Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ OPENCV ТА БІБЛІОТЕК PYTHON»

Виконав: студент IV курсу, групи 1АКІТ-21б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології
(шифр і назва спеціальності)
Іван КРИВОШЕЙ
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри АІІТ
Ольга СОФИНА
(прізвище та ініціали)
«12» 06 2025 р.

Рецензент: д.т.н., проф. кафедри ІКСУ
Марія ЮХИМЧУК
(прізвище та ініціали)
«13» 06 2025 р.

Допущено до захисту
завідувач кафедри АІІТ
Олег БІСІКАЛО
«16» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 15 – Автоматизація та приладобудування
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
_____ д.т.н., проф. Олег БІСІКАЛО
«16» 06 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кривошею Івану Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для автоматизації процесу сегментації зображень з використанням OpenCV та бібліотек Python»
керівник роботи Софина Ольга Юріївна, к.т.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «20» березня 2025 року №97
2. Термін подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи:
   - застосування алгоритмів сегментації зображень;
   - використання бібліотек Python для реалізації: OpenCV, scikit-image;
   - реалізація графічного інтерфейсу користувача (GUI) – через PyQt5;
   - кількість методів сегментації – не менше 4;
   - тип зображень – grayscale або RGB, формат JPEG/PNG.
4. Зміст текстової частини (перелік питань, які потрібно розробити) провести огляд сучасних методів сегментації зображень, проаналізувати бібліотеки OpenCV та scikit-image для обробки зображень, запропонувати архітектуру програмного забезпечення для сегментації зображень, розробити програмне забезпечення на Python з GUI для запуску алгоритмів сегментації, реалізувати систему оцінки якості сегментації за допомогою метрик
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Структурна схема програмного забезпечення для сегментації зображень; Принципова схема роботи модулів; Блок-схеми реалізації алгоритмів Graph Cut, K-means, Watershed, Region Growing; Графічний інтерфейс користувача; Приклади результатів сегментації із зазначенням вихідних зображень та результатів; Порівняльна діаграма якості сегментації за метриками IoU, F1-score, MSE)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Софина О. Ю., доц. каф. АІІТ		

7. Дата видачі завдання 20.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Термін виконання		Примітка
		початок	закінчення	вик
1	Вибір, узгодження та затвердження теми БКР	03.10.2024	09.10.2024	вик
2	Аналіз предметної області та опрацювання літературних джерел.	12.03.2025	29.03.2025	вик
3	Постановка задач для вирішення в БКР	01.04.2025	26.04.2025	вик
4	Вирішення поставлених задач	29.04.2025	10.05.2025	вик
5	Підтвердження достовірності отриманих результатів	13.05.2025	24.05.2025	вик
7	Оформлення пояснювальної записки та графічної частини	27.05.2025	31.05.2025	вик
8	Перевірка ПЗ на відповідність вимогам	03.06.2025	07.06.2025	вик
9	Попередній захист, доопрацювання, та рецензування БКР	10.06.2025	15.06.2025	вик
10	Захист БКР	16.06.2025	23.06.2025	вик

Студент _____ (підпис) Іван КРИВОШЕЙ (ім'я та прізвище)

Керівник роботи _____ (підпис) Ольга СОФИНА (ім'я та прізвище)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 102 сторінок формату А4, включаючи додатки, на яких представлено 24 рисунки, 4 таблиці, список використаних джерел містить 22 найменування.

У роботі розглянуто методи сегментації зображень, такі як порогова сегментація, кластеризація (k-means), нарощування областей, Graph Cut та Watershed, а також порівняно їх реалізацію у бібліотеках OpenCV та scikit-image. Запропоновано архітектуру програмного забезпечення для багатомодульної системи автоматичної обробки зображень з графічним інтерфейсом користувача на базі PyQt5. Програмне забезпечення реалізовано мовою Python з використанням сучасних бібліотек машинного навчання та комп'ютерного зору, що забезпечує високу продуктивність і масштабованість. Надано приклади обробки зображень, результати сегментації та метрики якості (IoU, F1-score, MSE), які підтверджують ефективність застосованих алгоритмів.

