Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Сегментація зображень

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
к.т.н., доцент кафедри
інформатики
Бучко О. А.

_____
(підпис)
«___» _____ 2022 р.

Виконав студент ІПЗ-3
Андрієнко Є. С.

«___» _____2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

_____

(прізвище та ініціали)

_____

(підпис)

«___» _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Андрієнко Єлизаветі Сергіївні

факультету інформатики 3 р.н. бакалаврської програми

ТЕМА: Сегментація зображень

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Вступ
4. Класифікація алгоритмів сегментації зображень
5. Алгоритми сегментації зображень
6. Реалізація алгоритмів сегментації
7. Висновок
8. Список використаних джерел

Дата видачі «___» _____ 2022 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи

№	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу.	22.10.2021	
2.	Огляд технічної літератури за темою роботи.	7.02.2022	
3.	Виконання аналізу основних алгоритмів сегментації зображення.	14.02.2022	
4.	Реалізація алгоритмів (алгоритм знаходження країв, алгоритм з використанням порогів, алгоритм на основі вододілів, кластеризація).	7.04.2022	
5.	Аналіз отриманих результатів з керівником.	18.04.2022	
6.	Покращення роботи алгоритмів з урахуванням зауважень керівника.	10.05.2022	
7.	Створення презентації.	31.05.2022	
8.	Захист роботи	7.06.2022	

Студент Андрієнко Є. С.

Керівник Бучко О. А.

«____» _____

Зміст

Вступ

Реалії сьогодення змушують людину щосекунди оброблювати незліченні об’єми інформації, які надходять до нас. Одним із видів такого аналізу є розпізнавання об’єктів та класифікація їх за певними критеріями. Мозок робить це автоматично. Але чи здатний на це комп’ютер? У сучасному світі техніки існує можливість відокремлювати предмети, їх форми та межі на зображеннях та відео. Цей процес називається сегментацією. Її головною метою є спрощення зображення для його легшого подальшого аналізу. Основний принцип роботи сегментації – це поділ вхідного зображення на певні частини (сегменти) за певними ознаками. У цифровій обробці ці частини називаються об’єктами зображення.

Як правило, більшість аналіз зображення починається саме із сегментації. Відсутність цього етапу унеможливлює наступні виконання програм комп’ютерного зору. Саме тому цей процес є таким важливим.

Завдяки різним алгоритмам сегментації пікселі на зображення можна розбивати і класифікувати, надавати їм певні мітки. Додатково можна розрізняти передній та задній фон, виділяти окремі об’єкти та їх межі.

На сьогодні сегментація зображень знайшла своє застосування у багатьох областях промисловості. Деякі з цих галузей включають виявлення дорожніх знаків, біологію, оцінку будівельних матеріалів або відеоспостереження. Крім того, автономні транспортні засоби та спеціальні системи допомоги водієві повинні розрізняти дорожні поверхні або виявляти пішоходів.

Також сегментація зображень широко застосовується в медичних програмах, таких як виділення кордонів пухлини або вимірювання об’ємів тканин. Тут є можливість створити стандартизовані бази даних зображень, які можна використовувати для оцінки нових хвороб і пандемій, що швидко поширюються.

Сегментація зображень успішно застосовується для аналізу супутникових знімків у сфері дистанційного зондування, включаючи методи міського планування або точного землеробства. Крім того, зображення, зібрані за допомогою дронів, були сегментовані за допомогою методів глибокого навчання (англ. deep learning), що дає можливість вирішити важливі екологічні проблеми, пов’язані зі зміною клімату.

Із щоденним розвитком технологій комп’ютерного зору паралельно зростає попит на користування різними алгоритмами сегментації зображень. Приміром, методи сегментації часто використовується промисловцями для пошуку неякісної продукції. У цьому випадку фіксуються потрібні частини зображення і потім групуються на пошкоджені та правильні. Завдяки такій системі можна знизити ризик людського фактору, а також зробити процес тестування набагато простішим та більш ефективним.

Існує досить багато алгоритмів сегментації зображень. У цій роботі будуть розглянуті можливі варіанти їх класифікації, а також будуть проаналізовані та реалізовані чотири найвідоміші методи, а саме: знаходження країв, алгоритм з використанням порогів, вододіли та кластеризація методом K-середніх.

Розділ 1. Класифікація алгоритмів сегментації зображень

Завдяки високій популярності темі сегментації наразі існує безліч різноманітних способів виконання даного процесу. Тому за певними параметрами можна виділити кілька груп алгоритмів сегментації.

1.1 Сегментація на основі підходу

У найпростішому сенсі сегментація зображення — це ідентифікація об’єктів[1]. Алгоритми не можуть класифікувати різні сегменти без попередньої ідентифікації об'єктів. І прості, і складні реалізації сегментації зображення працюють на основі розпізнавання об’єктів.

Тому можна класифікувати методи сегментації зображень за тим, як алгоритм розпізнає об’єкти, тобто збирає схожі пікселі та відокремлює їх від інших. Для реалізації цього завдання є два підходи:

1.1.1 Підхід на основі регіонів (виявлення подібності)

За допомогою даного підходу можна знайти схожі пікселі на зображенні залежно від обраного значення порогу, злиття або росту регіонів. Кластеризація та подібні методи машинного навчання використовують цей підхід для виявлення невідомих атрибутів. Алгоритми класифікації дотримуються цього підходу для визначення характеристик і відокремлення фрагментів зображень на їх основі.

1.1.2 Підхід на основі кордонів (виявлення розривів)

На противагу підходу на основі регіонів, де знаходяться пікселі зі схожими ознаками, тут необхідно знайти пікселі, які відрізняються один від одного на базі кордонів. Виділення країв, точок та інші схожі алгоритми слідують цьому підходу для пошуку країв окремих пікселів і відокремленні їх від решти зображення.

1.2 Сегментація на основі техніки

Алгоритми на основі техніки використовуються залежно від типу зображень, які необхідно обробити та проаналізувати, і результатів, які ми хочемо отримати.

