Programming, vol 1. P. 1–25. 2021.
14. Canny J. A Computational Approach To Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 8. P. 679–698. 1986.
15. Kaspers A. Blob Detection, Image Science Institute, UMC Utrecht. 2011.
16. Lindeberg T. Feature Detection with Automatic Scale Selection, KTH. 1996.
17. Bromley J., Bentz J. W., Bottou L., Guyon I., LeCun Y., Moore C., S¨ackinger E., Shah R. Signature verification using a siamese time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence, vol. 7. P. 669–688. 1993.
18. Gadetska S. V., Gorokhovatsky V. A., Statistical Measures for Computation of the Image Relevance of Visual Objects in the Structural Image Classification Methods, Telecommunications and Radio Engineering, vol. 77, P. 1041–1053. 2018.
19. Гороховатський В. О., Гадецька С.В. Статистичне оброблення та аналіз даних у структурних методах класифікації зображень (монографія), Харків, ФОП Панов А. Н., 128 с. 2020.
20. Gorokhovatsky V. Structural analysis and intellectual data processing in computer vision. SMIT: Kharkiv, P. 316. 2014.
21. Гороховатский В. А., Передрий Е. О. Корреляционные методы распознавания изображений путем голосования систем фрагментов, Радіоелектроніка. Інформатика. Управління, 1(20), С. 74-81.
22. Daradkeh Y. I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., Al-Dhaifallah M. Methods of Classification of Images on the Basis of the Values of Statistical

Distributions for the Composition of Structural Description Components, IEEE Access, 9, P. 92964-92973. 2021.
23. Гороховатський В. О., Творошенко І. С. Методи інтелектуального аналізу та оброблення даних: навч. посібник. Харків: ХНУРЕ, С. 92. 2021.
24. Lowe D. Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, vol. 60, P. 91–110. 2004.
25. Патер М., Піксельні методи порівняння зображень. – Матеріали Міжнародної науково-практичної конференції молодих учених, аспірантів та студентів “Інформаційні технології в сучасному світі: дослідження молодих вчених”: тези доповідей, 17 – 18 лютого 2022 р. – Х.: ХНЕУ імені Семена Кузнеця, 2022. – С. 90.
26. Rublee E., Rabaud V., Konolige K., Bradski G. ORB: an efficient alternative to SIFT or SURF, in Proceedings of the 2011 International Conference on Computer Vision, P. 2564–2571. 2011.
27. Гороховатский, В. А., Путятин, Е. П. Структурное распознавание изображений на основе моделей голосования признаков характерных точек. Реєстрація, зберігання і обробка даних. 2008.
28. Gorokhovatsky, V. O. and Gadetska, S. V. Determination of Relevance of Visual Object Images by Application of Statistical Analysis of Regarding Fragment Representation of their Descriptions, Telecommunications and Radio Engineering, vol. 78. P. 211–220. 2019.
29. Image Thresholding. OpenCV documentation. URL: https://docs.opencv.org/3.3.0/d7/d4d/tutorial_py_thresholding.html. (date of access: 02.05.2022).
30. Jegou H. INRIA Holidays dataset. Retrieved from http://lear.inrialpes.fr/~jegou/data.php.
31. Leutenegger S., Chli M., Siegwart R. Y. BRISK: binary robust invariant scalable keypoints, in Proceedings of the IEEE International Conference on Computer Vision (ICCV). P. 2548–2555. 2011.
32. Bhagat D. Duplicate Query Detection using Siamese Network. URL:

https://www.linkedin.com/pulse/duplicate-query-detection-using-siamese-network-drishti-bhagat. (date of access: 07.05.2022)

33. Rosten E., Drummond T. Machine learning for high speed corner detection, in Proceedings 9th Euproean Conference on Computer Vision, vol. 1. P. 430–443. 2006.
34. Midpoint circle algorithm. Wikipedia. URL: https://en.wikipedia.org/wiki/Midpoint_circle_algorithm. (date of access: 08.05.2022)
35. OpenCV compare images. OpenCV documentation. URL: https://www.delftstack.com/howto/python/opencv-compare-images. (date of access: 08.05.2022)
36. Testing different image hash functions. URL: https://content-blockchain.org/research/testing-different-image-hash-functions. (date of access: 10.05.2022).