Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ВИВЧЕННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ХЕШУВАННЯ СТРУКТУРНОГО ОПИСУ
(тема)

Виконав:
здобувач 2 року навчання,
групи ІНФм-24-1
Гема О. Г.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Науковий керівник д. т. н., проф.
Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гемі Олександру Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вивчення методів класифікації зображень з використанням хешування структурного опису

затверджена наказом університету від 14 листопада 2025 року № 1045Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 6 грудня 2025 р.
3. Вихідні дані до роботи методи класифікації зображень з використанням хешування структурного опису, літературні джерела щодо застосування методів класифікації, програмні засоби для реалізації обраних методів класифікації, зображення квітів сімейства лілійних

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних методів класифікації зображень з використанням хешування структурного опису.
2. Аналіз літературних джерел щодо апробації методів класифікації зображень.
3. Формування покрокового алгоритму для кожного із вибраних методів класифікації.
4. Візуалізація сформованих покрокових алгоритмів.
5. Розробка програми, що надасть змогу класифікувати зображення квітів сімейства лілійних кожним із вибраних методів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми класифікації зображень з використанням хешування структуррного опису, об'єкт та мета дослідження, постановка задачі, блок-схеми алгоритмів вибраних методів класифікації, приклад еталонних зображень для класу сформованого набору даних, ілюстрація результатів класифікації кожним із вибраних методів, висновки, апробація роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	29.09.2025	
2	Аналіз завдання, підбір літератури	30.09.25-07.10.25	
3	Аналіз літератури з досліджуваної проблеми	08.10.25-14.10.25	
4	Особливості методів класифікації зображень з використанням хешування структурного опису	15.10.25-20.10.25	
5	Дослідження методів класифікації зображень з використанням хешування структурного опису	21.10.25-27.10.25	
6	Програмна реалізація	28.10.25-05.11.25	
7	Обґрунтування отриманих результатів	06.11.25-11.11.25	
8	Оформлення пояснювальної записки	12.11.25-14.11.25	
9	Перевірка на нормоконтроль	26.11.25-27.11.25	
10	Перевірка на плагіат	27.11.25	
11	Рецензування	28.11.25	
12	Підготовка презентації та доповіді	28.11.25-29.12.25	
13	Занесення роботи в електронний архів	15.12.25	
14	Попередній захист кваліфікаційної роботи	15.12.25	

Дата видачі завдання 29 вересня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Гороховатський В. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 84 с., 12 табл., 15 рис., 48 джерел.

ВИПАДКОВЕ ПРОЄКТУВАННЯ, ДЕТЕКТОР AKAZE, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, КЛЮЧОВА ТОЧКА, КОМП'ЮТЕРНИЙ ЗІР, ЛОКАЛЬНО-ЧУТЛИВЕ ХЕШУВАННЯ, СТРУКТУРНИЙ ОПИС, ТОЧНІСТЬ КЛАСИФІКАЦІЇ, ШВИДКОДІЯ

Об'єктом дослідження є методи розпізнавання зображень з використанням ознак у формі множини дескрипторів ключових точок.

Предметом дослідження є впровадження хешування у методи розпізнавання.

Метою дослідження є підвищення ефективності різних методів класифікації зображень з використанням хешування структурного опису

Використано методи Random Projection LSH, LSH на основі кількості одиниць у хеш-коді, метод центрів та лінійний пошук для класифікації зображень.

Наукова новизна полягає в тому, що здобула подальший розвиток адаптація методів локально-чутливого хешування для задачі класифікації зображень за структурними описом у формі множини дескрипторів.

Взаємозв'язок з іншими роботами полягає у продовженні напрямку досліджень, присвячених оптимізації методів пошуку подібних об'єктів у багатомірних просторах.

Рекомендації щодо використання результатів роботи полягають у можливості застосування розроблених методів у системах комп'ютерного зору.

У результаті дослідження реалізовано метод класифікації зображень з використанням хешування структурного опису, що продемонстрував високу швидкодію та точність класифікації.

ABSTRACT

Explanatory note to the qualification work: 84 pages, 12 table, 15 figures, 48 sources.

AKAZE DETECTOR, CLASSIFICATION ACCURACY, COMPUTER VISION, IMAGE CLASSIFICATION, KEYPOINT, LOCALITY-SENSITIVE HASHING (LSH), PERFORMANCE, RANDOM PROJECTION, STRUCTURAL DESCRIPTION

The object of the research is image recognition methods that use features represented as a set of keypoint descriptors.

The subject of the research is the integration of hashing into recognition methods.

The aim of the research is to improve the efficiency of various image classification methods using structural descriptor hashing.

The Random Projection LSH, LSH based on the number of ones in the hash code, the center-based method, and linear search were applied for image classification.

The scientific novelty lies in the further development of the adaptation of locality-sensitive hashing methods for the task of image classification based on structural descriptions represented as sets of descriptors.

The relationship with other studies consists in continuing the research direction focused on optimizing methods for searching similar objects in multidimensional spaces.

Recommendations for the application of the results include the possibility of implementing the developed methods in computer vision systems.
As a result of the research, a classification method based on structural descriptor hashing was implemented, demonstrating high processing speed and classification accuracy.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВП – випадкове проєктування

КТ – ключова точка

КЗ – комп’ютерний зір

ШІ – штучний інтелект

AKAZE – Accelerated KAZE

BFS – Brute-force Search (пошук методом грубої сили)

LDB – Local Difference Binary (двійковий код локальної різниці)

LSH – Local Sensitivity Hashing (локально-чутливе хешування)

M-LDB – Modified Local Difference Binary (модифікований локальний різницевий двійковий код)

ВСТУП

Комп’ютерний зір є однією з найдинамічніших і найважливіших галузей сучасних інформаційних технологій, що активно розвивається на перетині штучного інтелекту, машинного навчання та обробки сигналів. Його основним завданням є моделювання здатності людського зору за допомогою обчислювальних засобів, тобто надання комп’ютеру можливості «бачити» навколишній світ, аналізувати його та приймати рішення на основі візуальної інформації. У найзагальнішому вигляді системи комп’ютерного зору здійснюють перетворення двовимірного зображення у структуроване подання даних про об’єкти, їх взаємне розташування, властивості та семантичний зміст .

Завдання комп’ютерного зору охоплюють широкий спектр напрямів – від базової обробки зображень (наприклад, фільтрації шумів, покращення контрасту або нормалізації освітлення) до складних операцій з аналізу сцен, виявлення об’єктів, відстеження їх у русі та розпізнавання. Одним із ключових напрямів у цій сфері є класифікація зображень – процес віднесення візуальних даних до певних категорій на основі характерних ознак об’єктів. Такі ознаки можуть бути як низькорівневими (колір, текстура, форма), так і високорівневими (структурні або семантичні дескриптори) [1].

Практичне значення комп’ютерного зору постійно зростає, адже він знаходить застосування у великій кількості галузей. Серед найпоширеніших прикладів можна виділити розпізнавання облич у системах безпеки, автоматичну класифікацію медичних знімків для діагностики захворювань, сортування продукції в промисловості, а також аналіз дорожніх ситуацій у системах автономного керування транспортом. Крім того, технології комп’ютерного зору активно використовуються у побудові пошукових систем за зразком зображення, у програмах доповненої та віртуальної реальності, у сільському господарстві для моніторингу стану рослин, у військовій техніці, а також у наукових дослідженнях з біології та екології.

Актуальність роботи полягає у зростанні обсягів цифрових зображень та необхідності швидкої та точної їх класифікації без застосування ресурсомістких методів повного перебору. Класичні методи аналізу зображень часто потребують великих обчислювальних ресурсів, що обмежує їх застосування у реальному часі. Використання структурних дескрипторів та методів локально-чутливого хешування (LSH) дозволяє значно прискорити процес пошуку подібних зображень та підвищити продуктивність систем автоматичної класифікації. Крім того, адаптація цих методів до задачі класифікації конкретних об’єктів, таких як зображення квітів родини лілійних, дозволяє оптимізувати процес навчання моделей та підвищити їхню точність.

Огляд сучасного стану проблеми показує, що існують різні підходи до класифікації об’єктів на основі дескрипторів. Серед них – Random Projection LSH, який застосовує випадкові проєкції у бінарний простір; LSH на основі кількості одиниць у хеш-коді, що спрощує обчислення подібності між ознаками; метод центрів, який формує хеші на основі відстані до попередньо обчислених центрів; та лінійний пошук, що використовується як еталон для порівняння результатів. Кожен з цих методів має свої переваги та обмеження щодо швидкодії та точності класифікації, що потребує їх комплексного порівняння та адаптації до конкретної задачі.

Наукова задача, що вирішується у роботі, полягає у порівнянні ефективності різних методів хешування структурного опису для задачі класифікації зображень квітів родини лілійних та розробці програмного застосунку, що демонструє практичну реалізацію цих методів. Дослідження спрямоване на виявлення оптимальних стратегій побудови хеш-ключів та підбору параметрів алгоритмів для досягнення високої точності при мінімальних обчислювальних витратах.

У межах роботи проведено аналіз сучасних методів класифікації об’єктів на зображеннях та відповідних літературних джерел, сформовано блок-схеми алгоритмів реалізації хешування структурних дескрипторів, а також виконано експериментальне порівняння методів за показниками швидкодії та точності.

1 КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ ЗА МНОЖИНОЮ ДЕСКРИПТОРІВ

1.1 Постановка задачі класифікації зображень

Задача класифікації зображень належить до фундаментальних напрямів комп'ютерного зору та штучного інтелекту. Її сутність полягає у визначенні належності зображення до одного з наперед визначених класів на основі його візуальних характеристик. Іншими словами, класифікаційна система має здатність автоматично аналізувати вхідне зображення та приймати рішення, яке з можливих понять – наприклад, «автомобіль», «будівля», «людина» або «тварина» – воно представляє. Така задача є базовою для багатьох прикладних систем: розпізнавання об'єктів, медичної діагностики, відеоспостереження, пошуку зображень за змістом, автономного водіння тощо [1].

У загальному вигляді процес класифікації можна подати як відображення:

$$f: X \rightarrow Y, \tag{1.1}$$

де X – простір усіх можливих зображень;

Y – множина можливих класів.

Мета полягає у побудові функції f, яка для будь-якого вхідного зображення $x \in X$ визначає правильну мітку класу $y \in Y$. Основна складність полягає у тому, що простір X має надзвичайно високу розмірність (кількість пікселів, колірні канали, варіації освітлення, положення тощо), а відмінності між класами часто не є лінійно роздільними [1, 2].

Типовий процес класифікації зображень складається з декількох послідовних етапів.

Спочатку відбувається попередня обробка зображення. Зображення нормалізується за розміром, контрастом, яскравістю або кольоровою гамою. Цей етап дозволяє зменшити вплив зовнішніх чинників – освітлення, масштабу або шуму – і зробити ознаки, що будуть виділені пізніше, більш стабільними.

Далі відбувається виділення ознак на зображенні. На цьому етапі відбувається перетворення вихідного зображення у компактне числове представлення, яке описує найважливіші структурні, геометричні та текстурні властивості об’єкта. У класичних методах це здійснюється за допомогою детекторів та дескрипторів локальних ознак – SIFT, SURF, BRISK, ORB, AKAZE тощо [1, 2].

Наступним етапом йде побудова моделі класифікації. Ознаки, отримані з навчальних прикладів, подаються на вхід класифікатора, який навчається відокремлювати приклади різних класів. Сучасні підходи до побудови моделі класифікації зображень можна умовно розділити на три групи з різними сильними та слабкими сторонами. Перший – глибоке навчання з кінцевим навчанням, яке автоматично витягує ознаки і навчає класифікатор одночасно; цей підхід забезпечує високі показники для великомасштабних наборів даних, але потребує значних обчислювальних ресурсів і великих наборів розмічених даних [3]. Другий – класичні статистичні методи і методи шаблонного порівняння, вони більш інтерпретовані і часто краще працюють на невеликих наборах або при обмеженій розмітці [2]. Третій – методи на основі локальних дескрипторів і їх агрегації або індексації, що передбачають порівняння з еталонними зразками, де для оцінки подібності використовується значення відстані або міри наближення між порівнюваними об’єктами [1].

Завершальним етапом класифікації є прийняття рішення. Для нового, невідомого зображення витягуються ознаки, після чого модель визначає клас, що має найбільшу ймовірність або найменшу відстань у просторі ознак.

Незалежно від конкретної моделі, основна мета класифікації полягає в тому, щоб знайти таке подання зображень, у якому відстань між елементами одного класу є малою, а між різними класами – великою. Від вибору цього подання залежить якість усієї системи. Якщо ознаки неадекватно описують структуру об’єктів, навіть найпотужніший класифікатор не забезпечить високої точності [3].

Для об’єкта зображення, описаного дескрипторами ключових точок, визначимо математичну постановку задачі розпізнавання об’єкту.

Нехай Ω – простір образів (зображень); $\omega \in \Omega$ – конкретний образ (візуальний об’єкт, зображення); $M[J] = \{1,2,...J\}$ – множина номерів для класів образів $\Omega_1,\Omega_2,...,\Omega_J$ таких, що $\Omega = \cup_{i=1}^{J} \Omega_i$, $\Omega_i \cap \Omega_k = \emptyset$, $\emptyset$ – пуста множина; $g: \Omega \to M$ – невідома індикаторна функція, яка ставить у відповідність кожному образу $\omega \in \Omega$ деякий номер $j \in M$ його класу.

Розглянемо $Z = z \mid z \in R^n$ – простір описів образів, де опис – це скінченна множина числових векторів (дескрипторів КТ). Задамо відображення $\Theta: \Omega \to Z$ із простору Ω у простір Z. Кожному $\omega \in \Omega$ відображення Θ ставить у відповідність його опис $z(\omega) = \{z_i\}_{i=1}^{s}$. Опис $z(\omega)$ у просторі Z – це скінченна множина: $z(\omega) = \{z_i\}_{i=1}^{s}$, $s = card\ z(\omega)$ – потужність $z(\omega)$; $Z_1, Z_2, \dots, Z_j$ – підмножини у просторі Z, такі, що $Z_i \cap Z_k = \emptyset$ і $Z = \cup_{i=1}^{J} Z_i$. Передбачається, що ці підмножини отримані дією відображення $\Theta: \Omega_k \to Z_k$.

Тепер розглянемо $\hat{g}: Z \to M$ – вирішальне правило, яке ставить у відповідність множині ознак образу $z(\omega)$ номер класу образу. Тоді процес класифікації можна подати як ланцюг перетворень даних із простору образів у множину класів, що має вид

$$\Omega \to \Theta(\Omega) \to Z \to \hat{g}(Z) \to M. \qquad (1.2)$$

Головною задачею процесу (1.1) є побудова вирішального правила $\hat{g}$, що базується на поданні у просторі Z, отриманому застосуванням Θ. Вибір Z та $z(\omega)$ прямо впливає на ефективність правила $\hat{g}$. При цьому критерієм якості класифікації виступає правильність прийняття рішення щодо елементів вихідного простору образів Ω.

У задачі розпізнавання, де описи представлені сукупностями КТ зображення, множини описів $z(\omega)$ формуються детекторами для побудови КТ.

У спрощеному випадку множина описів може бути подана єдиним вектором, тоді трактується як простір векторів.

На попередньому етапі здійснюємо формування бази еталонних образів і побудову для кожного з них структурних описів $Z_1, Z_2, \ldots, Z_j$.

Базовий метод класифікації полягає в наступному:

– побудова опису $O \in Z$ для зображення, що підлягає розпізнаванню;

– обчислення значень релевантності $d_j(Z_j, O)$ для опису O на множині $Z_1, Z_2, \ldots, Z_j$ описів еталонів;

– оптимізація на множині значень $d_j(Z_j, O)$, $j = 1,..., J$, і визначення номеру v класу еталона $v = arg \underset{M}{\text{opt}}\, d_j(Z_j, O)$ з найкращим значенням релевантності (максимум для подібності або мінімум для відстані) [4].

1.2 Структурний опис зображення

Методи структурного опису зображень відіграють важливу роль у задачах комп'ютерного зору та розпізнавання образів. Саме завдяки ним відбувається перетворення зображення від суто піксельного представлення до інваріантного та стійкого до змін зовнішніх умов способу подання візуальної інформації. Їх сутність полягає у виділенні та подальшому описі локальних елементів зображення, які зберігають характерні властивості об'єкта незалежно від положення, масштабу, освітлення або перспективи. Виходячи з цього, структурний опис можна розглядати як модель зображення, побудовану на основі множини ключових точок, кожна з яких має власний дескриптор, що відображає локальні ознаки у її околі [1, 7].

Процес побудови структурного опису починається з етапу виявлення ключових точок, або локальних інваріантних ознак, які є носіями найбільш інформативної частини зображення.

Ключові точки (keypoints або interest points) – це фундаментальний елемент структурного опису зображення. Вони визначають ті позиції на зображенні, які

є найбільш інформативними, стійкими та відтворюваними при різних перетвореннях. У загальному вигляді ключова точка – це піксель або невелика область, де спостерігаються різкі локальні зміни сигналу, такі як контрасти, кути, перетини ліній або характерні текстурні елементи [4, 5]. Математично детектори КТ шукають локальні екстремуми певної функції, що відображає «цікавість» пікселя – це може бути детермінант матриці Гаусса, модуль градієнта або різниця розмитих копій зображення.

Рисунок 1.1 –Зображення та його множина координат КТ

Ключові точки є ефективним інструментом побудови структурного опису завдяки своїм властивостям [5]:

– інваріантність (визначення не залежить від афінних перетворень);

– інтерпретація (представлена так, щоб можна було виокремити інформацію про неї);

– унікальність (відрізняється від усіх інших точок);

– стабільність (стійкість до геометричних видозмін зображення та шумів);

– локальність (розглядається невелика область, окіл);

– точність (порівняння відповідних точок двох зображень);

– чисельність (достатня кількість КТ для прийняття результативного

рішення);

– ефективність (робота у режимі он-лайн або наближений до нього);

– відтворюваність (КТ повинні стабільно виявлятися у тих самих або дуже близьких місцях об’єкта при повторних знімках).

Важливішою властивістю КТ є інваріантність. Це означає, що при будь-якому геометричному або фотометричному перетворенні об’єкта відповідні точки на різних зображеннях повинні залишатися розпізнаваними як ті самі. Для цього у процесі виявлення та опису кожної точки застосовується нормалізація її масштабу, орієнтації та контрасту.

