**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра прикладної математики**

«До захисту допущено»

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Р. Чертов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

з напряму підготовки 6.040301 «Прикладна математика»

на тему: Автоматизована система пошуку дублікатів зображень

Виконав: студент IV курсу, групи КМ-42

|  |  |  |
| --- | --- | --- |
|  | Волк Ілля Михайлович | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Керівник | асистент  Дрозденко О. М. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Консультант зі спеціальних питань | старший викладач  Мальчиков В.В. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Консультант із нормоконтролю | старший викладач Мальчиков В. В. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Рецензент | к-т техн. наук, доцент  Сапсай  Т. Г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Засвідчую, що в цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.

Волк І.М. \_\_\_\_\_\_\_\_\_\_\_\_

Київ — 2018**Національний технічний університет України**

**«Київський політехнічний інститут  
імені Ігоря Сікорського»**

Факультет прикладної математики

Кафедра прикладної математики

Рівень вищої освіти — перший (бакалаврський)

Напрям підготовки 6.040301 «Прикладна математика»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Р. Чертов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

Волку Іллі Михайловичу

1. Тема роботи: «Автоматизована система пошуку дублікатів зображень»,

керівник роботи Дрозденко Олександр Миколайович, асистент,

затверджені наказом по університету від «06» квітня 2018 р. № 1108-С.

2. Термін подання студентом роботи: «19» червня 2018 р.

3. Вихідні дані до роботи: розроблювана система повинна працювати з зображеннями, до яких застосовані фільтри та афінні перетворення.

4. Зміст роботи: виконати аналіз існуючих методів розв’язання задачі, вибрати метод пошуку дублікатів зображень, спроектувати автоматизовану систему пошуку схожих зображень, здійснити програмну реалізацію розробленої системи, провести тестування розробленої системи.

5. Перелік ілюстративного матеріалу: таблиці порівняння, послідовності виконання кроків методу, знімки екранних форм.

6. Консультанти розділів роботи:

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| Розділ 3. Математичне забезпечення | Мальчиков В.В. старший викладач |  |  |

7. Дата видачі завдання: «19» лютого 2018 р.

Календарний план

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
| --- | --- | --- | --- |
| 1 | Огляд літератури за тематикою та збір даних | 12.11.2018 |  |
| 2 | Проведення порівняльного аналізу математичних методів розпізнавання зображень | 14.12.2018 |  |
| 3 | Проведення порівняльного аналізу математичних методів визначення схожості зображень | 24.12.2018 |  |
| 4 | Підготовка матеріалів першого розділу роботи | 01.02.2018 |  |
| 5 | Розроблення математичного забезпечення для визначення схожості зображень | 01.03.2018 |  |
| 6 | Підготовка матеріалів другого розділу роботи | 15.03.2018 |  |
| 7 | Підготовка матеріалів третього розділу роботи | 05.04.2018 |  |
| 8 | Розроблення програмного забезпечення для пошуку схожих зображень | 15.04.2018 |  |
| 9 | Підготовка матеріалів четвертого розділу роботи | 03.05.2018 |  |
| 10 | Оформлення пояснювальної записки | 01.06.2018 |  |

|  |  |  |
| --- | --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Волк І.М. |
| Керівник роботи | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Дрозденко О.М. | |

Анотація

Дипломну роботу виконано на 58 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 12 найменувань. У роботі наведено 19 рисунків та 3 таблиці.

Метою даної дипломної роботи є створення автоматизованої системи виявлення схожих зображень.

У роботі проведено аналіз існуючих рішень указаної задачі —методів на основі водяного знака та на основі вмісту зображення. Було вибрано метод на основі перетворення Радона. Розроблено автоматизовану систему, що реалізує обраний метод. Виконано тестування розробленої системи.

Ключові слова: зображення, схожість, відбиток, створення відбитку, пошук схожих зображень.

ABSTRACT

The thesis is presented in 58 pages. It contains 2 appendixes and bibliography of 12 references. 19 figures and 3 tables are given in the thesis.

The goal of the thesis is to develop mathematical and software tools for solving the problem of duplicate image search.

In the thesis, existing solutions are analyzed, such as content-based methods and watermarking. Radon transform based method was chosen. An automated system implementing and choosed method was developed. The developed system is tested.

Keywords: image, similarity, fingerprint, fingerprinting, duplicate image search.

ЗМІСТ

[Вступ 8](#_Toc515792216)

[1. Постановка задачі 9](#_Toc515792217)

[2. Аналіз існуючих програмних рішень 10](#_Toc515792218)

[2.1 Огляд існуючих систем 10](#_Toc515792219)

[2.1.1 Tineye 10](#_Toc515792220)

[2.1.2 Dup Detector (Keronsoft). 12](#_Toc515792221)

[2.1.3 Image Comparer 13](#_Toc515792222)

[2.1.4 Google image search 14](#_Toc515792223)

[2.1.5 Порівняння існуючих програмних рішень 15](#_Toc515792224)

[2.2 Огляд методів 16](#_Toc515792225)

[2.2.1 Пошук на основі водяного знака 16](#_Toc515792226)

[2.2.2 Пошук особливостей 17](#_Toc515792227)

[2.2.2.1 Методи на основі відбитку. 18](#_Toc515792228)

[2.2.3 Порівняння програмних рішень для розв’язання задачі 23](#_Toc515792229)

[2.3 Висновки до розділу 24](#_Toc515792230)

[Математичне забезпечення 26](#_Toc515792231)

[3.1 Опис обраного методу 26](#_Toc515792232)

[3.1. Висновки до розділу 31](#_Toc515792233)

[4. Програмне забезпечення 32](#_Toc515792234)

[4.1. Структура програми 32](#_Toc515792235)

[4.2. Результати випробування 33](#_Toc515792236)

[4.3. Висновки до розділу 39](#_Toc515792237)

[Висновки 40](#_Toc515792238)

[Перелік посилань 41](#_Toc515792239)

[Додаток А Лістинг програми 43](#_Toc515792240)

[Додаток Б Ілюстративний матеріал 52](#_Toc515792241)

Вступ

Швидкий ріст обсягу даних які можна зберігати на персональних носіях, доступність засобів фотографування спричинили накопичення фотографій на персональних комп’ютерах. Це ускладнює пошук і перегляд зображень. Також, з’являється проблема коли не вистачає місця для нових зображень. Таке відбувається при фотографуванні одного і того ж місця декілька разів, при подальшій обробці зображення: використання фільтрів, підвищення контрастності, зміні розмірів.

Для вирішення цих проблем використовують програмне забезпечення для пошуку схожих зображень. Великі системи виконують пошук лише в мережі, а ті, що працюють на персональних комп’ютерах, використовують слабкі методи. Методи на основі водяних знаків використовується для пошуку підробок, для пошуку схожих зображень на персональному комп’ютері найбільше підходять методи на основі аналізу вмісту, а саме методи на основі цифрових відбитків, адже інші методи вимагають деякої підготовки, а методи цифрових відбитків побудовані на завчасно визначених алгоритмах. Програми, побудовані на таких методах, порівнюють зображення лише поверхнево, або є хмарними комерційними рішеннями. Саме тому актуальною є проблема створення програмного забезпечення для виявлення дублікатів зображень на персональному комп’ютері.

# Постановка задачі

Метою даної дипломної роботи є створення автоматизованої системи для виявлення схожих зображень.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Огляд існуючих систем.
2. Вибір методу і його алгоритмізація.
3. Програмна реалізація вибраного методу.
4. Тестування створеного ПО.

Вимоги до вхідних даних:

1. Зображення:
2. Формат: JPEG, PNG.
3. Розмір: до 1440×2560 пікселів.

Розроблена система повинна виконувати наступні функції:

1. Працювати на персональному комп’ютері користувача
2. Показувати звіт про свою роботу

Розроблена система повинна вважати схожими зображення з такими відмінностями:

1. Використання фільтрів
2. Поворот
3. Зсув
4. Зміна розмірів
5. Додавання об’єктів

# 2. Аналіз існуючих програмних рішень

## Огляд існуючих систем

### Tineye

Tineye – сервіс для пошуку візуально схожих зображень в колекціях. Має глобальну колекцію зображень з мережі, комерційна версія надає можливість створювати власні колекції.

Призначення сервісу – пошук дублікатів, модифікованих копій, перевірка зображення на ідентичність.

Технологія побудована на отриманні відбитка – невеликої кількості даних, що описують особливості, знайдені в зображенні. Відбитки зберігаються в їх внутрішніх базах даних. Для додавання зображення і пошуку використовуються виклики API. Технологія здатна скористатися схожістю частини відбитка.