На підставі даних параметрів можна класифікувати алгоритми сегментації зображень на наступні групи:

1.2.1 Структурні техніки

Для структурних технік важливими є дані зображення, з яким ми працюємо. До них належать пікселі, їх щільність, гістограма, розподіл кольору та інша відповідна інформація. Також вам знадобиться структурна інформація про регіон, який треба виділити на зображенні. Алгоритми, які використовуються для цього завдання, дотримуються підходу на основі регіону.

1.2.2 Стохастичні техніки

Для реалізації цих алгоритмів нам необхідна інформації про дискретні значення пікселів зображення, а не про структуру потрібної частини зображення. Це корисно, коли потрібно обробляти декілька зображень, оскільки не вимагається багато інформації для сегментації. До стохастичних технік можна віднести кластеризацію на основі K-середніх і алгоритми ANN.

1.2.3 Гібридні техніки

З назви можна здогадатися, що ці алгоритми використовують і структурні, і стохастичні методи. Тобто вони виконують сегментацію зображення, користуючись структурною інформацією про досліджувану область, а також дані про дискретні пікселі по всьому зображенню.

1.3 Сегментація на основі потреб

Оскільки сегментація зображень все частіше використовується, важливо зрозуміти, який вид методу сегментації найкраще відповідає потребам у тій чи іншій справі (для виконання різних завдань). Тому сегментацію можна

розділити на три різні види на основі наших потреб, а саме: на семантичну, сегментацію екземплярів та паноптичну[2].

1.3.1 Семантична сегментація

Семантична сегментація зображення – це техніка, яка передбачає виявлення об’єктів у зображенні та їх групування на основі визначених категорій. Це просто позначення кожного пікселя зображення відповідним класом. Наприклад, ви хочете класифікувати різні види одягу на основі кольору. Модель семантичної сегментації може ідентифікувати об’єкти на основі їх кольору, а пізніше розбити набір зображень із одягом однакового кольору на різні групи (наприклад, зображення з білим одягом в групі 1, зображення з червоним одягом у групі 2 тощо).

Важливим фактором, який відрізняє семантичну сегментацію від двох інших методів, є те, що ми не відокремлюємо екземпляри одного класу; ми розглядаємо лише категорію кожного пікселя. Тобто вони не розрізняються як окремі моделі сегментації, які розрізняють окремі об’єкти одного класу. Як згадувалося у наведеному прикладі про одяг, хоч у нас є два різних зображення з одягом одного класу (наприклад, сукня двох різних кольорів - білого та червоного), модель, орієнтована на класифікацію одягу на основі кольору, не буде ідентифікувати його на основі класу.

Моделі семантичної сегментації корисні для різних завдань, зокрема для автономних транспортних засобів, медичної діагностики.

1.3.2 Сегментація екземплярів

Сегментація екземплярів робить семантичну сегментацію ще на один крок вперед і включає виявлення об’єктів у визначених категоріях. На відміну від семантичної сегментації, сегментація екземплярів складається з локалізації конкретних об’єктів на основі асоціації належних їм пікселів. Це робить її більш складною в реалізації. Наприклад, метод сегментації зображення екземпляра можна використовувати для ідентифікації визначених об’єктів

(припустимо, що нам необхідно ідентифікувати повітряні кульки на зображенні), модель не тільки розпізнає всі повітряні кульки, але й допоможе нам розрізнити їх окремо. На відміну від семантичної сегментації, усі кульки мають різні кольори або мітки, оскільки семантична сегментація не розпізнає однакові об’єкти на одному зображенні як різні.

Сегментація екземплярів — це логічне продовження семантичної сегментації та одна зі найскладніших моделей для роботи в порівнянні з іншими методами сегментації. Оскільки мета сегментації екземплярів — отримати уявлення об’єктів одного класу, поділених на різні екземпляри, автоматизація цього процесу — непросте завдання. Сучасне тегування екземплярів надає користувачеві додаткову інформацію про дані та допомагає приймати ефективні рішення у невідомих ситуаціях.

Моделі сегментації екземплярів широко використовуються для морського нагляду (для захисту від незаконного промислу), контролю надходження нафти та моніторингу забруднення моря, а також у роздрібній торгівлі для оптимізації операцій, безпеки та спостереження.

1.3.3 Паноптична сегментація

Паноптична сегментація семантично розрізняє різні об’єкти, а також визначає окремі екземпляри кожного виду об’єктів. Іншими словами, паноптична сегментація призначає дві мітки кожному з пікселів зображення – семантичну мітку та ідентифікатор екземпляра. Вважається, що пікселі з однаковою міткою належать до одного семантичного класу, а ідентифікатори розрізняють його екземпляри. На відміну від сегментації екземплярів, кожному пікселю у паноптичній сегментації присвоєно унікальну мітку, яка відповідає екземпляру, що дозволяє уникнути неправильної інтерпретації інформації.

Попередні методи сегментації зображень зосереджувалися на використанні наскрізної архітектури для ідентифікації та розрізнення об’єктів на зображенні. Оскільки паноптична сегментація використовує взаємодоповнюваність між

завданнями семантичної та сегментації екземплярів для підвищення точності та продуктивності, модель необхідно реалізовувати за допомогою окремих алгоритмів із величезними обсягами даних.

Паноптичні моделі сегментації знайшли своє застосування в аналізі медичного зображення для раннього виявлення раку, цифровій обробці зображень, самокерованих автомобілях (для розуміння, по якій поверхні їде транспортний засіб, і перемикання між режимами водіння) тощо.

Розділ 2. Алгоритми сегментації зображень

2.1 Алгоритм виділення країв

Виділення країв є однією з основних операцій при виконанні обробки зображень. Це допомагає зменшити кількість даних (пікселів) для обробки та підтримки структурного аспекту зображення. Суть цього алгоритму полягає у виявлення перепадів яскравості.

