Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ МЕТОДУ КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ ВАГОВИХ ХАРАКТЕРИСТИК ДЛЯ ДАНИХ ОПИСУ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-19-2
Санжаровський А.В.
(прізвище, ініціали)

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

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір В.П.
(посада, прізвище, ініціали)

Допускається до захисту

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

2020 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

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

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Санжаровському Антону Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методу класифікації зображень з використанням вагових характеристик для даних опису

затверджена наказом по університету від «23» жовтня 2020 року № 1428Ст.

2. Термін подання студентом роботи до екзаменаційної комісії 1 грудня 2020 р.

3. Вихідні дані до роботи *Класифікація, кластеризація, детектування особливих точок на зображенні, особливі точки, класифікація зображень за допомогою гістограм, класифікація даних, бінарний дескриптор, вагові коефіцієнти*

4. Перелік питань, що потрібно опрацювати в роботі
1. Постановка задачі класифікації
2. Обчислення бінарних дескрипторів ключових точок
3. Моделі оброблення особливих точок на зображенні
4. Методи класифікації зображень на основі бінарних дескрипторів
5. Програмна реалізація, дослідження результативності методів та аналіз результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Дескриптор ORB, ключові точки на зображеннях еталонів, гістограми побудованих кластерів, результати експериментів, аналіз таблиці експериментальних досліджень, висновки.*

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	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	23.10.20	виконано
2	Аналіз завдання, підбір літератури	24.10.20-25.10.20	виконано
3	Аналіз літератури з досліджуваної проблеми	26.10.20-30.10.20	виконано
4	Аналіз технічних засобів	31.10.20-02.11.20	виконано
5	Розробка методу дослідження	02.11.20-07.11.20	виконано
6	Програмна реалізація	08.11.20-12.11.20	виконано
7	Оформлення пояснювальної записки	12.11.20-20.11.20	виконано
8	Перевірка на плагіат	26.11.20	виконано
9	Рецензування	26.11.20	виконано
10	Підготовка презентації та доповіді	27.11.20	виконано
11	Занесення роботи в електронний архів	01.12.20	виконано
12	Попередній захист атестаційної роботи	09.12.20	виконано

Дата видачі завдання _23__ ___жовтня___2020 р.

Студент _____ _____
(підпис)

Керівник роботи _____ _____проф. Машталір В.П._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка атестаційної роботи: 94 с., 3 табл., 44 рис., 41 джерело.

КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, КЛЮЧОВІ ТОЧКИ, ЕТАЛОН, ДЕСКРИПТОР ORB, ВАГОВІ КОЕФІЦІЄНТИ, РЕЛЕВАНТНІСТЬ ОПИСІВ, КЛАСТЕРИЗАЦІЯ, ГІСТОГРАМА, АЛГОРИТМ *K*-СЕРЕДНІХ.

Метою атестаційної магістерської роботи є дослідження методу класифікації зображень на підставі множини бінарних дескрипторів ключових точок.

Об’єктом дослідження є методи класифікації зображень на основі результатів кластеризації бінарних дескрипторів.

Проведено кластеризацію множини бінарних дескрипторів для еталону, знайдено гістограму кластерів. Знайдено дескриптори (для тестових зображень) та їх проекції на кластери для еталону. Класифіковано зображення за допомогою порівняння гістограм.

У результаті роботи розроблено програмне забезпечення для класифікації зображень на основі кластеризації бінарних дескрипторів.

IMAGE CLASSIFICATION, KEY POINTS, STANDARD, ORB DESCRIPTOR, WEIGHTS, RELEVANCE OF DESCRIPTIONS, CLUSTERING, HISTOGRAM, *K*-MEANS ALGORITHM.

The purpose of the master's thesis is to study the method of image classification based on a set of binary descriptors of key points.

The object of research is methods of image classification based on the results of clustering of binary descriptors.

The binary descriptors for the etalon are clustered, the histogram of clusters is found. Descriptors (for test images) and their projections on clusters for the etalon were found. Images are classified by comparing histograms.

As a result, software for image classification based on clustering of binary descriptors was developed.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КТ – ключові точки зображення

SIFT – Scale-invariant feature transform

SURF – Speeded-Up Robust Features

FAST – функції з прискореного тестування сегментів

ORB – Oriented FAST and Rotated BRIEF

ВСТУП

Зараз комп’ютерний зір широко використовується у багатьох напрямах: в медицині, військовій сфері, виробництві, системах спостереження та контролю, тощо [1-5]. В першу чергу, основна задача комп’ютерного зору – це розпізнавання об’єктів, що передбачає і їх класифікацію. При розпізнаванні образів використовуються еталони зображень треба порівнювати з тестовими, але тут виникає основна проблема класифікації – як класифікувати, за якими ознаками, з якою точністю. Проблеми класифікації при аналізі зображень і сигналів вимагають, з алгоритмічної сторони, враховувати складну інформацію, вбудовану в дані. Зображення можуть містити багато тисяч піксельних значень у кількох кольорових каналах; їх кореляція та взаємозв’язок характеризують клас і дозволяють скласти критерії відокремлення від інших класів. Як правило, неможливо інтегрувати всю цю інформацію в розумний час для проблем класифікації. Тому зображення та сигнали виділяються як представники кожного об’єкта та його класу. Ці ознаки, будь то граничне представлення, як дескриптори Фур’є, кути Гарріса або гауссові піки, утворюють представлення об’єкта нижчої розмірності і потрапляють в характерну область в просторі ознак, інколи, досить диференційовані від об’єктів інших класів, але подібні до об’єктів того ж класу [6-9]. Але буває, що ознаки для двох класів дуже схожі, і через це погіршується точність. У атестаційній роботі магістра буде розглянуто та проаналізовано ці проблеми.

1 ОГЛЯД МЕТОДІВ ПОБУДОВИ МНОЖИНИ КЛЮЧОВИХ ТОЧОК ТА ДЕСКРИПТОРІВ ЗОБРАЖЕННЯ

1.1 Дескриптори зображень

Ознаки зображення – це точки на зображенні, які суттєво відрізняються від інших. Результати функції повинні бути незмінними при застосуванні таких перетворень зображень, як обертання, переклад та масштабування. У контексті класифікації ознаки зразка об’єкта (зображення) не повинні змінюватися при обертанні зображення, зміні масштабу (рівнозначно зміні роздільної здатності або збільшення) або зміні кута отримання. Однак ці незмінності не безмежні. Крім того, функції повинні бути нечутливими до умов освітлення та кольору (якщо це не вимагається спеціально). Частіше за все, коли говориться про ознаки зображення, то мова йде про особливі (контрольні) точки на зображенні [2].

Ці точки мають наступні особливості та властивості [2, 3]:

– відмінність (distinctness) – особлива точка повинна бути відмінною в своїй околиці;

– інваріантність (invariance) – незалежність до афінних перетворень змін яскравості;

– стабільність (stability) – стійкість до шумів;

– унікальність (uniqueness) – унікальність серед повторюваних паттернів;

– інтерпретованість (interpretability) – виявлення інформації для аналізу;

– локальність (locality) – займати невелику область зображення;

– кількість (quantity) – оптимальна кількість точок в залежності від предмета їх використання;

– ефективність (efficiency) – час виявлення.

Для визначення контрольних точок застосовуються наступні підходи [3]:

– на основі інтенсивності точок зображення;

– використання контурів зображення;

– використання моделі-шаблону.

Ключові точки визначаються за допомогою [3]:

– детектор (feature detector) – здійснює пошук точок;

– дескриптор (descriptor) – описує знайдені точки;

– матчер (matcher) – встановлює відповідність між точками.

Добре відомі приклади особливостей зображення включають кути, SIFT, SURF, точки, краї. Не всі вони є незмінними і нечутливими при вищезгаданних перетвореннях. Однак, залежно від завдання на класифікацію та очікуваної геометрії об’єктів, ознаки можна розумно підбирати. Наприклад, на платі друкованої плати, яка втілює чітко визначені геометричні фігури, кутові особливості можуть бути гарною відправною точкою. Кути Харріса, обчислюють двовимірне власне значення гессія зображення (часткові похідні, розраховані, що згортають зображення з гауссовим ядром), і відповідно до їх значення визначають метрику «кутовості».

Дескриптори точок об’єктів супроводжують вилучення об’єктів і використовуються для порівняння між об’єктами, виділеними з різних зображень. Крім того, такі дескриптори, як SURF, дозволяють віднести об’єкт до певного класу на основі міри подібності. Вивчення розподілу значень ознак певного класу (типу друкованої плати, об’єктів, таких як собаки, коти, особи тощо) природно веде до класифікації за допомогою методологій машинного навчання. Застосування таких алгоритмів для класифікації нового зразка відбувається після етапу навчання, коли взяті ознаки еталонів, а ознаки тестових зображень витягуються та вводяться в класифікатор для кожного нового зображення.

Характеристика об’єктів за допомогою комбінації точок об’єктів та пов’язаних з ними дескрипторів також є звичайною практикою. Оскільки такі ознаки, як піки та краї, фокусуються на одному аспекті об’єкта, то SURF та бінарні дескриптори проливають світло на інші аспекти, нещодавно досліджене зображення, на якому об’єкт може виглядати у довільному положенні, повинно природно характеризуватися будь-якими можливими способами. Тоді розробники алгоритмів несуть відповідальність за осмислення значення і типу вилучених ознак та їх взаємозв’язку з метою адаптації процесу класифікації.

Окрім класифікації, дескриптори використовуються для зіставлення об’єктів [4]. Завдяки зменшенню розміру зображення або сигналу за кількома множинами особливих точок можна швидше порівнювати об’єкти. Порівняння на основі ознак знаходить своє застосування, наприклад у пошуку порушень авторських прав на зображення в мережі.

Останні кілька десятиліть ознаки об’єкта, знайшли постійне місце в наборі інструментів комп’ютерного зору. Однак методології не завжди працюють ефективно без етапів попередньої та подальшої обробки. У багатьох ознаках, якщо не у всіх, параметри потрібно налаштовувати, щоб дозволити розумний вибір ознак на основі їх індукованих метрик та дескрипторів. Будь то для цілей класифікації, отримання зображень, характеристики чи порівняння, вилучення ознак – це лише одна частина рішення проблеми: ефективне цілісне рішення потребує роботи експерта з комп’ютерного зору [1].

Для отримання дескрипторів найчастіше застосовують такі детектори (алгоритми):

- кути Гарріса;
- SIFT (Scale-invariant feature transform);
- SURF (Speeded-Up Robust Features);
- BRISK;
- BRIEF;

– FAST;

– ORB.

Проаналізуємо суть деяких із них.

1.2 Кути Гаррріса

Harris Corner Detector – це оператор виявлення кутів, який зазвичай використовується в алгоритмах комп'ютерного зору для вилучення кутів та виведення особливостей зображення. Вперше він був представлений Крісом Гаррісом та Майком Стівенсом у 1988 році після вдосконалення детектора кутів Моравека. Кутовий детектор Харріса враховує диференціал кутової оцінки з урахуванням напряму безпосередньо, замість того, щоб використовувати зсувні плями для кожних кутів 45 градусів, і, як було доведено, є більш точним у розрізненні країв та кути. З тих пір він був вдосконалений і прийнятий у багатьох алгоритмах для попередньої обробки зображень для наступних додатків [5].

Кут – це точка, місцева околиця якої знаходиться у двох домінуючих та різних напрямках краю. Іншими словами, кут можна інтерпретувати як стик двох країв, де край – це раптова зміна яскравості зображення. Кути – це важливі особливості зображення, і їх, як правило, називають точками інтересу, які незмінні щодо перекладу, обертання та освітлення [5].

Тепер треба розібратися, чому кути вважаються кращими характеристиками чи хорошими для картографування латок. На малюнку, якщо взяти рівну область, то зміни градієнта не спостерігається в будь-якому напрямку. Подібним чином в області ребра не спостерігається зміни градієнта вздовж напрямку ребра. Отже, як плоска область, так і область краю не дуже відрізняються, тому вони погано підходять для збігу патчів (у довжині краю в краю області є багато подібних плям). В кутовій області градієнта в будь-якому напрямку значно змінюється. Завдяки цьому ці кути

вважаються хорошими для збігу патчів (зсув вікна в будь-якому напрямку призводить до значних змін зовнішнього вигляду) і, як правило, більш стабільними при зміні точки зору [5].

Ідея виявлення кутів полягає в тому, щоб розглянути невелике віконце навколо кожного пікселя *p* на зображенні. Потрібно визначити всі такі унікальні вікна пікселів. Унікальність можна виміряти, зрушивши кожне вікно на невелику величину в заданому напрямку та вимірявши величину зміни, яка відбувається у значеннях пікселів. Більш формально, приймається різниця квадратів значень (SSD) значень пікселів до і після зсуву та ідентифікуються вікна пікселів, де SSD є великим для зсувів у всіх 8 напрямках. Тепер треба визначити функцію зміни *E(u, v)* як суму всіх сумарних квадратних різниць (SSD), де *u, v* – координати *x, y* кожного пікселя у нашому вікні 3×3, а *I* – значення інтенсивності пікселя. Характеристиками зображення є всі пікселі, які мають великі значення *E(u, v)*, як визначено деяким порогом [5]:

$$E(u,v)=\sum_{x,y} w(x,y)[I(x+u,y+v)-I(x,y)]^2. \qquad (1.1)$$

Далі потрібно максимізувати цю функцію *E(u, v)* для виявлення кутів. Застосовуючи розширення Тейлора до наведеного рівняння та використовуючи деякі математичні кроки, можна отримати остаточне рівняння як [5]:

$$E(u,v)\approx [u\,v]\left(\sum\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}\right)\begin{bmatrix} u \\ v \end{bmatrix}. \qquad (1.2)$$

Тепер перейменовується підсумована матриця і ставиться як *M*:

$$M = \sum w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}. \quad (1.3)$$

Отже, рівняння тепер стає:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}. \quad (1.4)$$

Потрібно, щоб SSD була суттєвою для всіх восьми напрямків, або навпаки, щоб не була малою для жодного з напрямків. Вирішуючи власні вектори *M*, можна отримати значення як найбільшого, так і найменшого збільшення SSD. Відповідні власні значення дають фактичну величину цих збільшення. Для кожного вікна обчислюється бал, *R* [5]:

$$R = \det M - k(trace\, M)^2, \quad (1.5)$$

$$\det M = \lambda_1 \lambda_2, \quad (1.6)$$

$$traceM = \lambda_1 + \lambda_2. \quad (1.7)$$

λ_1 і λ_2 є власними значеннями *M*. Отже, значення цих власних значень вирішують, регіон буде кутом, краєм чи плоскою:

– коли |*R*| малий, що трапляється, коли λ_1 і λ_2 малі, область плоска;

– коли $R < 0$, що трапляється, коли $\lambda_1 >> \lambda_2$ або навпаки, область є ребром;

– коли *R* велике, що трапляється, коли λ_1 і λ_2 великі і $\lambda_1 \sim \lambda_2$, область є кутом.

Високорівневий псевдокод (алгоритм) [5]:

– конвертуйте початкове зображення у відтінки сірого;

– застосуйте фільтр Гауса для усунення шумів;

– застосуйте оператор Sobel, щоб знайти значення градієнта *x* та *y* для кожного пікселя на зображенні у відтінках сірого;

– для кожного пікселя *p* на розгляньте вікно 3×3 навколо нього та обчисліть функцію сили кута, назвіть це значенням Харріса;

– знайдіть усі пікселі, які перевищують певний поріг і є локальними максимумами в межах певного вікна (для запобігання дублювання ознак);

– для кожного пікселя, знайденого на попередньому кроці, обчисліть дескриптор об’єкта.

1.3 SIFT

SIFT (Scale-invariant feature transform – масштабно-інваріантне перетворення ознак), яке має властивість незмінності масштабу, що робить його кращим за кути Гарріса. Детектор Гарріса не є інваріантом масштабу, кут може стати краєм, якщо масштаб зміниться [6].

Невід’ємною властивістю об’єктів у світі є те, що вони існують лише як значущі сутності в певних діапазонах масштабу. Простим прикладом є гілка дерева, яку є сенс розглядати лише в масштабі від, скажімо, кількох сантиметрів до максимум кількох метрів. Безглуздо розглядати гілку дерева на нанометрі чи кілометрі. У таких масштабах доречніше говорити про молекули, що утворюють листя дерева, або ліс, в якому дерево росте. Подібним чином, важливо лише говорити про хмару в певному діапазоні грубих масштабів. У більш точних масштабах доцільніше розглядати окремі краплі, які, в свою чергу, складаються з молекул води, які складаються з атомів, які складаються з протонів та електронів тощо [6].

Переваги SIFT [7]:

– ознаки є локальними, вони надійні до оклюзії та шуму (без попередньої сегментації);

– відмінність: окремі особливості можна підібрати до великої бази даних об’єктів;

– кількість: багато ознак можна створити навіть для невеликих об’єктів;

– ефективність: близька до продуктивності в режимі реального часу;

– розширюваність: може бути легко розширена до широкого спектру різних типів функцій, з додаванням кожної надійності.

SIFT – досить задіяний алгоритм. В основному в алгоритмі SIFT беруть участь наступні етапи [7]:

– вибір піку в масштабі простору (потенційне місце для пошуку об’єктів);

– локалізація ключових точок (точне визначення ключових точок об’єкта);

– призначення орієнтації (призначення орієнтації ключовим точкам);

– дескриптор ключових точок (опис ключових точок як великого розмірного вектору);

– підбір ключових точок.

Далі буде детальніше розглянуто кожен етап.

Вибір піку в масштабі простору. Об’єкти реального світу є сенс розглядати лише в певному масштабі. Цей багатомасштабний характер об’єктів досить часто зустрічається в природі. І масштабний простір намагається відтворити цю концепцію на цифрових зображеннях [7].

Масштабний простір зображення (рис. 1.1) – це функція $L(x, y, \sigma)$, яка отримується в результаті згортки ядра Гауса (Розмиття) у різних масштабах із вхідним зображенням. Простір масштабу розділений на октави, а кількість октав і масштаб залежить від розміру вихідного зображення. Отже, генерується кілька октав вихідного зображення. Розмір зображення кожної октави вдвічі менший від попереднього [7].

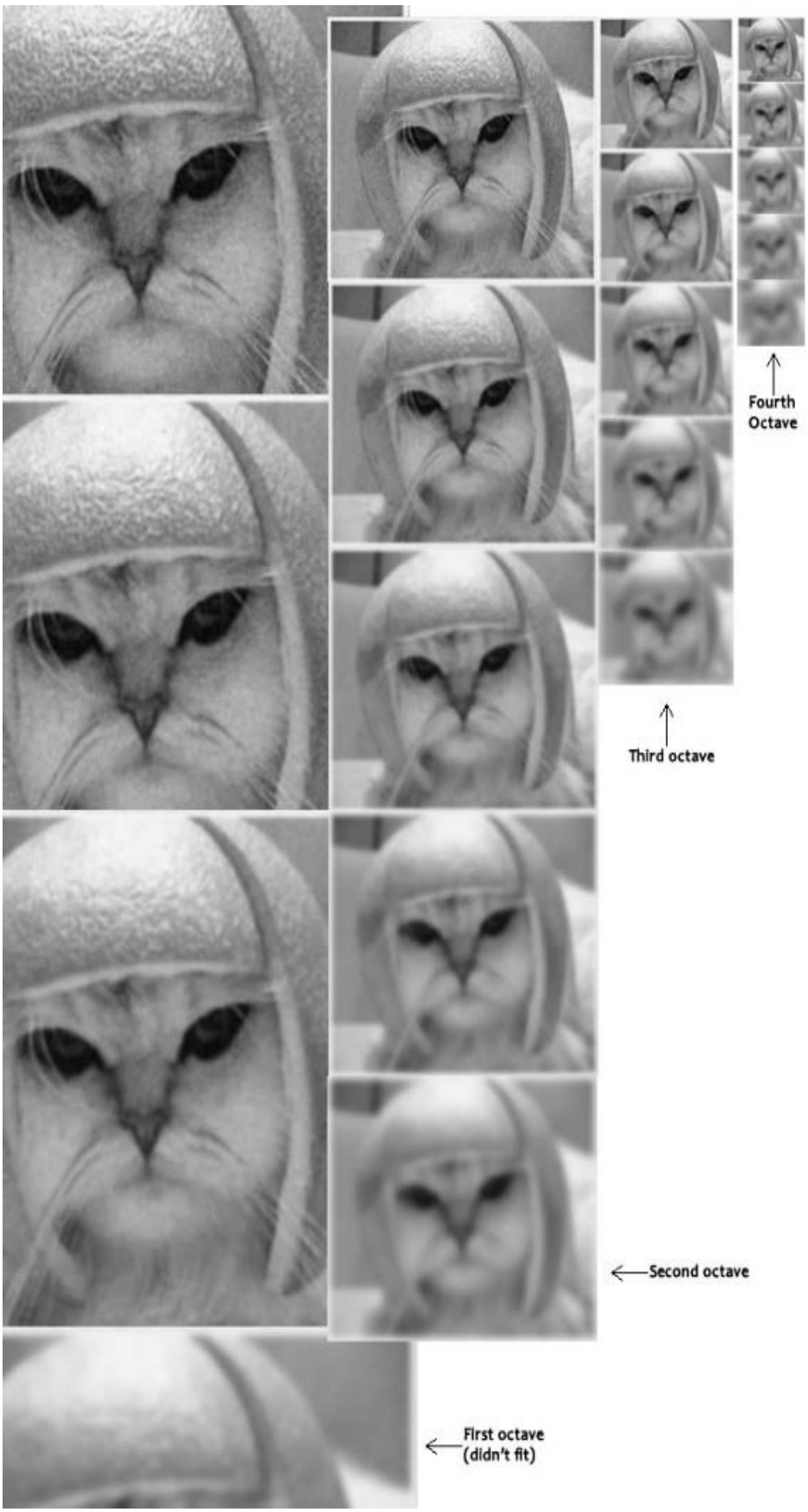


Рисунок 1.1 – Приклад масштабного простору зображення

В межах октави зображення поступово розмиваються за допомогою оператора розмиття Гауса. Математично «розмиття» називається згорткою гауссового оператора та зображення. Гаусова розмитість має певний вираз або «оператор», який застосовується до кожного пікселя. Результатом є розмитість зображення [7].

$$L(x, y, \sigma) = G(x, y, \sigma) + I(x, y), \tag{1.8}$$

де G – оператор розмиття за Гаусом;

I – зображення;

x, y – координати місцезнаходження;

σ – параметр «масштабу».