Можливості системи:

1. знаходить ідентичні зображення;
2. знаходить зображення, що мають схожу частину (Рис. 2.1.1.1);
3. шукає зображення в мережі;
4. не вважає схожими унікальні зображення (Рис. 2.1.1.2);

Недоліки:

1. посилання на публічне зображення не актуальне;
2. використання власних зображень для пошуку платне;
3. відсутність «агента» - для використання власних зображень спочатку їх потрібно завантажити в систему;

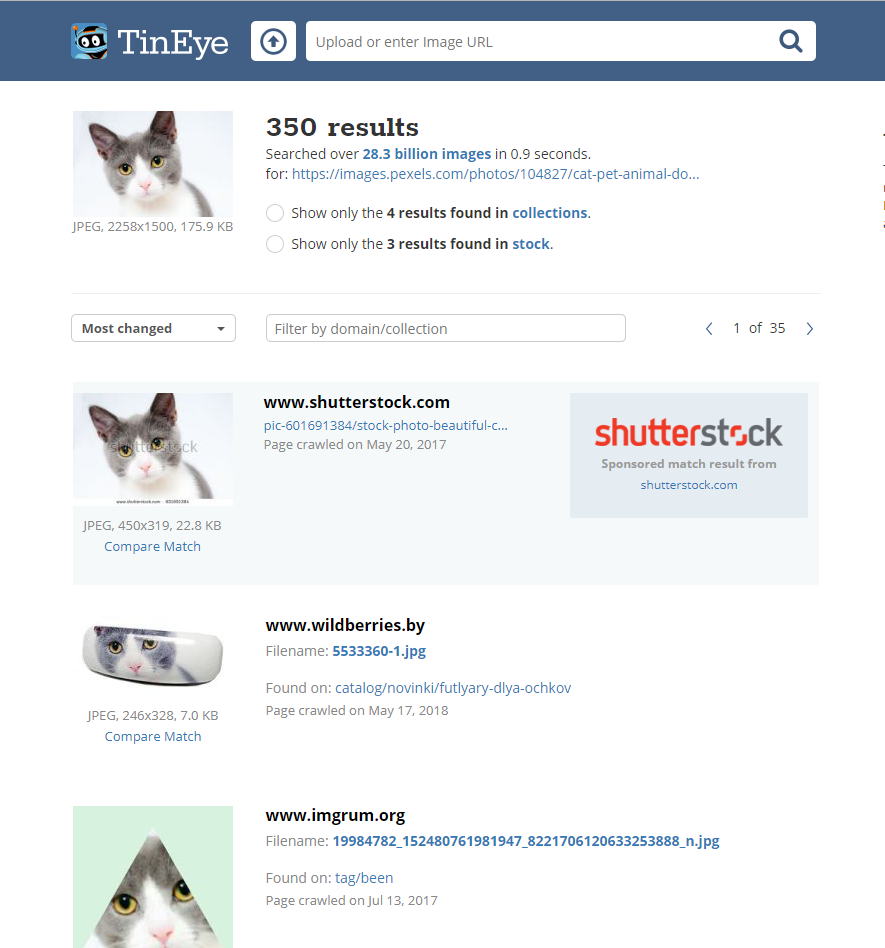


Рис. 2.1.1.1 – Результат пошуку Tineye – зображення з мережі

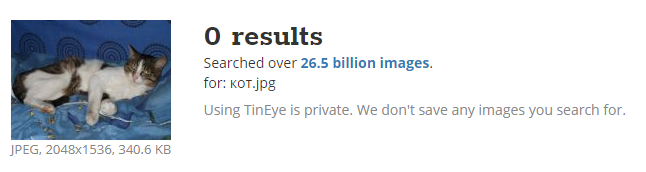


Рис. 2.1.1.2 – Результат пошуку Tineye – унікальне зображення

### Dup Detector (Keronsoft).

Keronsoft – команія, що спеціалізується на дослідженнях цифрових фотографій, відеозаписів, застосуваннях для створення медіаконтенту. Також вона розповсюджує мультимедійне програмне забезпечення.

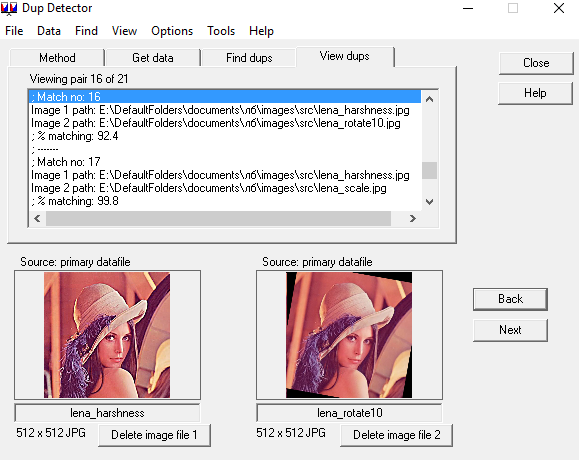


Рис. 2.1.2.1 – Інтерфейс програми Dup Detector

Програма змогла виявити схожість у випадках застосування фільтрів, додавання об’єктів, масштабування, повороту на 10, але не змогла виявити її у випадках зсуву, повороту на 90, 180.

### Image Comparer

Програма для windows для пошуку схожих зображень на машині користувача.

Судячи з результатів її роботи вона використовує ті ж методи що і tineye – на основі отриманні відбитка.

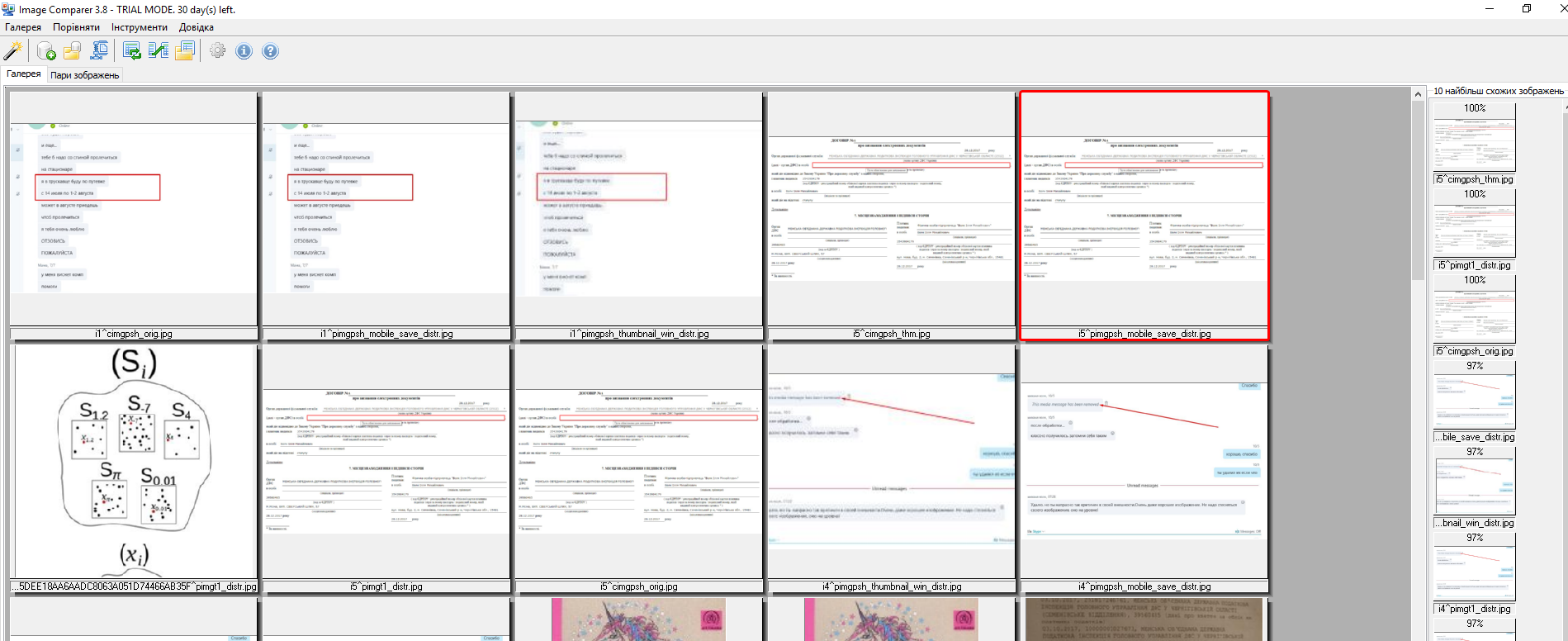


Рис. 2.1.3.1 – Інтерфейс програми Image Comparer

При тестуванні програми виявлено що вона добре визначає схожими масштабовані зображення, зображення з додаванням фільтрів, додаванням об’єктів, але не визначає схожими повернуті та зсунуті зображення.