Ключові слова: сегментація зображень, комп'ютерний зір, Python, OpenCV, scikit-image, Graph Cut, Watershed, кластеризація, графічний інтерфейс.

ANOTATION

The bachelor's thesis consists of 102 pagesof A4 format, including appendices, which contain 24 figures, 4 tables,the list of references contains 22 titles.

The work considers image segmentation methods such as threshold segmentation, clustering (k-means), region building, Graph Cut and Watershed, and compares their implementation in the OpenCV and scikit-image libraries. The software architecture for a multimodule automatic image processing system with a graphical user interface based on PyQt5 is proposed. The software is implemented in Python with the use of modern machine learning and computer vision libraries, which ensures high performance and scalability. Examples of image processing, segmentation results, and quality metrics (IoU, F1-score, MSE) are presented to confirm the effectiveness of the applied algorithms.

Keywords: image segmentation, computer vision, Python, OpenCV, scikit-image, Graph Cut, Watershed, clustering, graphical interface.

ЗМІСТ

ВСТУП

У сучасну еру цифрових технологій сегментація зображень виступає фундаментальною складовою комп'ютерного зору, що дозволяє розділяти вхідні зображення на смислові області для подальшого аналізу. Застосування методів сегментації охоплює широкий спектр галузей: від медичної діагностики до автономного водіння, де алгоритми сегментації допомагають виявляти дорожні об'єкти та перешкоди.

Об'єкт дослідження – процес сегментації зображень, що включає методи класичного машинного навчання та глибинних нейронних мереж. **Предмет дослідження** – програмні інструменти для сегментації зображень на основі бібліотек OpenCV, scikit–image та інших фреймворків, а також методи оцінки їх ефективності.

Класичні алгоритми сегментації вже довели свою ефективність у багатьох прикладних задачах. Водночас стрімкий розвиток глибинного навчання призвів до появи гібридних підходів, зокрема U–Net та трансформерних моделей, які забезпечують суттєве підвищення точності в умовах складних зображень.

Метою даної кваліфікаційної роботи є покращення програмного забезпечення для сегментації зображень на мові Python з використанням бібліотек OpenCV та scikit–image, а також впровадження інтуїтивного графічного інтерфейсу на базі PyQt5 для налаштування параметрів алгоритмів у реальному часі. Додатково передбачається оцінка якості сегментації за допомогою класичних метрик (IoU, Dice, F1–score) і оптимізація продуктивності з використанням апаратного прискорення (CUDA, OpenCL).

Для досягнення поставленої мети **потрібно вирішити такі задачі**:

1) Провести теоретичний аналіз класичних (порогова, k-means, GMM, Region Growing, Graph Cut, Watershed) та сучасних (U-Net) методів сегментації.

2) Реалізувати класичні алгоритми за допомогою OpenCV та scikit-image.

3) Розробити GUI (PyQt5) для завантаження зображень, вибору алгоритму, налаштування параметрів та візуалізації результатів.

4) Реалізувати оцінку якості за метриками IoU, Dice та F1-score.

5) Провести тестування та порівняльний аналіз ефективності алгоритмів на різних зображеннях.

6) Впровадити апаратне прискорення (CUDA/OpenCL) для оптимізації продуктивності.

Методи дослідження включають:

1) Теоретичний аналіз наукової літератури з питань сегментації зображень.

2) Експериментальне тестування алгоритмів (порогова сегментація, k–means, Gaussian Mixture Models, Region Growing, Graph Cut, Watershed) на різних наборах даних.

3) Програмна реалізація алгоритмів за допомогою бібліотек OpenCV, scikit–image, TensorFlow/PyTorch (для глибинних методів).

4) Порівняльний аналіз ефективності алгоритмів за метриками IoU, Dice та F1–score.

5) Оптимізація продуктивності за рахунок паралельних обчислень на GPU (CUDA, OpenCL).