Після визначення координат ключових точок починається формування дескрипторів – числових або бінарних векторів, які описують локальні властивості зображення в околі кожної точки. Спочатку для кожної ключової точки визначається локальна область – квадрат або коло, розмір якого може залежати від масштабу точки. Ця область виділяє фрагмент зображення, що відображає характерні структурні особливості об’єкта.

Далі у межах цієї області аналізуються локальні зміни інтенсивності або градієнти яскравості, що дозволяє оцінити текстуру, контури та орієнтацію деталей. Зібрані дані стискаються у компактний вектор, де кожен елемент кодує певну характеристику локальної структури. Основна мета такого кодування – забезпечити інваріантність дескриптора: він повинен залишатися стабільним при обертанні, масштабуванні, зміні освітлення або частковому перекритті об’єкта, і водночас бути достатньо дискримінативним, щоб розрізняти різні об’єкти або класи.

На завершальному етапі ключова точка отримує власний дескриптор – компактне та інформативне представлення локальної структури зображення. Ці дескриптори можна ефективно порівнювати між різними зображеннями, що дозволяє визначати відповідності, класифікувати об’єкти та будувати структурний опис зображення на основі набору дескрипторів [5, 6].

Після побудови дескрипторів для всіх ключових точок отримується множина локальних векторів, яка фактично є структурним представленням

зображення. Ця множина може мати значну розмірність, особливо коли об’єкт містить багато дрібних деталей, тому постає проблема ефективного зберігання і порівняння таких описів. Структурний опис, на відміну від глобального, не є єдиним вектором фіксованої довжини – це множина точок у просторі ознак, що описують різні частини одного зображення. Таким чином, порівняння двох структурних описів вимагає оцінки подібності між множинами векторів, а не між одиничними представленнями. У найпростішому випадку використовується пошук найближчих сусідів для кожного дескриптора, після чого визначається кількість або якість співпадінь між ними. Міра подібності може визначатися за евклідовою відстанню, мангеттенською метрикою або, у випадку бінарних дескрипторів, за відстанню Хемінга.

Формалізовано структурний опис можна розглядати як відображення, яке кожному зображенню ставить у відповідність скінченну множину дескрипторів, тобто векторних ознак, що утворюють своєрідний «відбиток» зображення. Це відображення має зберігати топологічні властивості об’єкта: зображення одного і того самого класу мають формувати близькі множини у просторі дескрипторів, тоді як представлення різних класів повинні бути віддаленими. На цьому принципі ґрунтується подальший етап класифікації, який полягає у визначенні подібності між описами шляхом обчислення відстаней або міри близькості. У простих випадках використовується порівняння з еталоном, тобто оцінюється відстань між описом вхідного зображення та структурним описом відомого об’єкта. Якщо відстань менша за заданий поріг, зображення вважається належним до цього класу [7].

1.3 Методи визначення ключових точок і їх дескрипторів

Сьогодні існує широкий спектр методів визначення ключових точок і побудови дескрипторів, кожен із яких має свої переваги та особливості. Основна відмінність між ними полягає у способі числового представлення дескриптора та алгоритмі його формування. У загальному випадку дескриптори поділяють на

дві великі групи: з плаваючою комою (наприклад, SIFT та SURF) та бінарні (BRIEF, BRISK, AKAZE, ORB). Дескриптори першої групи характеризуються високою точністю опису локальних ознак, однак потребують значних обчислювальних ресурсів. Бінарні ж дескриптори мають значно менший обсяг даних і забезпечують швидке порівняння шляхом використання двійкових операцій, що робить їх особливо зручними для апаратної реалізації та роботи в режимі реального часу.

Крім того, алгоритми опису можна класифікувати за напрямом пошуку: виділяють дескриптори кутів (наприклад, ORB, BRISK) і дескриптори областей (SIFT, SURF, KAZE, AKAZE). У першому випадку аналізуються різкі зміни інтенсивності, що дозволяє виявляти характерні кути об’єктів, у другому – локальні області з вираженими структурними відмінностями.

Одним із перших ефективних підходів до визначення ключових точок став метод FAST, у якому навколо кожного пікселя формується невелике коло, і точка вважається ключовою, якщо яскравість певної кількості її сусідів істотно більша або менша за власну. Цей алгоритм забезпечує високу швидкодію і став основою для ряду подальших методів [7, 8].

Метод BRIEF став наступним кроком у розвитку, оскільки запропонував простий, але ефективний спосіб опису ключових точок у бінарному вигляді. Дескриптори формуються шляхом порівняння яскравостей пар точок у межах локального околу. Якщо яскравість однієї точки більша за іншу, формується біт зі значенням 1, інакше – 0. У результаті отримується компактний бінарний вектор, який можна швидко порівнювати за допомогою відстані Хемінга. BRIEF вирізняється високою швидкістю та незалежністю від змін освітлення, проте залишається чутливим до поворотів [9].

Серед класичних методів найвідомішими є SIFT і SURF. Метод SIFT базується на пошуку локальних екстремумів різниці Гауссіанів у масштабному просторі. Ключові точки відбираються за контрастом і позицією на зображенні, а для кожної з них формується дескриптор на основі орієнтованих градієнтів.

SIFT забезпечує стійкість до поворотів, масштабування, шуму й змін освітлення, проте має високу обчислювальну складність [8].

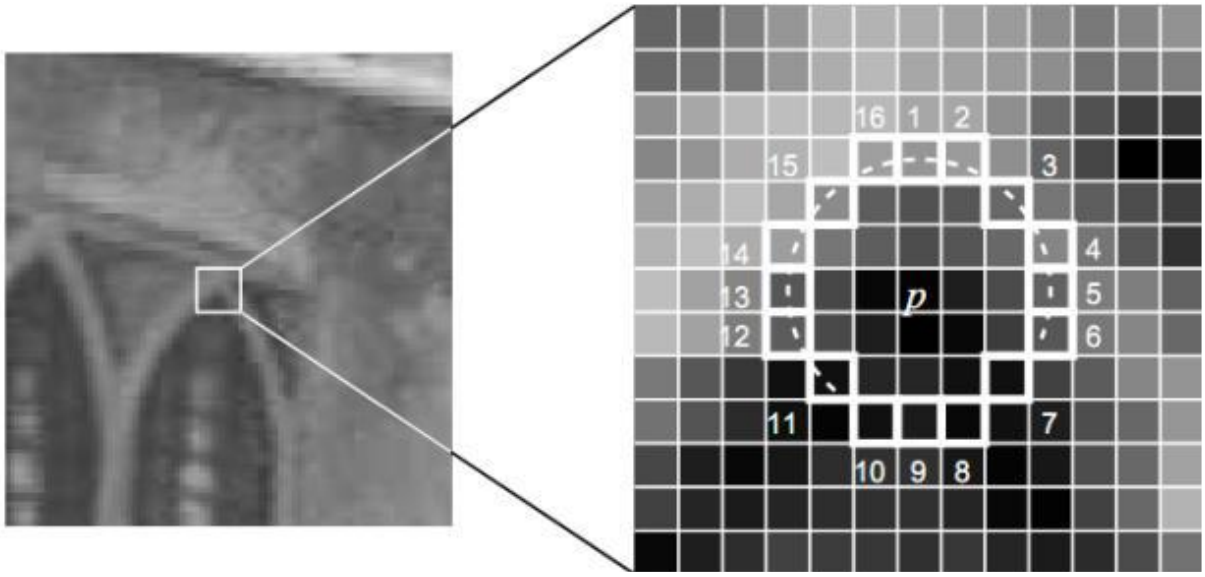

Рисунок 1.2. Знаходження КТ методом FAST

Метод SURF, створений як швидша альтернатива SIFT, використовує апроксимацію лапласіана Гаусіана через матрицю Гессе та побудову дескриптора на основі вейвлет-перетворень Хаара. Завдяки цьому SURF працює швидше, зберігаючи інваріантність до повороту та масштабу. Водночас через складність обчислень і великий обсяг даних метод не завжди придатний для систем із обмеженими ресурсами [6].

Прагнення поєднати точність традиційних методів із ефективністю бінарних призвело до появи сучасних алгоритмів ORB і BRISK. Метод ORB поєднує швидкий детектор FAST з модифікованим бінарним описом BRIEF, який орієнтується відповідно до кута повороту зображення. Це дозволяє зберігати інваріантність до обертання при мінімальних обчислювальних витратах. Метод ORB формує дескриптори розміром 256 біт, які зручно зберігати та порівнювати у двійковому вигляді [11].

Метод BRISK, у свою чергу, використовує пірамідальне представлення зображення для виявлення ключових точок у різних масштабах. Опис формується на основі концентричних кіл, де кожне кільце представляє

локальний контекст навколо точки. Дескриптор BRISK має розмір 512 біт і характеризується високою точністю, стабільністю та незалежністю від масштабу [10].

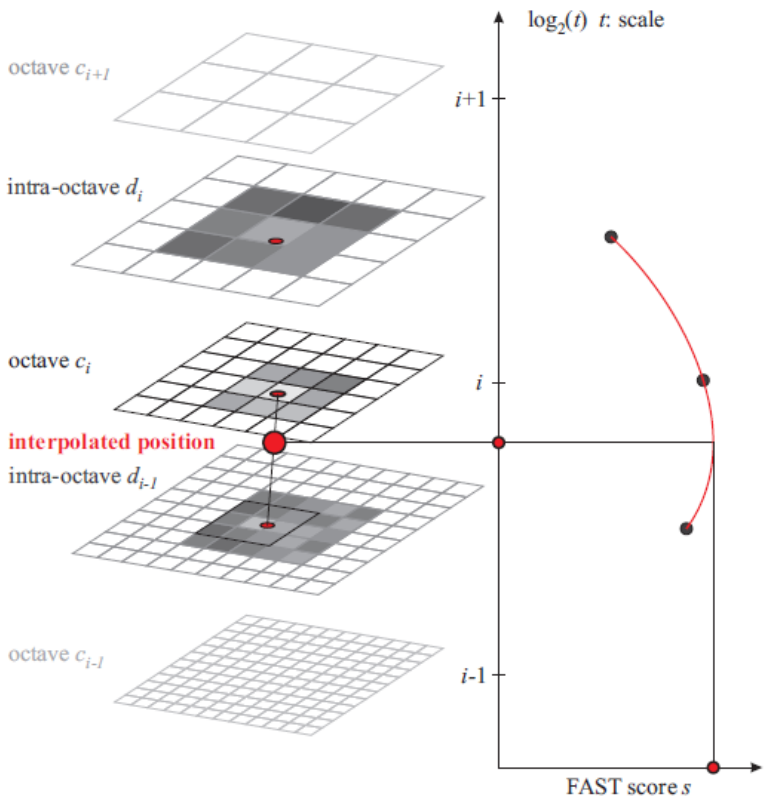

Рисунок 1.3. Знаходження КТ методом BRISK

Порівняльний аналіз показує, що алгоритми ORB і BRISK мають значно менші вимоги до обчислювальних ресурсів, ніж методи SIFT і SURF, забезпечуючи при цьому співставну або навіть вищу точність на окремих типах зображень. Їх бінарна природа дозволяє реалізовувати порівняння за допомогою простих логічних операцій, що суттєво підвищує швидкість і робить можливим використання у мобільних системах і в режимі реального часу.

Додатковою перевагою є модульність побудови: детектори ORB і BRISK дозволяють комбінувати різні підходи для пошуку ключових точок та формування дескрипторів, оптимізуючи роботу алгоритму під конкретне завдання. У BRISK також можливо керувати кількістю визначених ключових точок, змінюючи параметри масштабу, пороги яскравості або розміри околу [4].

Метод AKAZE, створений для покращення швидкодії та точності, ґрунтується на нелінійній дифузії, яка формує багатомасштабне представлення зображення. Він інваріантний до поворотів і змін масштабу, забезпечує якісні результати навіть на складних структурах і текстурах [14].

Загалом, еволюція методів побудови дескрипторів від SIFT і SURF до ORB, BRISK та AKAZE демонструє перехід від точності та складності до більшої швидкодії та гнучкості. Завдяки цим змінам сучасні підходи дозволяють забезпечити баланс між інваріантністю, дискримінативністю та ефективністю, що є критично важливим для практичних задач комп’ютерного зору.

1.4 Дескриптор AKAZE

Алгоритм AKAZE (Accelerated KAZE) є одним із сучасних і збалансованих методів побудови локальних дескрипторів, який поєднує точність класичних алгоритмів на кшталт SIFT із високою швидкодією бінарних методів, таких як ORB. Він був запропонований у 2013 році Пабло Фернандесом Алонсо та співавторами як покращена версія KAZE – алгоритму, що вперше ввів поняття нелінійного масштабного простору для аналізу зображень. Основна ідея KAZE полягала у побудові багаторівневого представлення зображення, де ступінь згладжування залежить від локальної структури: гладкі ділянки розмиваються сильніше, а області з різкими градієнтами (краї, кути, текстурні переходи) залишаються чіткими. Такий підхід дозволив зберегти дрібні деталі й підвищити стійкість алгоритму до шумів і зміни освітлення. Проте обчислення KAZE виявилися надто повільними для практичного застосування. Саме тому була розроблена прискорена версія – AKAZE, у якій використовується ефективніша дискретна апроксимація нелінійної дифузії, що дозволило суттєво зменшити час обробки без втрати якості виявлення ознак [13].

У процесі роботи AKAZE спочатку будує нелінійний масштабний простір на основі дифузійної фільтрації, після чого на кожному масштабі визначаються

контрольні точки – локальні екстремуми детермінанта матриці Гессе. Такі точки характеризують області з високою варіацією інтенсивності, тобто місця, де зображення містить найбільше структурної інформації. Завдяки цьому ключові точки AKAZE мають високу стабільність при зміні масштабу, освітлення або кута огляду. Після детектування точок переходять до етапу побудови дескрипторів, тобто числових векторів, які компактно описують локальний вигляд зображення в околі кожної ключової точки.

Однією з головних інновацій AKAZE є використання дескриптора M-LDB (Modified Local Difference Binary), який є модифікованою версією LDB (Local Difference Binary). Його принцип роботи полягає у побудові компактного бінарного опису структури зображення в локальному вікні навколо ключової точки. Для цього вибрана область поділяється на невеликі підблоки, і для кожного з них обчислюються прості статистичні характеристики – середня яскравість, горизонтальні та вертикальні градієнти. Потім ці характеристики попарно порівнюються: якщо, наприклад, середня яскравість одного блоку більша за інший, у відповідному бітовому векторі записується «1», інакше – «0». Таким чином формується бінарна послідовність, яка відображає просторові співвідношення інтенсивностей у межах локального околу точки.

M-LDB удосконалює цей підхід, додаючи декілька важливих елементів. Він використовує попереднє нормування напрямку локального градієнта, що забезпечує інваріантність до поворотів зображення. M-LDB комбінує інформацію про інтенсивність і градієнти, створюючи більш насичене представлення локальної структури. А за рахунок оптимізованої схеми вибору пар блоків для порівняння вдається мінімізувати надлишковість у векторі ознак і зменшити його розмір, що прискорює обчислення. Завдяки бінарній природі M-LDB дескриптори можна ефективно порівнювати за допомогою простої метрики – відстані Хемінга, яка обчислюється надзвичайно швидко навіть для великих наборів даних [14].

Порівняно з класичними методами, AKAZE забезпечує відмінне співвідношення між швидкістю та якістю. SIFT і SURF формують дескриптори

у вигляді дійсних чисел і мають велику довжину вектору, що дає високу точність, але потребує значних ресурсів пам’яті та часу для пошуку збігів. ORB і BRISK, навпаки, використовують бінарні вектори, забезпечуючи високу швидкість, але поступаючись у точності на складних текстурах або за сильних змін освітлення. AKAZE з дескриптором M-LDB займає проміжну позицію: він зберігає точність, близьку до SIFT, але працює зі швидкістю, порівнянною з ORB. Завдяки цьому він став одним із найпопулярніших методів для задач, де важливо знайти компроміс між надійністю, інваріантністю та продуктивністю.

1.5 Постановка задачі дослідження

Вивчення методів класифікації зображень з використанням хешування структурного опису є актуальним завданням у галузі комп’ютерного зору. Для вирішення цього завдання прийнято рішення щодо розроблення програмного застосунку, який реалізує класифікацію зображень із використанням методів локально-чутливого хешування (LSH) та порівняння їх ефективності за показниками швидкодії та точності.

Об’єктом дослідження є структурні методи класифікації зображень на основі множини дескрипторів ключових точок.

Предметом дослідження є методи локально-чутливого хешування, що застосовуються у методах класифікації зображень за їх структурними ознаками.

Метою дослідження є впровадження і вивчення ефективності методів хешування для задачі класифікації на прикладі набору зображень квітів родини лілійних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз наукових джерел щодо методів класифікації зображень на основі локальних дескрипторів;

– дослідити принципи роботи та властивості методів локально-чутливого хешування;

– реалізувати алгоритми класифікації зображень із використанням методів Random Projection, хеш-коду за кількістю одиниць, методу центрів та лінійного пошуку;

– здійснити порівняльний аналіз характеристик методів за критеріями швидкодії та точності класифікації;

– візуалізувати алгоритми класифікації у вигляді блок-схем;

– сформулювати рекомендації щодо вибору методу хешування структурного опису для практичного застосування.

Реалізація зазначених завдань дозволить оцінити можливості використання методів локально-чутливого хешування для задач класифікації зображень у системах комп’ютерного зору та сформувати висновки щодо доцільності їх застосування у практичних розробках.

2 КЛАСИФІКАЦІЯ З ВИКОРИСТАННЯМ ХЕШУВАННЯ

2.1 Лінійний пошук

Лінійний пошук найближчих сусідів (BFS) є фундаментальним підходом до класифікації об’єктів у багатовимірному просторі ознак. Його основна ідея полягає у прямому порівнянні запитного дескриптора із усіма дескрипторами бази даних з метою визначення найбільш схожих об’єктів. Такий метод не потребує попередньої підготовки даних або спеціалізованих структур даних, що робить його універсальним і зрозумілим для реалізації. Процес класифікації за допомогою лінійного пошуку включає кілька етапів. Спочатку кожне зображення, яке необхідно класифікувати, перетворюється на набір дескрипторів, наприклад, AKAZE, ORB або SIFT. Далі для кожного запитного дескриптора обчислюється відстань до кожного дескриптора бази даних, при цьому використовуються різні метрики схожості: Hamming distance для бінарних дескрипторів, яка визначає кількість різних бітів між векторами, або Euclidean distance для дескрипторів з плаваючою точкою. На основі обчислених відстаней визначаються дескриптори з мінімальною відстанню до запиту. У найпростішому варіанті вибирається один найближчий сусід, проте часто застосовується k-найближчих сусідів (k-NN), де рішення приймається за голосуванням міток цих сусідів. Кінцеве рішення про клас запитного зображення формується на основі міток найближчих сусідів, що може реалізовуватись через більшість голосів або вагові оцінки відстаней [15].