Найчастіше для виділення країв використовують оператор Собеля. Він також відомий як метод обрахунку похідних першого порядку для пошуку градієнта. Обробка зображення проходить у напрямках X та У окремо, а потім об'єднується у формі нового зображення як сума країв X і У. Однак, ці проміжні зображення можна використовувати як окремий результат[5].

При використанні оператора Собеля рекомендується спочатку перетворити зображення із шкали RGB (кольорове) у зображення Grayscale (у відтінках сірого). Також у ході виявлення країв використовується так звана згортка ядра. Ядро — це матриця розміром 3x3, яка складається з різно (або симетрично) зважених індексів. Вона буде представляти фільтр, який ми будемо реалізовувати для виявлення країв. Необхідні дві матриці, по одній для кожному напряму X та У. Якщо ми подивимось на них, то побачимо, що матриця для напрямку X має мінусові числа з лівого боку і додатні числа з правого боку. Аналогічно, матриця для напрямку Y має мінусові числа знизу і додатні зверху.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Рис. 1.1 Маски оператора Собеля

Основним завданням оператора Собеля є з’ясувати величину різниці, помістивши градієнтну матрицю над кожним пікселем нашого зображення. Ми отримуємо два зображення, одне для напрямку X, а інше для Y-напрямку. Використовуючи згортку ядра, ми бачимо, що на зображенні нижче є межа між стовпцем із значеннями пікселів 100 та 200.

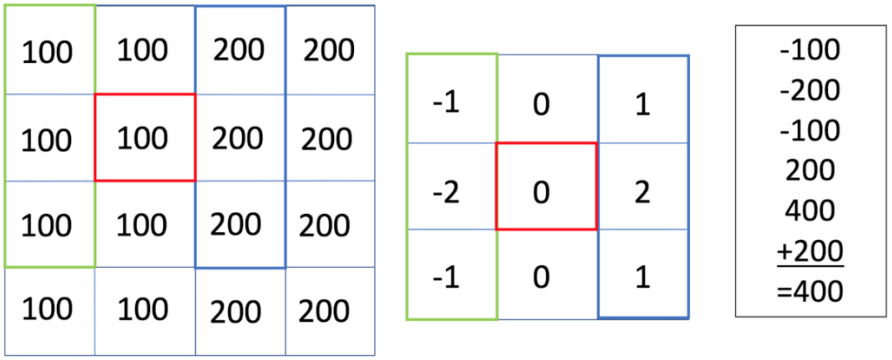

Рис. 1. 2 Приклад виконання згортки

Наведений вище приклад показує результат виконання згортки шляхом розміщення маски Gx над пікселями зі значенням 100, позначеними червоним кольором. Виконавши обчислення, які можна побачити справа, біло отримано значення 400, що не дорівнює нулю, отже, це є ребро. Якби всі пікселі зображень мали однакове значення, то згортка призвела б до підсумкової суми, що дорівнює нулю. Таким чином, градієнтна матриця забезпечить велике число за умови, коли одна сторона світліша або темніша за іншу. Також слід звернути увагу на те, що знак вихідного результату (+ або -) не має значення.

2.2 Алгоритм з використанням порогів

Використання порогів є доволі популярним при обробці зображень. Вони допомагають відрізнити передній план від заднього (фону). Обравши коректне порогове значення T, зображення в градаціях сірого кольору можна перетворити на двійкове (чорно-біле). Двійкове зображення повинно містити всю необхідну інформацію про положення і форму об'єктів, що цікавлять (передній план). Перевага отримання спочатку двійкового зображення полягає в тому, що це зменшує складність даних і спрощує їх процес розпізнавання та класифікації. Найбільш поширеним способом перетворення зображення в

градаціях сірого в двійкове зображення є обрання єдиного порогового значення (T). Потім усі пікселі, значення яких нижчі за цей поріг, будуть класифікуватися як чорні (0), а пікселі зі значенням вищим за поріг будуть білими (1)[6].

$$g(x,y) = f(x) = \begin{cases} 1, \text{якщо } f(x,y) > T \\ 0, \text{якщо } f(x,y) \leq T \end{cases} \quad (3.1)$$

Однак в даному алгоритмі постає проблема вибору правильного значення для порогу T. Для цього часто використовується метод японського ученого Нобуюки Оцу.

Даний метод обирає порогове значення, щоб мінімізувати внутрішньокласову дисперсію порогових чорно-білих пікселів. Спеціальна функція критерію використовується як міра статистичного поділу для описового аналізу. По-перше, для сегментації зображень пропонується критерій максимізації модифікованої міжкласової дисперсії, еквівалентний критерію максимізації звичайної міжкласової дисперсії. Далі, відповідно до нового критерію, розроблено рекурсивний алгоритм для ефективного пошуку оптимального порогу.

2.3 Алгоритм вододілу

Ще одним із найпопулярніших методів сегментації зображень є алгоритм вододілу. Його часто використовують, коли ми маємо справу з однією з найскладніших операцій в обробці зображень – розділенням схожих об’єктів на зображенні, які торкаються один одного. Щоб зрозуміти «філософію» алгоритму вододілу, нам потрібно уявити зображення у градації сірого як топографічну поверхню. На такому зображенні значення пікселів високої інтенсивності представляють найвищі точки (білі області), а значення низької інтенсивності представляють впадини – локальні мінімуми (чорні області)[7].

Тепер уявіть, що ми починаємо заповнювати кожну ізольовану долину водою. Що станеться? Вода з різних долин почне зливатися. Щоб уникнути цього, нам

потрібно спорудити бар’єри в місцях злиття води. Ці бар’єри використовуються для визначення меж сегментів. Потім ми продовжуємо наповнювати воду і будувати вододіл, поки рівень води не досягне висоти найвищого піку. Наприкінці процесу будуть видні лише лінії вододілу, і це буде кінцевим результатом сегментації. Отже, можна зробити висновок, що метою даного алгоритму є виявлення ліній вододілу.

По суті, ми розбиваємо зображення на два різні набори: темні області називаються басейнами водозбору – група пов’язаних пікселів з однаковим локальним мінімумом. Лінії, що відділяють один водозбір від іншого, називають лініями вододілу. Даний алгоритм вдало ілюструє зображення нижче[8].