Практичні результати роботи:

1) Розроблено програмне забезпечення для сегментації зображень з графічним інтерфейсом, що дозволяє інтерактивно налаштовувати параметри алгоритмів.

2) Проведено порівняльний аналіз класичних методів сегментації на основі метрик якості.

3) Впроваджено механізми апаратного прискорення для підвищення швидкодії алгоритмів.

4) Отримано рекомендації щодо вибору оптимального методу сегментації залежно від типу зображення.

Апробація результатів дослідження:

Результати роботи було представлено на LIV Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації (2025) [22].

1 ОГЛЯД МЕТОДІВ ТА РЕАЛІЗАЦІЙ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ

1.1 Вступ до сегментації зображень

Сегментація зображень є однією з найважливіших задач у сфері обробки та аналізу цифрових зображень. Вона полягає у розділенні зображення на значущі області або окремі об'єкти, які можуть мати певні візуальні, текстурні або семантичні характеристики [1]. Основною метою сегментації є спрощення або зміна структури представлення зображення, що робить його більш інформативним і придатним для подальшого аналізу, класифікації або обробки.

Ця задача має широкий спектр застосувань у різних галузях науки і техніки. Наприклад, у медичній діагностиці сегментація дозволяє виділяти органи, патологічні утворення (наприклад, пухлини, аневризми, аномалії тканин) на рентгенівських, магнітно–резонансних (МРТ) або комп'ютерно–томографічних (КТ) зображеннях [11-17]. Це значно покращує якість діагностики та допомагає лікарям приймати обґрунтовані рішення.

В автономних транспортних засобах алгоритми сегментації використовуються для виявлення дорожніх об'єктів, таких як пішоходи, автомобілі, дорожні знаки, розмітка та перешкоди [13].

Завдяки таким технологіям автомобілі з автопілотом можуть швидко та точно аналізувати навколишнє середовище, що є критично важливим для забезпечення безпеки руху.

Сегментація зображень у відеоспостереженні дозволяє виявляти та відстежувати рухомі об’єкти, що підвищує ефективність автоматизованого моніторингу. Вона допомагає виокремлювати об’єкти на тлі змінного середовища, зменшуючи вплив шуму та покращуючи розпізнавання [14].

Ці методи широко застосовуються у системах безпеки, криміналістиці та військовій сфері. Вони використовуються для розпізнавання осіб, аналізу натовпу та виявлення підозрілої поведінки, а також у військових системах для стеження за об’єктами на полі бою чи стратегічних територіях.

1.2 Класифікація методів сегментації

Методи сегментації зображень можна класифікувати за принципами їхньої роботи та підходами до обробки даних. Вони використовуються для розділення зображення на області, що мають схожі характеристики, такі як колір, текстура або інтенсивність [1].

Порогові методи (Thresholding) – один із найпростіших способів сегментації, що передбачає встановлення певного значення яскравості або кольору, за яким пікселі класифікуються як належні до об’єкта або фону. Застосовується в бінарній сегментації та може бути адаптивним або глобальним. Формула для простого глобального порогу [15]:

$$S(x,y) = \begin{cases} 1, & I(x,y) \geq T \\ 0, & I(x,y) < T \end{cases}, \tag{1.1}$$

де I(x,y) — інтенсивність пікселя, T — поріг.

Методи кластеризації – ґрунтуються на групуванні пікселів відповідно до їхніх характеристик, таких як інтенсивність або текстурні властивості. Найпоширеніші алгоритми включають k–means, ієрархічну кластеризацію та GMM (Gaussian Mixture Model), які дозволяють сегментувати зображення без попереднього знання про об’єкти. Формула для оновлення центрів кластерів:

$$\mu_k = \frac{1}{|C_k|} \sum_{x \in C_k} x, \tag{1.2}$$

де C_k — множина пікселів, що належать кластеру k, а μ_k — його центр.

Методи, засновані на аналізі областей (Region–Based Segmentation) – будуються на пошуку взаємопов’язаних груп пікселів, що мають спільні властивості. Наприклад, алгоритм зростання областей (Region Growing) починається з обраних "насіннєвих" точок і поступово розширює межі, додаючи сусідні пікселі, які відповідають певним критеріям подібності.