Лінійний пошук має низку переваг. По-перше, він забезпечує високу точність, оскільки всі дескриптори бази беруть участь у порівнянні, що гарантує знаходження справжніх найближчих сусідів. По-друге, метод простий у реалізації та не потребує складних структур даних чи додаткових процедур підготовки. По-третє, він універсальний і може працювати з будь-якими типами дескрипторів та метриками відстані. Однак прямий пошук виявляється неефективним при великих об’ємах даних, оскільки для бази з N дескрипторами

обчислювальна складність лінійного пошуку складає O(N) для кожного запитного дескриптора. У великих системах класифікації з десятками тисяч або сотнями тисяч дескрипторів це призводить до значного часу обробки і високих витрат обчислювальних ресурсів. Крім того, метод погано масштабується у багатовимірних просторах, де спостерігається феномен «прокляття вимірності», коли відстані між усіма парами точок стають подібними, а ефективність порівняння падає. Незважаючи на ці обмеження, лінійний пошук використовується як еталонний метод для оцінки точності та ефективності інших алгоритмів пошуку [16].

2.2 Метод класифікації на основі бінарних центрів

Метод пошуку за центрами передбачає побудову для кожного класу єдиного бінарного вектору, який називається глобальним центром. Глобальний центр формується шляхом агрегування всіх дескрипторів, що належать даному класу. Для цього кожен дескриптор переводиться у бінарну форму, після чого обчислюється усереднене значення кожної координати. Позначимо довжину дескриптора як D, а кількість дескрипторів класу як N. Нехай кожен дескриптор $x_i \in \{0,1\}^D$, де $i = 1 \dots N$, і його координати x_i, j для $j = 1 \dots D$.

Усереднення координати j обчислюється за формулою:

$$\mu_j = \frac{1}{N} \cdot \sum_{i=1}^{N} x_{i,j} \ . \tag{2.1}$$

Після цього біт центру c_j формується за правилом:

$$\begin{cases} c_j = 1, \text{якщо } \mu_j > 0.5 \\ c_j = 0, \text{якщо } \mu_j \leq 0.5 \end{cases}. \tag{2.2}$$

Таким чином, глобальний центр класу можна записати у вигляді вектору:

$$c = (c_1, c_2, \dots, c_D) \in \{0,1\}^D. \tag{2.3}$$

Класифікація нового зображення відбувається аналогічним чином. Зображення пропускається через AKAZE, бінарні дескриптори агрегуються у вектор запиту за правилом «>50% → 1» [17].

Далі для кожного класу обчислюється Hamming-відстань між вектором запиту q та глобальним центром c класу. Hamming-відстань визначає кількість різних бітів у двох векторах і обчислюється за формулою:

$$H(q,c) = \Sigma_{(j=1\ldots D)} |q_j - c_j|. \quad (2.4)$$

Клас, для якого Hamming-відстань мінімальна, обирається як результат класифікації. З точки зору обчислень, метод має складність $O(K \cdot D)$ для одного запиту, де K – кількість класів.

Теоретично цей підхід є ефективним для одномодальних класів, де дескриптори розташовані компактно у бінарному просторі, і взаємні залежності між бітами можна ігнорувати. Однак у випадках багатомодальної структури дескрипторів або наявності сильних кореляцій між бітами усереднення «згладжує» особливості окремих дескрипторів, що знижує дискримінативну здатність центру. Поодинокі помилки або шум у невеликих наборах даних можуть змістити бінарний центр, якщо ключові координати дескрипторів пошкоджені.

Переваги методу полягають у простоті реалізації та інтуїтивній зрозумілості класифікаційного правила: не потрібне додаткове тонке налаштування гіперпараметрів, а мінімізація Hamming-відстані легко інтерпретується і може бути оптимізована апаратно. Недоліком є обмежена гнучкість: єдиний центр не здатен відображати багатомодальні особливості класу та може «затирати» дрібні, але важливі характеристики ознак, що знижує точність класифікації для складних структурних дескрипторів.

2.3 Локально-чутливе хешування

Хешування – це фундаментальна концепція у комп’ютерних науках, що використовується для компактного представлення, зберігання та швидкого пошуку даних. Його суть полягає у застосуванні спеціальної функції, яка перетворює вхідні дані довільної довжини у фіксовану послідовність символів або бітів – хеш-код. Основна мета хешування – забезпечення швидкого доступу до даних без необхідності повного перебору, що особливо важливо при роботі з великими обсягами інформації. У найпростішому вигляді хеш-функція може використовуватись для реалізації хеш-таблиць – структур даних, які забезпечують майже миттєвий пошук, вставку або видалення елементів. Важливою властивістю хеш-функції є рівномірність розподілу результатів: вона повинна розподіляти дані таким чином, щоб уникати надмірних колізій, коли різні елементи отримують однаковий хеш [18].

Існує багато різновидів хешування, і їх вибір залежить від конкретної мети. Наприклад, криптографічне хешування (SHA, MD5 тощо) застосовується для перевірки цілісності та захисту даних – тут важливо, щоб навіть мінімальна зміна у вхідних даних призводила до повністю іншого хешу. У класичному пошуку чи зберіганні, навпаки, іноді бажано, щоб схожі об’єкти мали близькі хеші – саме ця властивість стала основою для появи сімейства методів локально чутливого хешування. Водночас хешування може виступати не лише як інструмент зберігання, а й як засіб зменшення розмірності даних, коли складні багатовимірні об’єкти представляються у вигляді коротших векторів для подальшої обробки [18].

У сфері комп’ютерного зору та машинного навчання хешування використовується як ефективний спосіб представлення ознак, особливо коли обсяг даних надто великий для прямого порівняння. Уявімо задачу пошуку схожих зображень у базі з мільйонами записів – класичні методи порівняння всіх пар ознак будуть надто повільними. Натомість хешування дозволяє швидко визначити, у яких частинах бази можуть міститися потенційно схожі об’єкти,

скорочуючи кількість порівнянь у тисячі разів. Такий підхід лежить в основі сучасних систем класифікації, пошуку зображень та відео, де хешування забезпечує поєднання швидкодії, компактності та практичної стійкості до варіацій у даних.

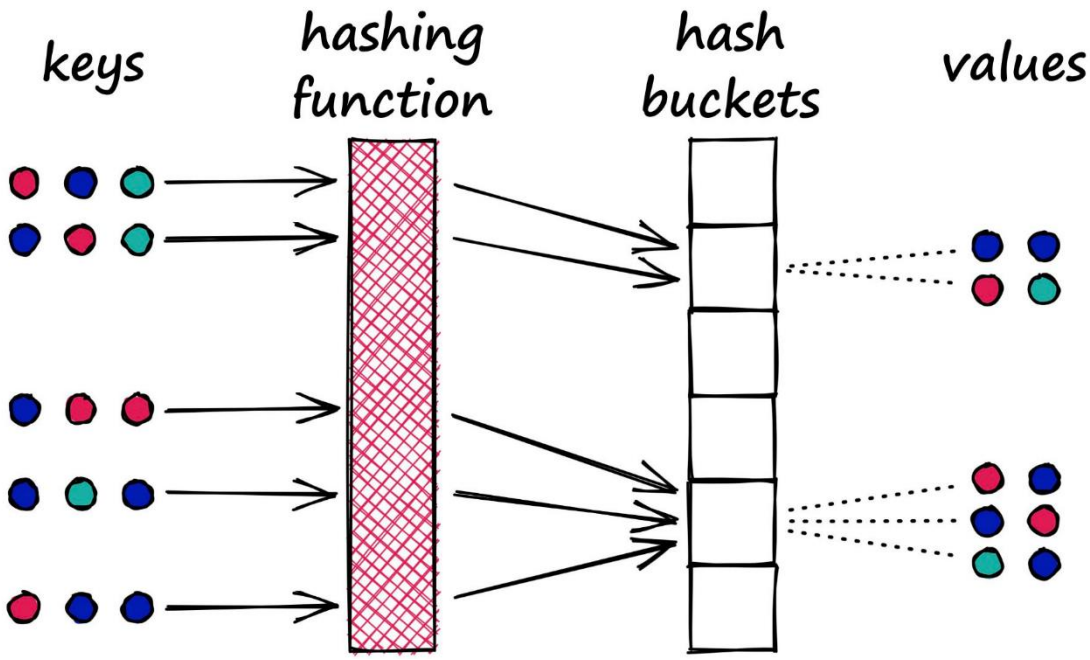

Рисунок 2.1. Процес хешування

Locality-Sensitive Hashing (LSH) – це підхід, який використовується для швидкого пошуку схожих об’єктів у великих багатовимірних просторах ознак. Його ключова ідея полягає в тому, щоб перетворювати дані у компактні хеш-представлення таким чином, щоб схожі об’єкти мали схожі або навіть однакові хеші. На відміну від звичайного хешування, де метою є уникнення колізій, у LSH навпаки колізії є корисними, адже вони сигналізують про потенційну схожість об’єктів. Це дозволяє значно зменшити кількість порівнянь, оскільки система замість повного перебору всіх елементів аналізує лише ті, що потрапили в однакові або близькі комірки у хеш-просторі. Загалом хешування – це процес перетворення довільного набору даних у фіксовану послідовність бітів, що називається хеш-кодом. Класичне хешування використовується для швидкого доступу до даних, створення хеш-таблиць або перевірки цілісності інформації.

Проте у випадку з даними високої розмірності, наприклад, із векторами ознак зображень, звичайне хешування неефективне, оскільки воно не зберігає інформацію про подібність. LSH вирішує цю проблему, забезпечуючи більшу ймовірність того, що об’єкти, які близькі у початковому просторі, матимуть однаковий хеш [19].

LSH складається з набору спеціальних хеш-функцій, які побудовані таким чином, щоб зберігати геометричну близькість між об’єктами. Кожен об’єкт проєктується у нижчовимірний простір, де простіше визначати схожість. Отримані хеш-коди потрапляють до певних «кошиків» або комірок, і під час пошуку схожих елементів система перевіряє лише ті об’єкти, що знаходяться у тому самому кошику, що й запит. Це дає можливість виконувати пошук із сублінійною складністю, що особливо важливо для великих наборів даних. Основна перевага LSH полягає у тому, що метод дозволяє знайти наближено найближчих сусідів із високою швидкістю, при цьому забезпечуючи прийнятний баланс між точністю та ефективністю.

У комп’ютерному зорі LSH знайшов широке застосування, зокрема у задачах класифікації та пошуку схожих зображень. Під час обробки зображень із них зазвичай спочатку витягуються локальні дескриптори, що відображають текстурні, геометричні або колірні особливості. Ці дескриптори часто мають велику розмірність, і без попередньої обробки пошук серед них був би надзвичайно повільним. LSH дозволяє суттєво скоротити кількість порівнянь, адже дескриптори, що мають подібну структуру, потрапляють у ті самі або близькі хеш-комірки. Таким чином, класифікація зображень із використанням LSH відбувається за принципом порівняння з еталонними зразками, де замість повного перебору порівнюються лише ті об’єкти, що опинилися у спільних кошиках. Це не тільки пришвидшує обчислення, а й підвищує масштабованість системи [20].

Кожен дескриптор відображається у хеш-код, який вказує, у який кошик потрапить цей об’єкт. Під час класифікації нового зображення система порівнює його хеш-коди з хешами еталонних об’єктів, обчислюючи кількість збігів у

бакетах. Об’єкт відноситься до того класу, чий еталон має найбільшу кількість подібних хешів. Такий підхід дозволяє системі швидко визначати схожість навіть при наявності шумів, незначних змін масштабу чи обертання.

Крім того, LSH у контексті класифікації відіграє роль не лише пошукового механізму, а й інструмента зниження розмірності. Оскільки високорозмірні дескриптори перетворюються у коротші хеші, це дозволяє зменшити обсяг збережених даних і прискорити подальшу обробку. При цьому зберігається головна властивість – близькі за змістом зображення залишаються близькими і в хеш-просторі. Завдяки цій властивості метод LSH ефективно використовується для класифікації об’єктів на основі порівняння з еталонами, коли потрібно не лише визначити, чи збігаються два зображення, а й знайти найбільш подібний клас серед великої кількості можливих варіантів [21].

2.4 Хешування за кількістю одиниць

Одним із найпростіших і водночас ефективних способів попереднього хешування бінарних дескрипторів є хешування за кількістю одиниць. Ідея цього методу полягає у використанні простої характеристики бінарного вектору – кількості бітів зі значенням «1» – як індексу для розподілу дескрипторів по кошиках. Основна ідея полягає в тому, що бінарні вектори, які мають схожу кількість одиниць, зазвичай є близькими за Hamming-відстанню, а отже – й за змістом. Таким чином, групування дескрипторів за кількістю одиниць дозволяє зменшити розмір простору пошуку, уникаючи необхідності порівнювати запит із кожним елементом бази [22, 23].

На етапі підготовки даних формується проста, але ефективна хеш-таблиця. Нехай кожен дескриптор має довжину D біт. Тоді створюється $D+$ кошиків, пронумерованих від 0 до D, де кожен кошик відповідає певному значенню кількості одиниць у векторі. Для кожного вектору D обчислюється кількість одиниць за формулою:

$$popcount(x_i) = \sum_{j=1}^{D} x_{i,j}. \quad (2.5)$$

Розподіл кількості одиниць у бінарних дескрипторах має наближено біноміальний характер. Якщо припустити, що всі біти незалежні й кожен має ймовірність p бути одиницею, то ймовірність того, що у векторі буде рівно k одиниць, описується біноміальним законом:

$$P(k) = \binom{D}{k} p^k (1-p)^{D-k}. \quad (2.6)$$

Отримане число визначає номер кошика, у який поміщається цей дескриптор. Наприклад, якщо в бінарному векторі 123 біти мають значення «1», то такий дескриптор потрапляє у кошик №123. Після заповнення всіх кошиків вони міститимуть різну кількість елементів: центральні – найбільш насичені, а крайні (0 і D) можуть бути практично порожніми.

Класифікація проводиться наступним чином:

– по черзі беремо кожен дескриптор із запитного зображення;

– швидко підраховуємо його кількість одиниць і дивимося у відповідний кошик;

– якщо цей кошик порожній – пропускаємо дескриптор (він не голосує);

– якщо ні – серед тих тренувальних дескрипторів, що в кошику, шукаємо найближчого за Hamming-відстанню (тобто мінімально відрізняється біт за бітом);

– дескриптор «віддає» свій голос за клас знайденого сусіда. Лінійний пошук;

– після обробки всіх дескрипторів підраховуємо голоси: якщо їх зовсім не було, то повертаємо «Unknown», інакше вирішує правило більшості (або можна зважувати голоси залежно від надійності кошика).

Метод класифікації за кількістю одиниць має низку вагомих переваг. Він є надзвичайно простим у реалізації, не вимагає ніяких гіперпараметрів або попереднього навчання, забезпечує високу швидкість завдяки використанню апаратно-оптимізованих операцій і добре масштабується на великі обсяги даних.

Крім того, цей підхід ефективний у випадках, коли класи істотно відрізняються за середньою кількістю одиниць у своїх дескрипторах, тобто коли гістограми popcount різних класів не перекриваються. У такій ситуації навіть просте розділення за кількістю одиниць забезпечує чітку дискримінацію між класами [22, 23].

Разом з тим метод має і суттєві обмеження. Найбільша його проблема – нерівномірний розподіл дескрипторів по кошиках. Оскільки більшість векторів концентрується у центральних областях біноміального розподілу, саме ці кошики містять найбільшу кількість елементів, що призводить до зростання колізій і збільшення часу пошуку. Натомість крайні кошики залишаються практично порожніми, через що певна частина запитних дескрипторів не знаходить відповідностей і не бере участі у голосуванні. Крім того, якщо різні класи мають подібні розподіли кількості одиниць, дискримінаційна здатність цього підходу суттєво зменшується, оскільки кількість одиниць у такому разі не несе достатньо інформації для надійного розмежування.

2.5 Метод Random Projection LSH

Метод Random Projection LSH (локально-чутливе хешування на основі випадкових проєкцій), або SimHash, є ефективним підходом до прискорення пошуку подібних об’єктів у високовимірних просторах. Основна ідея полягає у тому, щоб замість прямого порівняння усіх векторів ознак, що є обчислювально затратним при великих обсягах даних, виконувати їх проєкцію на набір випадкових гіперплощин. Результатом цієї проєкції є компактні бінарні хеші, що зберігають інформацію про кутову подібність між векторами. Такий підхід дозволяє значно зменшити обчислювальні витрати під час пошуку найближчих сусідів, зберігаючи при цьому інформацію про напрямок та відносну подібність між векторами.

Принцип роботи методу Random Projection LSH базується на геометричному підході: кожен вектор дескриптора розглядається як точка у D-

вимірному евклідовому просторі. Для побудови хеш-функції генерується набір випадкових гіперплощин, кожна з яких задається вектором нормалі, компоненти якого згенеровані з нормального розподілу $N(0,1)$ [24].

Генерація набору гіперплощин реалізується у кілька основних кроків [25].

Спочатку відбувається ініціалізація генератора випадкових чисел. Мета цього кроку полягає у забезпеченні відтворюваності результатів під час повторних запусків алгоритму. Це означає, що при одному й тому ж початковому значенні генератора (seed) має формуватися абсолютно ідентична матриця напрямів. Така властивість є критично важливою для проведення наукових експериментів і порівняльного аналізу, оскільки дозволяє уникнути випадкових варіацій у результатах. Для цього створюється локальний генератор випадкових чисел з фіксованим seed, незалежний від інших частин програми. Без фіксації початкового стану неможливо об'єктивно оцінити вплив параметрів методу, оскільки різні набори напрямів щоразу створювали б різні результати навіть при незмінних даних.

$$R_{i,j}^{\text{raw}} \sim N(0,1), \qquad i = 1, \dots, k, \quad j = 1, \dots, D. \tag{2.7}$$

Другим кроком є генерація матриці випадкових значень, що позначається як R_{raw}. Ця матриця має розмір $k \times D$, де D – розмірність простору ознак, а k – кількість напрямів або, відповідно, довжина бінарного хешу. Для кожного елемента матриці генерується випадкове число, розподілене за нормальним законом $N(0,1)$. Такий підхід обрано невипадково: нормальний розподіл гарантує рівномірне охоплення напрямів у багатовимірному просторі, що забезпечує статистичну неупередженість хешування. Кожний рядок матриці R_raw відповідає окремій гіперплощині у просторі дескрипторів. Наприклад, якщо ми маємо $k = 1$ і $D = 4$, то один рядок матриці може виглядати як [1,245, –0,672, 0,305, 0,888]. Такий набір чисел фактично визначає орієнтацію однієї гіперплощини у чотиривимірному просторі.