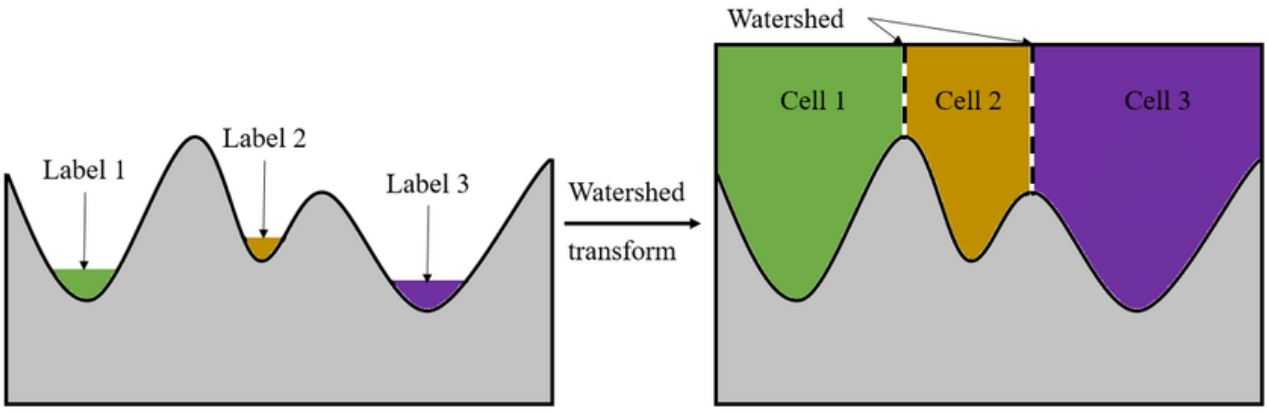

Рис. 1. 3 Алгоритм вододілу

Можна побачити, коли ми піднімемо жовтий поріг, в кінцевому підсумку сегмент 1 і сегмент 2 будуть об’єднані, тому нам потрібно поставити першу лінію вододілу. Після цього сегменти 2 і 3 будуть об’єднані, тому нам потрібно додати ще одну лінію вододілу.

2.4 Алгоритм кластеризації

Основна мета кластеризації полягає в тому, щоб відокремити групи пікселів на зображенні зі схожими ознаками та об’єднати їх у кластери.

Варто зазначити, що у кластеризації ви не знаєте, що шукаєте, і намагаєтеся визначити деякі сегменти або кластери у своїх даних. Коли ви використовуєте

алгоритми кластеризації у своєму наборі даних, можуть раптово з’явитися неочікувані речі (структури, кластери і т.п.). Ось чому алгоритм кластеризації є неконтрольованим.

Існує досить багато методів кластеризації. Вони поділяються на ієрархічні (алгоритм найближчого сусіда), ітеративні (K-середніх, C-середніх), алгоритми засновані на щільності (DBSCAN), модельні (EM), концептуальні (Cobweb), мережеві (WaveCluster)[9].

Одним із найвідоміших є метод K-середніх. Це алгоритм на основі центроїдів або алгоритм на основі відстані, де ми обчислюємо відстані, щоб призначити точку кластеру. В даному алгоритмі кожен кластер асоціюється з центроїдом. Головним завданням алгоритму K-середніх є мінімізування суми відстаней між точками та відповідним центроїдом кластера.

Тепер розглянемо як працює кластеризація K-середніх. Припустимо, у нас є набір даних (пікселів) без міток, і ми хочемо згрупувати ці дані в кластери. Тепер важливе питання полягає в тому, як обрати оптимальну кількість кластерів. Існує два можливі способи вибору кількості кластерів – самостійно (на основі ваших цілей) та метод «ліктя».

Можна декілька разів запустити алгоритм кластеризації на основі K-середніх, щоразу обираючи нову кількість кластерів, для того, щоб на власні очі побачити і обрати той результат, який вас влаштує залежно від мети, яку ви хочете досягти.

У методі «ліктя» ми фактично міняємо кількість кластерів (K) від 1 до 10. Для кожного значення K ми обчислюємо WCSS (Within-Cluster Sum of Square) - суму квадратів всередині кластера[10]. Іншими словами, WCSS — це сума квадратів відстані між кожною точкою і центроїдою у кластері. Намалювавши графік WCSS із значенням K, можна побачити, що він має вигляд ліктя. Зі збільшенням кількості кластерів значення WCSS почне зменшуватися. Значення WCSS є найбільшим, коли K = 1. Після аналізу графіка можна дійти

висновку, що він швидко змінюється в певній точці, створюючи таким чином форму ліктя. З цього моменту графік починає рухатися майже паралельно осі X. Значення K, що відповідає цій точці, є оптимальним значенням K, тобто оптимальною кількістю кластерів.

Розділ 3. Реалізація алгоритмів сегментації

У цій роботі алгоритми сегментації зображень було розроблено на мові програмування Java, а також використано бібліотеку OpenCV для порівняння результатів. OpenCV – це крос-платформна бібліотека, за допомогою якої можна створювати програми комп'ютерного зору. В основному він зосереджений на обробці зображень, захопленні та аналізі відео.