Подумайте про це як про кількість розмиття. Чим більше значення, тим більше розмиття. Нижче наведено оператор розмиття [7]:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{1.9}$$

Далі ці розмиті зображення використовуються для створення іншого набору зображень, «Відмінність Гауса» (DoG). Ці зображення DoG (рис. 1.2) чудово підходять для пошуку особливих точок на зображенні. Різниця Гаусса отримується як різниця розмиття Гауса зображення з двома різними σ, нехай це буде σ і $k\sigma$. Цей процес виконується для різних октав зображення в піраміді Гауса. Він представлений на зображенні нижче [7]:

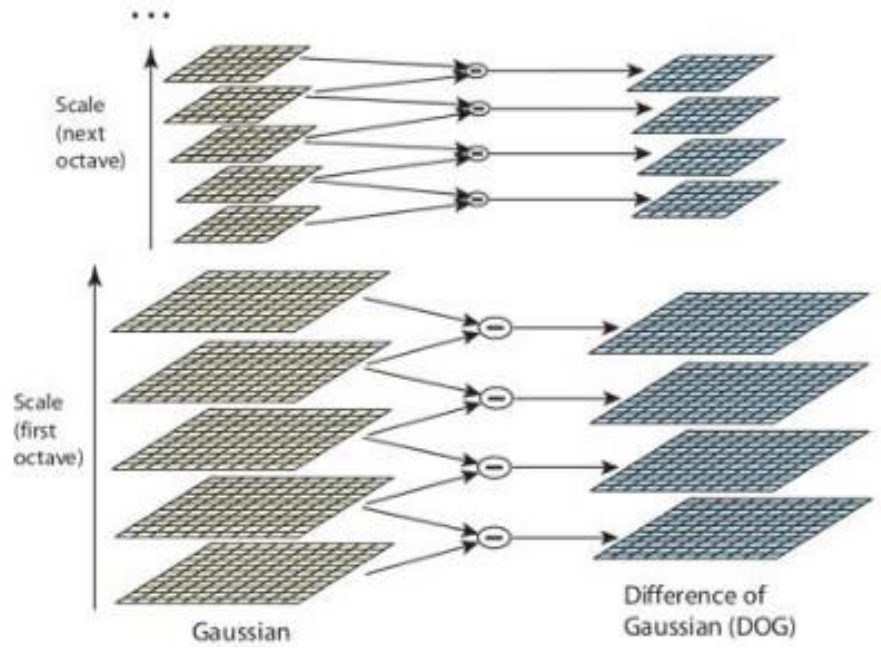


Рисунок 1.2 – Приклад DoG

Далі виконується пошук ключових точок (рис. 1.3) на основі згенерованого масштабного простору та обчисленої різниці Гаусса. Потім вони використовуються для обчислення лапласіанських наближень Гауса, які є інваріантними щодо масштабу [7].

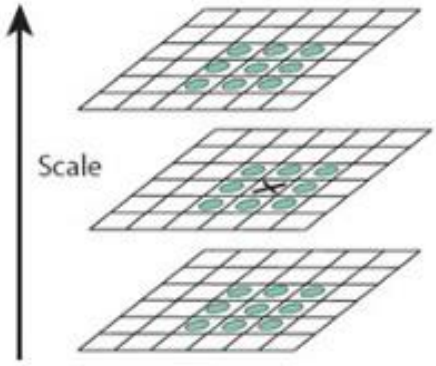


Рисунок 1.3 – Ілюстрація пошуку контрольних точок

Кожен піксель зображення порівнюється з 8 сусідніми, потім порівнюється з 9 пікселями в попередньому та наступному масштабах.

Зазвичай проводиться 26 таких перевірок. Якщо це локальний екстремум, це потенційний ключовий момент. Це в основному означає, що ключові точки найкраще представлені в цій шкалі [7].

Локалізація ключових точок. Деякі з ключових точок лежать уздовж краю, або їм недостатньо контрасту. Такі точки не мають користі у якості ознак і від них треба позбутися. Для цього застосовується підхід, подібний до підходу в кутовому детекторі Харріса для видалення крайових елементів. Для цього перевіряється інтенсивність ознак [7].

Вони використовували розширення масштабного простору серії Тейлора, щоб отримати більш точне розташування екстремумів, і якщо інтенсивність у цьому екстремумі менше порогового значення (0,03 відповідно до паперу), воно відхиляється. DoG має вищу реакцію на краї, тому краї також потрібно видалити. Вони використовували матрицю Гессія 2×2 (*H*) для обчислення головної кривизни (рис. 1.4) [7].

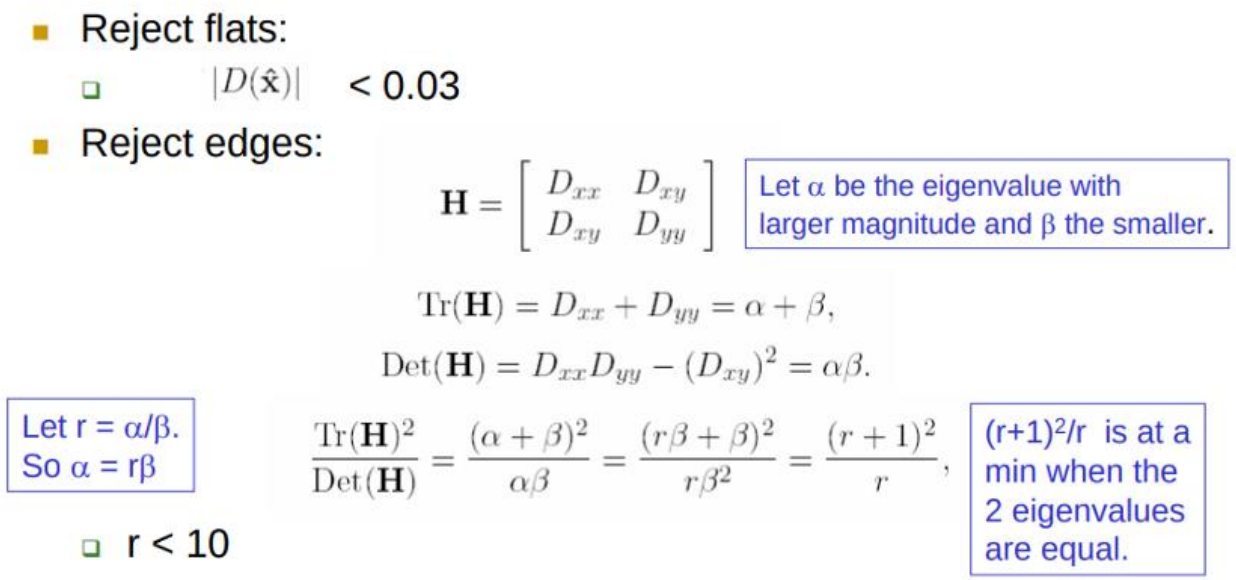


Рисунок 1.4 – Формули для обчислення головної кривизни

Призначення орієнтації. Зараз є законні ключові точки (рис. 1.5), які перевірені на стабільність. Вже відомо масштаб виявлення цих точок (це те

саме, що масштаб розмитого зображення). Іншими словами, масштаб – незмінний. Тепер потрібно призначити орієнтацію кожній ключовій точці, щоб зробити її інваріантністю обертання [7].

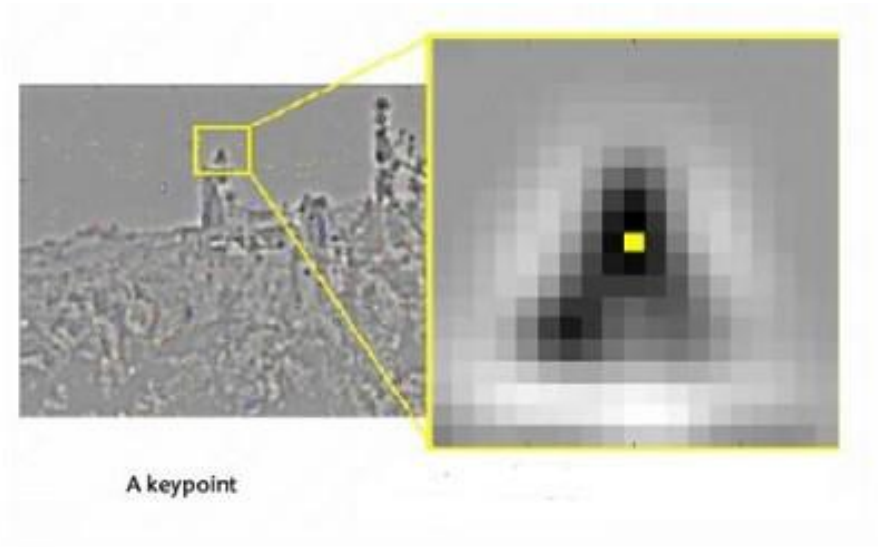


Рисунок 1.5 – Приклад контрольної точки

Навколо місця розташування ключової точки розглядається околиця в залежності від масштабу, а в ній обчислюються величина та напрямок градієнта. Створюється орієнтаційна гістограма (рис. 1.6) з 36 стовпцями, що охоплюють 360 градусів. Напрямок градієнта в певній точці (в «зоні збору орієнтації») становить 18,759 градусів, тоді він потрапить у смітник 10 – 19 градусів. І «сума», яка додається до смітника, пропорційна величині градієнта в цій точці. Після того, як це буде зроблено для всіх пікселів навколо ключової точки, гістограма в певний момент матиме пік [7].

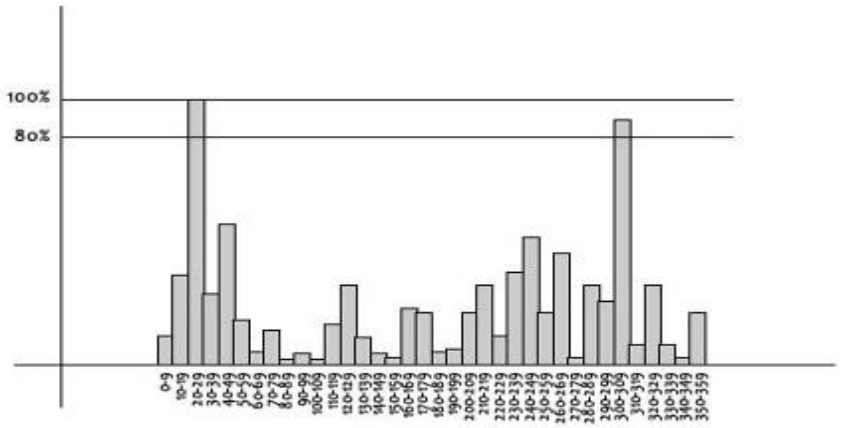


Рисунок 1.6 – Гістограма орієнтацій

Далі на гістограмі беруться найвищий пік та піки, що перевищують 80% від нього, що потім враховується для обчислення орієнтації. Ці піки створюють ключові точки з однаковим розташуванням і масштабом, але в різних напрямках, що забезпечує стабільность відповідності [7].

Кожна ключова точка має місце розташування, масштаб, орієнтацію. Далі потрібно обчислити дескриптор для локальної області зображення щодо кожної ключової точки, яка є надзвичайно відмітною та нечутливою до таких змін, як зміна точки зору та освітленості. Для цього береться вікно 16×16 і розділяється на 16 підблоків розміром 4×4 (рис. 1.7) [7].

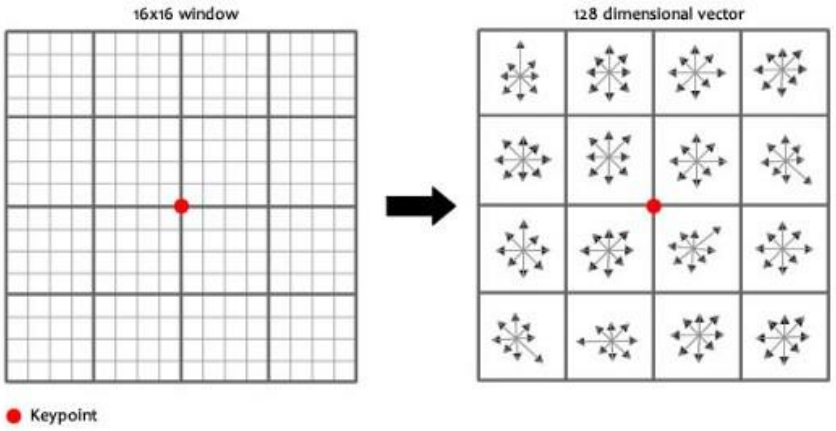


Рисунок 1.7 – Вікно навколо контрольної точки та підблоки

Для кожного підблоку створюється гістограма орієнтації на 8 частин.

Отже, на практиці було використано 4×4 дескриптори понад 16×16 вибіркового масиву. Напрямки 4×4×8 дають значення 128 частин. Він представлений у вигляді вектора ознак для формування дескриптора ключових точок. Цей вектор ознак вносить кілька ускладнень [7].

Вектор ознак використовує градієнтні орієнтації. Очевидно, що якщо ви обертаєте зображення, все змінюється. Всі орієнтації градієнта також змінюються. Для досягнення незалежності від обертання обертання ключової точки віднімається від кожної орієнтації. Таким чином, кожна орієнтація градієнта є відносно орієнтації ключової точки.

Залежність від освітленості. Якщо перевищуються великі цифри, можна досягти незалежності від освітленості. Отже, будь-яке число (із 128) більше 0,2 змінюється на 0,2. Цей результуючий вектор ознак знову нормалізується. І тепер у вас є незалежний від освітлення векторний елемент.

Ключові точки між двома зображеннями відповідають ідентифікації найближчих сусідів. Але в деяких випадках другий найближчий матч може бути дуже близьким до першого. Це може статися через шум або з інших причин. У цьому випадку приймається відношення найближчої відстані до другої найближчої відстані. Якщо воно більше 0,8, вони відхиляються. Він усуває близько 90% помилкових збігів, тоді як відкидає лише 5% правильних збігів [8].

1.4 SURF

Метод SURF (Speeded-Up Robust Features – прискорені надійні функції) – це швидкий та надійний алгоритм локального подання та порівняння зображень, що інваріантно подібні. Основний інтерес підходу SURF полягає в його швидкому обчисленні операторів з використанням полевих фільтрів, таким чином надаючи можливість реального часу додатків, таких як

відстеження та розпізнавання об’єктів. Структура SURF, описана в цій роботі, базується на дисертації Х. Бея [ETH Zurich, 2009 р.], а конкретніше на роботі, написаній у співавторстві Х. Бей, А. Ессом, Т. Туйтелаарсом та Л. Ван Гоолом [9].

SURF складається з двох етапів:

– вилучення функцій;

– опис функції.

Вилучення функцій. Підхід для виявлення точок інтересу використовує дуже базове наближення матриці Гесса [9, 10].

Цілісні зображення. Інтегральне зображення або таблиця із підсумковою областю було представлено в 1984 році. Інтегральне зображення використовується як швидкий та ефективний спосіб обчислення суми значень (піксельних значень) на зображенні – або прямокутної підмножини сітки. Вjуj також використовується для обчислення середньої інтенсивності в межах даного зображення [9].

Ці суми значень дозволяють швидко обчислювати згорткові фільтри коробчатого типу. Запис інтегрального зображення $I_{\sum}(x)$ у місці $x = (x, y)^T$ являє собою суму всіх пікселів у вхідному зображенні I у межах прямокутної області, утвореної початком та x [9].

$$I_{\sum}(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j). \tag{1.10}$$

При обчисленні I_{Σ} потрібно лише чотири складання для обчислення суми інтенсивностей на будь-якій прямокутній прямокутній площині, незалежно від її розміру [9].

Surf використовує гессіанську матрицю через швидке обчислення та точність. Замість того, щоб використовувати інший вимір для вибору місця та масштабу (детектор Гесія-Лапласа), SURF спирається на детермінанту

матриці Гесса для обох. Враховуючи піксель, гессіан цього пікселя виглядає приблизно так [9]:

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}. \quad (1.11)$$

Для адаптації до будь-якого масштабу було відфільтровано зображення за гауссовим ядром, тому з урахуванням точки $X = (x, y)$ матриця Гесса $H (x, \sigma)$ у x у масштабі σ визначається як [9]:

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) L_{yy}(x,\sigma) \end{bmatrix}, \quad (1.12)$$

де $L_{xx} (x, \sigma)$ – згортка похідної Гауса другого порядку із зображенням I у точці x, і аналогічно для $L_{xy} (x, \sigma)$ та $L_{yy} (x, \sigma)$.

Гаус є оптимальними для аналізу масштабного простору, але на практиці їх потрібно дискретизувати та обрізати. Це призводить до втрати повторюваності при обертанні зображення навколо непарних кратних $\pi / 4$. Ця слабкість стосується детекторів на основі Гессія загалом. Тим не менше, детектори все ще працюють добре, і незначне зниження продуктивності не переважує переваги швидких звивин, спричинених дискретизацією та обрізанням [9].

Спочатку для обчислення детермінанту матриці Гесса потрібно застосувати згортку з ядром Гауса, а потім похідну другого порядку. Після успіху з наближеннями LoG (SIFT), SURF ще більше підштовхує апроксимацію (як згортку, так і похідну другого порядку) за допомогою коробчастих фільтрів. Ці приблизні похідні Гауса другого порядку можуть

бути оцінені за дуже низьких обчислювальних витрат за допомогою інтегральних зображень і незалежно від розміру, і це є частиною причини, чому SURF є швидким [9].

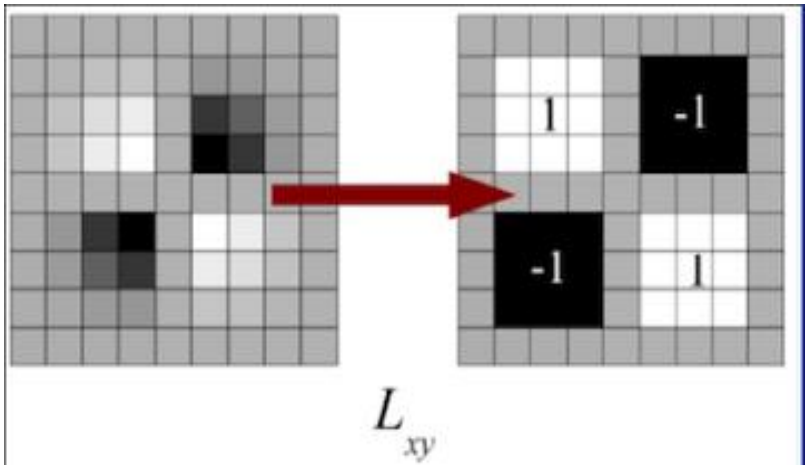


Рисунок 1.8 – Гауссова часткова похідна в *xy*

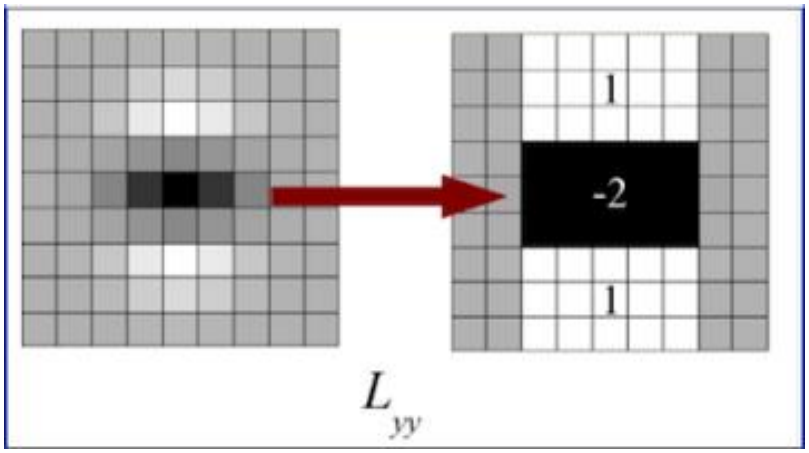


Рисунок 1.9 – Гауссова часткова похідна *y*

Фільтри 9×9 на наведених зображеннях (рис. 1.8, рис. 1.9) є наближеннями для похідних Гауса другого порядку з σ = 1,2. Позначимо ці наближення через *Dxx*, *Dyy* та *Dxy*. Тепер можна представити детермінант гессія (наближену) як [9]:

$$\det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2, \quad (1.13)$$

де $w = 0{,}9$ (пропозиція Бея).