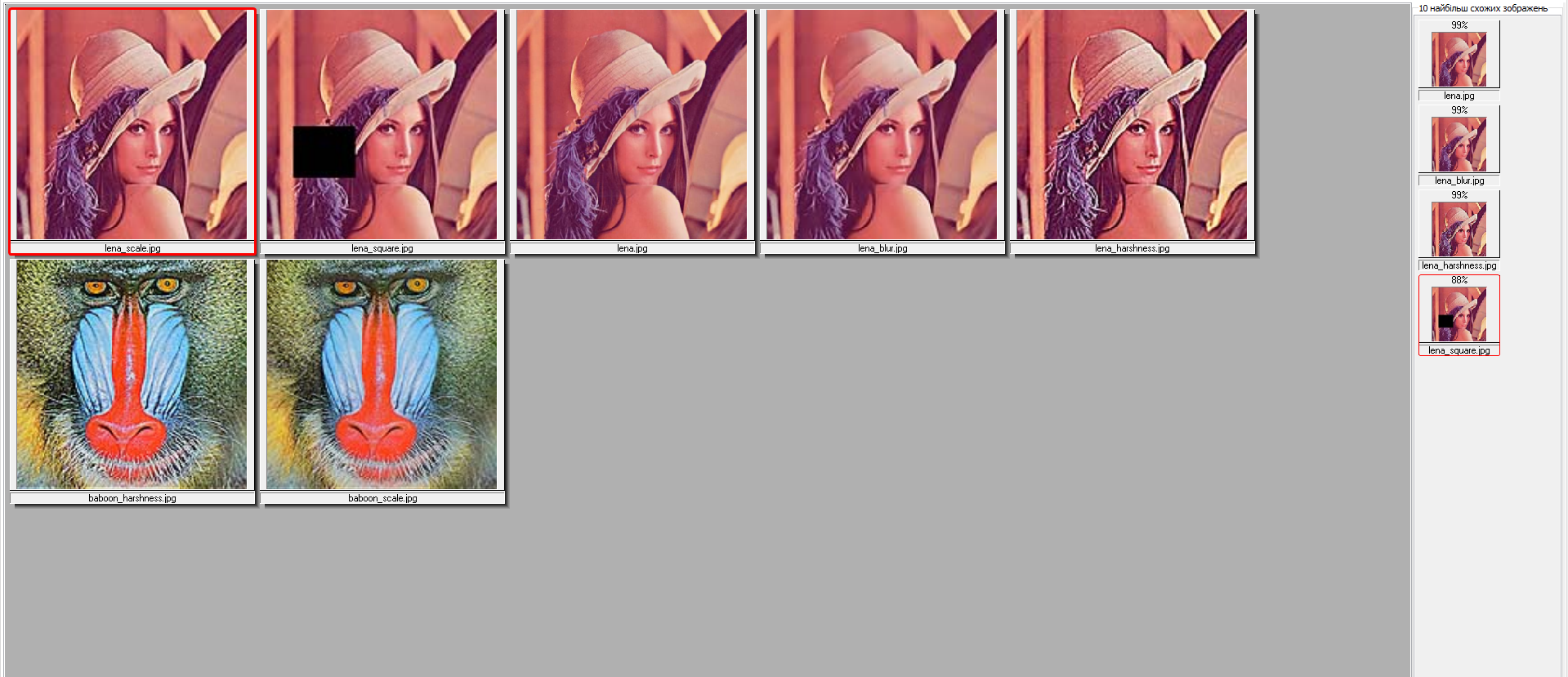


Рис. 2.1.3.2 – Тестування програми Image Comparer на зображеннях lena і baboon

### Google image search

Cервіс для пошуку схожих зображень в мережі. Він знаходить не лише візуально схожі зображення, а й логічно пов’язані. Система розпізнає об’єкти, що знаходяться на зображнні і повертає зображення з такими ж об’єктами.

Пошук побудований на ключових словах. Під час аналізу зображення система з’ясовує характеристики зображення і описує їх за допомогою слів. Результатами пошуку є зображення, опис яких є схожим.

На прикладі до tineye (зображення кота) google image search знаходить зображення, які б людина описала так само як і пошукове. В даному випадку це «Кіт на синьому фоні» (Напевно звіт чорно-білий, прошу повірити). Всі знайдені зображення містять котів і мають близький до синього (бодай не червоний) колір фону.

Рис. 2.1.4.1 – Зображення мого кота Рис. 2.1.4.2 – Результат пошуку Google

### Порівняння існуючих програмних рішень

Основні переваги і недоліки існуючих програмних рішень для пошуку схожих зображень наведено в таблиці 2.1.

Таблиця 2.1 – Порівняння існуючих програмних рішень

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Програмний продукт | Пошук в мережі | Пошук на персональному комп’ютері | Стійкий до афінних перетворень | Виведення всіх схожих зображень і їх місцезнаходження |
| Tineye | + | Платна можливість створення персональної хмарної колекції | + | + |
| Google image search | + | - | + | - |
| Dup Detector | - | + | + | + |
| Image Comparer | - | + | - | + |

## Огляд методів

Існує два протилежні підходи в пошуку схожих зображень – на основі водяного знаку і на основі вмісту. Підхід на основі водяного знака полягає в закарбовуванні підпису в зображенні перед його розповсюдженням. Згодом за цим підписом можна буде визначити чи є зображення дублікатами. З іншого боку, підхід на основі вмісту, як видно з його назви, полягає в аналізі зображення з метою визначення його особливостей.

### Пошук на основі водяного знака

Історично, дублікати шукалися за допомогою методів на основі водяного знаку.

Ідея водяного знака проста: автор зображення додає до нього стійкий непомітний секретний підпис перед його розповсюдженням. Прихований підпис виконує дві функції. По-перше, він дозволяє встановити власника зображення в суперечливих ситуаціях. По-друге, за допомогою підпису можна знайти копії зображення, наприклад в мережі, і водночас визначити чи є вони легальними. [5]

Останнім часом водяні знаки, як засіб захисту інтелектуальної власності, зазнали сильної критики. Херлі [[13]](#b13) розпочав дискусію щодо недоліків водяного знака з суперечливим документом під назвою "Чому водяний знак є нісенітницею"[[13]](#b13). Суть аргументації Херлі полягає в тому, що захищати всі об'єкти в невеликій околиці позначеного об'єкта, як це виконується в багатьох опублікованих алгоритмах на основі водяного знаку, є необхідним, але не достатнім. Він стверджує, що корисний алгоритм водяного знаку повинен захищати всі особливості, а не лише ті, які є близькими до позначеного об'єкта. Інші автори продовжували додавати до цієї дискусії, наприклад, Барні [[14]](#b14) підкреслював, що водяні знаки все ще є молодим напрямом обробки сигналів, і що жоден метод не зміг захистити вміст від усіх можливих атак. Однак, Моулін [[15]](#b15) частково відкинув заяву "водяний знак - нісенітниця" [[14]](#b14), помітивши, що це може бути досить знайти такі перетворення, які б допомогли уникнути виявлення дубліката. Крім того, він зауважив, що водяні знаки були досить корисними в областях з низькою безпекою, наприклад, в кабельній телевізійній системі, і що нові методи можуть ще більше покращити ефективність алгоритмів водяного знаку. [5].

На думку автора [[5],](#b5) водяний знак має важливі недоліки, як це описано в наступному. Хоча водяні знаки можуть бути корисними в певних ситуаціях, воно не може розглядатися як засіб для захисту вмісту у довгостроковій перспективі.

Методи на основі водяного знака мають кілька недоліків:

1. шахрай може видалити підпис з зображення, наприклад за допомогою зміни освітлення зображення, чи застосуванні того ж самого метода додавання водяного знака. Таке зображення можна законно розповсюджувати, метод на основі водяного знака виявиться марним;
2. методи на основі водяного знака вимагають модифікації зображення перед його розповсюдженням. Це не завжди можливо, і деякі автори можуть негативно ставитися до модифікацій [5];

### Пошук особливостей

Ці методи є достатньо молодими. На момент виходу [[5]](#b5) (2007) методи визначення схожості зображень на основі пошуку особливостей лише починали свій розвиток. Більшість робіт не достатньо зріла для визначення якості методів. Їхня продуктивність є низькою, а реалізація складною. Багато робіт не були опубліковані, методи є власністю компаній. Багато з опублікованих робіт протестовані лише поверхнево, для невеликої кількості зображень та перетворень. Далі буде розглянуто деякі з опублікованих робіт і узагальнено їх за такими критеріями як відгук (recall) та точність (precision) [[5]](#b5)

Хоча в кожній роботі і використовуються свої зображення для порівняння, до них застосовуються різні перетворення, все одно ці оцінки будуть вираховуватися виходячи з результатів представлених в самих роботах. Зрештою, більшість робіт використовують типові зображення, представлені на Рис. 2.1.1.1.



Рис. 2.1.1.1 – типові зображення для тестування методів.

### Методи на основі відбитку.

Більшість таких методів побудовані на створенні короткого опису зображення – відбитку.

Відбиток – це короткий опис зображення. Аналогом є криптографічні хеш-функції, які для будь-якого об’єму даних визначають невелику завчасно визначену кількість бітів даних.[[2]](file:///E:\DefaultFolders\Documents\dyp\дипломнаяv2.doc#b2) Але криптографічні функції не можуть бути застосовані для створення відбитків зображень. Над зображеннями виконують різні маніпуляції, такі як стиснення, покращення, геометричні перетворення. Криптографічні функції дуже чутливі: зміна одного біта зображення може значно змінити отриманий хеш. Через це криптографічні функції не можуть бути застосовані для отримання цифрових відбитків. Методи створення таких відбитків повинні дозволяти незначні зміни в зображенні. Щоб усунути ці недоліки ми приходимо до поняття multimedia fingerprinting, його ще називають robust hash functions.[[3]](file:///E:\DefaultFolders\Documents\dyp\дипломнаяv2.doc#b3)

Стійкість (інваріантність за подібністю до сприйняття): відбитки, що виникають внаслідок модифікації зображення, повинні бути схожими на відбиток на оригінальному знімку. Стійкість - це здатність ідентифікувати два подібні об'єкти як схожі. [[1]](file:///E:\\DefaultFolders\\Documents\\dyp\\дипломнаяv2.doc" \l "b1)

Відбитки можна порівнюються різними способами, наприклад, за допомогою норми Хемінга.

2, де f1, 72 – шістнадцяткові числа.



В деяких підходах кожен хеш описує конкретну частину зображення. Опис зображення стає багатшим. Кількість частин, їхнє положення і форма залежать від вмісту зображення. Таких частин може бути від кількох до кількох тисяч. Зображення вважаються схожими якщо кількість співпадаючих хешів є більшою за певний рівень. Такі методи є більш точними. Вони покладаються на конкретні особливості зображення і потребують великої кількості порівнянь.

Хоча ці методи є більш гнучкими ніж методи додавання водяного знака, і не вимагають попередньої модифікації зображення, вони роблять більше помилок ніж методи на основі водяного знака. В той же час вони будуть визначати два зображення як схожі якщо вони мають схожий зміст, навіть якщо вони є різними в сенсі авторства. Також особливості можуть бути визначені по-іншому після модифікації зображення, що спричинить створення різних відбитків.

* + - * 1. Створення відбитків на основі перетворення Радона.