3.1 Реалізація алгоритму виділення країв

3.1.1 Стандартний алгоритм за допомогою оператора Собеля

Для реалізації алгоритму виділення країв перетворюємо вхідне зображення із кольорової моделі RGB у модель Grayscale. Для цього використовуємо клас ColorConvertOp, який виконує піксельне перетворення кольору вихідного зображення, використовуючи простір сірого кольору.

```
private static BufferedImage grayscale(BufferedImage img) {
    BufferedImageOp bufferedImageOp = new ColorConvertOp(ColorSpace.getInstance(ColorSpace.CS_GRAY),  hints: null);
    return bufferedImageOp.filter(img, dest: null);
}
```

Рис. 3. 1 Перетворення зображення із кольорової моделі RGB у Grayscale

Наступним кроком у двох циклах ми беремо кожний піксель зображення, заносимо цей піксель і його 8 сусідів (оточуючі пікселі) до масиву data. Важливим є те, що цикл починаємо з пікселя, який має координати (1, 1), оскільки це перша точка, яка має усі 8 сусідів.

```
for(int x = 1; x < width - 1 ; x++){
    for(int y = 1; y < height - 1; y++){
        int[] data = {
                img.getRGB( x: x-1, y: y-1)&0xff,img.getRGB(x, y: y-1)&0xff,img.getRGB( x: x+1, y: y-1)&0xff,
                img.getRGB( x: x-1, y)&0xff,img.getRGB(x, y)&0xff,img.getRGB( x: x+1, y)&0xff,
                img.getRGB( x: x-1, y: y+1)&0xff,img.getRGB(x, y: y+1)&0xff,img.getRGB( x: x+1, y: y+1)&0xff
        };
        int value = convolution (matrix, data);
        result.setRGB(x,y,  rgb: 0xff000000|(value<<16)|(value<<8)|value);
    }
}
```

Рис. 3. 2 Реалізація алгоритму з використанням оператора Собеля

Далі необхідно застосувати згортку, спочатку для матриці напрямку X (MASK_X). У методі convolution в циклі, який проходить по отриманому масиву, сумуємо добуток значення пікселя і значення з матриці, що має відповідний індекс. Потім рахуємо середнє значення шляхом ділення результату на 9 (кількість елементів у масиві data).

```
private static int convolution(int[] matrix, int[] data){
    int result = 0;

    for (int i = 0; i < data.length; i++)
        result += matrix[i] * data[i];

    return Math.abs(result) / 9;
}
```

Рис. 3. 3 Виконання згортки

Отримане значення присвоюємо пікселю на вихідному зображенні edgeDetectionX.png.

Аналогічним чином використовуємо згортку для матриці напрямку У (MASK_У) і отримуємо зображення edgeDetectionY.png. Останнім етапом буде сумування абсолютного (невід'ємного) значення кожного пікселя зображення edgeDetectionX.png із відповідним абсолютним значенням пікселя на edgeDetectionY.png. Отримана сума є пікселем результуючого зображення.

```
for(int x = 0; x < width ; x++){
    for(int y = 0; y < height; y++){
        int data = Math.abs(edgesX.getRGB(x, y)&0xff) + Math.abs(edgesY.getRGB(x, y)&0xff);
        result.setRGB(x, y,  rgb: 0xff000000|(data<<16)|(data<<8)|data);
    }
}
```

Рис. 3. 4 Об'єднання попередніх результатів

Результат програми:

Рис. 3. 5 Вхідне зображення для алгоритму виділення країв

Рис. 3. 6 Результат застосування маски MASK_X

Рис. 3. 7 Результат застосування маски MASK_Y

Рис. 3. 8 Результат алгоритму виділення країв за допомогою оператора Собеля

Рис. 3. 9 Результат алгоритму виділення країв за допомогою оператора Собеля з використанням бібліотеки OpenCV

3.1.2 Оптимізований алгоритм

Стандартний метод з використанням оператора Собеля має певні проблеми. Оператор Собеля бере до уваги лише вертикальний напрямок і горизонтальний напрямок. Проте інформація зображення не обмежується лише горизонтальним і вертикальним напрямками. Звідси можна зробити висновок, що частина інформації зображення може бути втрачена. У цій роботі пропонується новий покращений алгоритм виділення країв на основі оператора Шара. Оператор Шара - це практично той самий оператор Собеля, однак з дещо іншими значеннями маски[13].

$$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}.$$

Рис. 3. 10 Маски оператора Шара

Для початку замінюємо оператор Собеля оператором Шара, створюючи маски у напрямку X та У, і додаємо ще дві маски у напрямку 45 і 135 градусів.
Далі застосовуємо згортку, таким чином отримуємо вже не два, а чотири зображення.
На останньому етапі враховуємо значення кожного пікселя з отриманих зображень для обрахунку фінального результату. Але оскільки тепер ми маємо у два рази більше значень, відповідно обраховане число ділимо на два.

```
private static BufferedImage edgeDetection(BufferedImage img0, BufferedImage img90, BufferedImage img45, BufferedImage img135){
    BufferedImage result = new BufferedImage(img0.getWidth(), img90.getHeight(), BufferedImage.TYPE_INT_RGB);
    int height = result.getHeight();
    int width = result.getWidth();

    for(int x = 0; x < width ; x++){
        for(int y = 0; y < height; y++){
            int data = (Math.abs(img0.getRGB(x, y)&0xff) + Math.abs(img90.getRGB(x, y)&0xff) +
                        Math.abs(img45.getRGB(x, y)&0xff) + Math.abs(img135.getRGB(x, y)&0xff))/2;
            result.setRGB(x, y, rgb: 0xff000000|(data<<16)|(data<<8)|data);
        }
    }
    return result;
}
```

Рис. 3. 11 Оптимізований алгоритм виділення країв

Результат програми:

Рис. 3. 12 Результат оптимізованого алгоритму виділення країв

3.2 Реалізація алгоритму з використанням порогів методом Оцу

Подібно до алгоритму, заснованому на виділенні країв, метод Оцу найкраще імплементувати на зображенні в градації сірого. Наступним кроком є знаходження гістограми. Гістограма – це графік розподілу значень яскравості пікселів зображення. У коді позначається масивом розміром 256, де кожний індекс відповідає певному рівню яскравості (0-255). Під час піксельного проходження зображення ми заносимо значення даного пікселя на відповідне місце у масиві. Таким чином рахуємо кількість пікселів з певним рівнем яскравості.

```
int[] histogram = new int[256];

for(int i = 0; i< width; i++) {
    for(int j = 0; j < height; j++) {
        int data = img.getRaster().getSample(i, j,  b: 0);
        histogram[data]++;
    }
}
```

Рис. 3. 13 Знаходження гістограми

З гістограми можна легко кількість класів, які чітко розділяються. Суть методу Оцу полягає в тому, щоб поріг між класами був виставлений таким чином, щоб кожен з них був якомога більш «щільним».