Третім кроком є обчислення евклідових норм рядків матриці (2.8), необхідне для того, щоб відокремити напрямок вектору від його довжини. Метою цього етапу є забезпечення того, щоб під час обчислення скалярного добутку з дескриптором впливала лише кутова різниця, а не абсолютна величина вектору. Для кожного рядка і матриці R_{raw} обчислюється його довжина за формулою евклідової норми:

$$L_i = \sqrt{\sum_{j=1}^{D} R_{\text{raw},i,j}^2}. \tag{2.8}$$

Цей крок можна проілюструвати на простому прикладі. Для вектору [1.245, –0.672, 0.305, 0.888] обчислюється сума квадратів його компонентів, що дорівнює 2.885, а потім з неї береться квадратний корінь – 1.699. Якщо цей етап пропустити, то вектори з більшою довжиною даватимуть непропорційно великі значення скалярних добутків, що призведе до викривлення подібності між дескрипторами. Тоді подібність перестане залежати виключно від кута між векторами, і метод втратить свою інваріантність до масштабування ознак.

Завершальною є нормалізація рядків до одиничної довжини. Цей крок полягає у тому, щоб кожен вектор-напрямок перетворити на одиничний, тобто такий, що має довжину рівно 1. Для цього кожен елемент рядка ділиться на обчислену раніше норму L_i (2.10). Таким чином отримуємо матрицю R, у якій кожен рядок є нормованим вектором. У наведеному прикладі для вектору [1,245, –0,672, 0,305, 0,888] з нормою 1,699 нормалізація дає результат [0,733, –0,396, 0,180, 0,523]. Перевірка підтверджує, що сума квадратів компонент цього вектору дорівнює одиниці. Завдяки цьому скалярний добуток між нормалізованим напрямом і вектором ознак x визначатиме лише кутову подібність між ними, а не залежатиме від масштабів числових значень. Нормалізація є критично важливим кроком, адже саме вона гарантує справедливість розподілу гіперплощин і стабільність роботи SimHash у різних експериментальних умовах.

$$R_{i,j} = \frac{R_{i,j}^{\text{raw}}}{L_i}. \quad (2.10)$$

У результаті виконання цих етапів формується матриця напрямів R, яка використовується для подальшої проєкції векторів ознак.

Після обчислення матриці напрямів можна переходити до обчислення хешу дескрипторів. Процес обчислення SimHash для одного дескриптора складається з чотирьох основних етапів [24, 26].

Перший етап – центрування дескриптора. Його мета полягає у видаленні зміщення навколо нуля, щоб подальші обчислення враховували лише напрям вектору, а не абсолютні значення його компонент. Для цього дескриптор представляється у вигляді бінарного вектору, а його кожен елемент $x_j \in \{0,1\}$ перетворюють за формулою $x'_j = x_j - 0.5$. Таким чином, нульові значення стають -0.5, а одиниці – +0.5. Це дозволяє зрівноважити вектор, зробивши його середнє значення нульовим. Наприклад, якщо вихідний вектор x = [1, 0, 1, 1], то після центрування отримаємо x' = [+0,5, -0,5, +0,5, +0,5]. Без цього кроку результати проєкції були б зміщені в залежності від кількості одиниць у векторі.

Другий етап – проекція на гіперплощини. На цьому кроці ми використовуємо заздалегідь згенеровану матрицю напрямів $R \in R^{k \times D}$, де кожен рядок R_i є одиничним вектором, який визначає орієнтацію однієї з гіперплощин у D-вимірному просторі. Для кожного напрямку обчислюється скалярний добуток за формулою:

$$s_i = \sum_{j=1}^{D} R_{i,j} \cdot x'_j, \quad \text{для } i = 1, \dots, k. \quad (2.11)$$

Це значення відображає, наскільки дескриптор «спрямований» у бік даної гіперплощини. Наприклад, якщо $R_1 = [0{,}3, -1{,}2, 0{,}5, 0{,}9]$, а x' = [+0,5, -0,5, +0,5, +0,5], тоді $s_1 = 0{,}3 \cdot 0{,}5 + (-1{,}2)(-0{,}5) + 0{,}5 \cdot 0{,}5 + 0{,}9 \cdot 0{,}5 = 1{,}45$. Додатне

значення свідчить, що вектор лежить по той бік гіперплощини, куди спрямований R_i, а від'ємне – що з протилежного боку.

Третій етап – бінаризація за знаком (2.12). Тепер кожне отримане значення s_i перетворюється на один біт b_i, який показує, з якого боку від гіперплощини розташований дескриптор. Це робиться за правилом: якщо $s_i > 0$, тоді $b_i = 1$; інакше $b_i = 0$. У результаті формується бітовий вектор $b = [b_1, b_2, \dots, b_k]$, який і є основою для майбутнього SimHash. Наприклад, якщо s = [2,25, -0,87, 0,03, -1,44], то відповідний бітовий вектор буде b = [1, 0, 1, 0]. Кожен біт цього вектору відповідає певній гіперплощині та зберігає інформацію про «бік» розташування дескриптора, завдяки чому SimHash добре відображає просторову близькість об'єктів.

$$b_i = \begin{cases} 1, & s_i > 0 \\ 0, & s_i \leq 0 \end{cases}, \; i = 1, \dots, k \quad . \tag{2.12}$$

Четвертий етап – формування цілого SimHash-ключа. Щоб зробити результат компактним і зручним для зберігання або порівняння, бітовий вектор b перетворюють на ціле число.

SimHash стискає вихідний дескриптор у компактне числове представлення, у якому близькі вектори в багатовимірному просторі мають схожі коди з малою гамінговою відстанню. Це дає змогу ефективно порівнювати великі обсяги даних і знаходити подібні об'єкти за допомогою простих бітових операцій.

Після того як хеш був обчислений для усіх дескрипторів, можна переходити до побудови хеш-таблиці. [18, 27, 29]

Побудова хеш-таблиці SimHash є ключовим етапом у структуризації великої кількості дескрипторів для подальшого швидкого пошуку подібних об'єктів. Вона дає змогу замість порівняння кожного дескриптора з усіма іншими зберігати їх у компактній асоціативній структурі, що дозволяє швидко знаходити кандидати з подібними хешами.

Перший етап – ініціалізація структури хеш-таблиці. На цьому кроці створюється порожня асоціативна структура даних – словник (hash map), який надалі зберігатиме відповідність між обчисленими SimHash-ключами та індексами дескрипторів, яким ці ключі належать. Ключами є цілі числа – значення SimHash, а значеннями — списки індексів тих дескрипторів, що мають однаковий хеш. Формально цю структуру можна описати як $H = \emptyset$ тобто на початку вона не містить жодного запису. Таке представлення дозволяє швидко створювати, оновлювати й отримувати доступ до кошиків — груп дескрипторів із однаковим SimHash.

Другий етап – групування дескрипторів за ключами. Для кожного дескриптора з індексом $i = 1 \ldots N$ обчислюється його SimHash-ключ: $key_i = SimHash(x_i)$. Після цього перевіряється, чи існує в хеш-таблиці кошик із цим ключем. Якщо ні – створюється новий порожній список, до якого додається поточний індекс iii. Якщо кошик уже існує, до нього просто додається індекс нового дескриптора. Формально процес можна описати як:

$$I_{\text{key}_i} \leftarrow I_{\text{key}_i} \cup \{i\}, \text{ де } I_k = \{j \mid SimHash(x_j) = k\}. \quad (2.13)$$

У результаті кожен дескриптор потрапляє у свою «групу подібності», сформовану за значенням хешу. Цей крок є основою для ефективного пошуку, адже тепер замість повного перебору всієї бази можна обмежитися перевіркою лише елементів із тим самим або близьким ключем.

Третій етап – фіналізація хеш-таблиці. Після обробки всіх N дескрипторів формується готова структура H, у якій кожен SimHash-ключ відповідає множині індексів дескрипторів, що мають однакове хеш-представлення.

Отримана структура повертається разом із матрицею напрямів R, щоб можна було надалі обчислювати хеші для нових дескрипторів під час запитів або класифікації. Таким чином, хеш-таблиця стає компактним індексом, який значно пришвидшує подальший пошук подібних зображень.

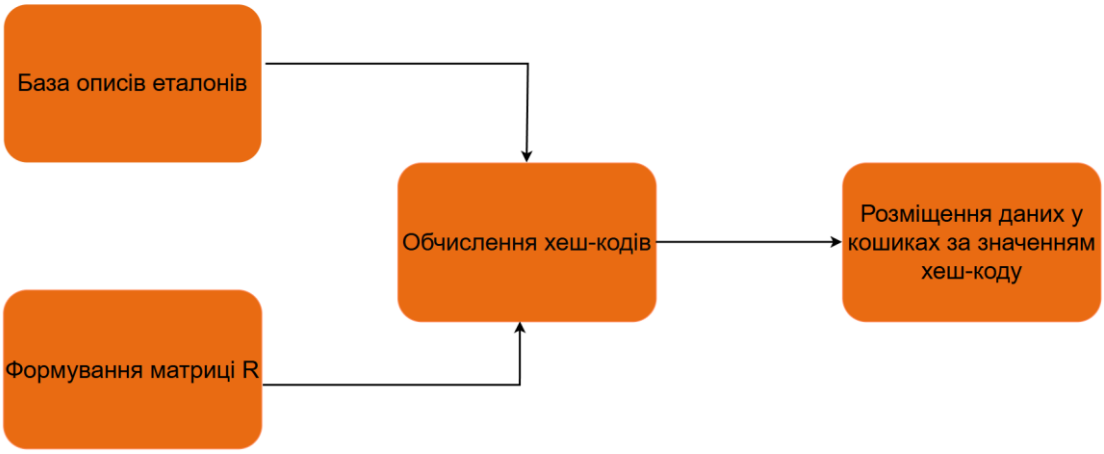

Рисунок 2.2. Схема аналізу даних при формуванні кошиків для бази еталонів

Останнім етапом є класифікація зображення за побудованою хеш-таблицею. Кожен дескриптор із запитного зображення обробляється по черзі: для нього обчислюється SimHash, а потім визначається відповідний кошик. Якщо кошик порожній, дескриптор не враховується. Якщо ні – серед еталонних дескрипторів у цьому кошику знаходиться найближчий за відстанню Хемінга, тобто той, що має мінімальні побітові відмінності. Дескриптор запиту «віддає голос» за клас знайденого сусіда. Після обробки всіх дескрипторів підраховується кількість голосів на користь кожного класу. Якщо голосів немає або їх недостатньо – система повертає результат «Unknown». Якщо ж більшість голосів належить певному класу, саме він визначається як підсумковий результат класифікації. За потреби голоси можна зважувати, наприклад, з урахуванням надійності або щільності відповідного кошику [28].

Рисунок 2.3. Схема встановлення класу для дескриптора об’єкту

2.6 Метод Multi-probe LSH

Метод Multi-Probe LSH – це вдосконалена модифікація класичного методу локально чутливого хешування випадкових проекцій, спрямована на підвищення точності пошуку найближчих сусідів у високовимірних просторах ознак. У стандартному LSH кожен вектор ознак перетворюється у компактне хеш-представлення за допомогою випадкових гіперплощин, що поділяють простір на області. Кожен вектор потрапляє до певної комірки (кошика) у хеш-таблиці залежно від свого хеш-коду. Під час пошуку об’єктів, схожих на запит, система шукає лише ті елементи, що знаходяться у тій самій комірці, тобто мають однаковий хеш-код. Такий підхід забезпечує дуже високу швидкість, однак має значний недолік – навіть незначна зміна вектору може призвести до потрапляння його в іншу комірку, через що схожі об’єкти можуть бути повністю втрачені [30].

Метод Multi-Probe LSH вирішує цю проблему, запроваджуючи ідею багатокрокового пошуку не лише у комірці, що точно відповідає хешу запиту, а й у кількох «сусідніх» комірках. Сусідні кошики визначаються за принципом мінімальної відстані у хеш-просторі, тобто перевіряються ті ключі, які відрізняються від базового хешу на невелику кількість бітів. Основна ідея методу полягає у формуванні впорядкованої послідовності можливих «підстановок» бітів у хеш-коді запиту, що задає пріоритет перевірки сусідніх комірок. Ця послідовність може бути отримана аналітично на основі геометрії випадкових гіперплощин або побудована емпірично на основі результатів тестування на вибірці даних. Кількість перевірок вибирається експериментально, щоб досягти оптимального компромісу між точністю та швидкодією.

Метод Multi-Probe LSH не потребує створення додаткових структур або таблиць: він використовує ту ж саму хеш-таблицю, що й класичний LSH, а зміни стосуються лише логіки пошуку. Алгоритм починається зі стандартного хешування запитного вектору для визначення базової комірки, після чого формується список сусідніх хешів, які можуть містити релевантні об’єкти. Перевірка сусідніх комірок проводиться пріоритетно, починаючи з тих бітів

хешу, які є найбільш «нестабільними», тобто для яких невелике відхилення даних може змінити біт у хеші. Далі перевіряються комбінації із двох і більше бітових змін, поступово розширюючи область пошуку. У деяких реалізаціях використовується ймовірнісний підхід: сусідні ключі генеруються з певною імовірністю залежно від відстані до меж гіперплощин, що дозволяє скоротити непотрібні перевірки і підвищити ефективність [30].

Метод підтримує гнучке налаштування глибини пошуку залежно від вимог конкретної задачі. Параметри, такі як максимальна кількість зондованих сусідніх комірок, порядок генерування варіацій бітів або критерії зупинки, можуть адаптуватися до бажаного рівня повноти та точності або часу відповіді системи. Це дозволяє інтегрувати метод у системи з різними обмеженнями — від режимів реального часу, де критичною є швидкодія, до офлайн-аналітики, де пріоритет надається точності.

Multi-Probe LSH здатен значно підвищувати точність пошуку найближчих сусідів без суттєвого збільшення обчислювальних витрат або використання пам’яті. Це забезпечує кращий баланс між швидкістю, стійкістю та точністю пошуку і робить метод ефективним у задачах класифікації та порівняння високовимірних ознак.

3 АНАЛІЗ РЕЗУЛЬТАТІВ КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ МЕТОДІВ

3.1 Обґрунтування програмних засобів

У ході виконання дослідження було розроблено та досліджено методи класифікації зображень квітів родини лілійних на основі хешування структурного опису. Для програмної реалізації обрано інтегроване середовище розробки PyCharm Community Edition 2025.2.1.1, мову програмування Python 3 та бібліотеки для обробки даних та роботи із зображеннями: OpenCV (cv2), NumPy, Matplotlib, а також стандартні модулі Python, такі як collections, os, time. Моделювання та оцінювання продуктивності здійснювалося на комп'ютері з процесором Intel Core i7-9700K (3.60 ГГц), оперативною пам'яттю 16 ГБ DDR4, під управлінням Windows 11 Pro (версія 24H2).

Інтегроване середовище розробки PyCharm надає широкий спектр інструментів для ефективної роботи з Python, включаючи підсвітку синтаксису, автодоповнення коду, інтегровану відладку та рефакторинг. Вибір редакції Community обумовлений її безкоштовністю та відкритістю, що забезпечує доступ до повного функціоналу для наукової розробки, включно з підтримкою інтерактивної консолі та керуванням бібліотеками для обробки даних і наукових обчислень.

Бібліотека OpenCV (Open Source Computer Vision Library) використовується для завантаження та масштабування зображень, обчислення ключових точок і дескрипторів (AKAZE), а також візуалізації результатів. Вона надає широкий набір інструментів для роботи з комп'ютерним зором, включаючи обробку зображень, виділення ознак, класифікацію об'єктів, відстеження рухомих об'єктів, з'єднання зображень у панорами та побудову пошуку за зразком. OpenCV має відкритий код і підтримує інтерфейси для Python, C++, Java та MATLAB, що робить її гнучким інструментом для досліджень у сфері комп'ютерного зору.

Бібліотека NumPy використовується для роботи з багатовимірними масивами та виконання ефективних чисельних операцій, включаючи матричні обчислення та побітові операції (наприклад, np.unpackbits). Matplotlib забезпечує побудову графічних відображень результатів, таких як гістограми та стовпчикові діаграми, що полегшує аналіз роботи алгоритмів. Модулі collections.Counter та defaultdict використовуються для підрахунку голосів та розподілу класів у хеш-кошиках, а модулі os та time – для роботи з файловою системою та оцінки часу виконання операцій.

Мова програмування Python обрана через простоту використання, високу читабельність коду, об’єктно-орієнтованість та відкритий код, що забезпечує гнучкість у створенні наукових додатків. Її інтеграція з бібліотеками для наукових обчислень і комп’ютерного зору дозволяє швидко реалізовувати алгоритми обробки та аналізу зображень без необхідності реалізовувати базові функції з нуля.

Таким чином, комбінація Python, PyCharm та зазначених бібліотек створює ефективне середовище для розробки, тестування та дослідження методів класифікації зображень на основі хешування структурного опису, забезпечуючи баланс між продуктивністю, зручністю розробки та науковою гнучкістю.

3.2 Особливості програмної реалізації

3.2.1 Реалізація методу Random Projection LSH

Першим етапом реалізації методу Random Projection LSH є підготовка бази дескрипторів еталонних зображень. На початку відбувається зчитування зображень, для яких обчислюються ключові точки (КТ) та дескриптори за допомогою детектора AKAZE. Для кожного зображення створюється конструктор детектора та встановлюється порогове значення виявлення КТ.

Другим етапом є підготовка хеш-функцій. Використовується функція gen_gaussian_planes(), яка генерує випадкові гауссові площини для хешування.

Кожна площина нормалізується для забезпечення стабільності обчислень. Кількість площин визначає розмір хеш-ключа, що безпосередньо впливає на роздільну здатність хешування та швидкодію пошуку.