Техніка, винайдена Lafebvre та опублікована в [[8]](#b8), [[9].](#b9) Спочатку застосовуєтся перетворення Радона [[7]](#b7). Потім обчислюється середня точка, інваріантна до подібних перетворень зображення. Відбиток утворюється складанням докупи всіх цих точок. Хеші мають вигляд як на [Рис. 2.2.1.1.1.1](#im2_2_1_1_1_1). Більш того, можна виявити як саме модифікували зображення. Щоб визначити, чи є зображення схожими, спочатку розраховується cross-correlation між хешами, а потім mean square error.

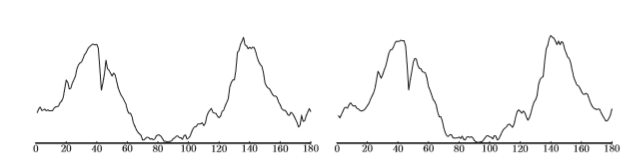


Рис. 2.2.1.1.1.1 – Приклади відбитків методу Lafebvre.

В статті показано, що метод гарно показує себе при застосуванні Гаусівського фільтру, JPER-стиснення на 25%, 15%, масштабування в 0.8, 1.2 разів, повороту на 1 і . За таких умов показник відгуку становить 0.975, а точність – 1. Хоча результати і гарні, кількість протестованих зображень мала, а перетворення, які застосовувалися до зображень, дуже мало змінюють зображення.

* + - * 1. Створення відбитків на основі ключових точок

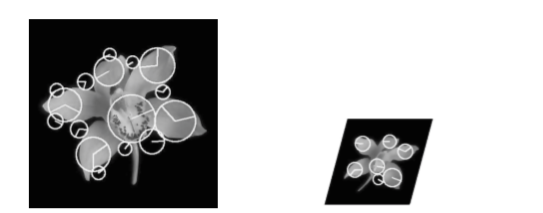
В [[10]](#b10) пропонується метод створення відбитку, що побудований на визначенні особливостей, названих ключовими точками (Key points). Їх властивості описані як «stable in a scale-space representation» [[10].](#b10) Зображення представляється у вигляді тисяч ключових точок. Зображення визначаються як схожі чи не схожі за допомогою співпадінь хешів, обчислених навколо ключових точок. Цей підхід є методом на основі отримання відбитку з використанням багатьох хешів, оскільки зображення представляється у вигляді тисяч частин.

Цей метод отримання відбитку покладається на стійкість ключових точок, які є також способом опису частин зображення в [[11]](#b11). Пошук ключових точок складається з таких етапів:

1. «scale-space maxima detection» [[10]](#b10);
2. локалізація ключових точок;
3. присвоєння напрямку;

Приклад визначених ключових точок зображено на [Рис. 2.2.1.1.2.1.](#im2_2_1_1_2_1)

Цей підхід досягає великої точності 0.99 і відгуку 0.98, правильно визначає зображення зі значними змінами. Але створення одного відбитка потребує кількох секунд, і такий відбиток складається з тисяч 150-елементних векторів. Один відбиток обчислюється кілька секунд, він вимагає багато місця для сберігання та багато часу для порівняння з іншими. Таким чином продуктивність методу є малою.



(а) Оригінальне зображення (б) модифіковане

Рис. 2.2.1.1.2.1 – визначення ключових точок

* + - * 1. Створення відбитків на основі сіткового представлення зображення.

Метод, описаний в [[12]](#b12), побудований на розрізанні зображення на трикутники, що не перетинаються і генерування хеша для кожного трикутника. Метод складається з двох етапів:

1. розділення зображення на трикутники;
2. генерування хеша для кожного трикутника;

Алгоритми надзвичайно складні, можна лише додати приклад розділення на трикутники зображення Lena ([Рис. 2.2.1.1.3.1](#im2_2_1_1_3_1)). Цей метод був протестований на великій базі зображень, його точність і відгук, відповідно, складають 0.82 і 0.82. Швидкість роботи методу прийнятна.



Рис. 2.2.1.1.3.1 – Розділення на трикутники зображення Lena

* + - * 1. Хешування на основі перетворення Радона – 2.

Метод, опублікований в [[1]](#b1). Він детально описаний в наступному розділі. Судячи з результатів, представлених в роботі, відгук=0.998, а точність – 0.02. Точність низька через те, що база зображень містила велику кількість схожих між собою груп зображень, і метод повертав всі зображення з групи. Це може бути зручно, коли зображення містять велику кількість копій і треба знайти їх усі.

### Порівняння програмних рішень для розв’язання задачі

Результати аналізу груп методів наведено в таблиці 2.2., а порівняння методів на основі аналізу вмісту зображення – в таблиці 2.3.

Таблиця 2.2 – Порівняння груп методів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Технологія | Вимагає підготовки технологія | Швидкість роботи | Кількість false-positives | Визначення схожими різних зображень з схожим вмістом |
| Fingerprinting | - | - | -(багато) | + |
| Watermarking | + | + | +(мало) | - |

Таблиця 2.3 – Порівняння груп методів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Точність | Відгук | Швидкість роботи | Якість тестування | Повертає багато можливих дублікатів |
| Радон - 1 | 1 | 0.975 | + | - | - |
| Ключові точки | 0.98 | 0.99 | - | - |  |
| Сіткове представлення | 0.82 | 0.82 | + | + |  |
| Радон - 2 | 0.02 | 0.998 | - | + | + |

## Висновки до розділу

На сьогодні існують програмні засоби для пошуку схожих зображень, проте одні з них – великі системи, що працюють на віддалених серверах, а інші – невеличкі програми з слабкими методами і незручними інтерфейсами, що працюють на персональних комп’ютерах.

В роботі для визначення схожості зображень буде використано метод на основі перетворення Радона через такі переваги:

1. відсутність необхідності підготовки перед роботою;
2. простота реалізації;
3. можливість виявлення сімейства дублікатів;
4. стійкість до застосування фільтрів;

# Математичне забезпечення

## Опис обраного методу

Для реалізації було обрано метод на основі перетворення Радона. [[1]](#b1) [[7]](#b7)

Це метод на основі отримання відбитку.

Цей метод є стійким до афінних перетворень, шумів, розмиття, стиснення.

Перетворення Радона представляє зображення як колекцію проекцій по різних напрямках. Перетворення Радона зображення визначається як інтеграл по лінії, нахиленої на кут вісі на відстані від початку координат. ([Рис. 2.3.1](#im1)) Математично записується так:

,

де .

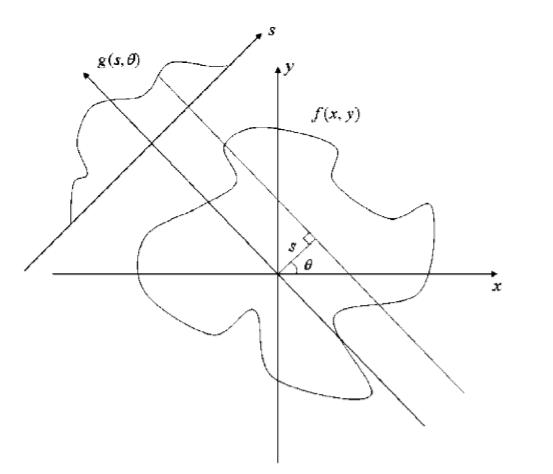


Рис. 2.3.1 – Інтеграл по напрямку

Перетворення Радона має такі корисні властивості, пов’язані з афінними перетвореннями зображення:

1. зсув зображення на викликає зсув перетворення Радона на s:
2. масштабування зображення в разів викликає масштабування перетворення в p разів. ;
3. обертання зображення на кут викликає зсув перетворення:

;

На [Рис. 2.3.3](#im2_3_3) зображено перетворення Радона зображення Lena ([Рис. 2.3.2](#im2_3_2))



Рис. 2.3.2 –Зображення Lena

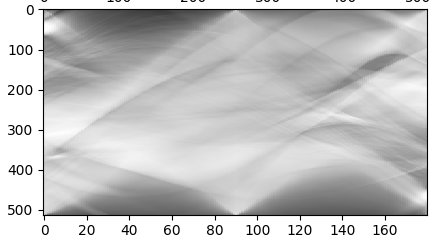
. 

Рис. 2.3.3 – Перетворення Радона зображення Lena

Розглянемо афінні перетворення: зсув, масштабування (зі збереженням кутів) та поворот. З 1 властивості слідує що зсув зображення спричиняє зсув перетворення Радона, але розмір зсуву для кожної проекції є різним. Для забезпечення незмінності зсуву до кожної проекції перетворення застосовується нормалізована автокореляція.

Після використання автокореляції ми отримуємо сигнал , стійкий до зсувів. Серед афінних перетворень залишились масштабування і поворот. З 2 і 3 властивості перетворення Радона, автокореляція масштабованого і повернутого зображення представлена як де та – розмір масштабування та повороту відповідно. Для того щоб досягти стійкості до масштабування і повороту використовується перетворення log-mapping [[4]](#b4) та двомірне перетворення Фур’є. Перетворення log-mapping перетворює масштабування сигналу в зсув. Потім перетворення Фур’є [[6]](#b6) відображає цей зсув в зміну фази. За допомогою log-mapping сигнал може бути записаний як

Потім використовується перетворення log mapping . Воно має перетворити масштабування в зсув. Потім використовується двомірне перетворення Фур’є.

Воно перетворює зсув зображення в зміну фази. ([Рис. 2.3.4](#im2_3_4)).

Рис. 2.3.4 – Послідовність застосування перетворень.