Представлення в масштабі простору. Масштабні простори, як правило, реалізуються як піраміди зображень. Зображення неодноразово згладжуються за допомогою фільтру Гауса і згодом вибираються зразки для досягнення більш високого рівня піраміди. Використання прямокутних фільтрів та інтегральних зображень дає наступну перевагу: SURF може застосовувати однакові фільтри будь-якого розміру з абсолютно однаковою швидкістю безпосередньо на вихідному зображенні, і навіть паралельно, замість ітеративного застосовування таких фільтрів до виходу попередньо відфільтрованого шару. Отже, масштабний простір аналізується шляхом збільшення масштабу фільтру (9×9 → 15×15 → 21×21 → 27×27 тощо), а не ітеративним зменшенням розміру зображення (рис. 1.10). Отже, для кожної нової октави збільшення розміру фільтра одночасно подвоюється. Інтервали вибірки для вилучення відсотків (σ) також можуть бути подвоєні, що дозволяє збільшувати масштаб фільтра при постійних витратах [9].

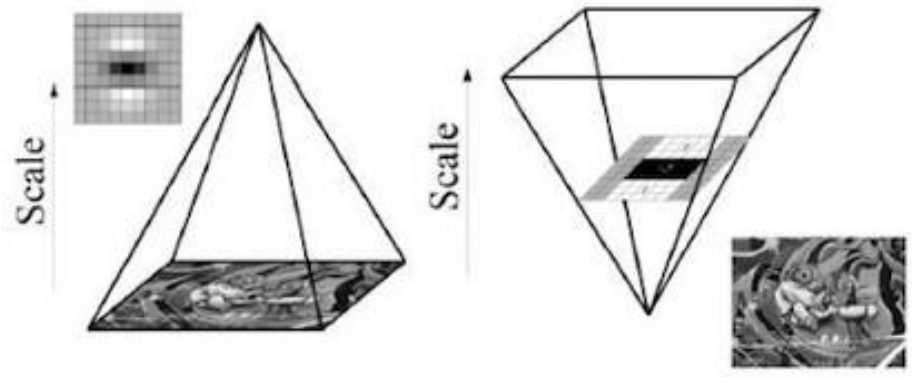


Рисунок 1.10 – Ітеративне зменшення розміру зображення (ліворуч) та збільшення масштабу фільтра при постійній вартості (праворуч)

Створення дескриптора SURF відбувається у два етапи. Перший крок складається з фіксації відтворюваної орієнтації на основі інформації з кругової області навколо ключової точки. Потім будуємо квадратну область, вирівняну до обраної орієнтації, і витягуємо з неї дескриптор SURF [9].

Призначення орієнтації. Для того, щоб не змінювати обертання, SURF намагається визначити відтворювану орієнтацію для точок інтересу. Це досягається у два етапи [9].

На першому етапі SURF спочатку обчислює значення вейвлета Хаара в напрямку *x* та *y*, і це в круговій околиці радіусом *6s* навколо ключової точки, із шкалою *s*, в якій була виявлена ключова точка. Крім того, крок дискретизації залежить від масштабу і обраний рівним *s*, а вейвлет-відгуки обчислюються в цьому поточному масштабі *s*. Відповідно, на великих масштабах розмір вейвлетів великий. Тому цілісні зображення знову використовуються для швидкої фільтрації;

На другому етапі обчислюється сума вертикальних і горизонтальних відповідей вейвлетів в зоні сканування, потім змінюємо орієнтацію сканування (додаємо *π/3*) і повторно обчислюємо, поки не знайдемо орієнтацію з найбільшим значенням суми, ця орієнтація є основною орієнтацією дескриптор функції.

Далі проводиться виймання дескриптора [9]. Спочатку будується квадратна область з центром навколо ключової точки та орієнтованою вздовж орієнтації, яку було отримано вище (розмір цього вікна – 20*s*);

Потім відбувається регулярний розподіл на менші частини розміром 4×4 квадрати (рис. 1.11). Для кожної частини обчислюються декілька простих характеристик у 5×5, що мають регулярний інтервал точок вибірки. З міркувань простоти називаємо *dx* вейвлет-відгуком Хаара в горизонтальному напрямку, а *dy* – вейвлет-відгуком Хаара у вертикальному напрямку (розмір фільтра *2s*). Щоб збільшити стійкість до геометричних деформацій та помилок локалізації, реакції *dx* та *dy* спочатку зважуються за допомогою Гауса (σ = 3,3 с) з центром у ключовій точці. Потім вейвлет-

відповіді *dx* і *dy* підсумовуються по кожної частини і утворюють перший набір записів до вектора ознак. Для того, щоб отримати інформацію про полярність змін інтенсивності, також виділяється суму абсолютних значень відповідей, |*dx*| та |*dy*|. Отже, кожна частина має чотиривимірний вектор дескриптора v для основної структури інтенсивності $V = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$. Це призводить до отримання вектора дескриптора для всіх 4-х частин довжиною 64 (у SIFT наш дескриптор є 128-D вектором, тому це частина причини, що SURF швидший, ніж SIFT) [11].

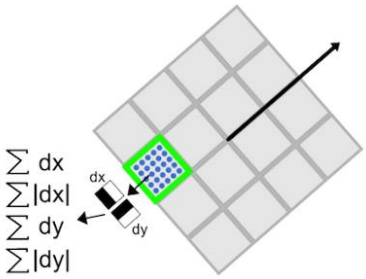


Рисунок 1.11 – Частина області з центром навколо ключової точки

1.5 FAST

Уже розглянуто кілька детекторів ознак, і багато з них справді хороші. Але ці детектори з точки зору програми в реальному часі недостатньо швидкі. Найкращим прикладом може бути мобільний робот SLAM (одночасна локалізація та відображення), який має обмежені обчислювальні ресурси [12].

Як вирішення цього, Функції з прискореного тестування сегментів (FAST) – це метод виявлення кутів, який можна використовувати для вилучення точок об’єкта, а згодом використовувати для відстеження та картографування об’єктів у багатьох завданнях комп’ютерного зору. Спочатку кутовий детектор FAST був розроблений Едвардом Ростен і Томом

Драммондом і був опублікований у 2006 році. Найголовнішою перевагою FAST детектора є його обчислювальна ефективність (швидке обчислення). Більше того, коли застосовуються методи машинного навчання, можна досягти чудової продуктивності з точки зору часу обчислення та ресурсів. Детектор кутів FAST дуже підходить для обробки відео в режимі реального часу завдяки цій високошвидкісній роботі [12].

Алгоритм виявлення ознак FAST [12]:

– виберіть піксель p на зображенні, який слід визначити як особливу точку чи ні (нехай його інтенсивність буде Ip);

– виберіть відповідне порогове значення t;

– розгляньте коло в 16 пікселів навколо тестового пікселя (це коло Брезенхама з радіусом 3);

– тепер піксель p є кутом, якщо в колі існує набір n суміжних пікселів (з 16 пікселів), які всі яскравіші за $Ip+t$ або всі темніші від $Ip-t$ (автори використали $n = 12$ у першій версії алгоритму);

– щоб зробити алгоритм швидким, спочатку порівняйте інтенсивність пікселів 1, 5, 9 та 13 кола з Ip (як видно з малюнка вище, принаймні три з цих чотирьох пікселів повинні відповідати пороговому критерію, щоб точка відсотка існувала);

– щоб зробити алгоритм швидким, спочатку порівняйте інтенсивність пікселів 1, 5, 9 та 13 кола з Ip (принаймні три з цих чотирьох пікселів повинні відповідати пороговому критерію, щоб точка відсотка існувала);

– повторіть процедуру для всіх пікселів на зображенні.

Існує кілька обмежень алгоритму. По-перше, для $n < 12$ алгоритм працює не дуже добре у всіх випадках, оскільки коли $n < 12$ кількість виявлених точок інтересу дуже велика. По-друге, порядок запиту 16 пікселів визначає швидкість алгоритму. До алгоритму вирішення цих проблем додано підхід до машинного навчання [12].

1.6 BRIEF

Дескриптори точок об’єкта зараз є основою багатьох технологій Computer Vision, таких як розпізнавання об’єктів, 3D-реконструкція, отримання зображень та локалізація камери. Оскільки виникає необхідність обробляти все більше даних або працювати на мобільних пристроях з обмеженими обчислювальними ресурсами, зростає потреба в локальних дескрипторах, які швидко обчислюються, швидко збігаються та ефективні в пам’яті [13].

Після виявлення ключової точки продовжується обчислення дескрипторів для кожного з них. Дескриптори ознак кодують цікаву інформацію в ряд цифр і виступають як свого роду числовий «відбиток пальця», який можна використовувати для диференціації однієї ознаки від іншої. Визначена околиця навколо пікселя (ключової точки) відома як патч, який являє собою квадрат певної ширини та висоти пікселя [13].

Плями зображення можуть бути ефективно класифіковані на основі порівняно невеликої кількості порівнянь інтенсивності в парі. Коротко перетворіть патчі зображень у двійковий векторний об’єкт, щоб разом вони могли представляти об’єкт. Вектор двійкових ознак також відомий як бінарний дескриптор ознак – це вектор ознак, який містить лише 1 і 0. Коротко, кожна ключова точка описується вектором ознак, який становить 128 біт – 512 біт [13].

Коротко йдеться про зображення на рівні пікселів, тому воно дуже чутливе. Попередньо згладжуючи патч, цю чутливість можна зменшити, збільшуючи тим самим стабільність і повторюваність дескрипторів. З тієї ж причини, що зображення потрібно згладжувати, перш ніж їх можна змістовно диференціювати при пошуку країв [13].

BRIEF використовує ядро Гаусса для згладжування зображення. На малюнку нижче порівнюється згладжування Гауса за коефіцієнтами розпізнавання для дисперсій ядра Гауса в діапазоні від 0 до 3. Чим складніше

збіг, тим важливішим згладжування стає досягнення гарних показників. Швидкість розпізнавання залишається відносно постійною в діапазоні від 1 до 3, і на практиці використовується значення 2 [13].

Тепер згладжено патч зображення, далі основна мета – створити бінарний векторний елемент з цього патча. Створюється бінарний вектор ознак відповідей двійкового тесту (τ). Бінарний тест τ визначається [10, 13].

$$\tau(p;x,y)=\begin{cases}1: p(x)<p(y)\\0: p(x)\geq p(y)\end{cases}, \tag{1.14}$$

де $p(x)$ – інтенсивність p у точці x.

Вибір набору з $n(x,y)$ – позиційних пар однозначно визначає набір двійкових тестів. Де n – довжина бінарного вектора ознак, і вона може становити 128, 256 і 512.

Як вибрати (x, y) пари? Залишається багато варіантів для вибору місць тестування ((x, y) пари) завдяки генеруванню n бітового вектора довжини. Пару (x, y) також називають випадковою парою, яка знаходиться всередині патча. Всього потрібно вибрати n тестів (випадкової пари) для створення бінарного вектора ознак, і потрібно вибрати цей n тест з одного з п’яти підходів (геометрії вибірки), наведено нижче (рис. 1.12) [13].

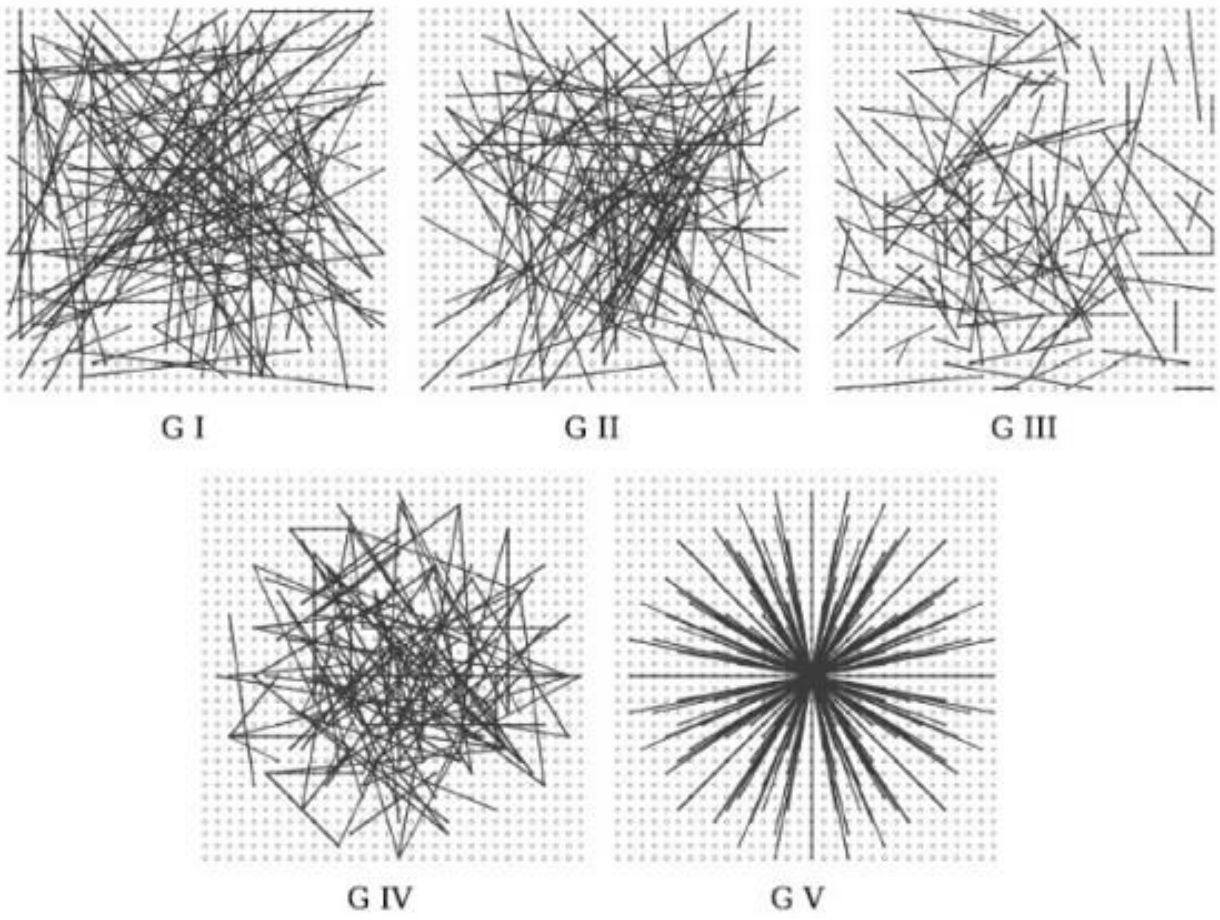


Рисунок 1.12 – Геометрія вибірки

Далі буде розглянуто розмір виправлення $p(S \times S)$ і припускається, що ключова точка знаходиться в центрі виправлення.

Рівномірний (G I) : Пікселі x та y у випадковій парі, вибраній за допомогою розподілу Unifrom або розподілу $S / 2$ навколо ключової точки. Пара (тест) може лежати близько до кордону патча [13].

Гаусова (G II) : І пікселі x, і y у випадковій парі витягуються з гауссового розподілу або розподілу $0,04 * S^2$ навколо ключової точки.

Гауссова (G III) : Перший піксель (x) у випадковій парі витягується з гауссового розподілу, зосередженого навколо ключової точки з багатогранним відхиленням або розподілом $0,04 * S^2$. Другий піксель (y) у випадковій парі береться з гауссового розподілу, зосередженого навколо першого пікселя (x) зі стандартним відхиленням або розмахом $0,01 * S^2$. Це

змушує тест (пару) бути більш локальним. Тестові (парні) місця поза патчем затискаються до краю патчу [13].

Груба полярна сітка (G IV): Пікселі *x* та *y* із випадкової пари відбираються з дискретних місць грубої полярної сітки, вводячи просторове квантування.

Груба полярна сітка (G V): перший піксель (*x*) у випадковій парі знаходиться на *(0, 0)*, а другий піксель (*y*) у випадковій парі витягується з дискретних місць грубої полярної сітки (рис. 1.13) [13].

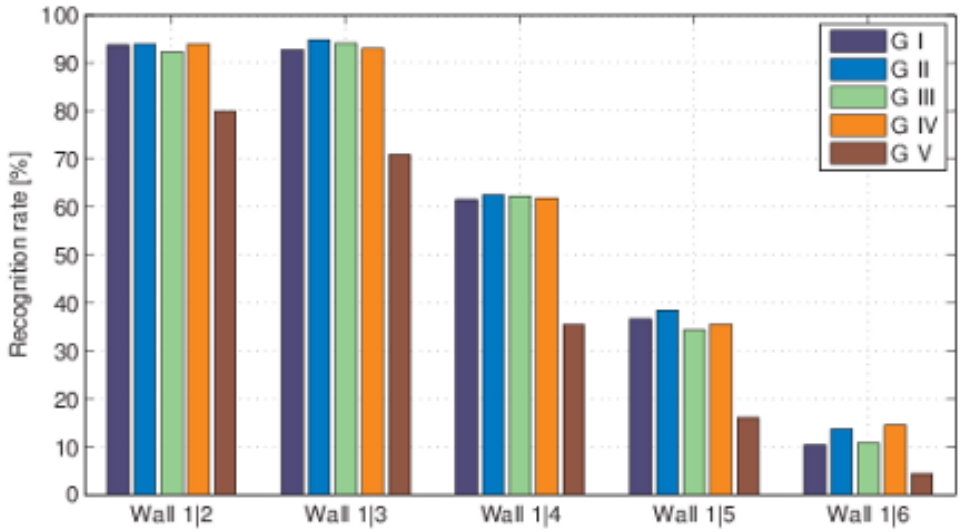


Рисунок 1.13 – Швидкість розпізнавання для п’яти різних геометричних тестів

Нарешті, BRIEF дескриптор виглядає так:

$$f_{n_d}(p) := \sum_{1 \le i \le n_d} 2^{i-1} \tau(p; x_i, y_i) . \qquad (1.15)$$

Переваги BRIEF дескрипторів. Короткий опис спирається на порівняно невелику кількість тестів на різницю інтенсивності, щоб представити патч зображення як двійковий рядок. Для цього дескриптора конструкція та узгодження не тільки набагато швидша, ніж для інших найсучасніших, але

також має тенденцію до вищих коефіцієнтів розпізнавання, доки інваріантність до великих обертань у площині не є вимогою [13].

1.7 ORB

Oriented FAST and Rotated BRIEF (ORB) був розроблений в лабораторіях OpenCV Ітаном Рублі, Вінсентом Рабо, Куртом Коноліге та Гері Р. Брадським у 2011 році як ефективна та життєздатна альтернатива SIFT та SURF. ORB був задуманий в основному тому, що SIFT та SURF є запатентованими алгоритмами. Однак ORB, на відміну від них, можна використовувати безкоштовно [8].

ORB виконує таку ж функцію, як SIFT, із завданням виявлення особливостей (і краще, ніж SURF), при цьому швидше майже на два порядки [14]. ORB спирається на добре відомий детектор ключових точок FAST і дескриптор BRIEF. Обидві ці техніки привабливі завдяки своїм хорошим характеристикам та низькій вартості. Основні внески ORB такі [8]:

– додавання компонента швидкої та точної орієнтації до FAST;

– ефективне обчислення орієнтованих коротких ознак;

– аналіз дисперсії та кореляції орієнтованих коротких ознак;

– метод навчання для декореляції BRIEF функцій за обертальної інваріантності, що призводить до кращої продуктивності в програмах найближчого сусіда.

Далі піде мова про деякі особливості FAST (функції прискореного та сегментного тесту), що використані в ORB.

ORB для кожного пікселя p швидко порівнює яскравість p із оточуючими 16 пікселями, які знаходяться в малому колі навколо p (рис. 1.14). Потім ці пікселі сортуються за трьома класами (світліші за p, темніші від p або подібні до p). Якщо більше 8 пікселів темніше або яскравіше за p, це буде вибрано як ключову точку. Тож ключові точки,

знайдені швидко, дають інформацію про розташування визначальних країв на зображенні [8].