Лістинг 3.1 Генерація гаусовських гіперплощин:

```
def gen_gaussian_planes(num_bits: int, D: int, seed: int = 42) -> np.ndarray:
    rng = np.random.RandomState(seed)
    R = rng.randn(num_bits, D).astype(np.float32)
    R /= np.linalg.norm(R, axis=1, keepdims=True)
    return R
```

На наступному етапі кожен дескриптор конвертується у бітовий формат та хешується функцією hash_simhash(). Дескриптор перетворюється на вектор із плаваючою точкою зі зсувом на 0.5, після чого обчислюється скалярний добуток із кожною випадковою площиною. Отримані значення переводяться в біти: якщо добуток більше нуля, встановлюється 1, інакше 0. Біти збираються в десятковий хеш-ключ, який буде використано для індексації дескриптора.

Лістинг 3.2 Функція хешування:

```
def hash_simhash(descriptor_bits: np.ndarray, R: np.ndarray) -> int:
    desc_f = descriptor_bits.astype(np.float32) - 0.5
    dots   = R.dot(desc_f)
    bits   = (dots > 0).astype(np.uint8)
    key = 0
    for i, b in enumerate(bits):
        if b:
            key |= (1 << i)
    return key
```

Після генерації хеш-ключів будується індекс хеш-таблиці за допомогою функції build_simhash_index(). Для кожного дескриптора обчислюється хеш-

ключ і додається до словника hash_table, де ключ – хеш-значення, а значення — список індексів дескрипторів, що потрапили у цей «кошик». Одночасно з цим зберігаються випадкові площини для повторного використання при класифікації нових зображень.

Лістинг 3.3 Побудова хеш-таблиці:

```
    def build_simhash_index(
descriptors_bits: np.ndarray,
num_bits: int = 16,
seed: int = 42
) -> Tuple[Dict[int, List[int]], np.ndarray]:
    start = time.time()
    N, D = descriptors_bits.shape
    R = gen_gaussian_planes(num_bits, D, seed)
    hash_table: Dict[int, List[int]] = defaultdict(list)
    for idx in range(N):
        key = hash_simhash(descriptors_bits[idx], R)
        hash_table[key].append(idx)
    elapsed = time.time() - start
    print(f"[build_simhash_index] buckets={len(hash_table)},
build_time={elapsed:.3f}s")
    return hash_table, R
```

Для збереження таблиці використовується функція save_index(), яка записує на диск словник хешів, бітові дескриптори, матрицю випадкових площин та відповідні мітки зображень. Це дозволяє уникнути повторного обчислення при наступних запусках програми.

Класифікація нового зображення реалізується функцією classify_image(). Спочатку зчитується вхідне зображення та обчислюються його дескриптори у бітовому форматі через допоміжну функцію extract_akaze_bits(). Для кожного

дескриптора обчислюється хеш-ключ методом hash_simhash. Потім знаходиться відповідна «кошик» у хеш-таблиці, і для дескрипторів із цієї кошика обчислюється Хеммінгова дистанція до дескриптора вхідного зображення.

Класифікація нового зображення реалізується функцією classify_image(). Спочатку зчитується вхідне зображення та обчислюються його дескриптори у бітовому форматі через допоміжну функцію extract_akaze_bits(). Для кожного дескриптора обчислюється хеш-ключ методом hash_simhash. Потім знаходиться відповідна «кошик» у хеш-таблиці, і для дескрипторів із цієї кошика обчислюється Хеммінгова дистанція до дескриптора вхідного зображення.

Для кожного дескриптора еталону, що показав найменше значення метрики, визначається мітка зображення. Після обробки всіх дескрипторів підраховується кількість «голосів» за кожне зображення-еталон, і вхідне зображення відноситься до класу, що набрав найбільшу кількість мінімумів. Під час виконання виводиться статистика по пустих «кошиках», час хешування та час голосування для оцінки продуктивності алгоритму.

Лістинг 3.5 Класифікація зображення:

```
    def classify_image(
  image_path: str,
  hash_table: Dict[int, List[int]],
  random_planes: np.ndarray,
  descriptors_bits: np.ndarray,
  labels: List[str]
) -> str:
  print(f"\n[classify_simhash] Classifying '{image_path}' with SimHash")
  q_bits = extract_akaze_bits(image_path)
  if q_bits.size == 0:
    print("  No descriptors → Unknown")
    return "Unknown"
  hash_time_ms = 0.0
```

```
vote_time_ms = 0.0
votes = Counter()
empty_buckets = 0
for i, qb in enumerate(q_bits):
    # Час для хешування
    t0 = time.perf_counter()
    key = hash_simhash(qb, random_planes)
    bucket = hash_table.get(key, [])
    t1 = time.perf_counter()
    hash_time_ms += (t1 - t0) * 1000
    if not bucket:
        empty_buckets += 1
        continue
    t2 = time.perf_counter()
    best_idx = min(
        bucket,
        key=lambda idx: int((qb != descriptors_bits[idx]).sum())
    )
    lab = labels[best_idx]
    votes[lab] += 1
    t3 = time.perf_counter()
    vote_time_ms += (t3 - t2) * 1000
print(f"Descriptors with empty buckets: {empty_buckets} / {len(q_bits)}")
print(f"  hash_time = {hash_time_ms:.3f} ms, vote_time = {vote_time_ms:.3f} ms")
if not votes:
    print("  No votes → Unknown")
    return "Unknown"
class_order = list(dict.fromkeys(labels))
vote_vector = [votes.get(cls, 0) for cls in class_order]
print("\nClass order:", class_order)
```

```
print("Vote vector:", ' '.join(map(str, vote_vector)))
pred = class_order[vote_vector.index(max(vote_vector))]
print(f"\n→ SimHash Prediction: {pred}\n")
return pred
```

Реалізація Random Projection LSH дозволяє ефективно прискорити пошук схожих дескрипторів у великій базі даних, зберігаючи високу точність класифікації. Параметри, такі як кількість біт хеш-ключа та випадкових площин, можуть бути налаштовані для оптимального балансу між швидкодією та точністю.

3.2.2 Особливості програмної реалізації методу LSH на основі кількості одиниць у хеш-коді

Першим етапом реалізації методу є підготовка бази дескрипторів еталонних зображень. На початку відбувається зчитування зображень, для яких обчислюються ключові точки та дескриптори за допомогою детектора AKAZE. Для кожного зображення створюється конструктор детектора та встановлюється порогове значення виявлення КТ. Дескриптори обчислюються через функцію AKAZE і конвертуються у бітовий формат за допомогою функції extract_akaze_bits(). Надалі масиви дескрипторів обрізаються до однакової розмірності для забезпечення коректності подальшого порівняння.

Другим етапом є побудова хеш-таблиці. Для цього використовується функція build_unitcount_index(), яка перебирає всі дескриптори та обчислює хеш-ключ за допомогою функції hash_unitcount(). Ключ формує десяткове значення шляхом підрахунку кількості одиниць у бітовому векторі дескриптора. Після обчислення ключа дескриптор додається до словника hash_table, де ключем виступає отримане число, а значенням — список індексів дескрипторів, що потрапили у відповідний «кошик». На етапі побудови індексу також виводиться

статистика щодо кількості «кошиків» та часу виконання, що дозволяє оцінити продуктивність методу.

Лістинг 3.6 Функції хешування та побудови таблиці за кількостю одиниць:

```
    def hash_unitcount(descriptor_bits: np.ndarray) -> int:
   return int(descriptor_bits.sum())

def build_unitcount_index(descriptors_bits: np.ndarray) -> Dict[int, List[int]]:
   start = time.time()
   hash_table: Dict[int,List[int]] = defaultdict(list)
   for idx, desc in enumerate(descriptors_bits):
      key = hash_unitcount(desc)
      hash_table[key].append(idx)
   elapsed = time.time() - start
   print(f"[build_unitcount_index] buckets={len(hash_table)},
build_time={elapsed:.3f}s")
   return hash_table
```

Класифікація нового зображення реалізується функцією classify_unitcount(). Спочатку відбувається зчитування вхідного зображення та обчислення його дескрипторів у бітовому форматі. Для кожного дескриптора обчислюється хеш-ключ через hash_unitcount, після чого визначається відповідний «кошик» у хеш-таблиці. Якщо кошик порожній, дескриптор пропускається. Для дескрипторів, що потрапили у кошик, обчислюється Хеммінгова дистанція до дескриптора вхідного зображення. Дескриптор еталону з мінімальною дистанцією визначає мітку зображення, яка додається до лічильника голосів. Після обробки всіх дескрипторів підраховується кількість голосів за кожне зображення-еталон, і вхідне зображення відноситься до класу, що набрав найбільшу кількість мінімумів. Під час виконання також виводиться

статистика по порожніх «кошиках», час хешування та час голосування, що дозволяє оцінити ефективність роботи алгоритму.

Лістинг 3.7 Класифікація зображення:

```
    def classify_unitcount(
    image_path: str,
    hash_table: Dict[int, List[int]],
    descriptors_bits: np.ndarray,
    labels: List[str]
) -> str:
    print(f"\n[classify_unitcount] Classifying '{image_path}' with UnitCount")
    q_bits = extract_akaze_bits(image_path)
    if q_bits.size == 0:
        print("  No descriptors → Unknown")
        return "Unknown"
    class_names = sorted(set(labels))
    label_to_idx = {lab: i for i, lab in enumerate(class_names)}
    vote_vector = np.zeros(len(class_names), dtype=int)
    hash_time_ms = 0.0
    vote_time_ms = 0.0
    votes = Counter()
    empty_buckets = 0
    for i, qb in enumerate(q_bits):
        t0 = time.time()
        key    = hash_unitcount(qb)
        bucket = hash_table.get(key, [])
        t1 = time.time()
        hash_time_ms += (t1 - t0) * 1000
        print(f"  Descriptor #{i:3d}: key={key}, bucket_size={len(bucket)}")
```

```
        if not bucket:
            empty_buckets += 1
            continue
        t2 = time.time()
        best_idx = min(bucket, key=lambda idx: int((qb != 
descriptors_bits[idx]).sum()))
        dist     = int((qb != descriptors_bits[best_idx]).sum())
        lab      = labels[best_idx]
        votes[lab] += 1
        vote_vector[label_to_idx[lab]] += 1
        t3 = time.time()
        vote_time_ms += (t3 - t2) * 1000
        print(f"   → best_idx={best_idx}, Hamming={dist}, label='{lab}'")
    print(f"\nDescriptors with empty buckets: {empty_buckets} / {len(q_bits)}")
    print(f"  hash_time = {hash_time_ms:.3f} ms, vote_time = {vote_time_ms:.3f} ms")
    if vote_vector.sum() == 0:
        print("  No votes → Unknown")
        return "Unknown"
    print("\nVote vector:")
    for idx, lab in enumerate(class_names):
        print(f"  {lab}: {vote_vector[idx]}")
    pred = votes.most_common(1)[0][0]
    print(f"\n→ UnitCount Prediction: {pred}\n")
    return pred
```

3.2.3 Програмна реалізація методу центрів

Першим етапом реалізації методу Binary-Centers є підготовка бази дескрипторів еталонних зображень. На початку відбувається зчитування зображень, для яких обчислюються ключові точки та дескриптори за допомогою

детектора AKAZE. Для кожного зображення створюється конструктор детектора та встановлюється порогове значення виявлення КТ. Дескриптори обчислюються через функцію AKAZE і конвертуються у бітовий формат за допомогою функції extract_akaze_bits(). Масиви дескрипторів обрізаються до однакової розмірності, що дозволяє проводити коректне порівняння та агрегацію бітів.

Другим етапом підготовки є обчислення бінарних центрів для кожного класу. Для цього виконується групування дескрипторів за їхніми мітками, після чого кожна група передається до функції majority_vote_hash. Функція обчислює середнє значення бітів по всіх дескрипторах класу та порівнює його із порогом 0,5, формуючи бінарний вектор-«центр» класу. Отримані центри зберігаються у словнику, де ключем виступає назва класу, а значенням — бінарний вектор центру. Під час виконання цієї операції виводиться статистика щодо часу обчислення, що дозволяє оцінити продуктивність формування центрів.

Лістинг 3.8 Розрахунок бінарних центрів:

```
    def majority_vote_hash(descriptors_bits: np.ndarray) -> np.ndarray:
    mean_bits  = np.mean(descriptors_bits, axis=0)
    bin_center = (mean_bits > 0.5).astype(np.uint8)
    return bin_center

def compute_binary_centers(
    descriptors_bits: np.ndarray,
    labels: List[str]
) -> Dict[str, np.ndarray]:
    start = time.time()
    label_to_descs: Dict[str, List[np.ndarray]] = defaultdict(list)
    for desc, lab in zip(descriptors_bits, labels):
        label_to_descs[lab].append(desc)
```

```
    centers = {
        lab: majority_vote_hash(np.vstack(descs))
        for lab, descs in label_to_descs.items()
    }
    elapsed = time.time() - start
    print(f"[compute_binary_centers] classes={list(centers.keys())},
time={elapsed:.3f}s")
    return centers
```

Класифікація нового зображення реалізується функцією classify_by_centers(). Спочатку відбувається зчитування дескрипторів вхідного зображення та перевірка їхньої наявності. Далі для дескрипторів запиту обчислюється бінарний «центр» за допомогою тієї ж функції majority_vote_hash. Цей центр порівнюється з центрами кожного класу шляхом підрахунку Хеммінгової відстані – кількості бітів, що відрізняються. Результати порівняння виводяться у вигляді відстаней до центрів усіх класів, а також показується час обчислення центру та порівняння.

Лістинг 3.9 Класифікація зображення:

```
        def classify_by_centers(
    image_path: str,
    centers: Dict[str, np.ndarray]
) -> str:
    print(f"\n[classify_by_centers] Classifying '{image_path}' with Binary-Centers")

    # 1) Витяг дескрипторів
    q_bits = extract_akaze_bits(image_path)
    if q_bits.size == 0:
        print("  No descriptors → Unknown")
        return "Unknown"
```

```
    # 2) Обчислення «центру» запиту
    t0 = time.perf_counter()
    q_center = majority_vote_hash(q_bits)
    t1 = time.perf_counter()
    center_time_ms = (t1 - t0) * 1000

    # 3) Порівняння з центрами кожного класу
    t2 = time.perf_counter()
    dists = {
        lab: int((q_center != center).sum())
        for lab, center in centers.items()
    }
    t3 = time.perf_counter()
    compare_time_ms = (t3 - t2) * 1000

    # 4) Виводимо відстані
    for lab, dist in dists.items():
        print(f"    Distance to '{lab}': {dist}")

    # 5) Підсумковий час і прогноз
    print(f"  center_time = {center_time_ms:.3f} ms, compare_time =
{compare_time_ms:.3f} ms")

    best_lab = min(dists, key=dists.get)
    print(f"\n→ Binary-Center Prediction: {best_lab}\n")
    return best_lab
```

Після завершення обчислень визначається клас з мінімальною відстанню до бінарного центру запиту. Вхідне зображення відноситься до цього класу, а

результат виводиться на екран разом із супровідною інформацією про час виконання операцій. Реалізація методу Binary-Centers дозволяє швидко класифікувати зображення за попередньо обчисленими бінарними векторами класів, забезпечуючи баланс між точністю та продуктивністю за рахунок зведення порівнянь до одного бінарного вектору на клас замість обробки всіх дескрипторів окремо.

3.3 Аналіз результатів комп'ютерного моделювання розроблених методів класифікації

Дослідження зазначених методів хешування виконувалося на зображеннях квітів родини лілійних, узятих з набору даних OxFord 102 Flower Dataset.

Рисунок 3.1 – Еталонні зображення квіток Ожинова лілія, Вогнівка, Латаття

Для визначення дескрипторів ключових точок застосовано детектор ключових точок AKAZE розмірністю n = 488. Дескриптори мають бінарний вид тому для зручного та простого порівняння векторів застосовано відстань Хеммінга. Розмір зображень склав 512x512 пікселів. Проілюстровано класи еталонних зображень (рис. 3.1) та координати сформованих ключових точок (рис. 3.2). Кількість обчислених дескрипторів у описі кожного із еталонів складає $s = 500$. Таким чином сформована база еталонів скала 1500 дескрипторів.

Для порівняння точності класифікації та швидкодії роботи досліджуваних методів була проведена низка експериментів.

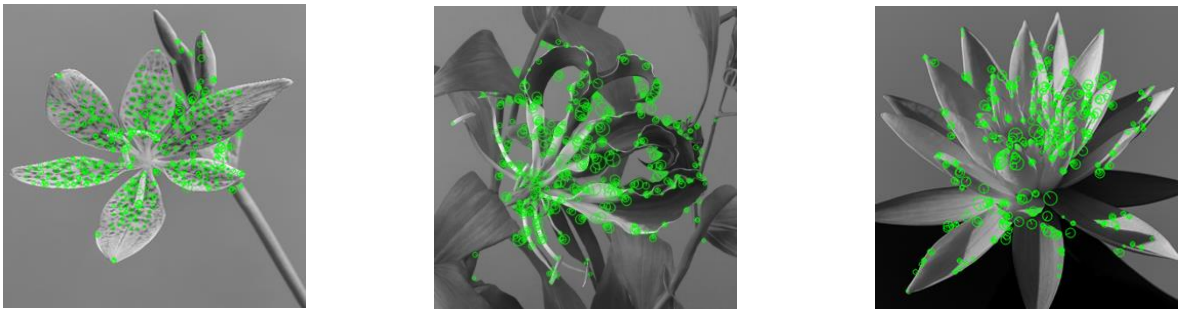

Рисунок 3.2 – Приклади виділених координат ключових точок дескриптором AKAZE (Ожинова лілія, Вогнівка, Латаття)

3.3.1 Аналіз сформованих хеш-таблиць

У методах класифікації, що базуються на хешуванні структурного опису зображень, властивості сформованої хеш-таблиці відіграють визначальну роль у подальшому пошуку та точності класифікації. Саме структура таблиці визначає, наскільки повно та адекватно відображена база даних у хеш-просторі. Тому одним із першочергових етапів аналізу є оцінка сформованих хеш-таблиць.

У методі класифікації за допомогою хешування за кількістю одиниць структура таблиці фіксована, жорстко визначена природою хеш-функції: кількість можливих значень ключа дорівнює довжині бінарного дескриптора (у випадку с дескриптором AKAZE це значення дорівнює 488), а отже змінити кількість кошиків або впливати на їхнє розсіювання неможливо.