Рис. 2.3.5 – Послідовність отримання відбитку.

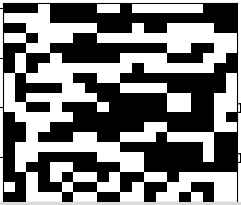


Рис. 2.3.6 – Відбиток зображення Lena

Потім потрібно вибрати перші 21х21 пікселя перетворення Фур’є. Найбільша енергія знаходиться в перших коефіцієнтах, решта є малими.

Потім потрібно розділити комплексні значення на модуль і аргумент.

До отриманих зображень потрібно застосувати 2Д фільтр .

В результуючих зображеннях замінити додатні значення на 1, а <=0 – на 0.

Об’єднати їх за допомогою двійкової функції виключаючого або. ([Рис. 2.3.5](#im2_3_5))

Отримані нулі і одиниці є відбитком. Для його зберігання достатньо 50 байт пам’яті. ([Рис. 2.3.6](#im2_3_6)).

## Висновки до розділу

У розділі розглянути математичне забезпечення системи пошуку схожих зображень. Для визначення схожості зображень використовується метод на основі зображення Радона, який включає в себе перетворення Радона, автокореляцію, log-mapping, перетворення Фур’є.

# Програмне забезпечення

## Структура програми

Програмне забезпечення для розв’язання поставленої задачі складається з 4 частин:

1. ввід даних;
2. обробка вводу;
3. зберігання даних;
4. вивід результатів роботи;

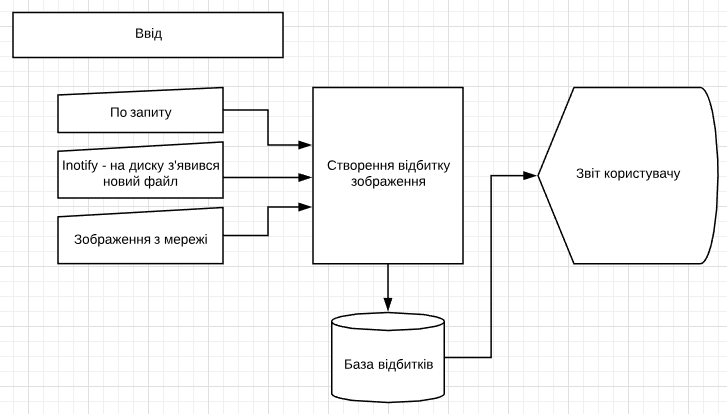


Рис. 4.1 – Схема системи

Наразі система підтримує один вид введення – шляху до директорії на диску персонального комп’ютера по запиту користувача. З вибраної директорії зчитуються файли формату .png та .jpg. Для кожного зображення виконується процедура отримання відбитку.

Детально алгоритм отримання відбитку було описано в попередньому розділі. Алгоритм реалізовано на мові програмування python. Для реалізації деяких кроків було використано бібліотеки skimage, numpy. Оскільки мова має низьку швидкість виконання, до написаних вручну кроків було застосовано jit компіляцію за допомогою бібліотеки numba.

База даних не реалізована. Для полегшення створення інтерфейсу користувача результати обробки зображень зберігаються в .json файлі.

Інтерфейс користувача створений за допомогою бібліотеки kivy.

Формат вхідних даних

* зображення;
* формат: JPEG;
* розмір: до 1440×2560 пікселів;

Формат результуючих даних

* формат: JSON;
* структура:

Array<{

text: string;

image: string;

diffs: Array<{

text: string;

image: string;

}]>

}>

## Результати випробування

* + 1. Контрольний приклад 1. Застосування фільтрів.

Вхідні дані:

1. Зображення lena.jpg.

* Формат зображення: JPEG;
* Розмір зображення: 512×512 пікселів.
* Оригінальне зображення класичного розміру.

1. Зображення lena\_blur.jpg.

* Формат зображення: JPEG;
* Розмір зображення: 512×512 пікселів.
* Зображення lena.jpg з застосуванням фільтру blur.

1. Зображення lena\_harshness.jpg.

* Формат зображення: JPEG;
* Розмір зображення: 512×512 пікселів.
* Зображення lena.jpg з застосуванням фільтру harshness.

1. Зображення lena\_yellow.jpg.

* Формат зображення: JPEG;
* Розмір зображення: 512×512 пікселів.
* Зображення lena.jpg з використанням виду редагування «кольоровий фон», після чого зображення стало більш жовтим.

Розглядаються 4 зображення – оригінальне, і з застосуванням різних фільтрів. Програма визначає зображення lena.jpg схожим на зображення lena\_blur.jpg на 82%, на lena\_harshness.jpg на 77% і на lena\_yellow.jpg на 49%. Результат роботи програми зображено на [Рис. 4.4.1.1](#im4_4_1_1)

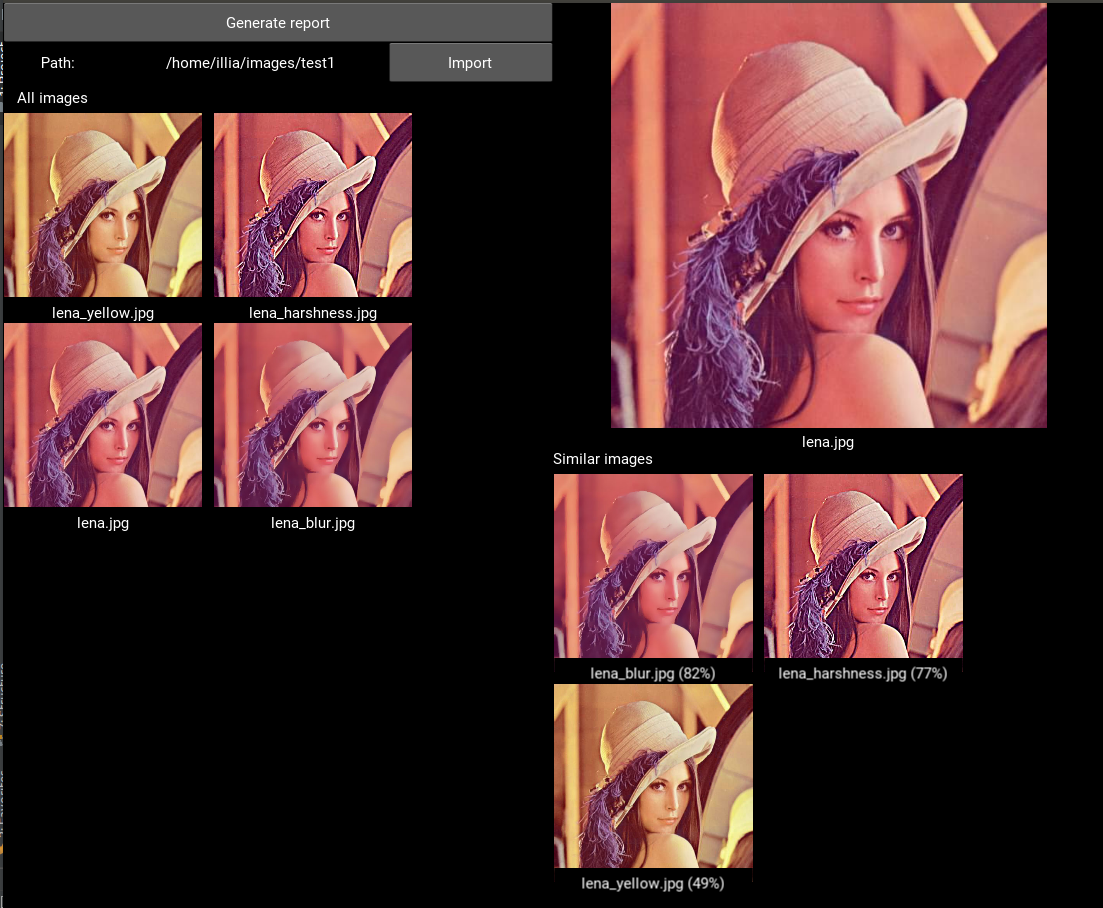


Рис. 4.4.1.1 – Контрольний приклад 1.

* + 1. Контрольний приклад 2. Зміна розмірів.

Вхідні дані:

1. Зображення lena.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* оригінальне зображення класичного розміру;

1. Зображення lena\_scale.jpg.

* формат зображення: JPEG;
* розмір зображення: 256×256 пікселів;
* зображення lena.jpg, стиснене в 4 рази;

Розглядаються оригінальне зображення і масштабоване. Вони визначилися схожими на 53%. Результати роботи програми зображено на [Рис. 4.4.2.1.](#im4_4_2_1)

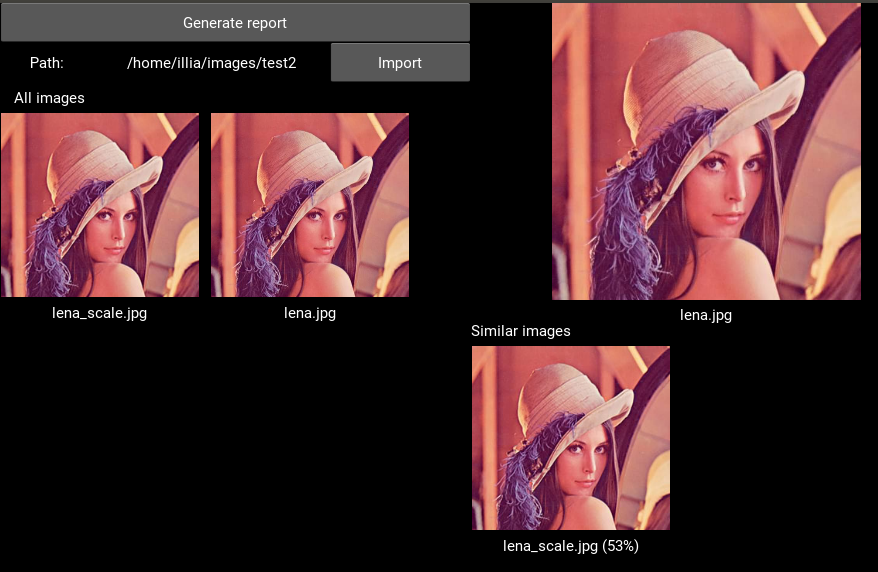


Рис. 4.4.2.1 – Контрольний приклад 2.