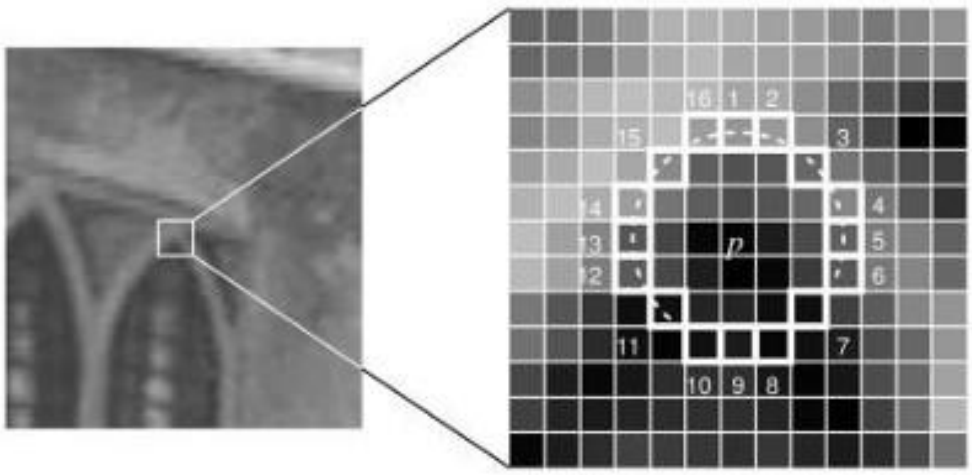


Рисунок 1.14 – Ключова точка (FAST)

Однак функції FAST не мають компонента орієнтації та багатомасштабних функцій. Отже, алгоритм ORB використовує багатомасштабну піраміду зображень [15]. Піраміда зображення (рис. 1.15) – це багатомасштабне версія зображення, яке складається з послідовності зображень, які є версіями зображення з різною роздільною здатністю. На кожному рівні піраміди міститься своя версія зображення. Коли створено піраміду, використовується алгоритм FAST для виявлення ключових точок на зображенні. Виявляючи ключові точки на кожному рівні, ORB ефективно визначає ключові точки в іншому масштабі. Таким чином, ORB є частковим інваріатом масштабу [8].

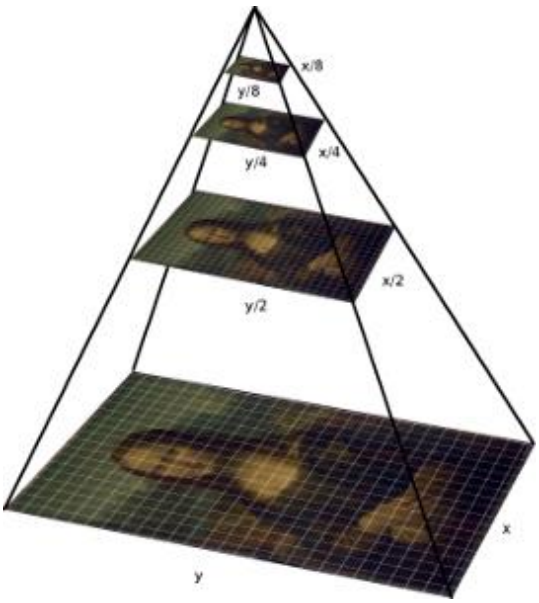


Рисунок 1.15 – Піраміда зображень (ORB)

Після пошуку ключових точок ORB тепер призначає орієнтацію кожній ключовій точці, як ліва або права сторона, залежно від того, як змінюються рівні інтенсивності навколо цієї точки. Для виявлення зміни інтенсивності шар використовує центроїд інтенсивності. Центроїд інтенсивності припускає, що інтенсивність кута зміщена від його центру, і цей вектор може бути використаний для обчислення орієнтації.

По-перше, моменти виправлення визначаються як:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y). \tag{1.16}$$

У ці моменти можна знайти центроїд, «центр маси» патчу, як:

$$C = \left(\frac{m_{10}}{m_{00}} \frac{m_{01}}{m_{00}} \right). \tag{1.17}$$

Можна побудувати вектор від центру кута *O* до центроїда – *OC*. Потім орієнтація патчу визначається:

$$\theta = a\tan 2(m_{01}, m_{10}). \tag{1.18}$$

Після того, як розрахували орієнтацію патча, можна повернути його до канонічного обертання, а потім обчислити дескриптор, отримуючи таким чином певну інваріантність обертання [8].

Далі піде мова про деякі особливості BRIEF, які використані у ORB. BRIEF (Binary robust independent elementary feature) бере всі ключові точки, знайдені швидким алгоритмом, і перетворює їх у бінарний вектор ознак, щоб разом вони могли представляти об’єкт. Вектор двійкових ознак також відомий як двійковий дескриптор ознак – це вектор ознак, який містить лише 1 і 0. Коротко, кожна ключова точка описується вектором ознак, який становить 128 біт – 512 біт.

BRIEF починається зі згладжування зображення за допомогою ядра Гауса, щоб запобігти чутливості дескриптора до високочастотних шумів. Потім BRIEF вибирає випадкову пару пікселів у визначеному районі навколо цієї ключової точки. Визначена околиця навколо пікселя відома як патч, який являє собою квадрат певної ширини та висоти пікселя. Перший піксель у випадковій парі береться з гауссового розподілу, зосередженого навколо ключової точки з багатогранним відхиленням або поширенням сигми. З гауссового розподілу, зосередженого навколо першого пікселя зі стандартним відхиленням або розповсюдженням сигми на два, береться другий піксель у випадковій парі. Тепер, якщо перший піксель яскравіший за другий, він присвоює значення 1 відповідному біту, інакше 0 [8].

BRIEF знову вибирає випадкову пару і призначає їм значення. Для 128-бітного вектора цей процес повторюється 128 разів для ключової точки. Після цього створюється такий вектор для кожної ключової точки на

зображенні. Однак BRIEF також не є інваріантом щодо обертання, тому куля використовує rBRIEF (BRIEF з урахуванням обертання). ORB намагається додати цю функціональність, не втрачаючи при цьому швидкості [8].

Далі буде розглянуто згладжене зображення. Бінарний тест τ визначається:

$$\tau(p;x,y)=\begin{cases}1: p(x)<p(y)\\ 0: p(x)\geq p(y)\end{cases}, \tag{1.19}$$

де $p(x)$ – інтенсивність p у точці x.

Характеристика визначається як вектор n двійкових тестів:

$$f(n)=\sum_{1<i<n}2^{i-1}\tau(p;x_i,y_i). \tag{1.20}$$

Показники відповідності BRIEF різко падають при обертанні в площині більше ніж на кілька градусів. ORB пропонує метод керування BRIEF відповідно до орієнтації ключових точок. Для будь-якого набору функцій з n двійкових тестів у місці (xi, yi) потрібна матриця $2xn$:

$$S=\begin{pmatrix}x1,\dots x_n\\ y1,\dots y_n\end{pmatrix}. \tag{1.21}$$

Він використовує орієнтацію патча θ та відповідну матрицю обертання $R\theta$ і створює керовану версію $S\theta$ S:

$$S_\theta=R_\theta S. \tag{1.22}$$

Тепер керований оператор BRIEF стає:

$$g_n(p,\theta) = f_n(p) \mid (x_i, y_i) \in S_0 . \quad (1.23)$$

Потім він дискретизує кут із збільшенням *2π/30* (12 градусів) і створює таблицю пошуку з попередньо обчислених коротких шаблонів. До тих пір, поки орієнтація ключової точки θ буде послідовною у поданнях, для обчислення її дескриптора буде використовуватися правильний набір точок $S\theta$.

ORB визначає алгоритм rBRIEF наступним чином:

– запустіть кожен тест на всіх навчальних патчах;

– впорядкуйте тести за їх відстанню від середнього значення 0,5, утворюючи вектор *T*;

– жадібний пошук.

Для жадібного пошуку:

– помістіть перший тест у вектор результату *R* і видаліть його з *T*;

– візьміть наступний тест з *T* і порівняйте його з усіма тестами в *P*. Якщо його абсолютна кореляція перевищує поріг, відкиньте його; ще додати його до *R*;

– повторюйте попередній крок, поки в *R* не буде 256 тестів (якщо їх менше 256, підніміть поріг і повторіть спробу).

rBRIEF демонструє значне поліпшення дисперсії та кореляції порівняно з керованим BRIEF [8].

1.8 Постановка задачі дослідження

Актуальність питання класифікації зображень зумовлена широким застосуванням комп’ютерного зору в багатьох сферах життя. А ще зумовлена питанням вибору ознак, за якими будуть класифікуватися зображення таким

чином, щоб ознаки для різних класів максимально відрізнялися одні від одного для забезпечення необхідної точності.

Об’єктом дослідження даної роботи є вагові характеристики зображень (бінарні дескриптори), обробка цих вагових характеристик та застосування для класифікації.

Основна мета – дослідити класифікацію зображень на основі бінарних дескрипторів за допомогою різних методик (SIFT, SURF, ORB, BRISK), використовуючи різні алгоритми класифікації [16 – 18]. Для цього необхідно вирішити такі завдання:

- вибрати зображення та обчислити їх бінарні дескриптори;
- сформувати початкову вибірку;
- провести класифікацію за різними алгоритмами;
- проаналізувати результати.

2 МЕТОДИ КЛАСИФІКАЦІЇ З ВИКОРИСТАННЯМ ВАГОВИХ ХАРАКТЕРИСТИК

2.1 Розроблення методу класифікації

Основним завданням атестаційної роботи є дослідження методу класифікації зображень з використанням вагових характеристик для даних опису. Тобто, основні дані – це вагові ознаки.

Дослідження буде складатися з трьох етапів:

– отримання бінарних дескрипторів еталону;

– обробка даних (дескрипторів);

– класифікація.

Спочатку про отримання дескрипторів. Найкращими є алгоритми отримання бінарних дескрипторів SURF та ORB. З одного боку, SURF максимально точний, ніж ORB. Проте алгоритм SURF має значно більші затрати по часу та пам’яті, ніж ORB і до того ж, в реалізація цього алгоритму знаходиться обмеженому доступі. З іншого боку, ORB – швидкий, у точності не дуже відстає від SURF і його реалізація знаходиться у вільному доступі. Отже, враховуючи всі переваги та недоліки методів отримання дескрипторів особливих точок, було обрано метод ORB (Oriented Fast and Rotated Brief). Бінарні дескриптори являють собою масив з векторів, які мають довжину 256 (кількість біт) та значення – 0 та 1.

Також не варто забувати про попередню обробку даних. Для того, щоб дослідження дали корректний результат, треба, щоб і в еталоні, і в тестовому зображенні бількість контрольних точок була однакова. В той же час, чим більше контрольних точок, тим точніший результат. Оскільки максимальна можлива кількість контрольних (особливих) точок в зображенні залежить від розміру зображення, то доцільно попередньо обчислювати максимально можливу кількість контрольних точок на еталоні та на тестовому зображенні, потім вибирати з них мінімум. Після цього, в свою чергу ще раз пройти по

еталону та тестовому зображенні та обчислити бінарні дескриптори контрольних точок, обмеживши їх кількість цим мінімумом.

Отже, що потрібно для подальших досліджень – бінарні дескриптори особливих точок еталону і тестового зображення, подальша обробка, та порівняння оброблених даних.

Тепер про класифікацію. Порівнювати самі дескриптори (величезні двомірні масиви) не є зручним. Зручнішим є наступний варіант – кластеризувати бінарні дескриптори еталону, знайти проекцію дескрипторів тестового зображення на еталон (тобто, віднести кожен вектор із дескрипторів тестового зображення до найближчого центру кластера еталону), потім знайти гістограми для кластерів та знайти їх подібність (відстань) [19 – 22].

Кластеризація – один із найпоширеніших методів дослідницького аналізу даних, який використовується для отримання уявлення про структуру даних. Це можна визначити як завдання ідентифікації підгруп у даних таким чином, що точки даних в одній і тій же підгрупі (кластері) дуже схожі, тоді як точки даних у різних кластерах дуже різні. Іншими словами, потрібно знайти в даних однорідні підгрупи, щоб точки даних у кожному кластері були якомога подібнішими за мірою подібності, такою як евклідова відстань або відстань на основі кореляції. Рішення, який засіб схожості використовувати, залежить від конкретної програми.

Кластерний аналіз може бути здійснений на основі ознак, де потрібно знайти підгрупи зразків на основі ознак, або на основі зразків. Тут розглянемо кластеризацію на основі функцій. Кластеризація використовується при сегментації ринку; де треба знайти клієнтів, схожих один на одного, чи то з точки зору поведінки чи атрибутів, сегментації / стиснення зображень; де треба згрупувати схожі регіони, кластеризувати документи за темами тощо.

На відміну від контрольованого навчання, кластеризація вважається некерованим методом навчання, оскільки немає основної істини для

порівняння результатів роботи алгоритму кластеризації із справжніми мітками для оцінки його ефективності. Далі буде досліджено структуру даних, групуючи точки даних у різні підгрупи [23].

2.2 Аналіз метрик для класифікації

Для знаходження відстаней між центрами кластерів, гістограмами і т.д. використовуються різні метрики:

- Евклідова;
- SAD;
- SSD;
- MAE;
- Хеммінга;
- Мінковського;
- манхеттенська;
- MSE;
- Канбери;
- косинусна;
- Пірсона;
- Чебишева.

На рис. 2.1 наведені приклади метрик. Показники відстані є ключовою частиною декількох алгоритмів машинного навчання. Ці показники відстані використовуються як під контролем, так і без нагляду, як правило, для обчислення подібності між точками даних [24].

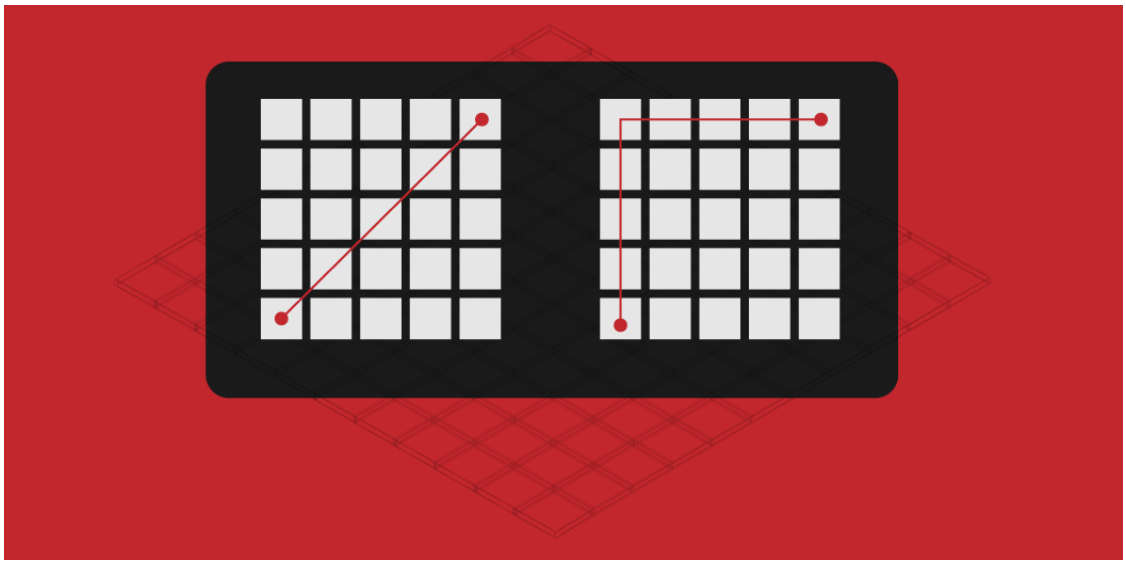


Рисунок 2.1 – Приклад відстаней

Треба створити кластери, використовуючи кластеризацію *K*-Means або *k*-Nearest Neighbor для вирішення проблеми класифікації або регресії. Як визначити подібність між різними спостереженнями тут? Як можна сказати, що два моменти схожі між собою? Це станеться, якщо їх особливості схожі, правда? Коли побудувати ці точки, вони будуть на відстані ближче один до одного (рис. 2.2) [24].

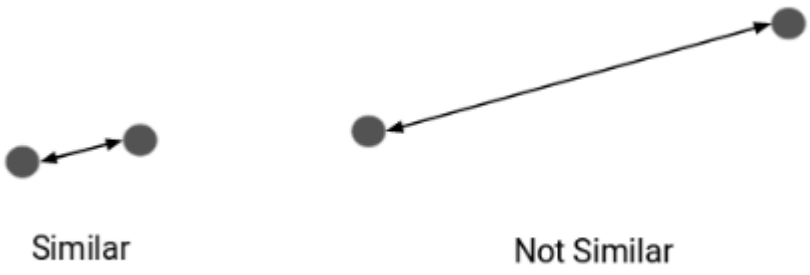


Рисунок 2.2 – Ілюстрація схожості за відстанню

Отже, можна обчислити відстань між точками, а потім визначити подібність між ними. Ось головне питання – як обчислити цю відстань і які різні показники відстані в машинному навчанні?

Метрична або функція відстані – це функція *d(x , y)*, що визначає відстань між елементами множини як невід'ємне дійсне число. Якщо відстань дорівнює нулю, обидва елементи еквівалентні за цією конкретною метрикою. Таким чином, функції відстані забезпечують спосіб вимірювання наближення двох елементів, де елементи не повинні бути числами, але також можуть бути векторами, матрицями або довільними об'єктами. Функції відстані часто використовуються як функції помилок або витрат, які слід мінімізувати в процесі оптимізації [25].

Існує кілька способів визначити метрику набору. Типовою відстанню для дійсних чисел є абсолютна різниця, $d:(x , y) \mapsto |x-y|$. Але масштабована версія абсолютної різниці, або навіть $d(x, y) = \{0$, якщо $x=y$, 1 – якщо $x \neq y\}$. є дійсними показниками. Кожен нормований векторний простір індукує відстань, задану $d(\vec{x},\vec{p}) = \|\vec{x}-\vec{p}\|$ [25].

Сума абсолютної різниці (SAD). Сума абсолютної різниці еквівалентна *L1*-норма різниці, також відома як норма Манхеттена або міських кварталів. Модуль робить цю метрику трохи складнішою для застосування під час аналізу, але вона більш надійна, ніж SSD [25].

$$d_{SAD} : (x, y) \to \| x - y \|_1 = \sum_{i=1}^{n} | x_i - y_i | . \qquad (2.1)$$

Сума квадратичної різниці (SSD). Сума різниці в квадраті еквівалентна квадрату *L2*-норма, також відома як евклідова норма. Тому вона також відома як квадрат Евклідової відстані. Це основна метрика в задачах найменших квадратів та лінійної алгебри. Відсутність abs функції робить цю метрику зручною для аналітичної роботи, але квадрати призводять до її чутливості до великих відхилень [25].

$$d_{SSD} : (x, y) \to \| x - y \|_2^2 = \langle x - y, x - y \rangle = \sum_{i=1}^{n} (x_i - y_i)^2 . \qquad (2.2)$$

Середня абсолютна помилка (MAE). Середня абсолютна похибка – це нормований варіант суми абсолютної різниці [25].

$$d_{MAE} : (x, y) \to \frac{d_{SAD}}{n} = \frac{\| x - y \|_1}{n} = \frac{1}{n} \sum_{i=1}^{n} | x_i - y_i | . \qquad (2.3)$$

Середньоквадратична помилка (MSE). Середня квадратична помилка – це нормований варіант суми квадратичної різниці [25].

$$d_{MSE} : (x, y) \to \frac{d_{SSD}}{n} = \frac{\| x - y \|_2^2}{n} = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2 . \qquad (2.4)$$

Відстань Чебишева – це $L\infty$ – норма різниці, окремий випадок відстані Мінковського, де p переходить у нескінченність. Вона також відома як Шахова дошка [25].

$$d_{\infty} : (x, y) \to \| x - y \|_{\infty} = \lim_{p \to \infty} \left(\sum_{i=1}^{n} | x_i - y_i |^{p} \right)^{\frac{1}{p}} = \max | x_i - y_i | . \qquad (2.5)$$

Відстань Канберри – це зважена версія манхеттенської відстані, запроваджена та вдосконалена в 1967 році Ленсом, Вільямсом та Адкінсом. Вона часто використовується для даних, розкиданих навколо джерела, оскільки є упередженим для вимірювань навколо джерела та дуже чутливим для значень, близьких до нуля [25].

$$d_{\infty} : (x, y) \to \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}. \quad (2.6)$$

Косинусна відстань містить точковий добуток, масштабований добутком евклідових відстаней від початку координат. Він представляє кутову відстань двох векторів, ігноруючи їх масштаб [25].

$$d_{\cos} : (x, y) \to 1 - \frac{\langle x, y \rangle}{\| x \|_2 \| y \|_2} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}}. \quad (2.7)$$

Відстань Пірсона – це кореляційна відстань, заснована на коефіцієнті кореляції продукту – імпульсу Пірсона для двох векторів вибірки. Оскільки коефіцієнт кореляції падає між [-1, 1], відстань Пірсона лежить у [0, 2] і вимірює лінійну залежність між двома векторами [25].

$$d_{Pearson} : (x, y) \to 1 - Corr(x, y). \quad (2.8)$$

Далі піде мова про метрики, які найчастіше застосовуються для машинного навчання. А саме, про наступні метрики [24]:

– Евклідова відстань;

– манхеттенська відстань;

– відстань Мінковського;

– відстань Хеммінга.