У той же час метод Random Projection LSH надає можливість гнучко налаштовувати структуру хеш-таблиці, змінюючи розмір її хеш-ключа. Наприклад, у досліджуваній базі еталонів таблиця з довжиною ключа 8 біт формує 245 заповнених кошиків (рис. 3.3), 16-бітний ключ дає вже 1431 кошик (рис. 3.4), а збільшення розрядності до 32 біт і більше приводить до того, що кількість унікальних хеш-значень дорівнює кількості дескрипторів у базі. Таким чином, зі зростанням довжини ключа хеш-функція забезпечує дедалі тонше розсіювання даних, і за достатньої розрядності кожен дескриптор може отримати фактично унікальний хеш-код. Хоча на практиці через те ще велика розрядність

ключа вимагає високих обчислювальних затрат потрібно досягати компромісу між якістю розсіювання та швидкодією, обираючи таку довжину хеш-коду, яка забезпечує достатню унікальність представлення дескрипторів, але не створює надмірного обчислювального навантаження.

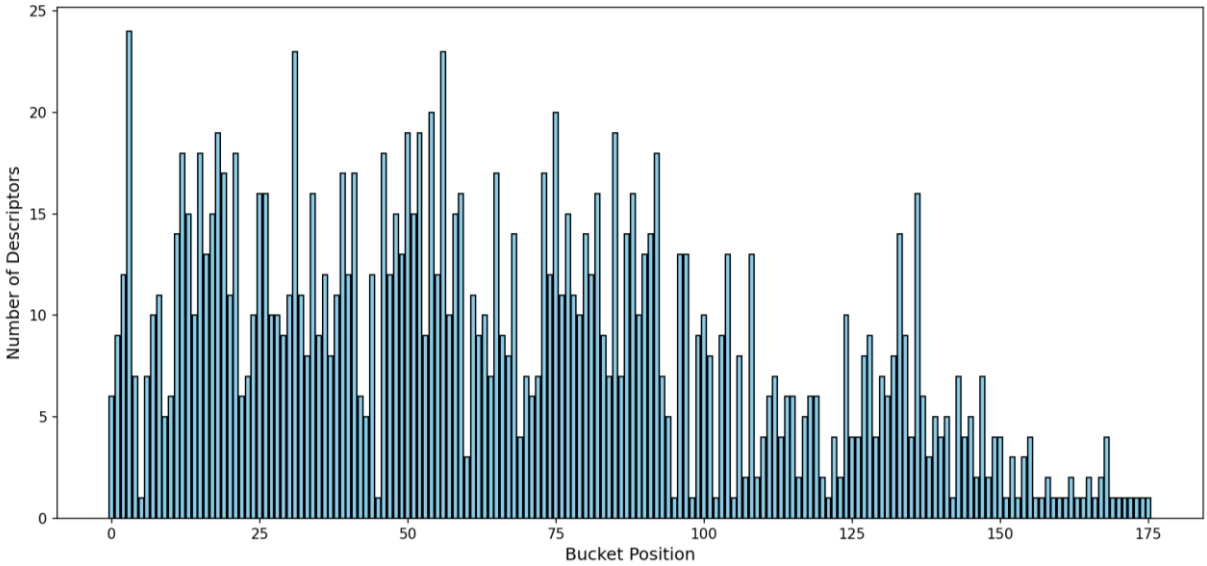

Рисунок 3.3 – Розподіл за хеш-кодом числа одиниць

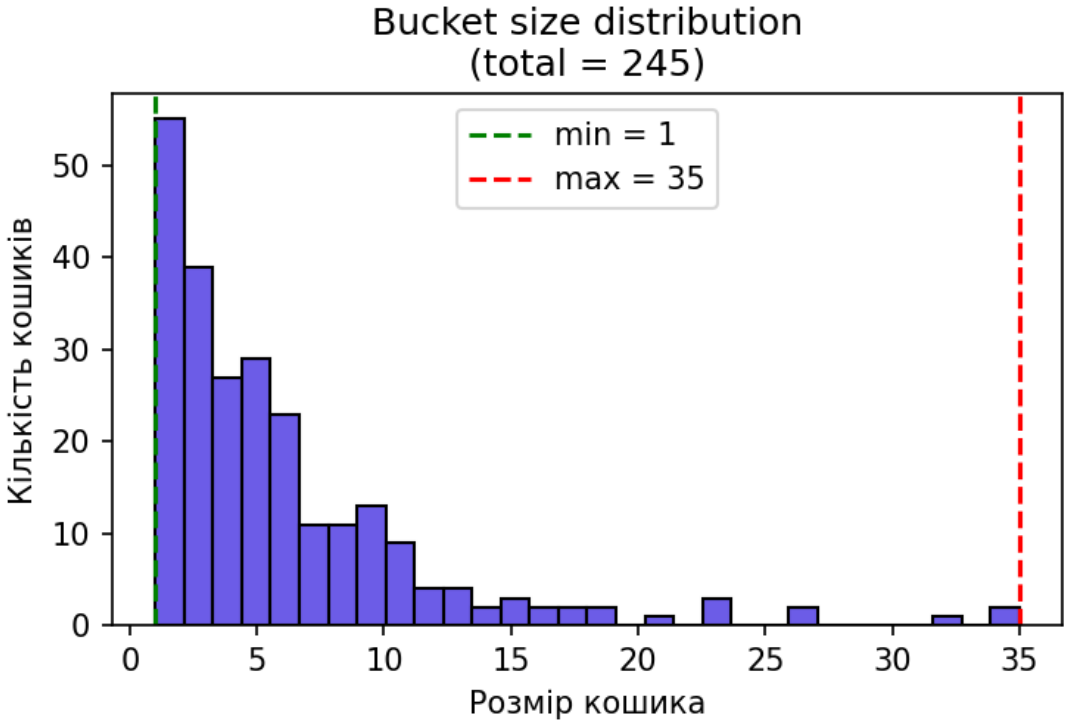

Рисунок 3.3 – Розподіл дескрипторів за LSH (розмір ключа 8)

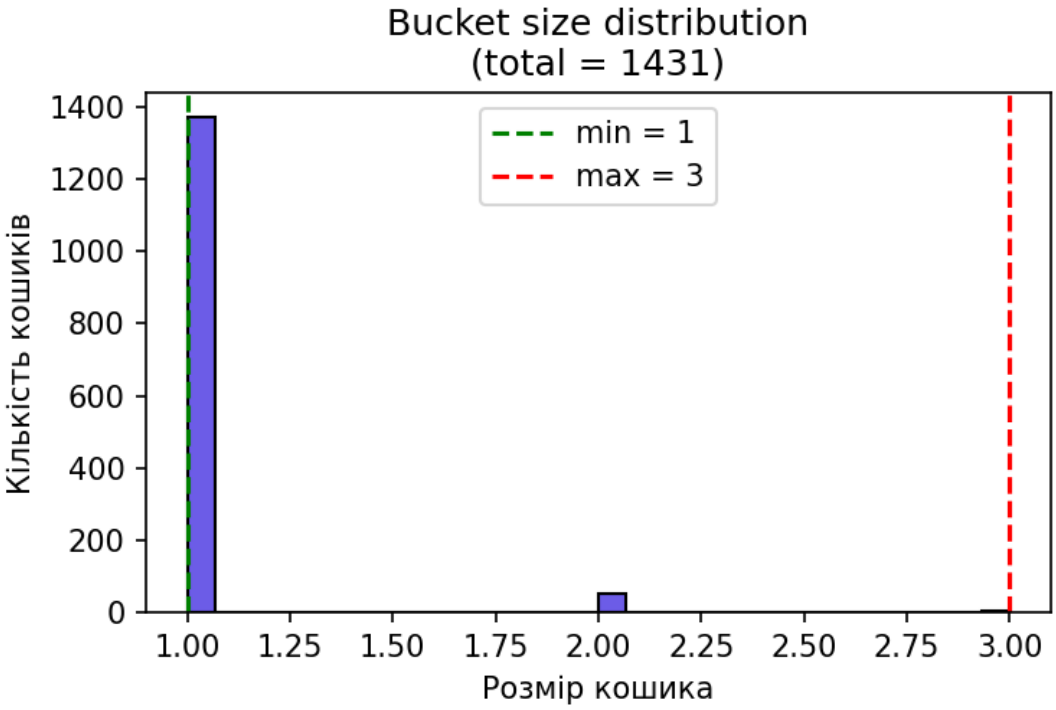

Рисунок 3.4 – Розподіл дескрипторів за LSH (розмір ключа 16)

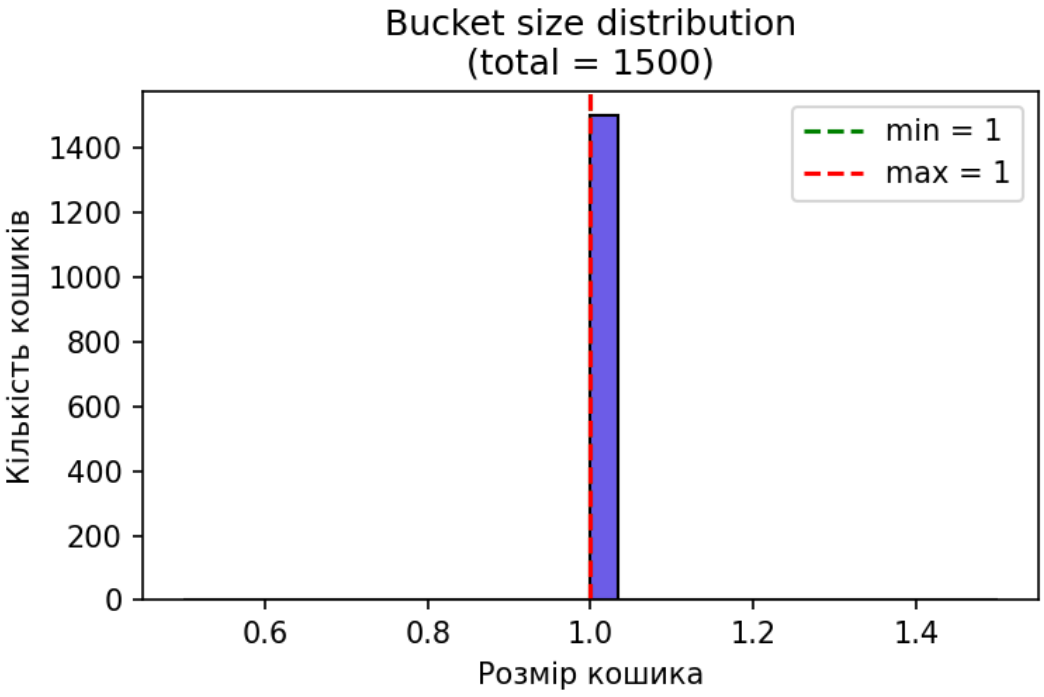

Рисунок 3.5 – Розподіл дескрипторів за LSH (розмір ключа 32)

Таблиця 3.1 Порівняння структури хеш-таблиці в залежності від довжини хеш-ключа

Довжина хеш-ключа	8 біт	16 біт	32 біт
Кількість кошиків	245	1431	1500
Середній розмір кошику	6,12	1,05	1
Максимальний розмір кошику	35	3	1
Мінімальний розмір кошику	1	1	1
Однорідні кошики (класс зображення: blackberrylily)	16	475	500
Однорідні кошики (класс зображення: firelily)	8	460	500
Однорідні кошики (класс зображення: waterlily)	17	459	500

3.3.2 Аналіз результатів класифікації на еталонних зображеннях

Першим експериментом дослідження було виконання класифікації еталонних зображень, тобто зображень, на основі яких формувалися бінарні дескриптори та будувалися відповідна еталонна база та хеш-таблиці. Такий експеримент є принципово важливим для оцінки коректності реалізації методів. Використання еталонних зразків у ролі вхідних даних усуває вплив зовнішніх факторів та зводить задачу до перевірки того, наскільки точно метод відтворює зв’язки, закладені під час формування структур даних. Цей експеримент виконував роль базового контрольного тесту, невдача в якому свідчила б про фундаментальні помилки в реалізації або про принципову непридатність обраного методу для задачі класифікації за допомогою хешування.

Результати експерименту показали, що всі розглянуті методи забезпечують стовідсоткову точність класифікації: кожен дескриптор був віднесений до свого еталонного класу без жодної помилки.

Тобто вектори голосів мають вигляд:

– для E_1 – (500; 0; 0);

– для E_2 – (0; 500; 0);

– для E_3 – (0; 0; 500).

Таблиця 3.2 Порівняння роботи методів класифікації на еталонних зображеннях

Метод	**Час (мс)**	**Точність (%)**
Unitcount	17,89	100
SimHash 16-bit	5,09	100
SimHash 8-bit	20,32	100
SimHash 32-bit	43,93	100
Linear Search	318,51	100
Binary Centers	0,33	100

Такий результат є очікуваним, оскільки в цьому сценарії хеш-таблиця містить вихідні значення ключів, а сам процес класифікації зводиться до пошуку відповідності всередині тих самих структур, на основі яких формувалися еталони. Завдяки цьому жоден із методів не продемонстрував змішаних кошиків чи невідповідностей між класами.

Попри однакову точність, методи суттєво відрізняються за швидкодією. Метод повного перебору очікувано був найповільнішим: для нього час класифікації перевищував 300 мс через необхідність попарного порівняння вхідного дескриптора з усіма еталонними десккрипторами. Найвищу швидкодію

продемонстрував підхід із використанням бінарних центрів класів: час класифікації становив близько 0.3 мс, що зумовлено відсутністю пошуку в хеш-таблиці та зведенням процедури до порівняння вектора ознак із наперед обчисленими центроїдами. Методи на основі SimHash показали час від 7 до 44 мс (але приблизно у 7–43 разів швидше за лінійний пошук) залежно від довжини хеш-ключа: збільшення розрядності закономірно призводило до зростання обчислювальних витрат на формування хешу, у той же час збільшуючи розсіювання, що зменшувало затрати на пошук дескрипторів. Тому, виходячи з експерименту, для цього методу на даному наборі даних довжина ключа у 16 бітів забезпечує найбільшу ефективність. Довжина ключа у 32 біти програє йому у швидкодії через занадто великі обчислювальні затрати на хешування, довжина ключа 8 бітів через недостатню ступінь розсіювання Тому саме 16 бітів були обрані основною довжиною ключа при проведенні подальших експериментів. Метод підрахунку одиниць працював повільніше за 16-бітний SimHash але приблизно у 5–6 разів швидше за лінійний пошук, що пояснюється специфікою обчислення хеш-значення та структурою відповідної хеш-таблиці.

Експеримент із класифікацією еталонних зображень підтвердив коректність реалізації всіх досліджуваних методів та продемонстрував відсутність у них внутрішніх колізій або помилок у роботі з хеш-ключами.

3.3.3 Встановлення порогів надійності

У методах класифікації, таких як LSH чи бінарні центри класів, рішення про належність зображення до певного класу приймається без оцінки ступеня впевненості. На еталонних даних це не створює проблем, оскільки всі дескриптори вже присутні в хеш-таблицях або класи добре представлені центрами. Проте у реальних умовах, коли вхідне зображення може містити лише частину дескрипторів або містити ознаки, близькі до кількох класів, алгоритм може віднести його до правильного класу випадково або на основі недостатньої інформації.

Наприклад, у методах класифікації за допомогою LSH навіть якщо переможець отримує більшість голосів, ця більшість може складатися з невеликої частини дескрипторів, а решта не знайде відповідності у хеш-таблиці. У методі бінарних центрів класів аналогічна проблема проявляється у випадках, коли відстань до найближчого центру відносно мала, але все одно близька до інших класів. У таких ситуаціях класифікація формально вірна, проте не можна бути впевненим у її надійності.

Введення порогів для ключових метрик вирішує цю проблему. Вони дозволяють: по-перше, фільтрувати випадкові або недостатньо підкріплені рішення; по-друге, забезпечити статистично обґрунтовану перевагу переможця над іншими класами; по-третє, контролювати, що рішення формуються на основі репрезентативної частини дескрипторів.

Таким чином, введення порогів дозволяє перетворити чисто формальне визначення класу на статистично обґрунтоване і надійне рішення, підвищуючи стійкість алгоритмів до неповних даних, шумів та зовнішніх зображень, і створює основу для порівняння ефективності різних методів у складних умовах класифікації.

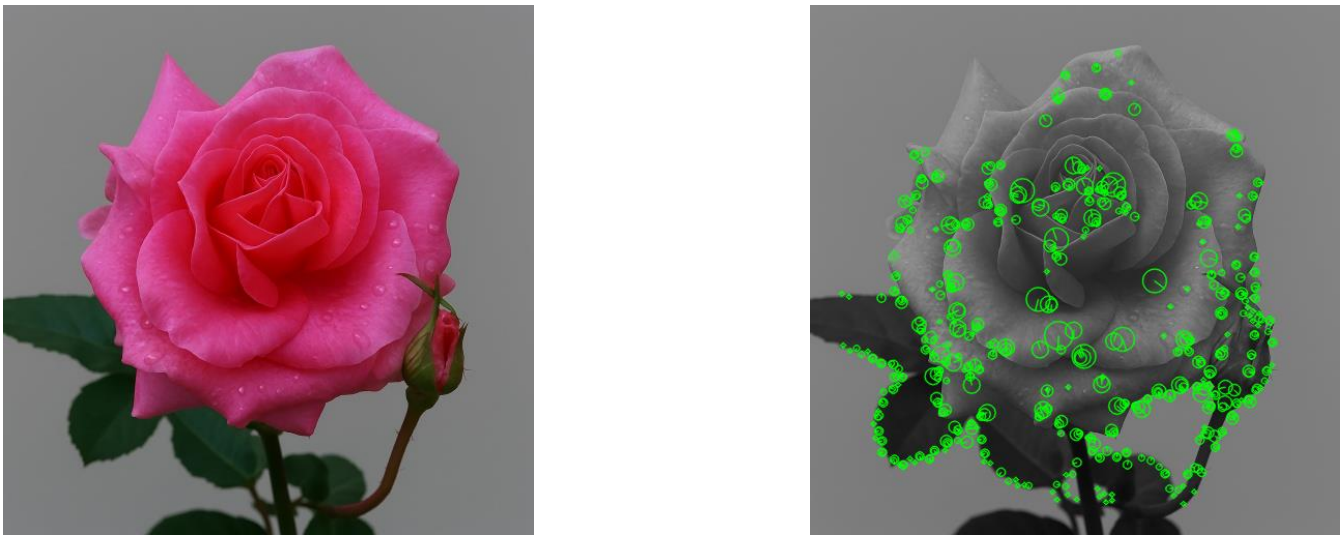

Рисунок 3.6 – Зображення поза базою еталонів

У методах класифікації для запобігання можливої неоднозначності введемо емпірично встановлений поріг у 0,75 для значення співвідношення

голосів при прийняті рішень. Це значення розраховується як відношення кількості голосів другого за величиною голосів класу до кількості голосів відданих за обраний клас. Тобто у нашому випадку коли це значення буде перевищувати 0,75, то рішення потрібно вважати ненадійним.