* + 1. Контрольний приклад 3. Поворот на 180 градусів.

1. Зображення lena.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* оригінальне зображення класичного розміру;

1. Зображення lena\_rotate180.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* зображення lena.jpg, перевернуте на 180;

Розглядаються оригінальне зображення і масштабоване. Вони визначилися схожими на 53%. Результати роботи програми зображено на [Рис. 4.4.3.1.](#im4_4_3_1)



Рис. 4.4.3.1 – Контрольний приклад 3.

* + 1. Контрольний приклад 4. Незначний поворот.

1. Зображення lena.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* оригінальне зображення класичного розміру;

1. Зображення lena\_rotate10.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* зображення lena.jpg, повернуте на 10, утворені чорні краї не обрізані;

1. Зображення lena\_rotate10\_noblack.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* зображення lena.jpg, повернуте на 10, утворені чорні краї обрізані;

1. Зображення lena\_rotate2.jpg.

* формат зображення: JPEG;
* розмір зображення: 512×512 пікселів;
* зображення lena.jpg повернуте на 2, накладене на нормальне зображення lena.jpg;

Розглядається нормальне зображення і повернуті. Метод виявляє мінімальне співпадіння для випадку повороту на 2, але скоріше через те що поворот на 2 градуси майже не змінює зображення, для інших випадків співпадінь не виявлено.

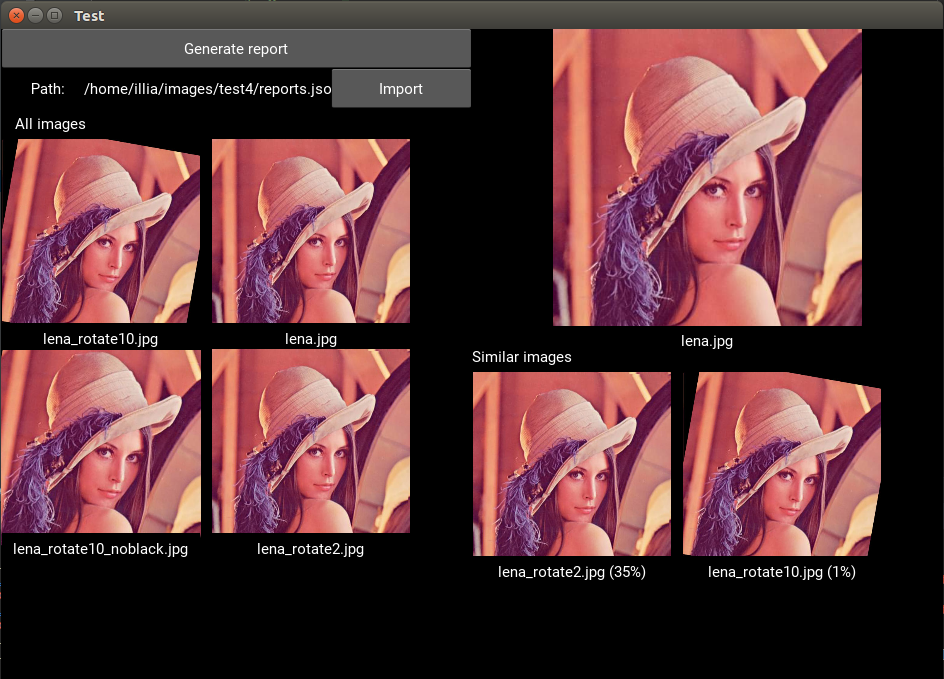


Рис. 4.4.4.1 – Контрольний приклад 4.

## Висновки до розділу

У цьому розділі розроблено програмне забезпечення для автоматизованої системи пошуку схожих зображень.

Розроблена система успішно визначає схожими зображення, що відрізняються застосуванням фільтрів розмиття, різкості, кольору, а також розміром, незначними афінними перетвореннями та поворотом на 180.

Висновки

Було розглянуто задачу виявлення схожості зображень. Проведено огляд та порівняння існуючих систем для пошуку схожих зображень. Проведено огляд і порівняння математичних методів. Виявлено, що для задачі пошуку схожих зображень на персональному комп’ютері користувача більш підходять content-based методи.

Для реалізації було обрано один з таких методів, описаний в [[1]](#b1). Було реалізовано систему, яка використовує обраний метод. Результати досліджень показали, що даний метод вважає схожими зображення з такими відмінностями як застосування фільтрів, зсуви, повороти, зміна розмірів.

Систему можна вдосконалювати в таких напрямках:

* додавання нових способів введення;
* покращення алгоритму отримання відбитка;
* централізоване збереження звітів сканування;
* синхронізація між різними приладами;

Перелік посилань

1. J.S. Seo et al. I Signal Processing: Image Communication 19 - 2004. - 325-339.
2. A. Menezes, P. Oorshot, S.Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, 1997.
3. J.A. Haitsma, T. Kalker, A highly robust audio ﬁnger- printing system, in: Proceedings of the International Conference on Music Information Retrieval (ISMIR) 2002, Paris, October 2002.
4. H.J. Seo, M. Sato, M. Takada, Dark energy from the Log-transformed Convergence Field: The Astrophysical Journal, September 2011.
5. Maret Yannick, Efficient image duplicate detection based on image analysis: Lausanne, 2007.
6. William L.Briggs, Van Emden Henson, The DFT: An Owner’s Manual for the Discrete Fourier Transform / William L.Briggs, Van Emden Henson – 1995.
7. S. Helgason: The Radon Transform / S. Helgason, 1999.
8. F. Lef`ebvre, B. Macq, J.-D. Legat (2002). RASH: Radon Soft Hash algorithm. In EURASIP European Signal Processing Conference, France.
9. F. Lefebvre, J. Czyz, B. Macq (2003). A robust soft hash algorithm for digital image signature. In Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on, vol. 2, pp. II–495–8 vol.3.4
10. Y. Ke, R. Sukthankar (2004). PCA-SIFT: a more distinctive representation for local image descriptors. In Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, vol. 2, pp. II–506–II–513Vol.2.
11. D. Lowe (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60(2):91–110.
12. C.-S. Lu, C.-Y. Hsu (2005). Geometric distortion-resilient image hashing scheme and its applications on copy detection and authentication. Multimedia Systems V11(2):159–173.
13. C. Herley (2002). Why watermarking is nonsense. Signal Processing Magazine, IEEE 19(5):10–11.
14. M. Barni (2003a). What is the future for watermarking? (part I). Signal Processing Magazine, IEEE 20(5):55–60.
15. P. Moulin (2003). Comments on “Why watermarking is nonsense”. Signal Processing Magazine, IEEE 20(6):57–59.
16. Лістинг програми

Лістинг файлу autocorrelation.py – алгоритм автокореляції

import numpy

from numba import jit

@jit

def autocorrelation(image):

'''

:param image: numpy.ndarray

:return: npumpy.ndarray

computes autocorrelation by y axis for given image

'''

result = numpy.empty(image.shape)

m, n = image.shape

for tetha in range(n):

s2 = 0.0

for s in range(m):

s2 += image[s, tetha] \*\* 2

for l in range(m):

s1 = 0.0

for s in range(l, m):

s1 += image[s, tetha]\*image[s-l, tetha]

result[l, tetha] = s1 / s2 if s2 != 0 else 0

return result

Лістинг файлу diff.py – визначення різниці між відбитками.

from numba import jit

@jit

def diff(im1, im2):

'''

computes Hemming distance between 2 fingerprints

:param im1: numpy.ndarray

:param im2: numpy.ndarray

:return: int

'''

res = 0

m, n = im1.shape

for i in range(m):

for j in range(n):

if im1[i][j] != im2[i][j]:

res += 1

return res

Лістинг файлу interleaving.py – алгоритм interleaving.

from numba import jit, types

import numpy

from cmath import phase

@jit(types.UniTuple(types.float64[:,:], 2)(types.complex128[:,:]), nopython=True)

def interleaving(image):

'''

computes interleaving for given image

:param image: numpy.ndarray

:return: (numpy.ndarray, numpy.ndarray)

'''

n = 21

image = image[:n,:n]

magnitudes = numpy.empty((n, n))

angles = numpy.empty((n, n))

for i in range(n):

for j in range(n):

x = image[i, j].real

y = image[i, j].imag

# magnitudes[i, j] = x

# angles[i, j] = y

magnitudes[i,j] = numpy.sqrt(x\*\*2 + y\*\*2)

angles[i,j] = phase(image[i, j])

return magnitudes, angles

Лістинг файлу trueshold.py – алгоритм trueshold.

from numba import jit

import numpy

@jit

def trueshold(image):

'''

applies trueshold algorithm for given image

:param image: numpy.ndarray

:return: numpy.ndarray

'''

result = numpy.empty(image.shape)

n, m = image.shape

for i in range(n):

for j in range(m):

result[i,j] = 1 if image[i,j] > 0 else 0

return result

Лістинг файлу twodfilter.py – застосування фільтру до зображення.