Формула для Graph Cut через функцію енергетичної мінімізації:

$$E(A) = R(A) + B(A), \quad (1.3)$$

де R(A) — штраф за відхилення інтенсивностей, B(A) — вага зв’язків між пікселями.

На рисунку 1.1 зображено приклад Gaussian Mixture Model з 2 компонентів, що відповідають двовимірним розподілам, з відповідними розподілами ймовірностей на спільних осях:

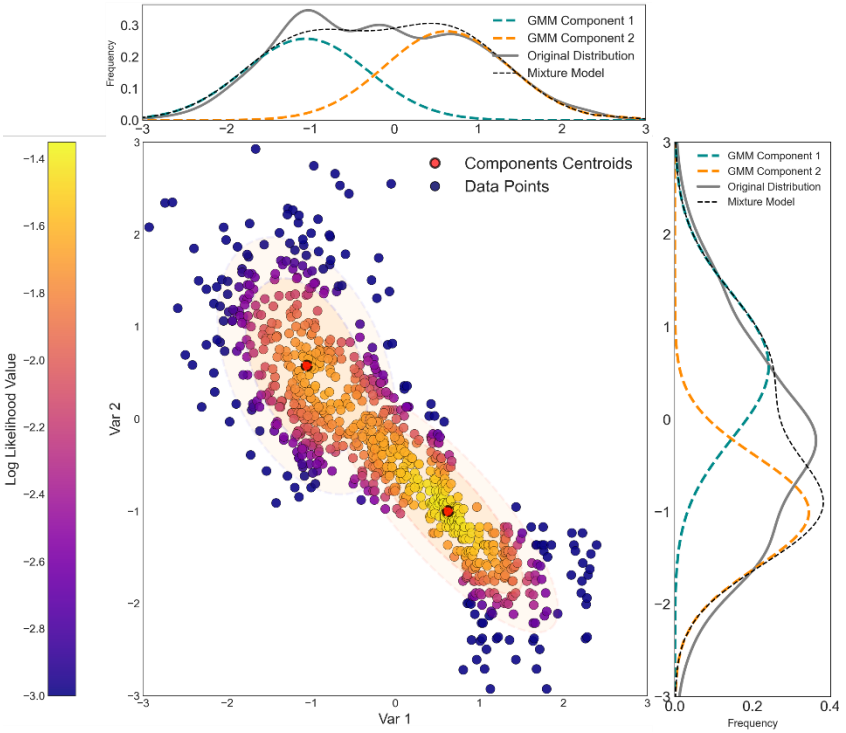

Рисунок 1.1 – GMM з 2 компонентів і відповідними розподілами

Методи, засновані на аналізі областей (Region–Based Segmentation) будуються на пошуку взаємопов’язаних груп пікселів, що мають спільні властивості.

Наприклад, алгоритм зростання областей (Region Growing) починається з обраних "насіннєвих" точок і поступово розширює межі, додаючи сусідні пікселі, які відповідають певним критеріям подібності.

Приклад застосування Region–Based Segmentation: градієнт зображення обчислений за допомогою фільтра Собела. Стек S, де координата z позначає L.

Колір маркування – G Average (L, c). Проекція максимуму стека S. Однак, інтенсивність була спрощена, а межі областей мають однакове значення для заданого L (більш чітко видно при порівнянні (E) з (B)).

Обчислення регіональних максимумів (D) повертає ті області пікселів, які оточені строго меншими значеннями пікселів (чорним кольором, F), а заповнення областей показано в (G).