Для початку необхідно знайти імовірності появи всіх класів (тобто імовірність того, що піксель буде належати даному класу).

$$P_1(k) = \sum_{i=0}^{k} p_i \qquad (3.2)\ [6]$$

де k – максимальне значення яскравості в даному класі.

Потім обраховуємо середнє значення яскравості пікселів кожного класу C.

$$m_1(k) = \sum_{i=0}^{k} iP(i|C_1) = \sum_{i=0}^{k} iP(C_1|i)P(i)/P(C_1) = \frac{1}{P_1(k)} \sum_{i=0}^{k} ip_i \qquad (3.3)\ [6]$$

Таким чином середня яскравість зображення знаходиться за формулою

$$m_G = \sum_{i=0}^{L-1} ip_i, \qquad (3.4)\ [6]$$

де L - градації яскравості.

Для того, щоб отримати оптимальний поріг, необхідно максимізувати міжкласову дисперсію $\sigma_B^2(\mathrm{k})$

$$\sigma_B^2(\mathrm{k}) = \frac{[m_G P_1(k) - m(k)]^2}{P_1(k)[1 - P_1(k)]} \quad (3.5)\ [6]$$

Даний метод реалізований у методі findTreshold:

```
private static int findTreshold(BufferedImage img) {
    int newSize, brightnessAmount = 0, threshold = 0;
    float sum = 0, maxVal = 0, sumBrightness = 0;
    int size = img.getHeight() * img.getWidth();
    int[] histogram = histogram(img);

    for(int i = 0; i < 256; i++)
        sum += i * histogram[i];

    for(int i = 0; i < 256; i++) {
        brightnessAmount += histogram[i];
        if(brightnessAmount == 0) continue;
        newSize = size - brightnessAmount;
        if(newSize == 0) break;
        sumBrightness += (float) (i * histogram[i]);
        float avgBrightness = sumBrightness / brightnessAmount;
        float avg = (sum - sumBrightness) / newSize;
        float dispersion = (float) brightnessAmount * (float) newSize * (avgBrightness - avg) * (avgBrightness - avg);
        if(dispersion > maxVal) {
            maxVal = dispersion;
            threshold = i;
        }
    }
    return threshold;
}
```

Рис. 3. 14 Знаходження оптимального порогу методом Оцу

Тепер можна легко отримати результат алгоритму шляхом порівняння значення кожного пікселя зображення зі знайденим порогом. Якщо це значення буде перевищувати поріг, то присвоюємо відповідному пікселю вихідного зображення 255. Якщо ж значення пікселя менше за значення порогу, то у результаті він буде дорівнювати 0.

```
int data, newValue, threshold = findTreshold(img);

for(int i = 0; i < width; i++) {
    for(int j = 0; j < height; j++) {
        data = img.getRaster().getSample(i, j,  b: 0);
        if(data > threshold)
            newValue = 255;
        else
            newValue = 0;
        result.setRGB(i, j,  rgb: 0xff000000 | (newValue << 16) | (newValue << 8) | newValue);

    }
}
```

Рис. 3. 15 Реалізація алгоритму з використанням оптимального порогу

Результат програми:

Рис. 3. 16 Вхідне зображення для алгоритму з використанням порогів

Рис. 3. 17 Результат алгоритму з використанням порогів методом Оцу для зображення у градації сірого

Рис. 3. 18 Результат алгоритму з використанням порогів методом Оцу для початкового зображення

Рис. 3. 19 Результат алгоритму з використанням порогів методом Оцу з використанням бібліотеки OpenCV

3.3 Реалізація алгоритму вододілу

Для початку реалізації даного алгоритму необхідно знайти усі мінімальні точки на зображенні. Це означає, що треба пройтись по всім пікселям зображення і знайти найтемніший, який ще не був обраний і не знаходиться близько до вже обраного мінімального пікселя. Можна знайти будь-яку кількість таких мінімумів. Вони будуть найнижчими точками «басейнів», які будуть затоплені. Також мінімуми потрібно позначити мітками (label). Міткою може бути, наприклад, номер індексу або значення цього пікселя на початковому зображенні (в нашому випадку це номер індексу).

```
while (foundMinimums < MIN_POINTS) {
    int minValue = 256;
    int minPosition = -1;
    for (int i = 0; i < processedPixels.length; i++) {
        if ((processedPixels[i] == 1) && (data[i] < minValue)) {
            minPosition = i;
            minValue = data[i];
        }
    }

    if (minPosition != -1) {
        int x = minPosition % width;
        int y = minPosition / width;
        int label = foundMinimums;
        int grey = data[x + width*y]&0xff;
        minPixels.add(new WatershedPixel(x,y,grey,label));
        int w = LOCATION / 2;
        fill(width, height,  x1: x-w, y1: y-w, x2: x+w, y2: y+w, value: 0, processedPixels);
        processedPixels[minPosition] = 0;
        foundMinimums++;
    } else {
        break;
    }
}
```

Рис. 3. 20 Пошук мінімальних значень

Для кожної знайденої мінімальної точки ми додаємо навколишні пікселі до черги пріоритетів, встановлюємо початкові мітки і сортуємо цю чергу в порядку зростання від темних до світлих пікселів.