from numba import jit

import numpy

@jit

def twodfilter(image):

'''

applies filter [[-1, 1]

[1, -1]]

for given image

:param image: numpy.ndarray

:return: numpy.ndarray

'''

n, m = image.shape

n, m = n-1, m-1

result = numpy.empty((n, m))

for i in range(n):

for j in range(m):

result[i,j] = -image[i][j]-image[i+1][j+1]+image[i+1][j]+image[i][j+1]

return result

Лістинг файлу xor.py – застосування виключаючого або до зображення.

from numba import jit

import numpy

@jit

def xor(image1, image2):

'''

applies XOR function for given fingerprint surrogate

:param image1: numpy.ndarray

:param image2: numpy.ndarray

:return: numpy.ndarray

'''

n, m = image1.shape

result = numpy.empty((n, m))

for i in range(n):

for j in range(m):

result[i, j] = int(bool(image1[i,j]) != bool(image2[i,j]))

return result

Лістинг файлу log.py – застосування log-mapping до зображення.

from numba import jit

import numpy

k0 = 0.001

@jit

def log\_mapping(im):

'''

computes log mapping for given image

:param im: numpy.ngarray

:return: numpy.ndarray

'''

return k0\*numpy.log(1 + im/k0)

Лістинг файлу report\_generator.py – функції для створення звіту.

from skimage.io import imread, imsave

import numpy

from skimage.transform import radon

from steps.autocorellation import autocorrelation

from steps.interleaving import interleaving

from steps.twodfilter import twodfilter

from steps.trueshold import trueshold

from steps.xor import xor

from steps.log import log\_mapping

import os.path

def compare(data1, data2):

diff = 0

for d1, d2 in zip(data1, data2):

if d1 != d2:

diff += 1

return diff

def generate\_report(files, for\_html=True):

images = [imread(f, as\_grey=True) for f in files]

results = []

for i in range(len(images)):

image = images[i]

file = files[i]

sinogram = radon(image, circle=True)

sinogram = autocorrelation(sinogram)

sinogram = log\_mapping(sinogram)

sinogram = numpy.fft.fft2(sinogram)

real, imaginary = interleaving(sinogram)

real, imaginary = twodfilter(real), twodfilter(imaginary)

real, imaginary = trueshold(real), trueshold(imaginary)

sinogram = xor(real, imaginary)

sign\_file = list(os.path.split(file))

if for\_html:

sign\_file = 'sign\_{}'.format(sign\_file[-1])

imsave('static/' + sign\_file, sinogram)

results.append({

"image": os.path.split(file)[-1],

"sign": sign\_file,

"sign\_data": [x for x in sinogram.reshape((1, 400))[0]],

})

else:

results.append({

'text': os.path.split(file)[-1],

'image': os.path.abspath(file),

'sign\_data': [x for x in sinogram.reshape((1, 400))[0]],

'diffs': [],

})

if not for\_html:

for d1 in results:

for d2 in results:

if d1 != d2:

diff = compare(d1['sign\_data'], d2['sign\_data'])

if diff <= 150:

diff = round(100 - diff/150\*100)

item = {

'text': '{} ({}%)'.format(d2['text'], diff),

'image': d2['image'],

'diffs': diff

}

d1['diffs'].append(item)

for d in results:

d['diffs'].sort(key=lambda x: x['diffs'])

d['diffs'].reverse()

d.pop('sign\_data')

for d in d['diffs']:

d.pop('diffs')

return results

Лістинг файлу main.py – графічний інтерфейс.

from ui.report\_generator import generate\_report

from kivy.app import App

from kivy.uix.button import Button, Label

from kivy.uix.boxlayout import BoxLayout

from kivy.garden.filebrowser import FileBrowser

from kivy.uix.popup import Popup

from fs import open\_fs

from fs.walk import Walker

from kivy.uix.behaviors import ButtonBehavior

from kivy.uix.image import Image

from kivy.uix.stacklayout import StackLayout

from kivy.uix.scrollview import ScrollView

import os.path

import json

REPORT\_FILENAME = 'reports.json'

def file\_chosen(selected, filters, popup, cb):

result = []

State.imported\_path.text = selected

if os.path.isdir(selected):

for file in Walker(filter=filters).files(open\_fs(selected)):

if 'sign' not in file:

result.append(selected + file)

print('file chosen', result)

cb(result, selected)

else:

cb(selected, selected)

popup.dismiss()

def choose\_file(filters, cb, dirselect=True):

browser = FileBrowser(

select\_string='Select',

filters=filters,

path='/home/illia/images',

dirselect=dirselect)

browser.bind(

on\_success=lambda instance: file\_chosen(

instance.selection[0],

filters,

popup,

cb,

),

on\_canceled=lambda \_: popup.dismiss(),

)

popup = Popup(

title='Choose directory',

content=browser

)

popup.open()

def save\_json\_file(files, dirpath):

report = generate\_report(files, False)

with open(os.path.join(dirpath, REPORT\_FILENAME), 'w') as f:

json.dump(report, f)

return report

class State:

pass

def set\_main\_image(image):

if 'diffs' in image:

State.main\_image.set\_image(image)

State.similar\_images.set\_images(image['diffs'])

class ImageButtonWithLabel(BoxLayout):

def \_\_init\_\_(self, source):

super(ImageButtonWithLabel, self).\_\_init\_\_(

size\_hint=(None, None),

orientation='vertical',

)

self.height = 200

self.width = 200

def pressed(button):

set\_main\_image(source)

image = ImageButton(pressed, source=source['image'])

self.add\_widget(image)

self.add\_widget(Button(text=source['text'], size\_hint\_y=0.01, background\_color=(0, 0, 0, 1)))

class ImageButton(ButtonBehavior, Image):

def \_\_init\_\_(self, pressed=None, \*\*kw):

super(ImageButton, self).\_\_init\_\_(\*\*kw)

self.pressed = pressed

def on\_press(self):

if self.pressed is not None:

self.pressed(self)

class PathInput(BoxLayout):

def \_\_init\_\_(self):

super(PathInput, self).\_\_init\_\_()

self.size\_hint\_y = None

height = 40

self.size = (self.width, height)

self.add\_widget(Label(text='Path:', size\_hint=(.2, None), size=(self.width, height)))

self.add\_widget(State.imported\_path)

self.button = Button(text='Import',

size\_hint=(.3, None),

size=(self.width, height))

def on\_save(file, dirpath):

filename = file

with open(filename) as f:

report = json.load(f)

State.all\_images.set\_images(report)

self.button.bind(on\_press=lambda \_: choose\_file(

['\*.json'],

on\_save, # State.all\_images.set\_images(result),

))

self.add\_widget(self.button)

class MainImage(BoxLayout):

def \_\_init\_\_(self):

super(MainImage, self).\_\_init\_\_()

self.orientation = 'vertical'

self.image\_button = ImageButton()

self.add\_widget(self.image\_button)

self.button = Button(text='', size\_hint\_y=0.01, background\_color=(0, 0, 0, 1))

self.add\_widget(self.button)

def set\_image(self, image):

self.button.text = image['text']

self.image\_button.source = image['image']

class ImagesColumn(BoxLayout):

def init(self, text):

self.add\_widget(Label(text=text, size=(self.width, 30), size\_hint=(None, None)))

scroll\_view = ScrollView()

scroll\_view.add\_widget(self.images\_list)

self.add\_widget(scroll\_view)

def \_\_init\_\_(self):

super(ImagesColumn, self).\_\_init\_\_()

self.orientation = 'vertical'

self.images\_list = ImagesList(size\_hint\_y=None)

self.images\_list.bind(minimum\_height=self.images\_list.setter('height'))

def set\_images(self, images):

self.images\_list.set\_images(images)

if len(images):

set\_main\_image(images[0])

class AllImagesColumn(ImagesColumn):

def \_\_init\_\_(self):

super(AllImagesColumn, self).\_\_init\_\_()

generate\_report\_button = Button(text='Generate report', size\_hint\_y=None, height=40)

def load(files, dirpath):

report = save\_json\_file(files, dirpath)

State.all\_images.set\_images(report)

def on\_press(\_):

choose\_file(['\*.jpg', '\*.png'], load)

generate\_report\_button.bind(

on\_press=on\_press

)

self.add\_widget(generate\_report\_button)

self.add\_widget(PathInput())

self.init('All images')

class SimilarImagesColumn(ImagesColumn):

def \_\_init\_\_(self):

super(SimilarImagesColumn, self).\_\_init\_\_()

self.add\_widget(State.main\_image)

self.init('Similar images')

class ImagesList(StackLayout):

def \_\_init\_\_(self, \*\*kw):

super(ImagesList, self).\_\_init\_\_(\*\*kw)

self.spacing = 10

def set\_images(self, images):

self.clear\_widgets()

for image in images:

self.add\_widget(ImageButtonWithLabel(source=image))

class Root(BoxLayout):

def \_\_init\_\_(self):

super(Root, self).\_\_init\_\_()

self.add\_widget(State.all\_images)

self.add\_widget(State.similar\_images)

State.imported\_path = Label(size\_hint=(.5, None))

State.imported\_path.height = 40

State.main\_image = MainImage()

State.all\_images = AllImagesColumn()

State.similar\_images = SimilarImagesColumn()

class TestApp(App):

def build(self):

return Root()

if \_\_name\_\_ == '\_\_main\_\_':

TestApp().run()