Евклідова відстань. Евклідова відстань є найкоротшою відстанню між двома точками. Більшість алгоритмів машинного навчання, включаючи *K*-Means, використовують цю метрику для вимірювання подібності між

спостереженнями. Наприклад, є два моменти, як показано нижче (рис. 2.3) [24]:

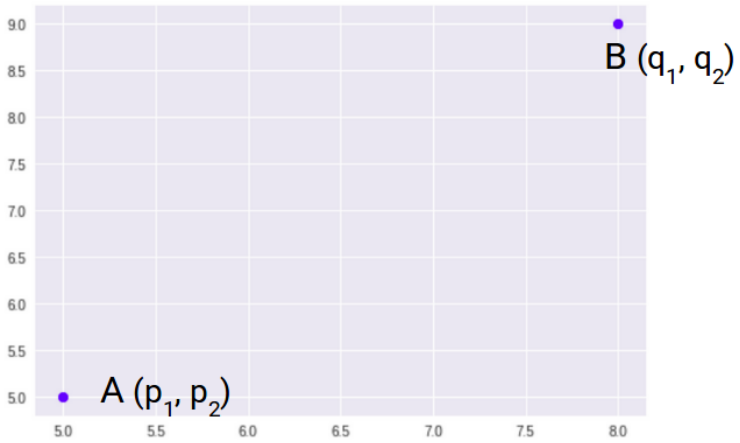


Рисунок 2.3 – Ілюстрація знаходження точок

Отже, евклідова відстань між цими точками буде (рис. 2.4) [24]:

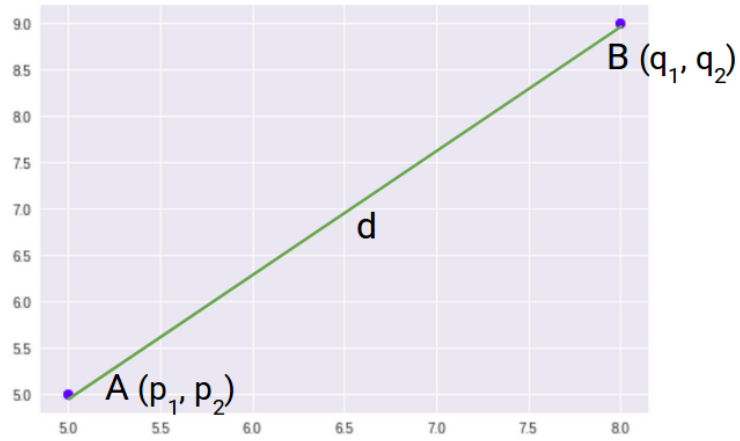


Рисунок 2.4 – Ілюстрація евклідової відстані

Ось формула для евклідової відстані:

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{\frac{1}{2}}. \quad (2.9)$$

Ця формула використовується, коли є з двома вимірами. Можна узагальнити це для n-мірного простору як [24]:

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{\frac{1}{2}}, \tag{2.10}$$

де n – кількість розмірів;

p_i, q_i – точки даних.

Манхеттенська відстань – це сума абсолютних різниць між точками за всіма вимірами (рис. 2.5). Можна представити відстань як [24].

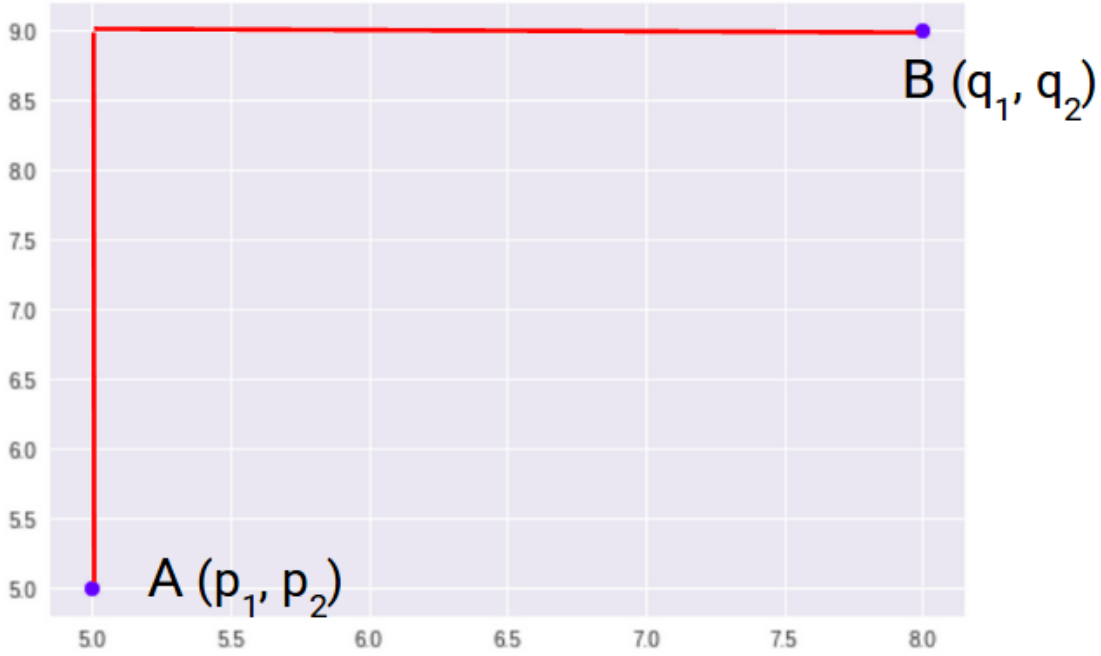


Рисунок 2.5 – Ілюстрація манхеттенської відстані

Оскільки вищезазначене подання є двовимірним, для обчислення манхеттенської відстані береться сума абсолютних відстаней в напрямках x та y. Отже, манхеттенська відстань у двовимірному просторі подається як [24]:

$$d = | p_1 - q_1 | + | p_2 - q_2 |. \tag{2.11}$$

І узагальнена формула для *n*-мірного простору подається як [24]:

$$D_n = \sum_{i=1}^{n} | p_i - q_i |, \tag{2.12}$$

де n – кількість розмірів;

p_i, q_i – точки даних.

Відстань Мінковського. Відстань Мінковського – узагальнена форма евклідової та манхеттенської відстані. Формула відстані Мінковського подана як [24]:

$$D = \left(\sum_{i=1}^{n} | p_i - q_i |^{p} \right)^{\frac{1}{p}}. \tag{2.13}$$

Параметр p метрики відстані Мінковського в SciPy представляє порядок норми. Коли порядок (p) дорівнює 1, це буде представляти манхеттенську відстань, а коли порядок у наведеній вище формулі дорівнює 2, це буде представляти Евклідову відстань. Коли порядок дорівнює 2, можна побачити, що відстань Мінковського та Евклідова відстань однакові [24].

Відстань Хеммінга вимірює схожість двох рядків однакової довжини. Відстань Хеммінга між двома рядками однакової довжини – це кількість позицій, у яких відповідні символи відрізняються [24].

Далі буде розглянуто приклад. Є два рядки: «euclidean» та «manhattan». Оскільки довжина цих рядків однакова, можна обчислити відстань Хеммінга. Далі буде посимвольне проходження і встановлення відповідності рядків. Перший символ обох рядків (*e* та *m* відповідно) відрізняється. Аналогічним чином, другий символ обох рядків (*u* та *a*) відрізняється. і так далі. Сім символів різні, тоді як два символи (два останні символи) схожі. Отже,

відстань Хеммінга тут дорівнюватиме 7. Чим більша відстань Хеммінга між двома рядками, тим більше різницею будуть ці рядки (і навпаки) [24].

Тепер постає питання, яку відстань обрати. Для аналізу схожості гістограм найкраще застосовувати манхеттенську відстань. А ось для кластеризації – тут велике питання, оскільки є дані (масив), які складаються з векторів, які містять лише нулі та одиниці.

2.3 Вибір алгоритму кластеризації для дослідження

Для кластеризації найбільш популярні наступні алгоритми:

– *k*-means;

– *k*-medians;

– ієрархічні;

– DBSCAN;

– fuzzy *C*-means.

Далі буде детальніше розглянуто ці алгоритми. Спочатку про алгоритм *K-means* (*K*-середніх), який є найпоширенішим для кластеризації. Алгоритм *K-means* – це ітераційний алгоритм, який намагається розділити набір даних на *K* заздалегідь визначених окремих неперекриваючих підгруп (кластерів), де кожна точка даних належить лише одній групі . Він намагається зробити точки даних внутрішнього кластера якомога подібнішими, одночасно зберігаючи кластери якомога більш різними (далекими). Він призначає точки кластеру таким чином, що сума квадратної відстані між точками даних і центроїдом кластера (середнє арифметичне всіх точок даних, що належать до цього кластера) є мінімальною. Чим менше варіацій у кластерах, тим однорідніші (подібнішими) є дані в межах одного кластера [23].

Принцип роботи алгоритму *k*-means такий [23]:

– вкажіть кількість кластерів *K*;

– ініціалізуйте центроїди, спочатку перетасувавши набір даних, а потім випадковим чином вибравши *K* точок даних для центроїдів без заміни;

– продовжуйте повторювати, поки центроїди не будуть змінені. тобто присвоєння точок даних кластерам не змінюється;

– обчисліть суму квадратичної відстані між точками даних та усіма центроїдами;

– призначте кожну точку даних найближчому кластеру (центроїду);

– обчисліть центроїди для кластерів, взявши середнє значення всіх точок даних, що належать кожному кластеру.

Викликається підхід *k-means* для вирішення проблеми Очікування-максимізація. *E*-крок призначає точки даних найближчому кластеру. *M*-крок обчислює центроїд кожного кластера. Нижче наведено приклад того, як можна це вирішити математично.

Цільовою функцією є [23]:

$$J = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} \| x_i - \mu_k \|^2, \qquad (2.14)$$

де w_{ik} = 1 для точки даних x_i, якщо вона належить кластеру k; в іншому випадку w_{ik} = 0.

Крім того, μk є центроїдом кластера xi.

Це задача мінімізації двох частин. Спочатку мінімізуються *J wrt wik* і обробляється μk фіксовано. Тоді мінімізуються *J wrt* μk і обробляється *wik* фіксовано. Технічно кажучи, спочатку треба диференціювати *J wrt wik* та оновити призначення кластера (*E*-крок). Потім диференціюються *J wrt* μk і перераховуються центроїди після призначення кластера з попереднього кроку (*M*-крок). Отже, *E*-крок. Іншими словами, треба призначити точку даних *xi* найближчому скупченню, судячи з його суми квадратної відстані від центроїда кластера.

Що означає перерахунок центроїда кожного кластера для відображення нових призначень. Тут можна відзначити кілька речей [23]:

– оскільки алгоритми кластеризації, включаючи *k*-means, використовують вимірювання на основі відстані для визначення схожості між точками даних, рекомендується стандартизувати дані, щоб мати середнє значення нуля та стандартне відхилення одиниці, оскільки майже завжди функції будь-якого набору даних матимуть різні одиниці виміру такі як вік проти доходу;

– враховуючи ітераційний характер *k*-means і випадкову ініціалізацію центроїдів на початку алгоритму, різні ініціалізації можуть призвести до різних кластерів, оскільки алгоритм *k*-means може застрягти в локальному оптимумі і може не сходитися до глобального оптимуму; тому рекомендується запустити алгоритм, використовуючи різні ініціалізації центроїдів, і вибрати результати прогону, які дали нижчу суму квадратної відстані;

– призначення прикладів не змінюється – це те саме, що відсутність змін у варіації всередині кластера.

$$\frac{1}{m_k} = \sum_{i=1}^{m_k} \| x^i - \mu_{c^k} \|^2 . \qquad (2.15)$$

Цей алгоритм кластеризації має наступні переваги [26]:

– немічені набори даних;

– нелінійно відокремлені дані;

– простота;

– доступність;

– швидкість.

Далі буде розглянуто детальніше ці переваги.

Немічені набори даних. Багато реальних даних надходять без маркування, без певного класу. Перевага використання такого алгоритму, як кластеризація *K*-means, полягає в тому, що часто не відомо, як слід групувати екземпляри в наборі даних. Наприклад, є проблема спроби групувати глядачів Netflix у кластери на основі подібної поведінки перегляду. Відомо, що є кластери, але не відомо, що це за кластери. Лінійні моделі зовсім не допоможуть у вирішенні подібних питань [26].

Нелінійно відокремлені дані. Є набір даних, що містить набір із трьох концентричних кіл. Цей набір нелінійно розділяється. Іншими словами, немає прямої лінії чи площини, яку можна б намалювати на графіку нижче, яка легко розрізняє кольорові класи червоного, синього та зеленого. Використовуючи кластеризацію *K*-means та перетворюючи систему координат нижче з декартових координат у полярні координати, можна було б використовувати інформацію про радіус для створення концентричних кластерів [26].

Простота. Основна частина алгоритму кластеризації *K*-means лише два кроки, крок призначення кластера та крок переміщення центроїда. Якщо шукати алгоритм навчання без нагляду, який легко реалізувати і може обробляти великі набори даних, кластеризація *K*-means є гарною відправною точкою [26].

Доступність. Більшість популярних пакетів машинного навчання містять реалізацію кластеризації *K*-means.

Швидкість. Алгоритм виконує свою роботу швидко, навіть для дуже великих наборів даних [26].

Але, окрім цього, алгоритм *K*-means має недоліки [27]:

– *k* вибирається вручну;
– залежність від початкових центрів;
– дані кластеризації різного розміру та щільності;
– кластеризаційні викиди;
– масштабування за кількістю розмірів.

Тепер про ці недоліки.

K вибирається вручну. Треба вручну задавати кількість кластерів. Крім того, від кількості кластерів залежить точність [27].

Залежність від початкових значень. Для низького *K*, можна пом'якшити цю залежність, запустивши *k-means* кілька разів з різними початковими значеннями та вибравши найкращий результат. Зі збільшенням кількості кластерів (*K*), потрібні вдосконалені версії *k-means*, щоб вибрати кращі значення початкових центрів (так звані посіви *k-means*) [27].

Дані кластеризації різного розміру та щільності. *k-means* має проблеми з кластеризацією даних, коли кластери мають різний розмір і щільність [27].

Кластеризаційні викиди (елементи, які не входять в жоден кластер, або знаходяться між кількома кластерами). Центроїди можуть перетягуватись викидами, або викиди можуть отримати власний кластер замість того, щоб їх ігнорувати. Тому треба подумати про видалення або відсікання викидів перед кластеризацією [27].

Масштабування за кількістю розмірів. Зі збільшенням кількості розмірів показник подібності на відстані зближується до постійного значення між будь-якими наведеними прикладами. Зменшіть розмірність або за допомогою PCA для даних об'єкта, або за допомогою «спектральної кластеризації» для модифікації алгоритму кластеризації, як пояснено нижче [27].

K-medians. Як уже згадувалося вище, наявність численних випадаючих значень може суттєво перешкодити ефективності алгоритму. *K*-Means використовує середнє значення кожної точки кластера для обчислення центру кожного кластера. Однак середнє значення не є надійною метрикою. Отже, наявність викидів перекосить центри до них.

K-Medians базується на тому ж алгоритмі, що і *K*-Means, з тією різницею, що замість обчислення середнього значення координат усіх точок у даному кластері він використовує медіану. Як результат, скупчення стануть більш щільними та стійкими до перевищення [28].

Ієрархічна кластеризація. Ієрархічна кластеризація, мабуть, найінтуїтивніший алгоритм із усіх, що забезпечує велику гнучкість. Ієрархічна кластеризація є агломеративним алгоритмом. По суті, на початку процесу кожна точка даних знаходиться у своєму кластері. Використовуючи функцію несхожості, алгоритм знаходить дві точки в наборі даних, які є найбільш схожими, та об'єднує їх у групи. Алгоритм працює ітераційно так, поки всі дані не будуть кластеризовані. На цьому етапі можна використовувати дендрограму для інтерпретації різних кластерів та вибору кількості кластерів за бажанням.

Цей алгоритм найкраще підходить для [28]:

– категоричних даних;

– пошуку викидів.

Гнучкість є завдяки різним наявним функціям несхожості (тобто повний зв'язок, одинарний зв'язок, середній зв'язок, мінімальна дисперсія тощо), кожна з яких дає дуже різні результати [28].

Є ще одна особливість цього алгоритму. Зі збільшенням обсягу даних він стає дуже інтенсивним за часом / ресурсами, оскільки обробляє кожну точку даних ітеративно, щоразу перебираючи весь набір даних. Ієрархічна кластеризація погано масштабується [28].

Далі піде мова про алгоритм *fuzzy C-means* (FCM). Це алгоритм нечіткої кластеризації с-середніх , який дуже схожий на алгоритм k-середніх і метою є мінімізація цільової функції, визначеної наступним чином [29]:

$$J = \sum_{j=1}^{k} \sum_{i \in Cj} u_{ij}^{m} \| x_i - \mu_j \|^2, \qquad (2.16)$$

де *uij* – це ступінь спостереження *xi* належить кластеру *cj, μj* є центром;

m – це розмивач.

Видно, що FCM відрізняється від *k*-середніх за допомогою значень членства *uij* і розмивач *m*. Ступінь приналежності, *uij*, пов'язана обернено до відстані від x до центру скупчення.

Параметр *m* є дійсним числом більше 1 ($1{,}0 < m < \infty$) і визначає рівень розмитості кластера. Значення близьке до 1 дає результат, який стає дедалі подібнішим до результату жорсткої кластеризації, такої як *k*-means; тоді як значенням близьке до нескінченного призводить до повної нечіткості. Найкращим вибором є використання $m = 2{,}0$ (Hathaway and Bezdek 2001 р.) [29].

У нечіткому скупченні центроїд скупчення означає середнє значення всіх точок, зважених за ступенем їхньої належності до скупчення:

$$Cj = \frac{\sum u_{ij}^{m} x}{\sum u_{ij}^{m}}, \tag{2.17}$$

де *Cj* – центроїд кластеру *j*;

uij – це ступінь спостереження *xi* належить кластеру *cj*.

Алгоритм нечіткої кластеризації можна узагальнити таким чином [29]:

– вкажіть кількість кластерів *k* (аналітиком);

– призначте випадково кожному коефіцієнту точки для перебування в кластерах;

– обчисліть центроїд для кожного кластера, використовуючи формулу вище;

– для кожної точки обчисліть її коефіцієнти перебування в кластерах, використовуючи формулу вище;

– повторюйте третій та четвертий кроки, доки не буде досягнуто максимальної кількості ітерацій (заданих «maxit»), або коли алгоритм зійдеться (тобто зміна коефіцієнтів між двома ітераціями не перевищує ϵ, заданий поріг чутливості).

Алгоритм також мінімізує дисперсію між кластерами, але має ті самі проблеми, що і *k*-means; мінімум – це локальний мінімум, а результати залежать від початкового вибору ваг. Отже, різні ініціалізації можуть призвести до різних результатів.

Використання суміші Гаусса разом із алгоритмом максимізації очікувань є більш статистично формалізованим методом, який включає деякі з цих ідей: часткове членство в класах [29].

Переваги FCM [30]:

– дає найкращий результат для накладеного набору даних і порівняно кращий, ніж алгоритм *k-means* значень;

– на відміну від *k-means*, де точка даних повинна належати виключно одному центру кластера, тут присвоюється точка даних;

– членство в кожному центрі кластера, в результаті якого точка даних може належати більше ніж одному центру кластера.

Недоліки [30]:

– відмінність (distinctness) – особлива точка повинна бути відмінною в своїй околиці;

– інваріантність (invariance) – незалежність до афінних перетворень змін яскравості;

– апріорі специфікація кількості кластерів;

– при меншому значенні β отримується кращий результат, але за рахунок більшої кількості ітерацій;

– евклідова міра відстані може нерівномірно зважувати основні фактори.

Проаналізувавши переваги та недоліки віще перераховиних алгоритмів кластеризації, було вирішено застосувати алгоритм k-means для кластеризації дескрипторів [31].

3 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Обґрунтування вибору середовища програмної реалізації

Для досліджень буде використано середовище Jupyter Notebook на сервісі Google Colaboratory.