Тепер починаємо так зване "затоплення". Перший піксель (найтемніший) видаляється із черги пріоритетів. При цьому перевіряємо сусідні пікселі на наявність цього пікселя. Якщо всі позначені пікселі мають однакову мітку, позначаємо поточний піксель такою ж міткою. Таким чином поточний піксель знаходиться «всередині» басейну. Важливим моментом є те, що має бути принаймні один позначений піксель (той, який раніше був поруч із поточним пікселем).

```
while (vector.size() > 0) {
    WatershedPixel minPixel = null;
    minPixel = (WatershedPixel) vector.remove( position: 0);
    int x = minPixel.x;
    int y = minPixel.y;
    int label = minPixel.label;
    int[] labels = new int[NEIGHBOUR_PIXELS];

    labels[0] = getLabel(width, height, x, y: y-1, processedPixels);
    labels[1] = getLabel(width, height, x: x+1, y, processedPixels);
    labels[2] = getLabel(width, height, x, y: y+1, processedPixels);
    labels[3] = getLabel(width, height, x: x-1, y, processedPixels);

    if (NEIGHBOUR_PIXELS == 8) {
        labels[4] = getLabel(width, height, x: x-1, y: y-1, processedPixels);
        labels[5] = getLabel(width, height, x: x+1, y: y-1, processedPixels);
        labels[6] = getLabel(width, height, x: x+1, y: y+1, processedPixels);
        labels[7] = getLabel(width, height, x: x-1, y: y+1, processedPixels);
    }

    if (!isBorder(minPixel, labels)) {
        processedPixels[x + width*y] = minPixel.label;
    }
```

Рис. 3. 21 "Затоплення" басейну

Якщо є принаймні два пікселі з різними мітками, це означає, що знайшлася межа між двома різними затопленнями. Тоді даний піксель позначається як «край». Наприкінці додаємо всі сусідні пікселі, які не мають мітки, до черги пріоритетів.

Цей крок ("затоплення") необхідно повторювати допоки черга пріоритетів не стане порожньою, тобто всі пікселі зображення не будуть оброблені.

Результат програми:

Рис. 3. 22 Вхідне зображення для алгоритму вододілу

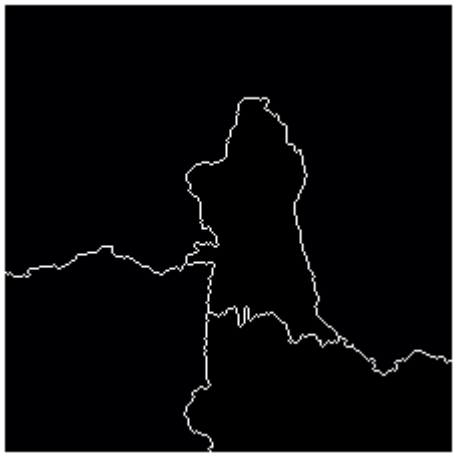

Рис. 3. 23 Результат алгоритму вододілу

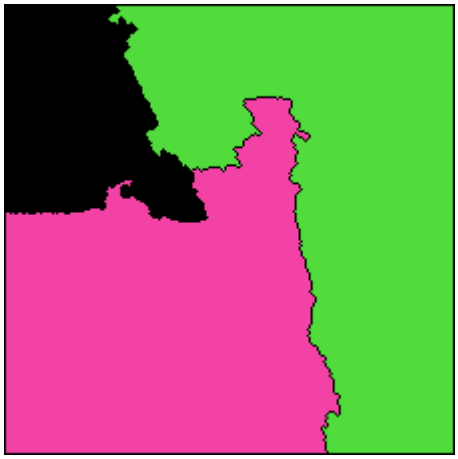

Рис. 3. 24 Результат алгоритму вододілу з використанням бібліотеки OpenCV

3.4 Реалізація алгоритму кластеризації методом K-середніх

Під час реалізації даного методу було вирішено створити окремий клас Cluster, який зберігає інформацію про RGB значення пікселя (центру кластера), та ідентифікатор (індекс кластера в масиві усіх кластерів). Відповідно до заданої кількості кластерів створюємо їх згідно з діаграмою Вороного (обраний піксель є центром даного кластера).

```
public static Cluster[] createClusters(BufferedImage img) {
    int x = 0;
    int y = 0;
    int xDist = img.getWidth() / AMOUNT_OF_CLUSTERS;
    int yDist = img.getHeight() / AMOUNT_OF_CLUSTERS;
    Cluster[] result = new Cluster[AMOUNT_OF_CLUSTERS];

    for (int i = 0; i < AMOUNT_OF_CLUSTERS; i++) {
        result[i] = new Cluster(i, img.getRGB(x, y));
        x += xDist;
        y += yDist;
    }
    return result;
}
```

Рис. 3. 25 Створення кластерів

Вводимо змінну changeCluster типу boolean, яка на початку алгоритму приймає значення true. Вона позначає, що у нас є пікселі, які знаходяться у неправильному кластері. Далі у циклі для кожного пікселя на зображенні необхідно знайти найближчий кластер (тобто той, який має найменшу відстань до даного пікселя). Також треба перевірити, чи належить цей піксель іншому кластеру. Якщо так, видаляємо його зі старого кластера і додаємо до нового.

```
while (changeCluster) {
    changeCluster = false;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int pixel = image.getRGB(x, y);
            Cluster cluster = findMinimalCluster(pixel);
            if (processedPixels[width*y + x] != cluster.getId()) {
                changeCluster = true;
                processedPixels[width*y + x] = cluster.getId();
            }
        }
    }

    for (Cluster cluster : clusters)
        cluster.clear();

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int clusterId = processedPixels[width*y + x];
            clusters[clusterId].addPixel(image.getRGB(x, y));
        }
    }
}
```

Рис. 3. 26 Додавання пікселя до кластера

Коли новий піксель додано до кластера, коректуємо значення центрального пікселя, врахувавши значення нового. І видаляємо значення цього пікселя із колишнього кластера.

```
void addPixel(int data) {
    redSum += data>>16&0x000000FF;
    greenSum += data>>8&0x000000FF;
    blueSum += data&0x000000FF;
    amountOfPixels++;
    red = redSum / amountOfPixels;
    green = greenSum / amountOfPixels;
    blue = blueSum / amountOfPixels;
}
```

Рис. 3. 27 Додавання пікселя до кластера

Повторюємо цикл, доки не залишиться пікселів, для яких треба змінити кластер.