У методі бінарного центру встановимо аналогічний поріг у 0,75 для значення співвідношення відстаней, що розраховується як відношення відстані до обраного класу до відстані наступного ближчого за відстанню класу. Також для методу бінарних центрів введемо поріг на мінімальну відстань необхідну для того, щоб прийняте рішення вважалося прийнятим. Для цього візьмемо зображення рози, що знаходиться поза базою еталонів, та спробуємо класифікувати її методом бінарних центрів, після чого візьмемо мінімальну здобуту відстань, що дорівнює 79, та помножемо його на коефіцієнт запасу, що був емпірично встановлений як 0,8, отримуючи таким чином мінімальну необхідну відстань – 63.

3.3.4 Проведення експериментів з завадами

Наступним етапом експериментальних досліджень було поставлено завдання оцінити стійкість методів класифікації дескрипторів до типових спотворень, що виникають у реальних умовах формування зображень. Незважаючи на те, що попередній експеримент із використанням еталонних зображень продемонстрував 100% точність для всіх досліджуваних підходів, такі результати не дозволяють зробити висновки щодо їх практичної надійності. У реальних сценаріях дані рідко надходять у «ідеальному» вигляді: на зображення впливають шум, зміни масштабу через відстань до об’єкта, а також повороти, зумовлені довільною орієнтацією камери. Відповідно, цей етап є принципово необхідним для перевірки того, чи зберігається коректність класифікації за умов появи таких спотворень.

Основна мета експериментів полягала у визначенні того, наскільки різні методи класифікації здатні відтворювати належність дескрипторів до

відповідних класів за наявності контрольованих завад. Для цього були сформовані декілька груп трансформацій, що охоплюють найбільш репрезентативні види спотворень: додавання гаусівського шуму з різними рівнями інтенсивності, повороти зображень, а також зміни масштабу.

Першим з низки цих експериментів було проведено експеримент з додавання гаусівського шуму до еталонних зображень. Для репрезентативності результатів було вирішено провести цей експеримент з 3 різними рівнями інтенсивності шуму. На класифікацію було подано по 100 зображень кожного еталону з різним рівнем шуму (по 100 зображень з математичним очікуванням адитивного шуму, рівним нулю, та середньоквадратичним відхиленням сігма σ =26, 43,127 (рис.3.7)).

Введення шуму продемонструвало суттєві відмінності у поведінці досліджуваних методів, що не проявлялися на еталонних зображеннях, що можна побачити на таблицях 3.3 та 3.4. Уже за σ = 26 стає помітним розходження між підходами, хоча частина алгоритмів усе ще зберігає близьку до ідеальної точність. Точність методу лінійного пошуку протягом усього експерименту залишається на рівні 100% при σ = 26, σ = 43, та σ = 127, а співвідношення голосів залишається значно нижчим за критичний поріг. Однак, через великий час обчислення цей підхід виконує роль контрольного, а не практичного орієнтиру.

Рисунок 3.7 Зображення з різним рівнем шуму (σ = 26, σ =43, σ = 127)

Найбільш різкий спад у ефективності спостерігається у методу хешування за кількістю одиниць, точність якого вже при σ = 26 поступалася іншим методам, а при збільшенні σ від 26 до 127 зменшується до 56,33%.

Таблиця 3.3 Результати класифікації методу бінарних центрів на зображеннях з шумом

σ	Середній час (ms)	Точність (%)	Співвідношення відстаней	Середня Мінімальна відстань до центру
26	0,284	100,000	0,415	35,860
43	0,274	99,333	0,508	43,177
127	0,264	63,667	0,851	65,667

Таблиця 3.4 Результати класифікації на зображеннях з шумом

σ	Метод	Середній час (ms)	Точність (%)	Співвідношення голосів	Середня кількість порожніх кошиків
26	LinearSearch	306,060	100,000	0,056	0
26	Unitcount	19,383	92,667	0,762	8,060
26	SimHash	5,07	100	0,266	408,580
26	SimHash-8 bit	18,719	100,000	0,457	4,490
43	LinearSearch	315,019	100,000	0,099	0
43	Unitcount	19,796	86,667	0,793	8,267
43	SimHash	4,89	98,7	0,431	426,757
43	SimHash-8 bit	19,918	100,000	0,559	4,923
127	LinearSearch	308,188	100,000	0,450	0
127	Unitcount	20,580	56,333	0,867	9,463
127	SimHash	4,92	57,3	0,788	449,040
127	SimHash-8 bit	20,165	75,333	0,822	5,987

На відміну від Simhash-методу, падіння не супроводжується зростанням кількості порожніх кошиків, що залишаються на рівні 7–9 елементів. Це означає, що проблема полягає не у втраті доступу до даних, а у принциповій нестійкості ознаки: додавання шуму порушує бітову структуру дескриптора, зміщуючи кількісні характеристики, на яких ґрунтується класифікація. Високе значення співвідношення голосів (до 0,86), що в усіх випадках перевищує заданий поріг у

0,75 свідчить про те, що навіть у випадках правильної класифікації рішення є ненадійним і близьким до випадкового. Із цього можна зробити висновок, що метод хешування за кількістю одиниць є фактично непридатним для класифікації зашумлених зображень.

Метод SimHash із довжиною ключа 16 бітів на перших двох рівнях шуму демонструє близьку до еталонної поведінку (100% та 98,7%), проте при σ = 127 точність значно знижується до 57,3%. Характерною особливістю є значне зростання кількості порожніх кошиків — від 409 до 450 із 500 можливих, що означає, що більшість дескрипторів узагалі не можуть бути співставлені з існуючими хешами. Тобто прийняті рішення приймають участь приблизно 20% дескрипторів. Причина полягає у високій чутливості SimHash до випадкових бітових коливань: зміна навіть частини бітів призводить до переходу хешу в інший кошик, що різко зменшує кількість активних голосів. Це ставить питання про надійність прийняття рішень цим методом. З іншого боку низьке значення співвідношення голосів вказує на надійне прийняття рішення при σ = 26 та σ = 43, і навіть при σ = 127 воно не значно вище встановленого порогу у 0,75.

Цікавим є порівняння з SimHash із довжиною ключа 8 бітів, у якого ключ має вдвічі меншу розрядність. Незважаючи на загальне зниження дискримінативної здатності, при коротшому хеші зберігається висока точність методу, а при σ = 127 метод демонструє навіть більшу точність. Проте значення співвідношення голосів значно зросло порівняно з 16-бітним ключем та при σ = 127 досягає 0,822, тобто навіть формально правильні рішення будуть ненадійними. У той же час кількість порожніх кошиків зменшується майже у 70 разів порівняно з SimHash (приблизно 5–6 проти 400+), що забезпечує стабільнішу участь дескрипторів у голосуванні.

Метод бінарних центрів протягом експерименту демонструє високу точність при σ = 26 та σ = 43, але очікувано суттєво втрачає її при σ = 127. Збільшення середньої мінімальної відстані до центру до 65,667 перебільшує встановлений поріг надійності (63), що означає, що результати залишаються формально коректними, але перебувають у ненадійному стані. На відміну від

хеш-методів, зміна параметра σ впливає не на кількість використаних дескрипторів, а на ступінь розмиття відстаней між класами, що забезпечує передбачувану поведінку без різких провалів.

Усі методи, окрім хешування за кількістю одиниць, зберігають високу надійність та точність при σ = 26 та σ = 43, але при σ = 127 ці показники різко спадають.

Загалом швидкодія усіх методів зберігається на тому ж рівні, що й була під час експериментів на еталонних зображеннях.

Таким чином експерименти з адитивним гаусівським шумом вказують на те, що метод хешування за кількістю одиниць через високе значення співвідношення голосів виявився ненадійними для класифікації зашумлених зображень. У той же час Simhash метод виявився стійким до шуму, зберігаючи високу швидкодію та точність, хоча остання очікувано суттєво падає при σ = 127. Незважаючи на це, постає питання щодо надійності 16-бітного Simhash через низьку частку дескрипторів, що беруть участь у прийнятті рішення.

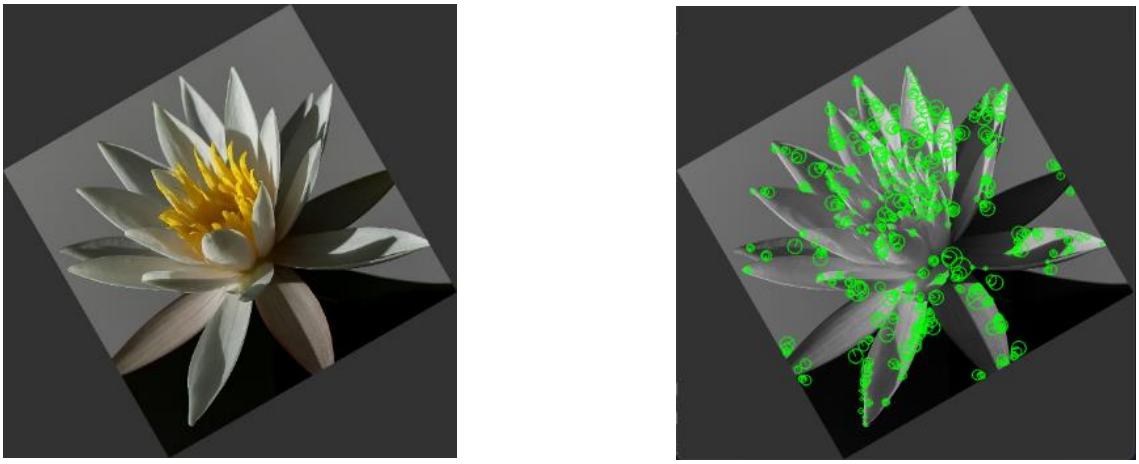

Рисунок 3.8 Приклад повернутого на 30 градусів зображення

Далі було вирішено провести експеримент з завадостійкості методів при поворотах еталонних зображень на 15, 30 та 45 градусів. Результати цих експериментів можна побачити на таблицях 3.5 та 3.6. На відміну від шумових спотворень, обертання зображення демонструє значно м’якший вплив на класифікацію більшості методів, що свідчить про вищу інваріантність ознак до

геометричних трансформацій порівняно з випадковими інтенсивностними змінами. Так практично усі методи показують 100-відстокову точність класифікації на усіх значеннях кута.

Таблиця 3.5 Результати класифікації методу бінарних центрів на зображеннях спотворених поворотами

Кут	**Середній час (ms)**	**Точність (%)**	**Співвідношення відстаней**	**Середня Мінімальна відстань до центру**
15	0,309	66,667	0,461	40,667
30	0,243	66,667	0,455	40,333
45	0,287	100,000	0,409	36,667

Таблиця 3.6 Результати класифікації на зображеннях спотворених поворотами

Кут	**Метод**	**Середній час (ms)**	**Точність (%)**	**Співвідношення голосів**	**Середня кількість порожніх кошиків**
1	2	3	4	5	6
15	LinearSearch	322,797	100	0,123	0
15	Unitcount	19,824	100	0,800	8,000
15	SimHash	4,751	100	0,691	454
15	SimHash-8 bit	20,641	100	0,710	8
30	LinearSearch	314,296	100	0,170	0
30	Unitcount	17,957	100	0,809	9,333
30	SimHash	4,641	100	0,583	438,667
30	SimHash-8 bit	19,757	100	0,629	7
45	LinearSearch	336,375	100	0,179	0

Продовження таблиці 3.6

1	2	3	4	5	6
45	Unitcount	18,852	100	0,831	11
45	SimHash	4,937	66,667	0,754	439,667
45	SimHash-8 bit	19,302	100	0,684	4,333

У той же час, незважаючи на високу точність, метод хешування за кількістю одиниць як і у попередньому експерименті має високе значення співвідношення голосів, через що його рішення не можуть вважатись надійними.

Бінарні центри значно втратили у точності порівняно з попереднім експериментом. При 15° та 30° цей показник дорівнює лише 66,67%. Хоча вони протягом усього експерименту зберігали непогані показники співвідношення відстаней та середньої мінімальної відстань до центру (табл. 3.5).

Стосовно методу Simhash, то він цілком зберігає закономірності встановлені у попередньому експерименті. Метод продовжує демонструвати високу точність, хоча й при 16-бітному хеші відбувається різке падіння точності при куті 45° до 66,67%. Співвідношення голосів 16-бітного та 8-бітного хешів залишаються меншими за заданий поріг, хоча у випадку 16-бітного розміру кількість порожніх кошиків перевищує 440–454, що означає втрату понад 90% дескрипторів під час голосування. 8-бітний хешовий простір, попри меншу дискримінативність, виявився добре пристосованим до обертання.

Загалом швидкодія усіх методів зберігається на тому ж рівні, що й була під час експериментів на еталонних зображеннях.

Таким чином повороти мають значно менший вплив на класифікацію, ніж завади інтенсивності, однак виявляють фундаментальні відмінності між досліджуваними методами. 8-бітний SimHash демонструє найкраще поєднання точності та надійності, тоді як 16-бітний SimHash хоч і значно виграє по швидкості (приблизно у 4 рази) але трохи програє по точності. Бінарні центри виявилися обмеженими за точністю, а метод хешування за кількості одиниць знов продемонстрував відсутність у прийнятих їм рішень надійності.

Завершальним експериментом стало масштабування зображень. Було проведено перевірку завадостійкості методів при зменшенні та збільшенні зображень у два рази. Результати цих експериментів можна побачити на таблицях 3.7 та 3.8.

Масштабування виявилося найбільш м’якою завадою для досліджувальних методів. Лише метод хешування за допомогою кількості одиниць не зміг досягти 100-відсоткової точності. Значення співвідношення голосів 8-бітного Simhash почало трохи перевищувати зазначений поріг. А метод бінарних центрів перевищив зазначену мінімальну відстань (69>63).

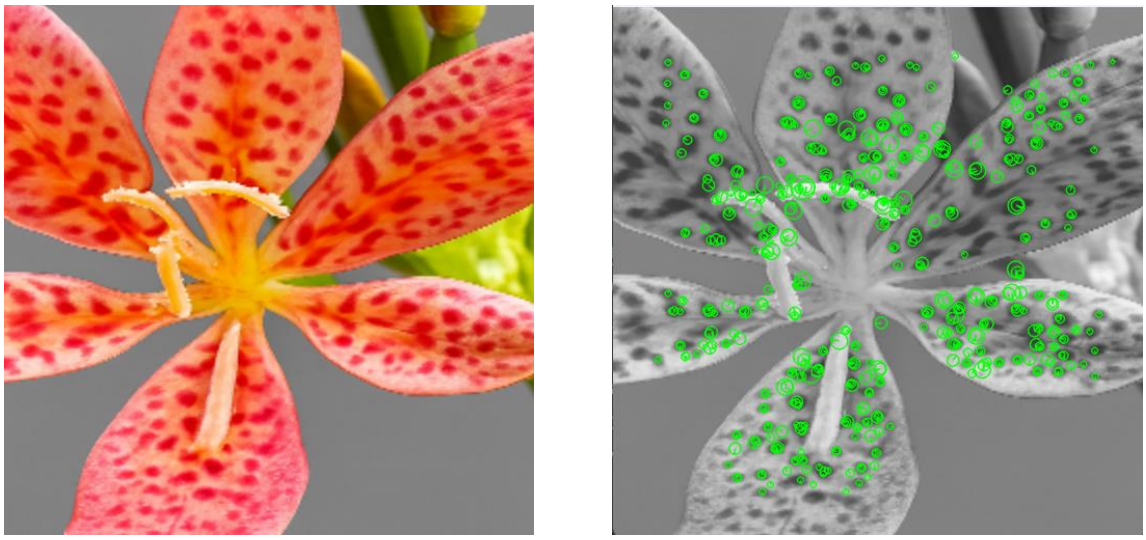

Рисунок 3.9 Приклад масштабованого зображення (масштаб 2)

Таблиця 3.7 Результати класифікації методу бінарних центрів на масштабованих зображеннях

Масштаб	**Середній час (ms)**	**Точність (%)**	**Співвідношення відстаней**	**Середня Мінімальна відстань до центру**
0,5	0,271	100	0,701	69
2	0,297	100	0,613	57,667

Проведені експерименти дозволили комплексно оцінити стійкість методів класифікації дескрипторів до основних типів спотворень, які виникають у реальних умовах формування зображень.

Таблиця 3.8 Результати класифікації на масштабованих зображеннях

Масштаб	**Метод**	**Середній час (ms)**	**Точність (%)**	**Співвідношення голосів**	**Середня кількість порожніх кошиків**
0,5	LinearSearch	334,1543	100	0,385	0
0,5	Unitcount	17,13324	66,667	0,790	17,667
0,5	SimHash	4,755	100	0,511	425,667
0,5	SimHash-8 bit	21,347	100	0,792	4,667
2	LinearSearch	350,466	100	0,466	0
2	Unitcount	19,755	100	0,789	9
2	SimHash	5,028	100	0,562	444
2	SimHash-8 bit	19,801	100	0,808	4,667

Лінійний пошук протягом усіх експериментів демонстрував еталонну поведінку, гарантуючи 100-відсоткову точність. Проте надзвичайно високі часові витрати роблять його непридатним для практичного використання, тому він розглядається лише як контрольний метод для оцінки якості.

Найбільш збалансованим за сукупністю показників – точністю, швидкодією та надійністю – виявився 8-бітний SimHash. Він зберіг високу точність при помірних рівнях шуму, продемонстрував стійкість до поворотів і практично не втрачає дескрипторів під час голосування, оскільки має дуже малу кількість порожніх кошиків, а його швидкодія майже у 15 разів вища за лінійний пошук. Менша розрядність ключа забезпечує меншу чутливість до випадкових

бітових змін, що робить цей метод добре пристосованим до реальних умов роботи. При сильних спотвореннях відзначається підвищення співвідношення голосів, однак у більшості випадків воно залишається прийнятним.

16-бітний SimHash показав досить високу точність та швидкожії (приблизно у 45 разів швидше за лінійний пошук). При слабких спотвореннях він забезпечує високу точність і низьке співвідношення голосів, що свідчить про надійність прийнятих рішень. Але «Ахіллесовою п’ятою» метода стало різке збільшення кількості порожніх кошиків – інколи понад 90% – унаслідок чого рішення приймаються на основі дуже малої кількості голосів.