Цей вибір зумовлений наступними причинами [32 – 36]:

– Jupyter Notebook є середовищем, у якому запускаються файли .ipynb (файли з текстом та фрагментами коду програми);

– Jupyter Notebook розрахований на мову програмування Python, але можна при спеціальних налаштуваннях використовувати інші мови програмування;

– можна запускати не всю програму, а окремі її фрагменти (комірки);

– візуалізація даних;

– попередньо встановлена велика кількість бібліотек python;

– для Google Colaboratory зайві налаштування не потрібне;

– Google Colaboratory дозволяє безкоштовно використовувати графічний процесор;

– Google Colaboratory може запускати ваш код протягом 24 годин без перерв, але не більше того;

– програми зберігаються лише на Google Drive.

Далі піде мова про те, які бібліотеки можуть знадобитися. В python є дві такі бібліотеки для роботи з зображеннями – scikit-image та OpenCV. Для даного дослідження можуть знадобитися обидві бібліотеки тому, що [37]:

– в scikit-image, на відміну від OpenCV, можна обчислювати саме бінарні дескриптори ORB, а не OpenCV дескриптори;

– в scikit-image немає можливості візуалізувати контрольні точки на зображенні, а в OpenCV – є.

3.2 Особливості програмної реалізації

Розглянемо особливості програмної реалізації. Спочатку завантажується зображення (рис. 3.1), конвертується у відтінки сірого (рис. 3.2).

Рисунок 3.1 – Основне зображення

Рисунок 3.2 – Основне зображення у відтінках сірого

Розмір цього зображення 600×314 пікселів. Далі береться ¼ зображення, де зображено перше цуценя. І використовується це зображення у якості еталону (рис. 3.3).

Рисунок 3.3 – Еталон

Але перед тим, як обчислювати бінарні дескриптори ORB для еталону, треба перевірити можливі кількості контрольних точок, оскільки при різній кількості контрольних точок проводити порівняння дескрипторів не має сенсу. Спочатку робиться проходження по зображенню і знаходиться максимально можлива кількість контрольних точок для кожного фрагменту зображення (рис. 3.4).

```
def checkOrbPoints(img):
    descriptor_extractor = ORB(n_keypoints=10000)
    descriptor_extractor.detect_and_extract(img)
    h_keypoints = descriptor_extractor.keypoints
    return h_keypoints.shape[0]
```

Рисунок 3.4 – Код для підрахунку максимальної кількості КТ

Отримано такий масив: [881, 505, 841, 634, 1027, 511, 1164]. Мінімальна кількість – 505, але для зручності було обрано 500. Після цього знаходяться бінарні дескриптори ORB для еталону (рис. 3.5, рис. 3.6).

```
def getOrb(img, keypoints_count):
    descriptor_extractor = ORB(n_keypoints=keypoints_count)
    descriptor_extractor.detect_and_extract(img)

    h_descriptors = descriptor_extractor.descriptors
    h_keypoints = descriptor_extractor.keypoints

    print('Keypoints count: ', h_descriptors.shape[0])

    kps = []
    for kp in h_keypoints:
      kps.append(cv2.KeyPoint(kp[1], kp[0], 20))

    return kps, h_descriptors
```

Рисунок 3.5 – Код для знаходження бінарних дескрипторів

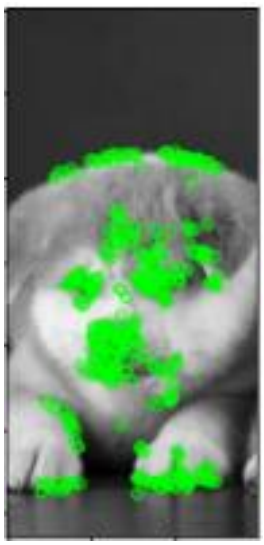


Рисунок 3.6 – КТ на еталоні

Після цього для еталону було отримано бінарні дескриптори ORB (лістинг 3.1) наступного виду:

```
[[0 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 0 1 0 0 1
  0 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 0
  0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0
  1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 1 0 1 0 0 0 1 0 1 0 0 1
  1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0
  1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1
  0 1 0 1 1 0 0 1 0 1 0 0 1 0 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 0
  1 1 1 1]

 [0 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0
  1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0
  0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1
  0 1 0 1 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0
  1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0
  1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 1
  0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1
  1 1 1 1]

 [1 1 0 1 0 0 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0
  1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
  0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1
  0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1 1 1 0
  1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0 0 1 0 0 1 0
  1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 1
  0 1 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1
  1 1 1 1]

 . . . . . .

 [1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 1
  0 1 1 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 1
  1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1
  0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1
  1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1
  0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1
  1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1
  1 0 1 0]

[1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 1
  0 1 1 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 1 1
  1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1
  0 1 1 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1
  1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1
  0 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 1
  1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1
  1 0 1 0]

 [1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1
  0 1 1 1 0 0 1 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1 1 1 0 0 1 0 0 1 1
  0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 0 1
  0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 0
  1 1 0 1 0 1 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1
  0 1 0 1 1 1 0 0 0 1 1 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 1
  1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0
  1 0 1 1]]
```

Лістинг 3.1 – Бінарні дескриптори ORB

Далі ці дескриптори кластеризуються. Для цього застосовується алгоритм *k*-means з манхеттенською метрикою (рис. 3.7) для обчислення відстаней між дескрипторами (тут кожен вектор із бінарного дескриптора розглядається як один елемент множини даних) і 3 кластерами, при цьому обираються випадковим чином їх початкові центри (лістинг 3.2), які потрібні для алгоритму *k*-means [38]. Але ці початкові центри можуть впливати на результат кластеризації і для різних початкових центрів можуть бути різні результати.

```
[
 [0 1 1 1 1 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1
  1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 0 0 0 0 1
  0 0 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 1 1
  0 0 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 1 1
  1 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 0 0 1 0
  1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1
  1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1
  1 1 1 1]

 [0 1 0 1 0 1 0 0 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 1
  1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0
  1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 1 0 0 1 1
  1 1 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0
  1 1 0 0 0 1 1 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0
  1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0
  1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0 1 1 1 1 1
  1 0 1 1]

 [0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 1
  1 0 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0
  1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1
  1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0
  1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 0
  0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 0 1
  1 1 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 1 0 1
  0 1 0 1]
]
```

Лістинг 3.2 – Початкові центри

```
from copy import deepcopy

centers_old = np.zeros(centers.shape) # to store old centers
centers_new = deepcopy(centers) # Store new centers

data.shape
clusters = np.zeros(n)
distances = np.zeros((n,k))

error = np.linalg.norm(centers_new != centers_old)

# When, after an update, the estimate of that center stays the same, exit loop
while error != 0:
    # Measure the distance to every center
    for i in range(k):
        for j in range(n):
          distances[j][i] = np.count_nonzero(data[j] != centers[i])

    # print(distances[:,0])

    # Assign all training data to closest center
    clusters = np.argmin(distances, axis = 1)

    centers_old = deepcopy(centers_new)
    # Calculate mean for every cluster and update the center
    for i in range(k):
        centers_new[i] = get_center(data[clusters == i])
        # print(centers_new)
    error = np.linalg.norm(centers_new != centers_old)
```

Рисунок 3.7 – Код кластеризації для першого експерименту

У результаті такої кластеризації було отримано нові центри кластерів (лістинг 3.3). Ці центри є векторами із бінарних дескрипторів.

```
[1 1 0 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 0 1 1 0
  1 1 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0
  0 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1
  0 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1
  1 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 1 0 1 1 0 1 0
  0 1 0 0 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 1
  0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1
  1 1 1 1]

 [0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 0 1 1 1 1 1
  1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0
  1 0 1 1 1 0 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1
  1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0
  1 1 0 0 1 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 0 1 0 0
  1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 1 0 0 1 1 1 1 0 0 1
  1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1
  1 0 1 1]

 [0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1
  1 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1
  1 0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1
  1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 1 0 1 1 1 0 1
  1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0
  1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 1 0 1 0 1 0 0 0 1
  1 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 1
  0 1 0 1]
```

Лістинг 3.3 – Знайдені центри

Далі було знайдено гістограму (рис. 3.8) для кластерів (кількість елементів у кожному кластері). Гістограма буде далі використовуватися для порівняння з гістограмами проекцій дескрипторів тестових зображень на кластери для еталону.

$$n_k = \sum_{i=1}^{m_k} v_i \,, \tag{3.1}$$

де k – кількість кластерів;

mk – елементів у кластері k.

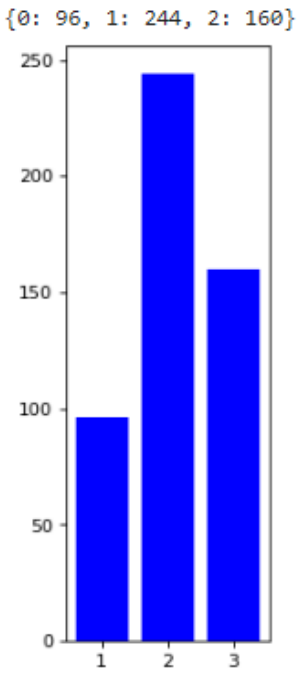


Рисунок 3.8 – Гістограма кластерів

Далі було пройдено ковзаючим вікном по зображенню, для кожного фрагменту знайдено бінарні дескриптори, проекцію на еталон (віднесення дескрипторів до кластерів, знайдених для еталону) та знайдено гістограми цих проекцій, а також манхеттенську відстань від гістограм проекцій до гістограми еталону. Ця процедура буде повторена і для наступних двох експериментів.

Тепер змінюється підхід до кластеризації. Бінарні дескриптори розглядаються як числові дані, де рядки – це вектори, довжиною 256, а кожен біт із 256 – це атрибут (стовбець) даних. Елементи цих векторів – це нулі та одиниці. Тут буде обчислюватися стандартна евклідова відстань для кожного атрибуту даних. Центрами можливо будуть вектори (масиви), які не є векторами із бінарних дескрипторів [39, 40].

Спочатку проводиться така кластеризацію бінарних дескрипторів (рис. 3.9).

```
from sklearn.cluster import KMeans

data = des1.astype(int)
#n_clusters is the number of clusters you want to use to classify your data
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=0).fit(data)

#you can see the labels with:
print(kmeans.cluster_centers_)
```

Рисунок 3.9 – Код кластеризації для другого і третього експериментів

Отримано центри кластерів (лістинг 3.4), що не є векторами в бінарних дескрипторах:

```
[0.27642276 0.66666667 0.45528455 0.71544715 0.45528455 0.27642276
  0.57723577 0.4796748  0.31707317 0.48780488 0.43089431 0.62601626
  0.76422764 0.60162602 0.21138211 0.66666667 0.37398374 0.6097561
  0.50406504 0.69105691 0.7398374  0.73170732 0.21138211 0.7804878
  0.43902439 0.41463415 0.27642276 0.69105691 0.82926829 0.61788618
  . . . . . .
  0.73170732 0.46341463 0.75609756 0.56910569 0.40650407 0.80487805
  0.45528455 0.7398374  0.82113821 0.3495935  0.58536585 0.43902439
  0.80487805 0.77235772 0.50406504 0.69105691 0.50406504 0.58536585
  0.65853659 0.53658537 0.24390244 0.60162602 0.32520325 0.35772358
  0.63414634 0.31707317 0.3495935  0.79674797 0.45528455 0.22764228
  0.54471545 0.34146341 0.33333333 0.33333333 0.64227642 0.27642276
  0.81300813 0.51219512 0.80487805 0.53658537 0.27642276 0.65853659
  0.24390244 0.57723577 0.63414634 0.85365854 0.45528455 0.64227642
  0.60162602 0.4796748  0.75609756 0.52845528]
 [0.52727273 0.93333333 0.23030303 0.53939394 0.28484848 0.55757576
  0.09090909 0.14545455 0.75151515 0.42424242 0.54545455 0.69090909
  0.65454545 0.51515152 0.34545455 0.76969697 0.43636364 0.76363636
  0.13939394 0.44242424 0.38787879 0.8        0.75757576 0.33333333
  . . . . . .

  0.56363636 0.62424242 0.58787879 0.64242424 0.72121212 0.77575758
  0.71515152 0.40606061 0.73333333 0.94545455]

 [0.59433962 0.37264151 0.54716981 0.45283019 0.76886792 0.58490566
  0.40566038 0.36320755 0.58018868 0.50943396 0.5        0.50471698
  0.52830189 0.29716981 0.64150943 0.31603774 0.70283019 0.61792453
  0.47169811 0.63207547 0.49528302 0.59433962 0.75943396 0.3254717
  . . . . . .
  0.52358491 0.77358491 0.48584906 0.24528302 0.56132075 0.60849057
  0.27358491 0.71226415 0.59433962 0.72169811 0.31603774 0.74528302
  0.38207547 0.38679245 0.4009434  0.28773585 0.59433962 0.70283019
  0.66037736 0.27358491 0.46698113 0.33018868 0.31132075 0.46698113
  0.53301887 0.68396226 0.51886792 0.56132075]
```

Лістинг 3.4 – Знайдені центри, що не є векторами з дескрипторів

Далі знову побудовано гістограму (рис. 3.10) для кластерів (кількість елементів у кожному кластері).

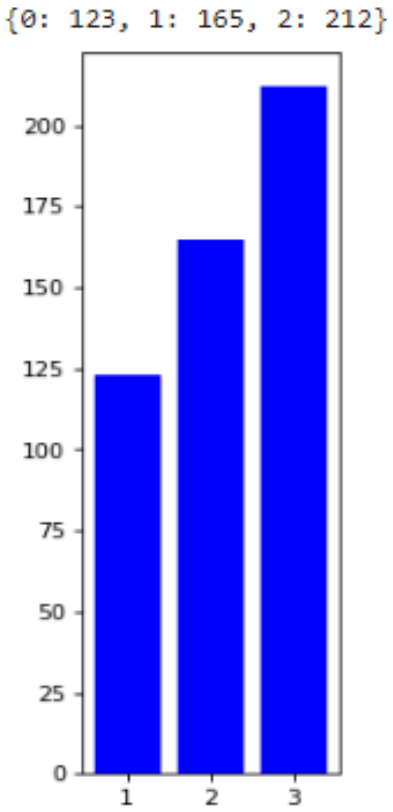


Рисунок 3.10 – Гістограма нових кластерів

Потім знову було пройдено ковзаючим вікном по зображенню, для кожного фрагменту знайдено бінарні дескриптори, проекцію на еталон (віднесення дескрипторів до кластерів, знайдених для еталону) та знайдено гістограми цих проекцій, а також манхеттенську відстань від гістограм проекцій до гістограми еталону [41, 42].

Потім було задано менший ліміт контрольних точок – 200. Після цього було повторено попередній експеримент з цим обмеженням. Так само було знайдено бінарні дескриптори ORB вже для 200 контрольних точок (рис. 3.11). А потім все було зроблено так само, як і в попередньому експерименті.

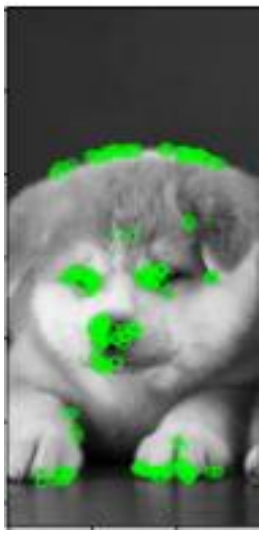


Рисунок 3.11 – Нові КТ на еталоні

Далі знову було знайдено знаходимо гістограму (рис. 3.12) для кластерів (кількість елементів у кожному кластері).

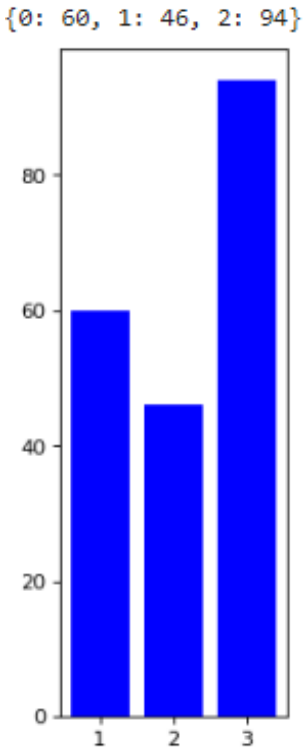


Рисунок 3.12 – Гістограма кластерів

Усі результати цих експериментів буде розглянуто у відповідному розділі.

3.3 Інструкція користувача

Для того, щоб запустити програму, де реалізовано дослідження, треба виконати наступні кроки:

– заходимо в аккаунт google;

– переходимо в Google Colaboratory;

– обираємо в головному меню «Upload notebook» та завантажуємо файл .ipynb;

– завантажуємо зображення в Google Drive, в комірці з фрагментами коду drive.mount(), та full_img_path= … прописуємо шлях до дирукторії та шлях до файлу;

– запускаємо комірки з кодом.

3.4 Аналіз результатів дослідження

Тепер настав час проаналізувати результати дослідження. Уже було 3 спроби класифікації фрагментів зображення:

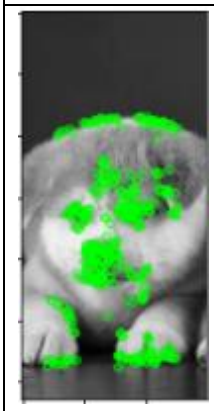
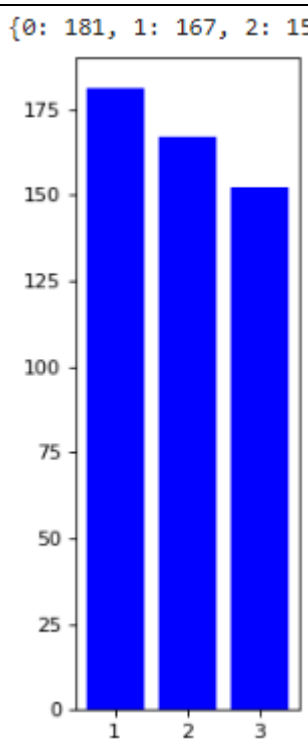
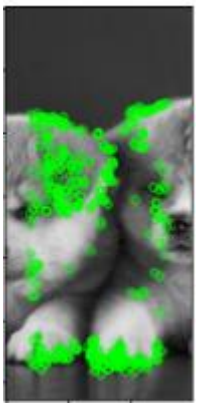
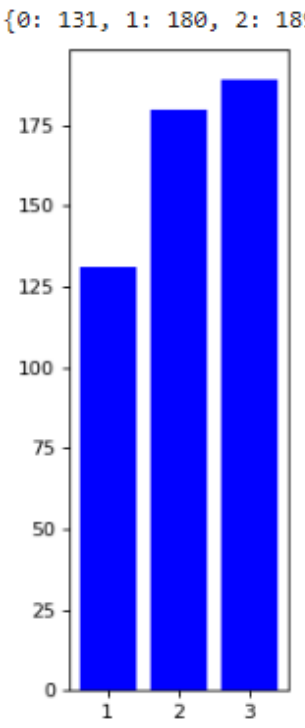
– за допомогою кластеризації *k*-means з манхеттенською метрикою для обчислення центрів (500 КТ);

– за допомогою стандартної кластеризації *k*-means, де елементи векторів розглядаються як числові атрибути даних (500 КТ);

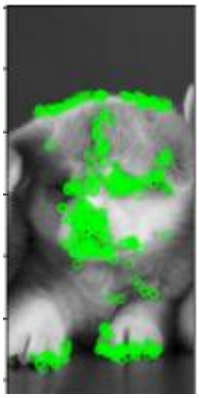
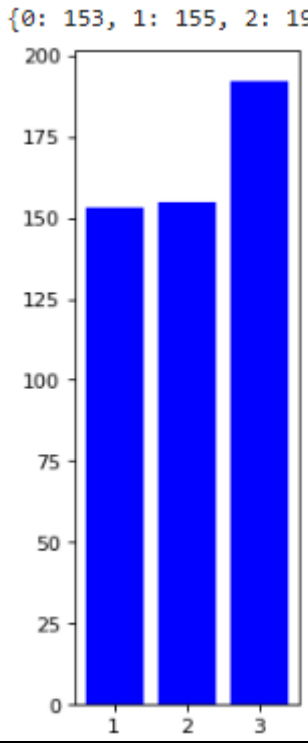
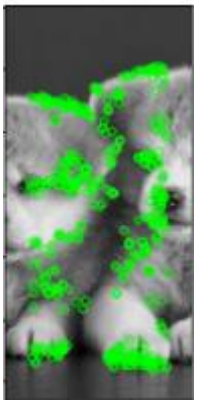
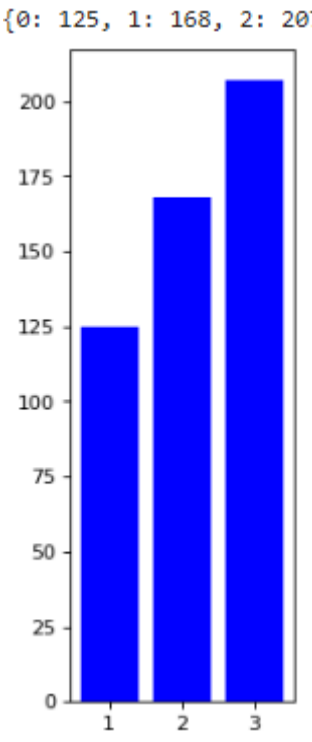
– за допомогою стандартної кластеризації *k*-means, де елементи векторів розглядаються як числові атрибути даних (200 КТ).