Результат програми:

Рис. 3. 28 Вхідне зображення для алгоритму кластеризації методом К-середніх

Рис. 3. 29 Результат алгоритму кластеризації методом К-середніх, де К = 2

Рис. 3. 30 Результат алгоритму кластеризації методом K-середніх, де K = 5

Рис. 3. 31 Результат алгоритму кластеризації методом K-середніх, де K = 50

Рис. 3. 32 Результат алгоритму кластеризації методом K-середніх з використанням бібліотеки OpenCV, де K = 2

Рис. 3. 33 Результат алгоритму кластеризації методом K-середніх з використанням бібліотеки OpenCV, де K = 5

Рис. 3. 34 Результат алгоритму кластеризації методом K-середніх з використанням бібліотеки OpenCV, де K = 50

Висновок

У результаті виконання даної роботи було досліджено різні техніки та підходи сегментації зображень, варіанти їх класифікації. Наприклад, прийнято розрізняти сегментацію на основі підходу та техніки. Серед першого варіанту можна виділити регіональний та підхід на основі кордонів. А до сегментації на основі техніки належать конструкційні стохастичні та гібридні алгоритми. Крім того, існує поділ сегментації на такі групи: семантична, сегментація екземплярів та паноптична.

Також було реалізовано чотири основні алгоритми.

Алгоритм виділення країв з використанням оператора Собеля досить повільно обчислюється, його ядро переважно згладжує вхідне зображення і таким чином робить оператор менш чутливим до шуму. Оскільки оператор Собеля заснований на згортанні зображення з невеликим, роздільним і цілочисельним фільтром у горизонтальному та вертикальному напрямку, тому є порівняно недорогим з точки зору обрахунків. З іншого боку, градієнтна апроксимація, яку він створює, є відносно грубою, зокрема для високочастотних варіацій зображення.

Для оптимізації алгоритму виділення країв було запропоновано використовувати оператор Шара, який за своєю логікою нагадує оператор Собеля, однак використовує інші значення маски. Також було обчислено градієнти у чотирьох напрямках (0, 45, 90 та 135 градусів). Це забезпечило уникнення втрати інформації з вхідного зображення, таким чином додавши точності у знаходженні країв.

Алгоритм на основі порогів можна вважати одним із найпростіших у реалізації, адже його основне завдання полягає знаходженні оптимального значення порогу. Є можливість власноруч перевіряти різні значення та вибрати одне, проте це неефективно та доволі незручно. Метод Оцу дозволив знайти його автоматично. Проте він має декілька обмежень. Наприклад, ми не можемо

використовувати його для зображень, які не є бімодальними (зображення, гістограми яких мають більше ніж два піки).

Під час реалізації алгоритму вододілу було виявлено, що утворені межі формують замкнуті та зв'язані регіони, що можна віднести до переваг даного алгоритму. Також контури на результуючому зображенні завжди відповідають реальним контурам предметів на вхідному. Не менш важливим є і те, об'єднання всіх окремих сегментів утворюють єдиний регіон зображення. Однак даний алгоритм часто створює надмірну сегментацію, особливо це характерно для реальних фотографій.

Кластеризація методом К-середніх на відміну від більшості інших алгоритмів підходить для роботи з великою кількістю даних і оброблює їх навіть швидше, ніж менші набори. Однак головним недоліком є те, що необхідно самостійно вказувати число кластерів на початку алгоритму. Це означає, що у більшості випадків необхідно буде декілька спроб для отримання бажаного результату.

Сегментація часто використовується як перший крок до отримання інформації із зображень, тому це все ще активна область для дослідження.

Список використаних джерел

1. Image Segmentation Techniques [Електронний ресурс] https://www.upgrad.com/blog/image-segmentation-techniques/#:~:text=The%20primary%20goal%20of%20image,first%20step%20for%20image%20analysis.
2. Semantic vs Instance vs Panoptic: Which Image Segmentation to Choose? [Електронний ресурс] https://soulpageit.com/semantic-vs-instance-vs-panoptic-which-image-segmentation-to-choose/
3. An Introduction to Image Segmentation: Deep Learning vs. Traditional [Електронний ресурс] https://www.v7labs.com/blog/image-segmentation-guide
4. Computer Vision Tutorial [Електронний ресурс] https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/
5. Understanding Edge Detection (Sobel Operator) [Електронний ресурс] https://medium.datadriveninvestor.com/understanding-edge-detection-sobel-operator-2aada303b900
6. Gonzalez R. Digital Image Processing. 4Th Edition / R. Gonzalez, R. Wood., 2018.
7. Image segmentation with Watershed algorithm [Електронний ресурс] https://datahacker.rs/007-opencv-projects-image-segmentation-with-watershed-algorithm/
8. Research on Distance Transform and Neural Network Lidar Information Sampling Classification-Based Semantic Segmentation of 2D Indoor Room Maps [Електронний ресурс] https://www.researchgate.net/publication/349323744_Research_on_Distance_Transform_and_Neural_Network_Lidar_Information_Sampling_Classification-Based_Semantic_Segmentation_of_2D_Indoor_Room_Maps
9. Кластеризація. Типи алгоритмів [Електронний ресурс]

http://bourabai.ru/tpoi/analysis6.htm

10. In-depth Intuition of K-Means Clustering Algorithm in Machine Learning [Електронний ресурс] https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning/
11. OpenCV Sobel Derivatives [Електронний ресурс] https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
12. Edge Detection Based on Improved Sobel Operator [Електронний ресурс] https://www.atlantis-press.com/proceedings/ceis-16/25867843
13. Edge detection using Prewitt, Scharr and Sobel Operator https://www.geeksforgeeks.org/edge-detection-using-prewitt-scharr-and-sobel-operator/ OpenCV Image Thresholding [Електронний ресурс] https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html
14. OpenCV Image Segmentation with Distance Transform and Watershed Algorithm [Електронний ресурс] https://docs.opencv.org/4.x/d2/dbd/tutorial_distance_transform.html
15. OpenCV K-means [Електронний ресурс] https://juejin.cn/post/6966609647054618655