Метод бінарних центрів показав ефективність при інтенсивнісних спотвореннях малого та середнього рівня. Утім, метод виявився дуже чутливим до поворотів зображень: уже при 15° або 30° точність зменшувалася до рівня близько 66%, що робить його малопридатним у сценаріях, де камера може мати довільну орієнтацію. Також при сильному шумі мінімальна відстань до центру перевищувала поріг надійності, що додатково знижувало стабільність прийнятих рішень.

Найгірше себе проявив метод хешування за кількістю одиниць. Він показав різке падіння точності вже при невеликому шумі, а при серйозних спотвореннях точність знижувалася до приблизно 56%. Високе значення співвідношення голосів, яке часом досягало 0,86, означає, що навіть формально правильні результати були ненадійними і близькими до випадкових. Повороти та масштабування не спричиняли значного провалу точності, однак надмірно високе співвідношення голосів унеможливлювало використання методу в реальних сценаріях. Загальною причиною слабкої поведінки є нестійкість самої ознаки: адитивний шум суттєво порушує кількість одиниць у бітовому рядку, на якій базується класифікація, що робить метод принципово вразливим.

Порівняння різних типів спотворень показало, що адитивний гаусівський шум є найскладнішим випробуванням для всіх методів класифікації. Повороти виявилися значно м’якшими за характером впливу, а масштабування – найменш критичною завадою. У підсумку можна зробити висновок, що 8-бітний SimHash

забезпечує оптимальний баланс між точністю, стійкістю до спотворень і продуктивністю, тоді як 16-бітний варіант, незважаючи на точність та швидкодію, потребує покращення механізму голосування через втрату великої кількості дескрипторів. Бінарні центри доцільно використовувати переважно у задачах із мінімальною кількістю геометричних трансформацій, а хешування за кількістю одиниць слід вважати непридатним для реальних сценаріїв через низьку стійкість та ненадійні рішення.

3.3.5 Модифікація методу Simhash

Задля вирішення проблеми 16-бітного Simhash, коли більшість дескрипторів не приймає участь у прийнятті рішення, було запропоновано можливі модифікації Simhash, що можуть підвищити покриття методу.

Першим таким методом став Multi-probe LSH, механізм роботи якого був детально розкритий у підрозділі 2.6. Якщо коротко цей метод пропонує окрім точного співпадіння ще й перевіряти кілька сусідніх корзин у межах однієї таблиці, що відрізняються від оригінального значення хеш-функції на певну встановлену кількість бітів. У рамках нашого дослідження експерименти, враховуючи велику довжину ключа, для збереження швидкодії метода було обрано провести експерименти з встановленої кількістю бітів 1 та 2, що вимагатиме перевірки 16 та 120 додаткових кошиків відповідно.

Іншим варіантом модифікації стала ідея зміни принципу пошуку кошиків під час класифікації, так якщо кошик знайдений за повним співпадінням виявляється порожнім, алгоритм може виконати пошук, знаходячи найближчий ключ за Хеммінговою відстанню – тобто, перевіряє ті кошики, хеш яких відрізняється від запитного лише на кілька бітів. Це дає змогу відновити пропущені збіги, якщо окремі біти хешу були змінені через шум, освітлення або незначні трансформації зображення. З іншого боку існує ризик падіння точності класифікації у випадках, коли відстань до найближчого хеш-ключа стає досить

великою. Щоб запобігти цієї проблеми було введено поріг на максимальну допустиму відстань Хемінга. У випадку з 16-бітним хеш-ключем було вирішено провести експерименти, коли значення максимально допустимої відстані дорівнює 2 та 4.

Проведемо на описаних методах попередні експерименти з класифікації еталонних зображень та з класифікації з завадами.

Таблиця 3.9 Порівняння роботи методів класифікації на еталонних зображеннях

Метод	**Час (мс)**	**Точність (%)**
SimHash	5,09	100
Multi-probe (Кількість бітів - 2)	37,93	100
Multi-probe (Кількість бітів - 1)	10,32	100
Hamming (відстань – 2)	38,51	100
Hamming (відстань – 4)	39,51	100

Як можна побачити на таблиці 3.9 ціною підвищення частки дескрипторів, що приймають участь у прийнятті рішень виявилося погіршення швидкодії. Хоча вона все ще досить висока та значно перевищує лінійний пошук, усі методи, окрім Multi-probe з кількістю бітів – 1, стали уступати 8-бітному Simhash майже у два рази.

Таблиця 3.10 Результати класифікації на зображеннях з шумом

σ	**Метод**	**Середній час (ms)**	**Точність (%)**	**Співвідношення голосів**	**Середня кількість порожніх кошиків**
1	2	3	4	5	6
26	SimHash	5,07	100	0,266	409,15
26	MultiProbe (Кількість бітів - 2)	36,74	100	0,249	10,36

Продовження таблиці 3.10

1	2	3	4	5	6
26	Multi-probe (Кількість бітів - 1)	10,30762	100	0,474131	174,7433
26	Hamming (відстань – 2)	35,65	100	0,502	10,35
26	Hamming (відстань – 4)	36,52015	100	0,652547	0
43	SimHash	4,89	98,7	0,431	426,79
43	MultiProbe (Кількість бітів - 2)	36,63	100	0,356	12,11
43	Multi-probe (Кількість бітів - 1)	10,34079	100	0,453562	173,6267
43	Hamming (відстань – 2)	35,78	100	0,62	11,94
43	Hamming (відстань – 4)	37,46289	100	0,616434	0
127	SimHash	4,92	57,3	0,788	450,43
127	MultiProbe (Кількість бітів - 2)	37,14	89	0,761	16,07
127	Multi-probe (Кількість бітів - 1)	10,4062	76,33333	0,820359	203,0967
127	Hamming (відстань – 2)	37,72	79,7	0,895	16,12
127	Hamming (відстань – 4)	39,36214	80,33333	0,892103	0

Таблиця 3.11 Результати класифікації на зображеннях спотворених поворотами

Кут	**Метод**	**Середній час (ms)**	**Точність (%)**	**Співвідношення голосів**	**Середня кількість порожніх кошиків**
1	2	3	4	5	6
15	SimHash	4,751100195	100	0,691470258	454
15	MultiProbe (Кількість бітів - 2)	37,5181666	100	0,564527403	18
15	Multi-probe (Кількість бітів - 1)	9,863899924	100	0,683144044	189
15	Hamming (відстань – 2)	37,95630013	100	0,839183608	18
15	Hamming (відстань – 4)	37,69103345	100	0,859845579	0
30	SimHash	4,641300067	100	0,582927225	438,6666667
30	MultiProbe (Кількість бітів - 2)	38,1125002	100	0,466562956	10,66666667
30	Multi-probe (Кількість бітів - 1)	9,672633304	100	0,622795284	177,6666667
30	Hamming (відстань – 2)	37,39020031	100	0,780140317	10,66666667
30	Hamming (відстань – 4)	36,81973363	100	0,765646228	0
45	SimHash	4,937200002	66,66666667	0,754336918	439,6666667
45	MultiProbe (Кількість бітів - 2)	37,07666682	100	0,497951606	15,33333333

Продовження таблиці 3.11

1	2	3	4	5	6
45	Multi-probe (Кількість бітів - 1)	10,21576696	100	0,726715888	186
45	Hamming (відстань – 2)	37,8915997	100	0,859268861	15,33333333
45	Hamming (відстань – 4)	38,46153369	100	0,808784849	0

Таблиця 3.12 Результати класифікації на масштабованих зображеннях

Масштаб	**Метод**	**Середній час (ms)**	**Точність (%)**	**Співвідношення голосів**	**Середня кількість порожніх кошиків**
0,5	SimHash	4,755	100	0,511969	425,6667
0,5	MultiProbe (Кількість бітів - 2)	37,4793	100	0,646748	10,33333
0,5	Multi-probe (Кількість бітів - 1)	10,0468	100	0,67222	178,6667
0,5	Hamming (відстань – 2)	39,8471	100	0,770692	10,33333
0,5	Hamming (відстань – 4)	37,0634	100	0,799436	0
2	SimHash	5,028733	100	0,562381	444
2	MultiProbe (Кількість бітів - 2)	37,76333	100	0,621048	15,66667
2	Multi-probe (Кількість бітів - 1)	9,618767	100	0,693821	191,6667
2	Hamming (відстань – 2)	37,23627	100	0,862848	15,66667
2	Hamming (відстань – 4)	39,22673	100	0,826979	0

Як бачимо з проведених експериментів запропоновані модифікації вирішують зазначену проблему, середня кількість порожніх кошиків значно впала і навіть у найгіршого за цим показником Multi-probe кількість дескрипторів що приймають рішення становлять більше 50%, що є достатнім для того, щоб вважати рішення надійним, у випадку з іншими методами цей показник становить більше 90%. Разом з цим можна побачити зростання точності класифікації при використанні цих методів, фактично тільки при експериментів з шумом при $\sigma = 127$ методи не дають стовідсоткову точність. Але навіть $\sigma = 127$ точність класифікації значно зросла порівняно з попередніми методами. Проте стоїть відмітити, що при експериментах з шумом при $\sigma = 127$, з поворотами та масштабуванням у метод знаходження кошиків через відстань хемінга значення співвідношення голосів перевищує зазначений поріг, тому

прийняті їх рішення не можна вважати надійними. У методів Multi-probe така проблема виникає тільки при експериментах з шумом при σ = 127 і перевищення там незначне.

Таким чином, запропоновані методи вирішують проблему 16-бітного Simhash, коли більшість дескрипторів не приймає участь у прийнятті рішення. Проте метод зі метод знаходження кошиків через відстань хемінга виявився ненадійним при роботі з завадами. У той же час метод Multi-probe продемонстрував високу точність, надійність та завадостійкість. А у випадку, коли значення кількості бітів дорівнює 1, метод майже не уступає по швидкодії класичному Simhash при цьому трохи, втрачаючи у точності. Тому використання Multi-probe з кількістю бітів 1 виглядає гарним компромісом між швидкістю, точністю та надійністю класифікації.

ВИСНОВКИ

Таким чином, у кваліфікаційній роботі досліджено методи класифікації зображень з використанням хешування структурного опису:

– проведено аналіз літературних джерел щодо апробації методів класифікації зображень, що дало можливість виявити сучасний стан дослідженої проблематики, недоліки та переваги аналогічних напрямків;

– проведено аналіз сучасних методів класифікації зображень, що дало можливість детально вивчити їх недоліки та переваги для подальшого вибору найкращих для вирішення поставленої задачі;

– сформовано покроковий алгоритм для кожного із вибраних методів класифікації, що дало можливість запрограмувати кожний із вибраних методів;

– візуалізовано покроковий алгоритми методів блок-схемою, що дозволило наочно зобразити кожний крок та, за можливості, оптимізувати або удосконалити окремі кроки;

У рамках кваліфікаційної роботи було проведено дослідження методів Random Projection LSH, Multi-probe LSH, LSH на основі кількості одиниць у хеш-коді, метод центрів та лінійний пошук для класифікації зображень квітів родини лілійних.

Побудовано покроковий алгоритм для кожного з методів та візуалізовано алгоритми за допомогою блок-схем.

Наукова новизна полягає у тому, що здобула подальший розвиток адаптація методів локально-чутливого хешування для задачі класифікації зображень за структурними дескрипторами.

Результати роботи апробовано у вигляді 2 тез доповідей під час IX Міжнародної студентської наукової конференції «РОЗВИТОК СУСПІЛЬСТВА ТА НАУКИ В УМОВАХ ЦИФРОВОЇ ТРАНСФОРМАЦІЇ» [31] та II Міжнародної науково-практичної студентської конференції «ІТ-ПРОСТІР СЬОГОДЕННЯ: ТЕНДЕНЦІЇ, ІННОВАЦІЇ ТА ПЕРСПЕКТИВИ РОЗВИТКУ» [32].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Szeliski, R. (2022). Computer vision: algorithms and applications. Springer Nature.

2. FORSYTH, David A.; PONCE, Jean. Computer vision: a modern approach. prentice hall professional technical reference, 2002.

3. Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press.

4. Гороховатський, В. О., & Гадецька, С. В. (2020). Статистичне оброблення та аналіз даних у структурних методах класифікації зображень.

5. Tuytelaars, T., & Mikolajczyk, K. (2008). Local invariant feature detectors: a survey. Foundations and trends® in computer graphics and vision, 3(3), 177-280.

6. Bay, H., Tuytelaars, T., & Van Gool, L. (2006, May). Surf: Speeded up robust features. In European conference on computer vision (pp. 404-417). Berlin, Heidelberg: Springer Berlin Heidelberg.

7. Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91-110.

8. Lowe, D. G. (1999, September). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision* (Vol. 2, pp. 1150-1157). Ieee.

9. Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010, September). Brief: Binary robust independent elementary features. In *European conference on computer vision* (pp. 778-792). Berlin, Heidelberg: Springer Berlin Heidelberg.

10. Leutenegger, S., Chli, M., & Siegwart, R. Y. (2011, November). BRISK: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision* (pp. 2548-2555). IEEE.

11. Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision* (pp. 2564-2571). IEEE.

12. Rosten, E., Porter, R., & Drummond, T. (2008). Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, *32*(1), 105-119.

13. Alcantarilla, P. F., & Solutions, T. (2011). Fast explicit diffusion for accelerated features in nonlinear scale spaces. IEEE Trans. Patt. Anal. Mach. Intell, 34(7), 1281-1298.

14. Nickel, M., Kalms, L., Häring, T., & Göhringer, D. (2022, July). High-performance AKAZE implementation including parametrizable and generic HLS modules. In 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP) (pp. 139-147). IEEE.

15. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21-27.

16. Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc.

17. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning.

18. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3-rd edition). *MIT Press and McGraw-Hill*.

19. Indyk, P., & Motwani, R. (1998, May). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 604-613).

20. Gionis, A., Indyk, P., & Motwani, R. (1999, September). Similarity search in high dimensions via hashing. In *Vldb* (Vol. 99, No. 6, pp. 518-529).

21. Andoni, A., & Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, *51*(1), 117-122.

22. Wang, J., Zhang, T., Sebe, N., & Shen, H. T. (2017). A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, *40*(4), 769-790.

23. Norouzi, M., & Blei, D. M. (2011). Minimal loss hashing for compact binary codes. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 353-360).

24. Charikar, M. S. (2002, May). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing* (pp. 380-388).

25.Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, *26*(189-206), 1.

26. Achlioptas, D. (2003). Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of computer and System Sciences*, *66*(4), 671-687.

27. Konoshima, M., & Noma, Y. (2012). Hyperplane arrangements and locality-sensitive hashing with lift. *arXiv preprint arXiv:1212.6110*.

28. Bingham, E., & Mannila, H. (2001, August). Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 245-250).

29. Rajaraman, A., & Ullman, J. D. (2011). *Mining of massive datasets*. Autoedicion.

30. Lv, Q., Josephson, W., Wang, Z., Charikar, M., & Li, K. (2007, September). Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases* (pp. 950-961).

31. Гема О. Г. Класифікація зображень з використанням хешування / О. Г. Гема, наук. керівник В. О. Гороховатський // ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку: зб. тез ІІ Міжн. науково-практичної студентської конференції, 15 жовтня 2025 р. – Харків : ХНУ ім. В.Н. Каразіна. – С. 204-208.

32. Гема О. Г. Оцінка завадостійкості методів класифікації зображень на основі хешування даних / О. Г. Гема, наук. керівник В. О. Гороховатський //

Розвиток суспільства та науки в умовах цифрової трансформації: матер. IX Міжнародної студентської наукової конференції, м. Тернопіль, 31 жовтня 2025 р. / ГО «Молодіжна наукова ліга». – Вінниця: ТОВ «УКРЛОГОС Груп».– С. 275-277.

33. Gorokhovatskyi V., Gadetska S., Stiahlyk N. (2020) Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2020), 2608, 1027-1039.

34. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, Computers, Materials & Continua, vol. 80, no. 2, 3085-3106.

35. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, International Journal of Academic and Applied Research, 7(11), pp. 134-145.

36. Gadetska S., Gorokhovatskyi V., Stiahlyk N., Vlasenko N. (2022) Aggregate Parametric Representation of Image Structural Description in Statistical Classification Methods. In CEUR Workshop Proceedings: Computer Modeling and Intelligent Systems (CMIS-2022), 3137, pp. 68-77.

37. Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, №2, 100–107.

38. Gorokhovatsky V.A. Efficient Estimation of Visual Object Relevance during Recognition through their Vector Descriptions. Telecommunications and Radio Engineering. – 2016, Vol. 75, No 14. – P. 1271–1283.

39. Gorokhovatskyi, V. A. (2003). Recognition of images in conditions of incomplete information. KNURE, Kharkov.

40. Gorokhovatskyi, V., Chmutov, Y., Tvoroshenko, I., & Kobylin, O. (2025). Reducing computational costs by compressing the structural description in image

classification methods. Advanced Information Systems, 9(1), 5–12. https://doi.org/10.20998/2522-9052.2025.1.01

41. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2025) Image description compression in classification structural methods, IEEE Access, vol. 13, pp. 43631-43641, doi: 10.1109/ACCESS.2025.3548910.

42. Gadetska, S.V., Gorokhovatskyi, V. O., Stiahlyk, N. I., Vlasenko, N.V. Statistical data analysis tools in image classification methods based on the description as a set of binary descriptors of key points. Radio Electronics, Computer Science, Control, 2021, №4, 58-68.

43. Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), Explainable Deep Learning Artificial Intelligence, pp. 125-146, Academic Press, 346 p.

44. Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. International Journal of Computing, 22(4), 485-492.

45. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, IEEE Access, 11, 126938-126949.

46. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., Hudáková M., and Gorokhovatskyi O. (2024) Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set, IEEE Access, vol. 12, 73376-73385.

47. Гороховатський В.О., Творошенко І.С. (2025) Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. Проблеми інформатики та моделювання (ПІМ-2025). Тези 25-ї міжнародної науково-технічної конференції (25 – 28 вересня 2025). Харків: НТУ «ХПІ», с. 38-43.

48. Кобилін О.А., Гороховатський В.О., Запорожченко А.П. (2025) Нейромережа Хемінга для класифікації зображень за множиною дескрипторів.

Проблеми інформатики та моделювання (ПІМ-2025). Тези 25-ї міжнародної науково-технічної конференції (25 – 28 вересня 2025). Харків: НТУ «ХПІ», с. 57-62.