Спочатку про перший експеримент (класифікація за допомогою кластеризації *k*-means з манхеттенською метрикою для обчислення центрів для 500 контрольних точок). Розглянемо результати цього експерименту (табл. 3.1).

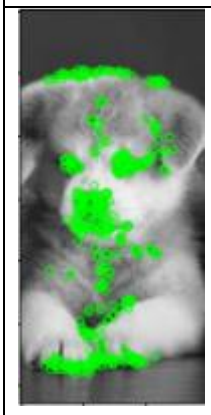
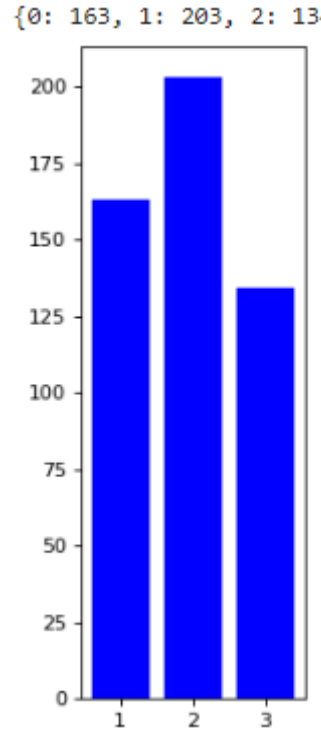
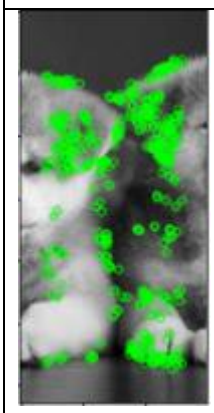
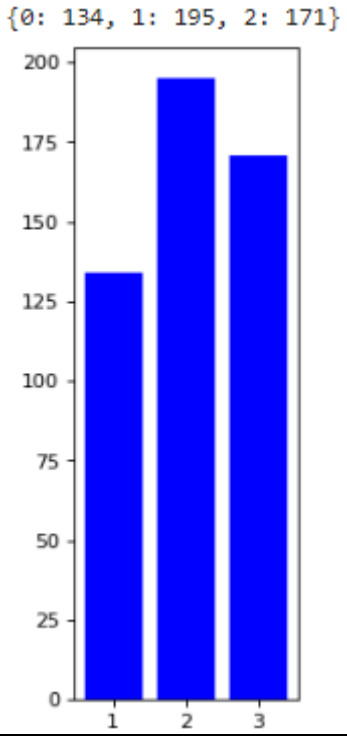
Таблиця 3.1 – Результати експерименту

Фрагмент зображення	**Гістограма**	**Манхеттенська відстань від гістограми еталону**
1	2	3
		118
		150

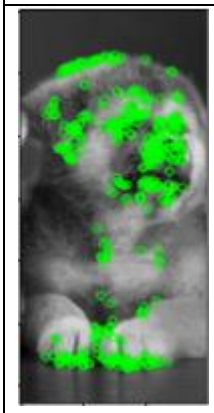
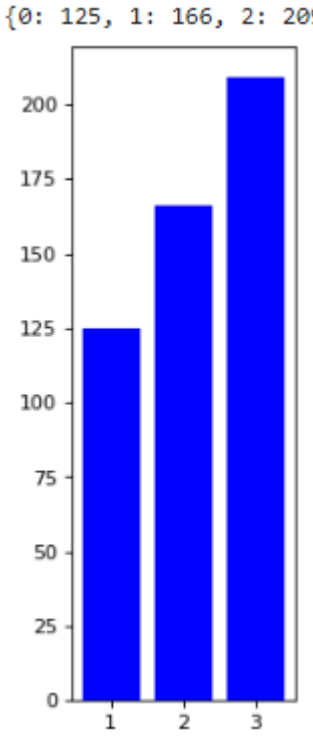
Продовження таблиці 3.1

1	2	3
		156
		186

Продовження таблиці 3.1

1	2	3
		46
		114

Кінець таблиці 3.1

1	2	3
		190

Далі відображено 4 частини з найменшою манхеттенською відстанню (рис. 3.13 – рис. 3.16).

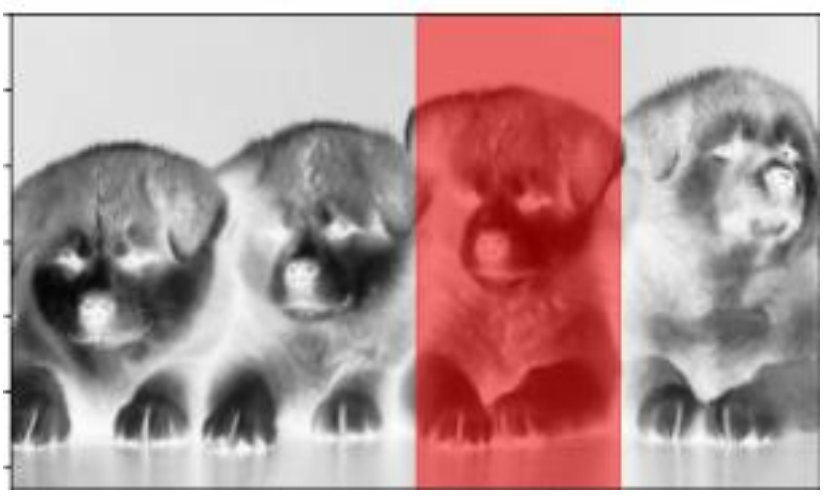


Рисунок 3.13 – Фрагмент з відстанню 46

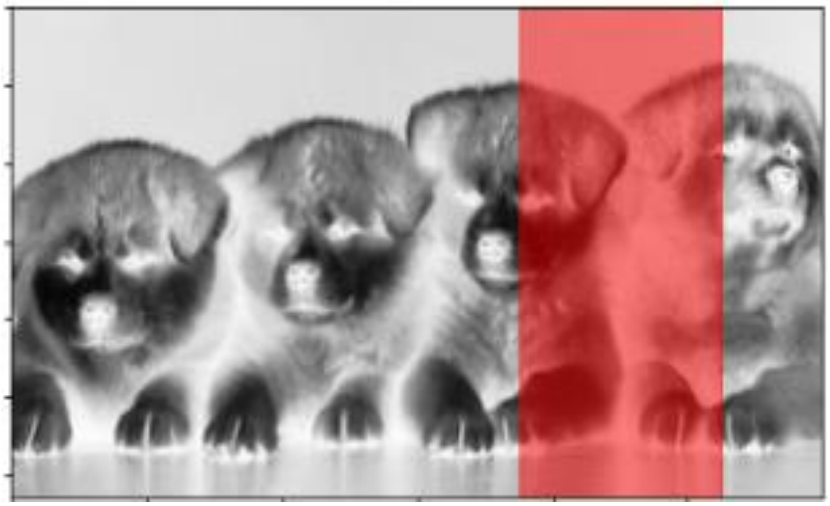


Рисунок 3.14 – Фрагмент з відстанню 114

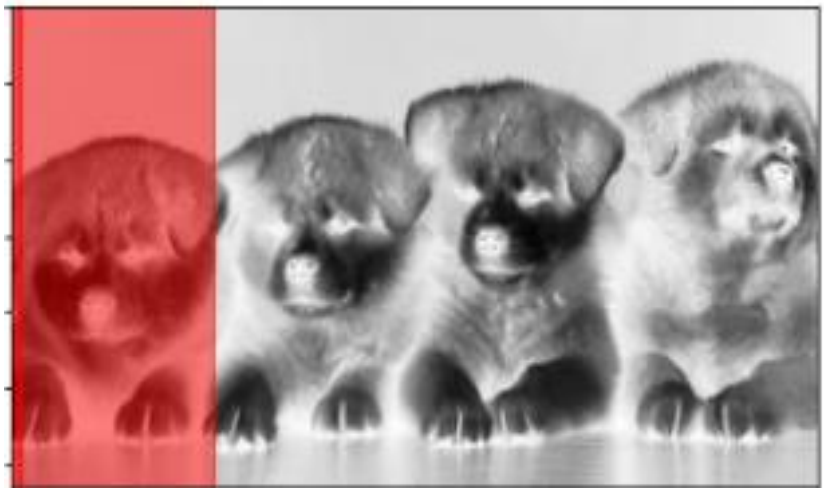


Рисунок 3.15 – Фрагмент з відстанню 118

Рисунок 3.16 – Фрагмент з відстанню 150

Гістограми подібні тоді, коли в ковзаюче вікно вміщається зображення першого (рис. 3.15), третього (рис. 3.13) цуценя та проміжні зображення між першим та другим (рис. 3.16), а також між третім та четвертим (рис. 3.14).

Зображення першого та третього цуценят були правильно класифіковані, але проміжні зображення між першим та другим і між третім та четвертим цуценятами виявилися більш схожими на еталон, ніж зображення другого і четвертого цуценят згідно з порівнянням гістограм.

Це виникає через похибки у кластеризації, внаслідок яких гістограма проекції еталону на себе та гістограма кластерів для еталону не співпадають.

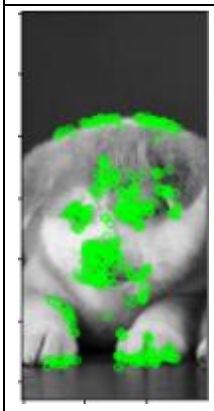
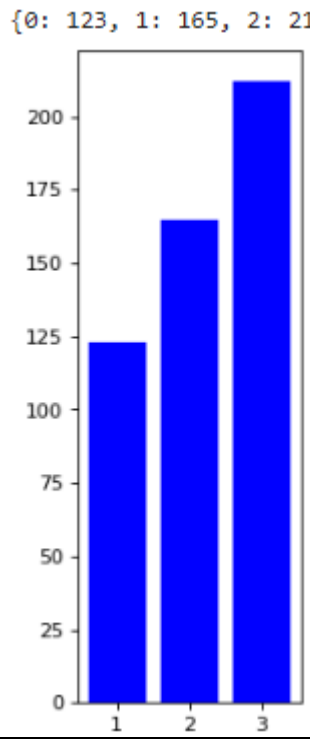
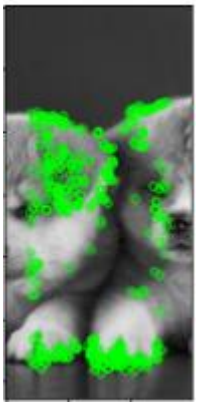
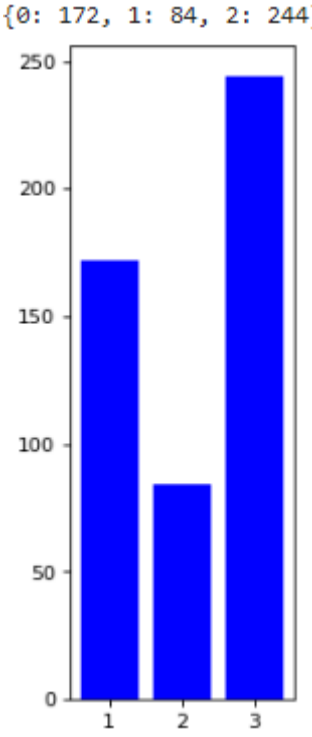
Крім того, для різних початкових центрів були отримані кластери з різними гістограмами. Тобто, тут чітко проявляється недолік алгоритму k-means, а саме, що на результат впливає вибір початкових центрів кластерів.

Є ще одна проблема. Кластеризація завершується на другій ітерації, а якщо умову з допустимою похибкою замінити на умову з обмеженням кількості ітерацій, то після другої ітерації нові центри постійно співпадають зі старими.

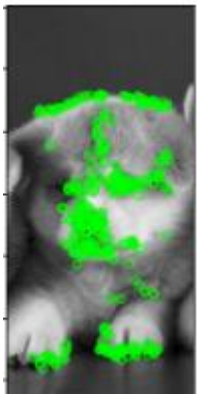
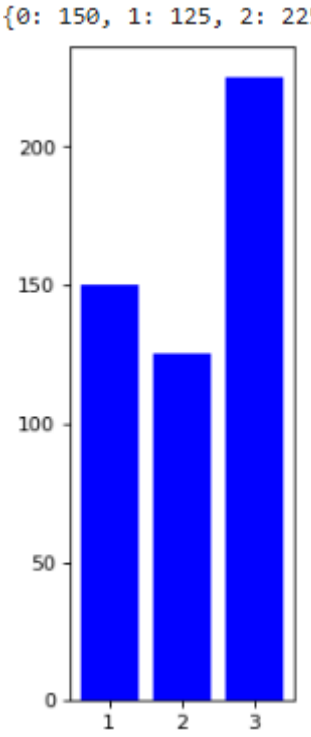
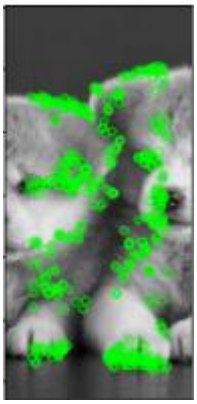
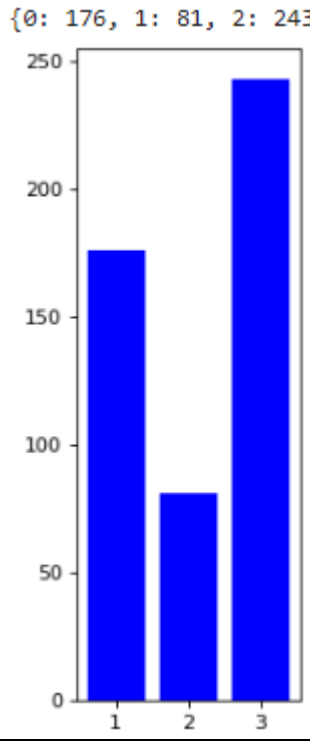
Отже, такий результат не задовольняє, оскільки в самій кластеризації є великі похибки.

Тепер буде розглянуто результати другого експерименту. Цей експеримент має наступну відмінність – використовується класифікація за допомогою стандартної кластеризації *k*-means, де елементи векторів розглядаються як числові атрибути даних для 500 контрольних точок). Ці результати можна побачити в табл. 3.2 і вони суттєво відрізняються від результатів попереднього експерименту.

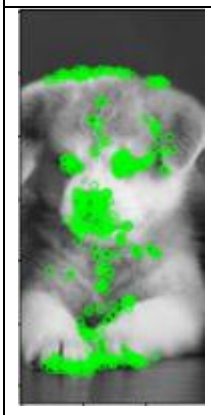
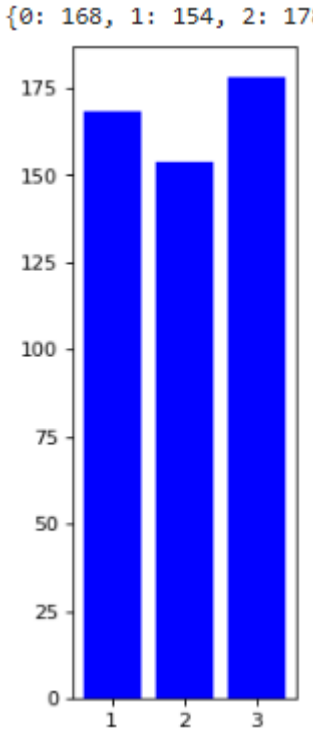
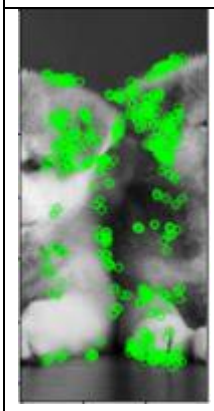
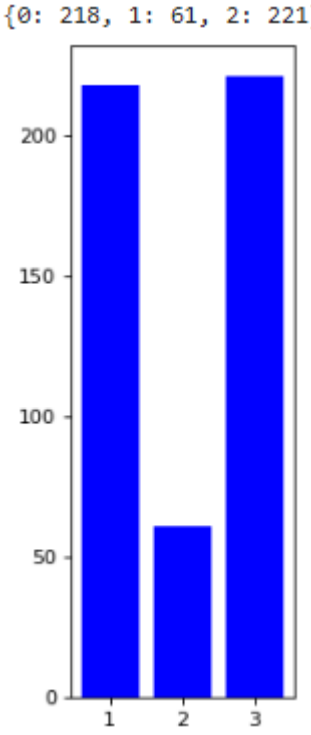
Таблиця 3.2 – Результати експерименту

Фрагмент зображення	**Гістограма**	**Манхеттенська відстань від гістограми еталону**
1	2	3
		0
		162

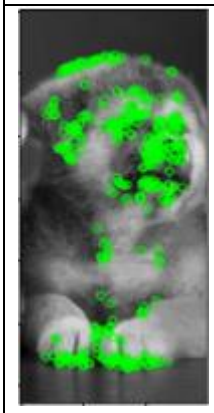
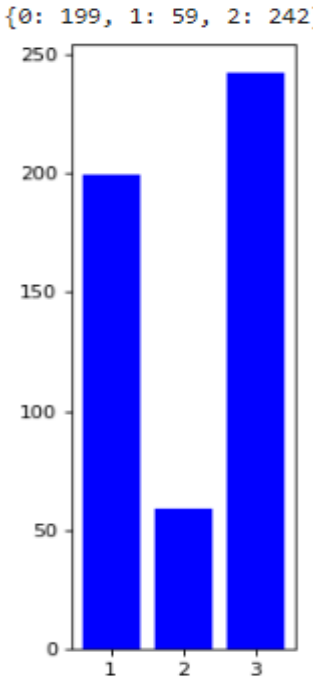
Продовження таблиці 3.2

1	2	3
		80
		168

Продовження таблиці 3.2

1	2	3
		90
		208

Кінець таблиці 3.2

1	2	3
		212

Далі відображено 4 частини з найменшою манхеттенською відстанню (рис. 3.17 – рис. 3.20).

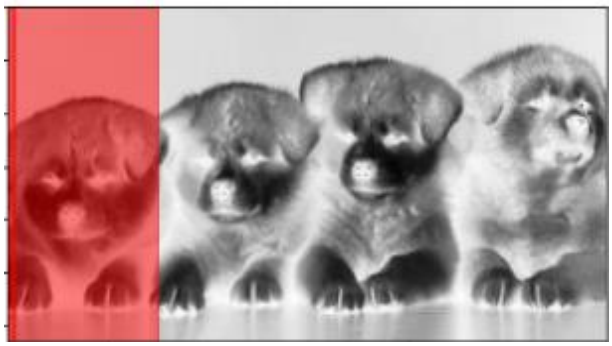


Рисунок 3.17 – Фрагмент з відстанню 0

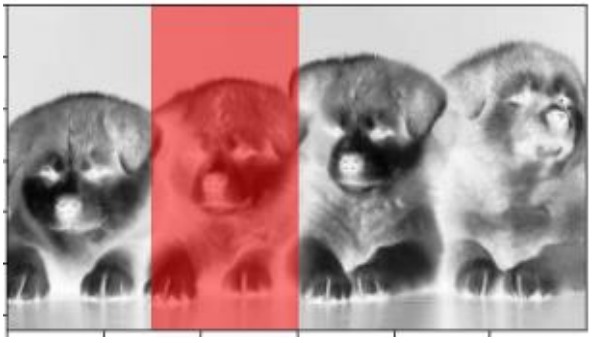


Рисунок 3.18 – Фрагмент з відстанню 80

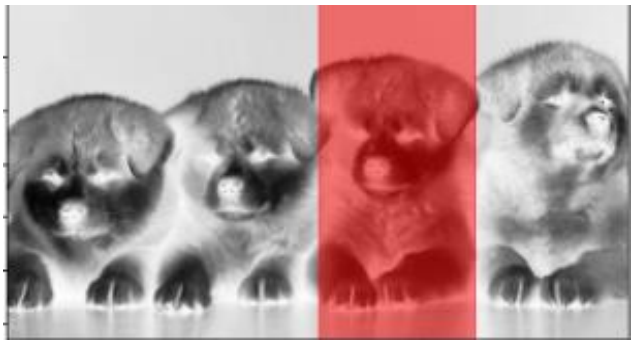


Рисунок 3.19 – Фрагмент з відстанню 90

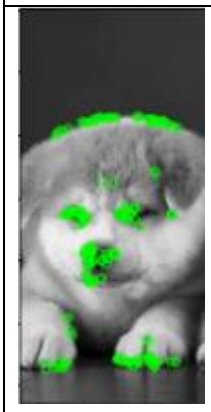
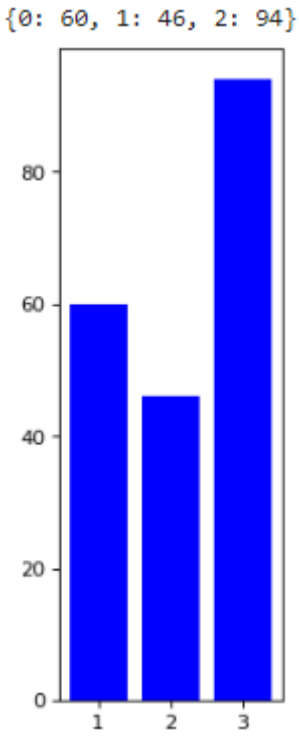
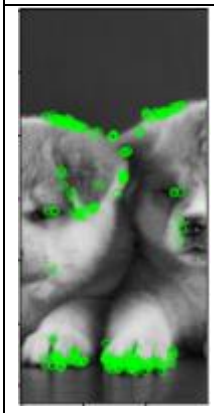
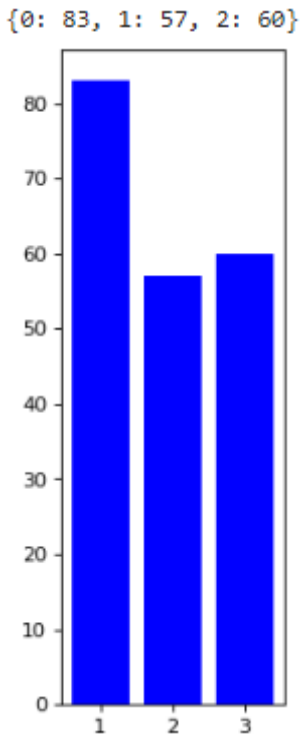
Рисунок 3.20 – Фрагмент з відстанню 162

Як бачимо, тут результат уже кращий – зображення перших 3-х цуценят (рис. 3.17 – рис. 3.19) віднесло до еталону. Але зображення останнього не було віднесено до еталону.