1. Ілюстративний матеріал

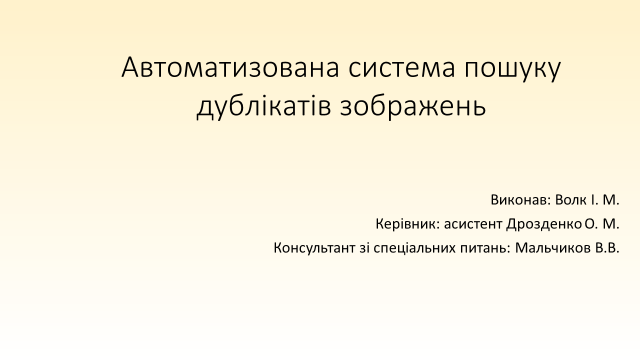


Рисунок Б.1 – Слайд 1

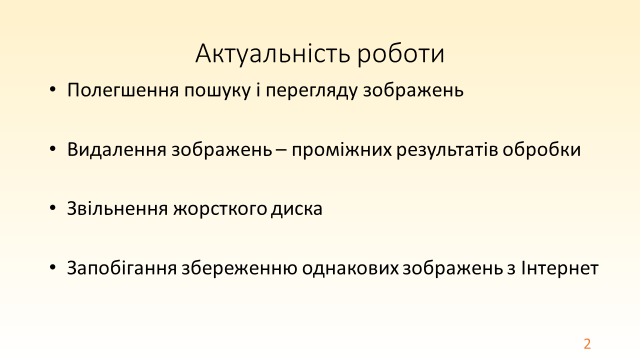


Рисунок Б.2 – Слайд 2

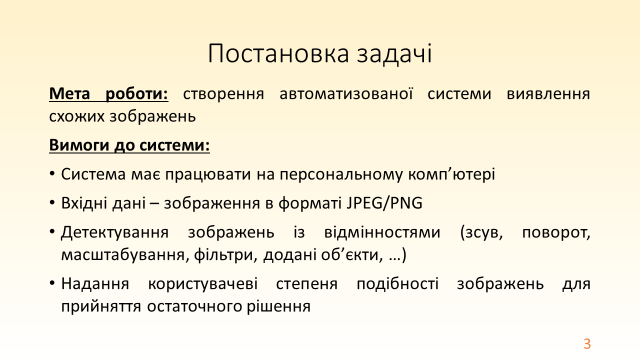


Рисунок Б.3 – Слайд 3

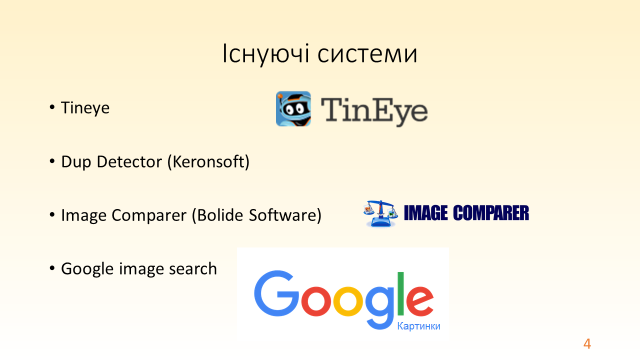


Рисунок Б.4 – Слайд 4

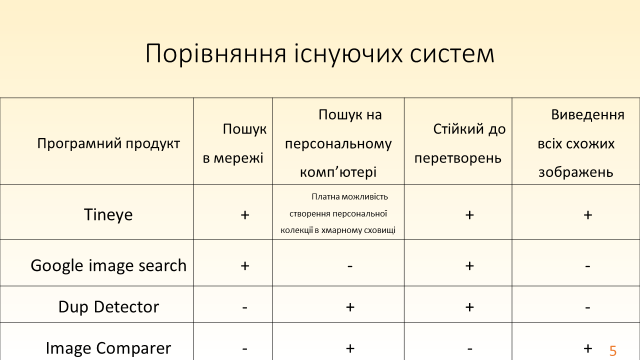


Рисунок Б.5 – Слайд 5



Рисунок Б.6 – Слайд 6

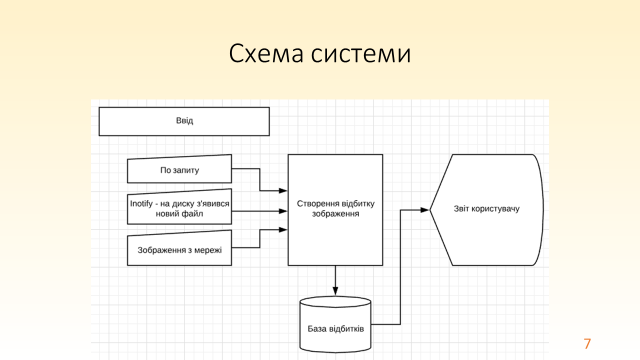


Рисунок Б.7 – Слайд 7

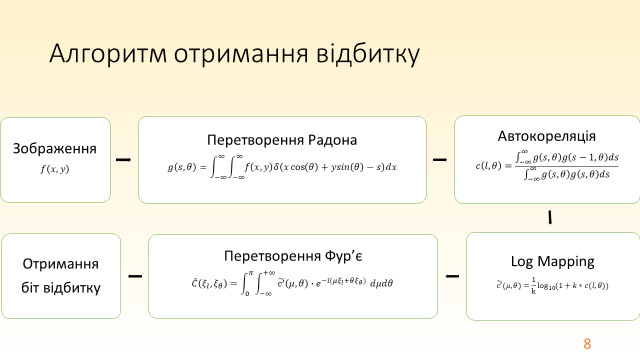


Рисунок Б.8 – Слайд 8

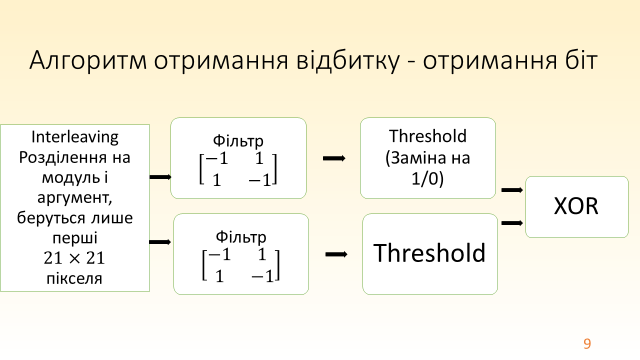


Рисунок Б.9 – Слайд 9



Рисунок Б.10 – Слайд 10

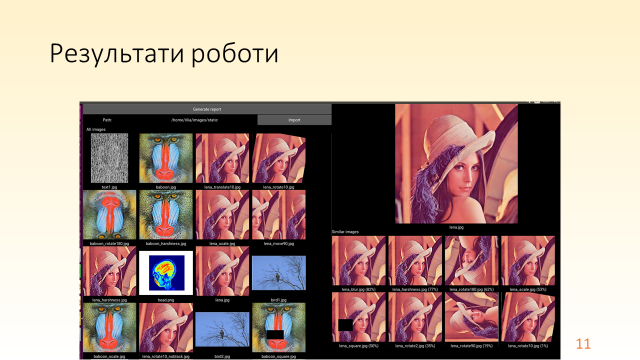


Рисунок Б.11 – Слайд 11

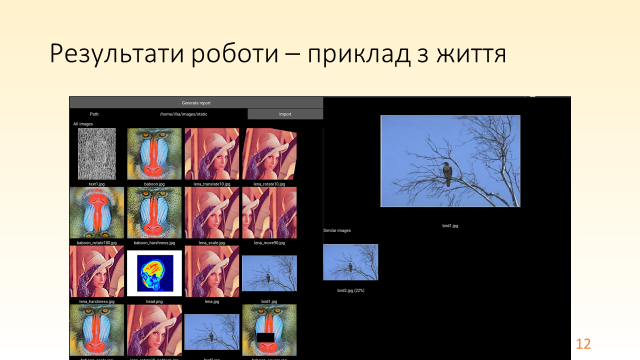


Рисунок Б.12 – Слайд 12

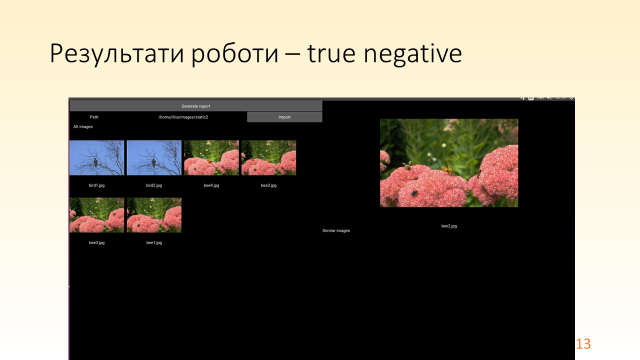


Рисунок Б.13 – Слайд 13



Рисунок Б.14 – Слайд 14

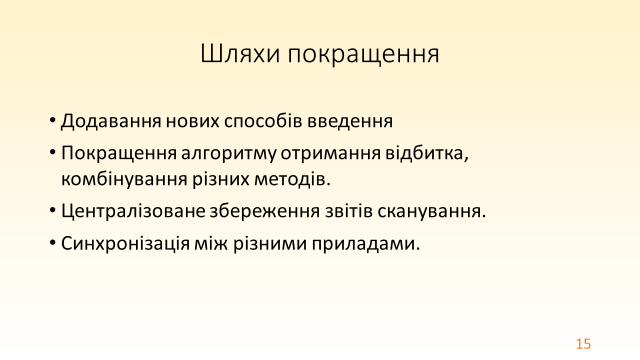


Рисунок Б.15 – Слайд 15



Рисунок Б.16 – Слайд 16