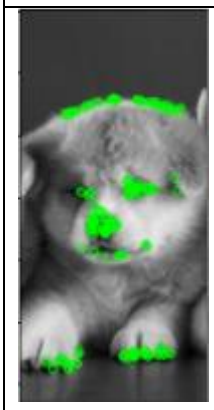
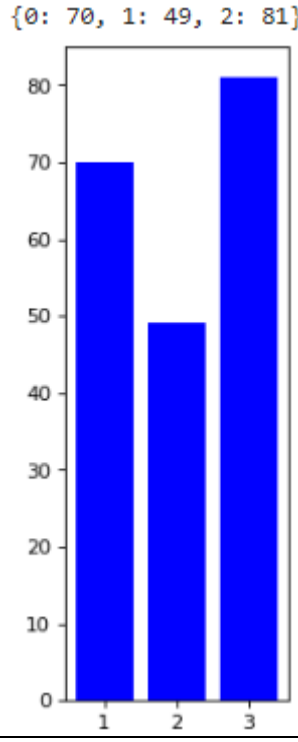
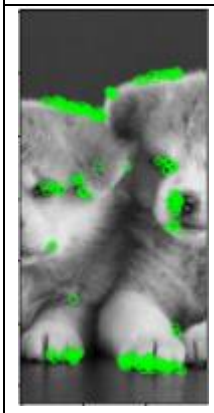
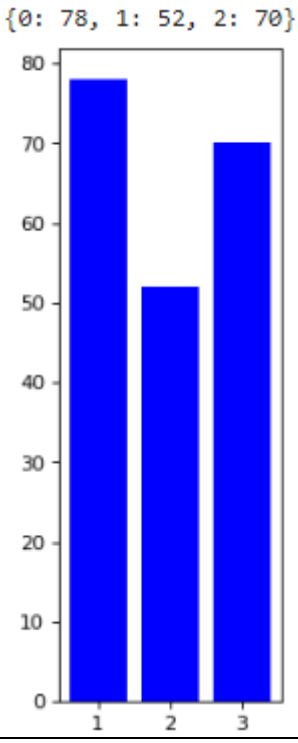
Натомість проміжне зображення між першим та другим (рис. 3.20) цуценятами виявилося більш схожим на еталон (четверте за найменшою відстанню), на відміну від зображення останнього цуценя. Проте тут про похибку кластеризації говорити не варто, оскільки гістограма дескрипторів еталону та гістограма проекції дескрипторів першого фрагменту на еталон співпадають. Можлива причина – знаходження особливих точок на зображенні четвертого цуценя та похибки ORB детектора.

Тепер розглянемо результат (табл. 3.3) третього експерименту (аналогічний другому, тільки для 200 контрольних точок).

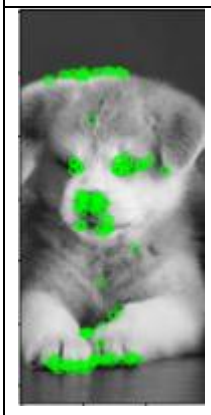
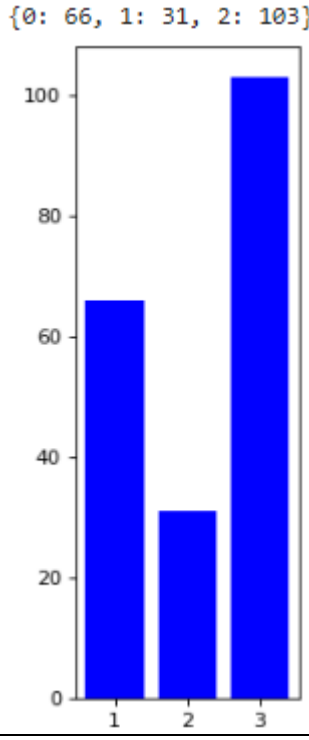
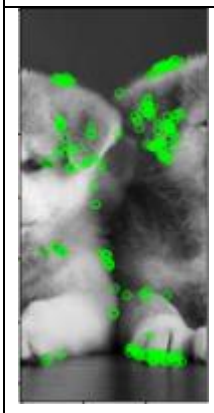
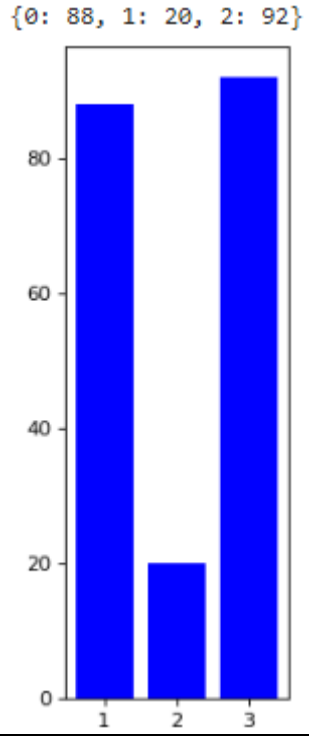
Таблиця 3.3 – Результати експерименту

Фрагмент зображення	**Гістограма**	**Манхеттенська відстань від гістограми еталону**
1	2	3
		0
		68

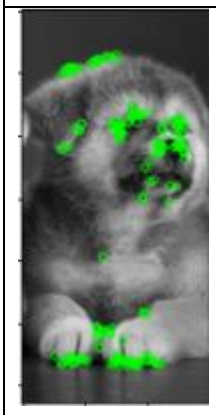
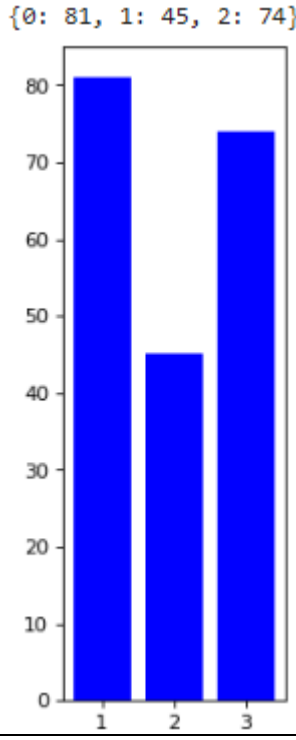
Продовження таблиці 3.3

1	2	3
		26
		48

Продовження таблиці 3.3

1	2	3
		30
		56

Кінець таблиці 3.3

1	2	3
		42

Далі знову було відображено 4 частини з найменшою манхеттенською відстанню (рис. 3.21 – рис. 3.24).

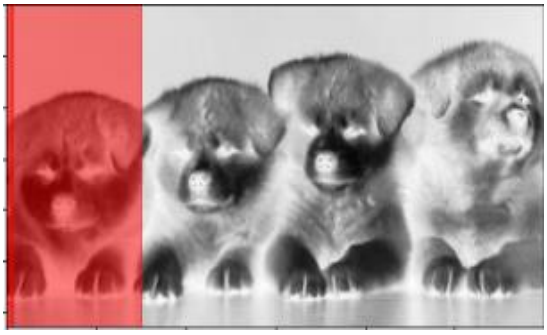


Рисунок 3.21 – Фрагмент з відстанню 0

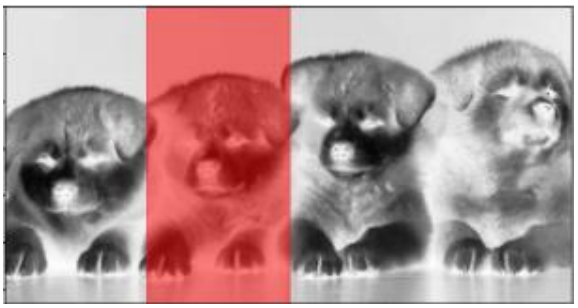


Рисунок 3.22 – Фрагмент з відстанню 26

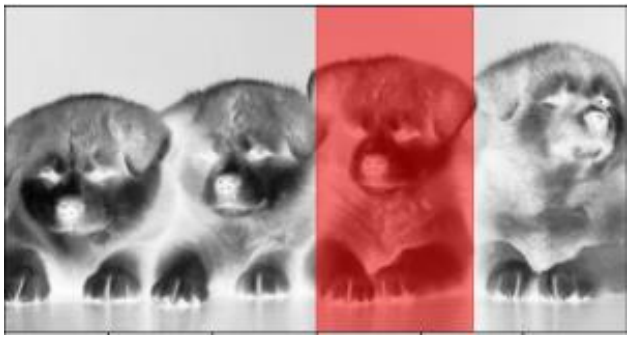


Рисунок 3.23 – Фрагмент з відстанню 30

Рисунок 3.24 – Фрагмент з відстанню 42

На цей раз результат уже практично ідеальний – усі зображення цуценят (рис. 3.21 – рис. 3.24), навіть останнє, були правильно класифіковані. Але зі зменшенням кількості контрольних точок в цілому точність падає, що власне видно по тому, що гістограми менше відрізняються одна від одної. В нашому випадку точність впала несуттєво, оскільки в четвірку схожих на еталон зображень не потрапило жодне проміжне, до того ж, було віднесено до еталону зображення відвернутого цуценя.

ВИСНОВКИ

У рамках атестаційної роботи досліджено метод класифікації зображень на основі вагових характеристик. Було вивчено теоретичний матеріал, підібрано алгоритми, розроблено метод досліджень. Програмно реалізовано три експерименти з класифікацією зображень. Ці експерименти відрізняються лише способом кластеризації та кількість контрольних точок. У ході цих експериментів було обчислено та кластеризовано бінарні дескриптори еталону, після чого для фрагментів тестового зображення бінарні дескриптори та проекції цих дескрипторів на кластери для дескрипторів еталону (віднесення векторів із дескрипторів до кластерів еталону). Були порівняні гістограми проекцій та кластерів еталону, а також знайдені манхетенські відстані між ними. Із результатів цього дослідження випливає наступне: при кластеризації методом *k*-means краще розглядати бінарні дескриптори як набір даних, де 256 біт – це окремі атрибути з числовими значеннями 0 або 1. Тут є наступне зауваження – центри кластерів не є векторами із бінарних дескрипторів, але точність такої кластеризації – краща, оскільки гістограми кластерів і проекції еталону на себе співпадають, що було доведено під час експерименту. Щодо кількості контрольних точок – якщо брати максимально можливу їх кількість, то можна відрізнити дуже схожі зображення, але в такому випадку зображення, які відрізняються лише тим, що об’єкти, які на низ зображені, направлені в різні сторони, не класифікуються, як однакові. Відповідно для класифікації зображень, коли зображення з одного класу мають незначні відмінності, то кількість контрольних точок потрібно зменшити в межах допустимого, а якщо треба перевірити повну ідентичність зображень (наприклад, QR-код, чи інші зображення, які повинні повністю співпадати) краще брати максимально можливу кількість КТ, але щоб їхня кількість була однаковою, як для еталону, так і для зображень.

Результати були апробовані на 24-ому Міжнародному молодіжному форуму [43].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Image Features for Classification Characterization and Retrieval. RSIP Vision. (2020). Retrieved 8 September 2020, from https://www.rsipvision.com/image-features-for-classification.

2. Гороховатский, В. А., & Путятин, Е. П. (2008). Структурное распознавание изображений на основе моделей голосования признаков характерных точек. Реєстрація, зберігання і обробка даних.

3. Выделение особенностей на изображении (Highlighting features in the image) — Моделирование и распознавание 2D/3D образов. (Modeling and recognition of 2D/3D images). Api-2d3d-cad.com. (2020). Retrieved 10 September 2020, from https://api-2d3d-cad.com/features.

4. Гороховатский, В. А. (2017). Методы определения релевантности изображений на основе медианной обработки структурных описаний. Радіоелектроніка, інформатика, управління, (1), 100 – 106.

5. Introduction to Harris Corner Detector. Medium. (2020). Retrieved 10 September 2020, from https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6.

6. SIFT (Scale-invariant feature transform). Medium. (2020). Retrieved 12 September 2020, from https://towardsdatascience.com/sift-scale-invariant-feature-transform-c7233dc60f37.

7. Introduction to SIFT (Scale Invariant Feature Transform). Medium. (2020). Retrieved 12 September 2020, from https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40.

8. Introduction to ORB (Oriented FAST and Rotated BRIEF). Medium. (2020). Retrieved 5 October 2020, from https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf.

9. Introduction to SURF (Speeded-Up Robust Features). Medium. (2020). Retrieved 5 October 2020, from https://medium.com/data-breach/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e/.

10. Гороховатский, В. А., & Полякова, Т. В. (2018). Применение пространственных структур признаков для классификации изображений в компьютерном зрении.

11. Гороховатский, В. А. (2014). Структурный анализ и интеллектуальная обработка данных в компьютерном зрении.

12. Introduction to FAST (Features from Accelerated Segment Test). Medium. (2020). Retrieved 7 October 2020, from https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65.

13. Introduction to BRIEF (Binary Robust Independent Elementary Features). Medium. (2020). Retrieved 7 October 2020, from https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6.

14. Gorokhovatskyi, V. O., & Gadetska, S. V. (2019). Determination of Relevance of Visual Object Images by Application of Statistical Analysis of Regarding Fragment Representation of their Descriptions. Telecommunications and Radio Engineering, 78(3).

15. Гороховатський, О. В., Пупченко, Д. В., & Солодченко, К. Г. (2018). Аналіз властивостей, характеристик та результатів застосування новітніх детекторів для визначення особливих точок зображення.

16. Gorokhovatskiy, V. A. (2016). Efficient Estimation of Visual Object Relevance during Recognition through their Vector Descriptions. Telecommunications and Radio Engineering, 75(14).

17. Гадецька, С. В, & Гороховатський., В. О., & Стяглик, Н. І. (2020) Вивчення критеріїв інформативності даних при впровадженні апарату дерев рішень у методах структурної класифікації зображень. Радіоелектроніка, інформатика, управління, (3), 78 – 87.

18. Gorokhovatskyi, O., Gorokhovatskyi, V., & Peredrii, O. (2018). Analysis of Application of Cluster Descriptions in Space of Characteristic Image Features. Data, 3(4), 52.

19. Гороховатський, В. О., & Пономаренко, Р. П. (2020). Класифікація зображень на підставі формування незалежної системи кластерів у складі структурних описів бази еталонів.

20. Гороховатский, В. А., Путятин, Е. П., & Столяров, В. С. (2017). Исследование результативности структурных методов классификации изображений с применением кластерной модели данных. Радіоелектроніка, інформатика, управління, (3), 78 – 85.

21. Гороховатський, В. А., & Куликов, Ю. А. (2010). Модели обработки дескрипторов характерных признаков изображений на основе анализа гистограмм. Системи обробки інформації, (9), 145 – 148.

22. Гороховатский В.А. Распознавание изображений в условиях неполной информации/ В.А. Гороховатский.– Х.: ХНУРЭ, 2003. – 112с.

23. K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks. Medium. (2020). Retrieved 7 October 2020, from https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

24. Sharma, P. (2020). Distance Metrics | Different Distance Metrics In Machine Learning. Analytics Vidhya. Retrieved 11 October 2020, from https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/.

25. Christoph Ruegg, J. (2019). Distance Metrics. Numerics.mathdotnet.com. Retrieved 13 October 2020, from https://numerics.mathdotnet.com/Distance.html.

26. Advantages of K-Means Clustering – Automatic Addison. Automaticaddison.com. (2020). Retrieved 11 October 2020, from https://automaticaddison.com/advantages-of-k-means-clustering.

27. k-Means Advantages and Disadvantages | Clustering in Machine Learning. Google Developers. (2020). Retrieved 11 October 2020, from https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages.

28. Clustering Algorithms: A One-Stop-Shop. Medium. (2019). Retrieved 13 October 2020, from https://towardsdatascience.com/clustering-algorithms-a-one-stop-shop-6cd0959f9b8f.

29. Fuzzy C-Means Clustering Algorithm – Datanovia. Datanovia. (2018). Retrieved 13 October 2020, from https://www.datanovia.com/en/lessons/fuzzy-clustering-essentials/fuzzy-c-means-clustering-algorithm/.

30. Data Clustering Algorithms – Fuzzy c-means clustering algorithm. Sites.google.com. (2019). Retrieved 15 October 2020, from https://sites.google.com/site/dataclusteringalgorithms/fuzzy-c-means-clustering-algorithm.

31. Gorokhovatskyi V.A. Employment of Intelligent Technologies in Multiparametric Control Systems/ V.A. Gorokhovatskyi, A.A. Zamula // Telecommunications and Radio Engineering. – 2016, Vol. 75, No 19. – P. 1775–1785.

32. 10 reasons why data scientists love Jupyter notebooks | Packt Hub. Packt Hub. (2020). Retrieved 20 October 2020, from https://hub.packtpub.com/10-reasons-data-scientists-love-jupyter-notebooks/.

33. Google Colab (A free GPU enabled Jupyer Notebook) — Pros and Cons : learnpython. Reddit.com. (2020). Retrieved 20 October 2020, from https://www.reddit.com/r/learnpython/comments/7wkrnl/google_colab_a_free_gpu_enabled_jupyer_notebook.

34. Benefits of using Google Colab (Python). Medium. (2020). Retrieved 20 October 2020, from https://medium.com/@sagihaider/benefits-of-using-google-colab-python-8f246c91bc52.

35. 7 Advantages of Using Google Colab for Python. Medium. (2020). Retrieved 20 October 2020, from https://medium.com/python-in-plain-english/7-advantages-of-using-google-colab-for-python-82ac5166fd4b.

36. The Pros and Cons of Using Jupyter Notebooks as Your Editor for Data Science Work. Medium. (2020). Retrieved 21 October 2020, from

https://medium.com/better-programming/pros-and-cons-for-jupyter-notebooks-as-your-editor-for-data-science-work-tip-pycharm-is-probably-40e88f7827cb.

37. ORB feature detector and binary descriptor — skimage v0.12.2 docs. Scikit-image.org. (2020). Retrieved 21 October 2020, from https://scikit-image.org/docs/0.12.x/auto_examples/features_detection/plot_orb.html.

38. K-Means Clustering Implementation in Python. Kaggle.com. (2019). Retrieved 21 October 2020, from https://www.kaggle.com/andyxie/k-means-clustering-implementation-in-python.

39. Python, R. (2020). K-Means Clustering in Python: A Practical Guide – Real Python. Realpython.com. Retrieved 21 October 2020, from https://realpython.com/k-means-clustering-python.

40. sklearn.cluster.KMeans — scikit-learn 0.23.2 documentation. Scikit-learn.org. (2020). Retrieved 21 October 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

41. Гороховатский, В. А., & Передрий, Е. О. (2009). Корреляционные методы распознавания изображений путем голосования систем фрагментов. Радіоелектроніка, інформатика, управління, (1 (20)).

42. Гороховатский, В. А. (2008). Иерархия пространственных отношений структурных признаков в задачах сопоставления визуальных объектов. Системи управління, навігації та зв'язку.—К.: Центральний наук.-досл. інститут навігації і управління, 85-89.

43. Санжаровський, А.В. & Гороховатський, В.О. (2020). Математичне моделювання при розробленні високонавантажених програмних систем. XXIV Міжнародний молодіжний форум Радіоелектроніка та молодь у XXI столітті. Зб. Матеріалів форуму. Т.7. – Харків: ХНУРЕ, 8–9.