Таке визначення регіональних максимумів означає, що дотичні об'єкти потенційно можуть бути не виявлені, якщо вони були позначені різними значеннями (тобто обидва об'єкти не можуть бути оточені пікселями зі строго меншим значенням) [12]. Дійсно, три об'єкти в (D) не були сегментовані в (F); однак, це можна успішно вирішити за допомогою обмежень специфікації об'єктів, що відображені на рисунку 1.2:

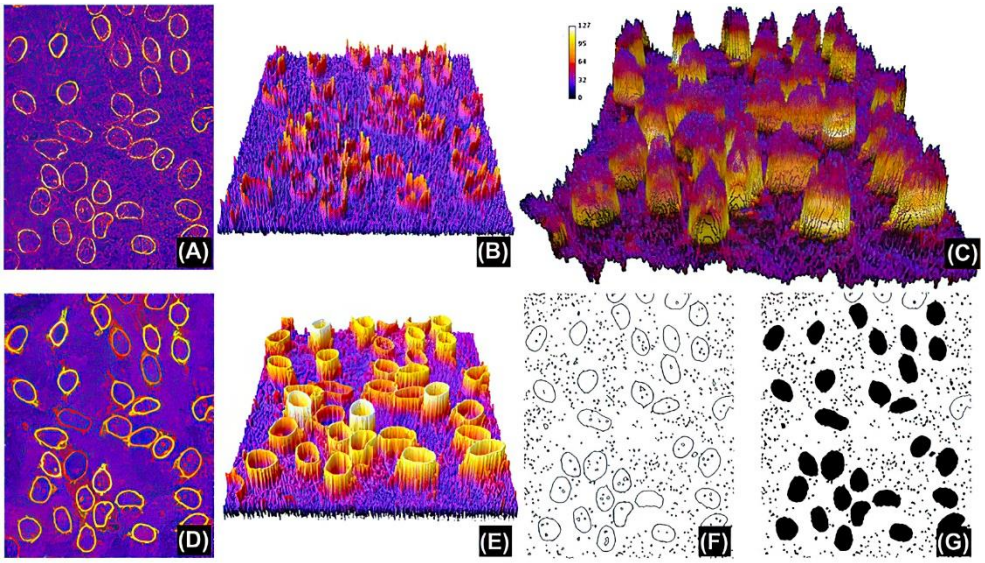

Рисунок 1.2 – Сегментація та 3D–візуалізація об'єктів

Графові методи сегментації – трактують зображення як математичний граф, де пікселі є вершинами, а зв’язки між ними визначаються подібністю.

Застосовуються методи мінімального розрізу графа (Graph Cut) або нормалізованого розрізу (Normalized Cut), які дозволяють виконувати складні види сегментації, наприклад, багатошарову або контекстно–залежну.

Метод водозливного перетворення (Watershed Transform) – розглядає зображення як топографічну карту, де інтенсивність пікселів відповідає висоті [18].

Алгоритм «заповнює» зображення водою з певних точок, що дозволяє розділити області відповідно до локальних мінімумів яскравості, що особливо ефективно для обробки рентгенівських і томографічних знімків.

На рисунку 1.3 зображено приклад процесу трансформації вододілу (Watershed Transform) вихідного зображення, де (A) – зображення видалення фону,
(B) – відфільтроване градієнтне амплітудне зображення, (C) – зображення трансформації вододілу:

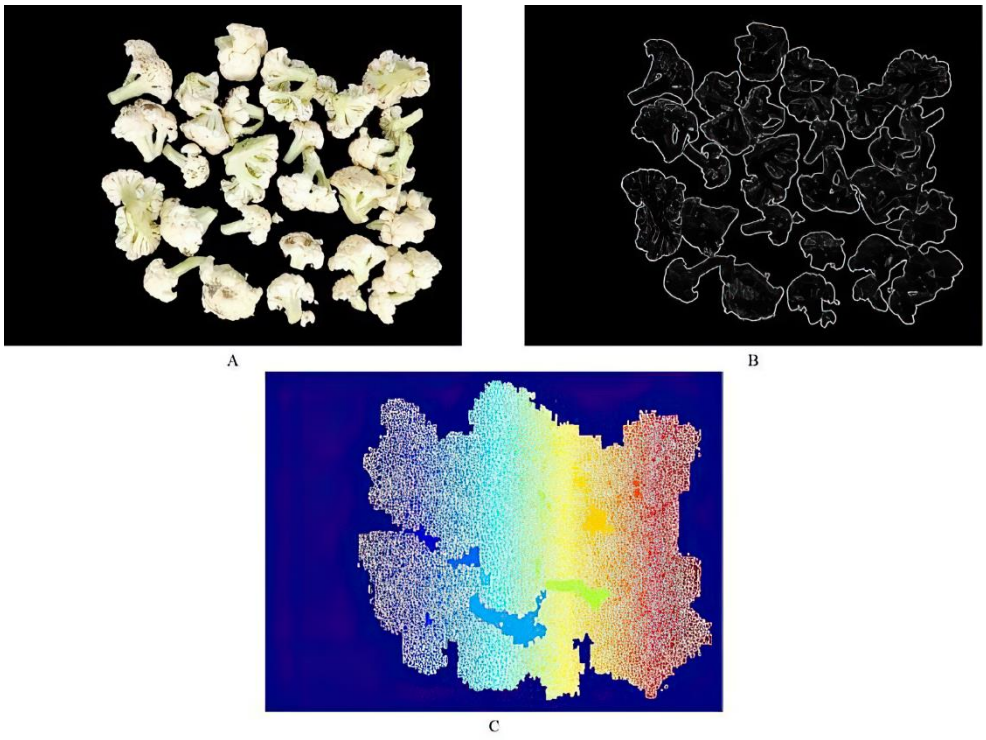

Рисунок 1.3 – Аналіз форми та глибини об'єктів

У сучасних дослідженнях активно використовуються гібридні методи та глибокі нейронні мережі (Deep Learning), які поєднують класичні алгоритми з можливостями штучного інтелекту для покращення точності та адаптивності до складних зображень [2-16].

1.3 Огляд основних методів сегментації

Порогова сегментація є методом аналізу зображень, що ґрунтується на визначенні рівня інтенсивності, який поділяє пікселі на області – об’єкти та фон [5].

Її головною перевагою є простота реалізації та швидкість обчислень, що робить цей метод особливо привабливим для обробки великої кількості зображень у режимі реального часу.

Серед методів автоматичного встановлення порогового значення особливо виділяється метод Оцу, який дозволяє визначити оптимальний поріг, аналізуючи гістограму інтенсивностей. Його алгоритм базується на розрахунку порога, що мінімізує внутрішньокласову дисперсію яскравості пікселів у двох групах: тієї, що відповідає об’єкту, і тієї, що відповідає фону.

Ці приклади, які ілюстровані на рисунках 1.4 та 1.5, демострують автоматичне Otsu thresholding:

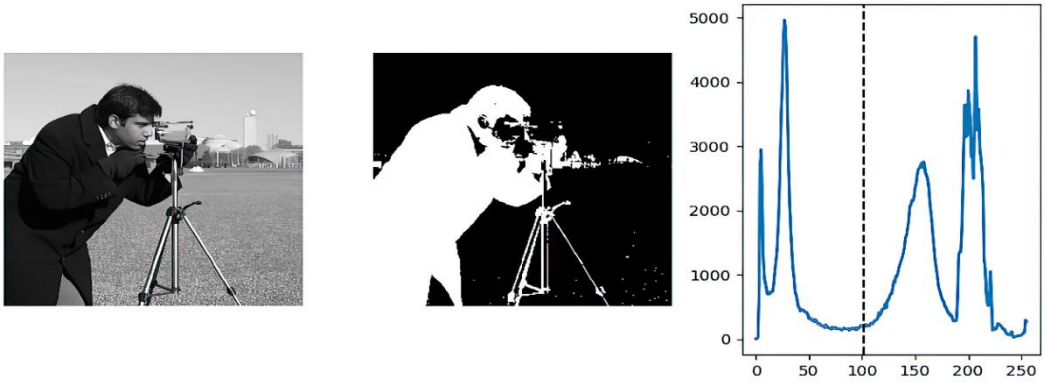

Рисунок 1.4 – Обробка зображень та аналіз інтенсивності

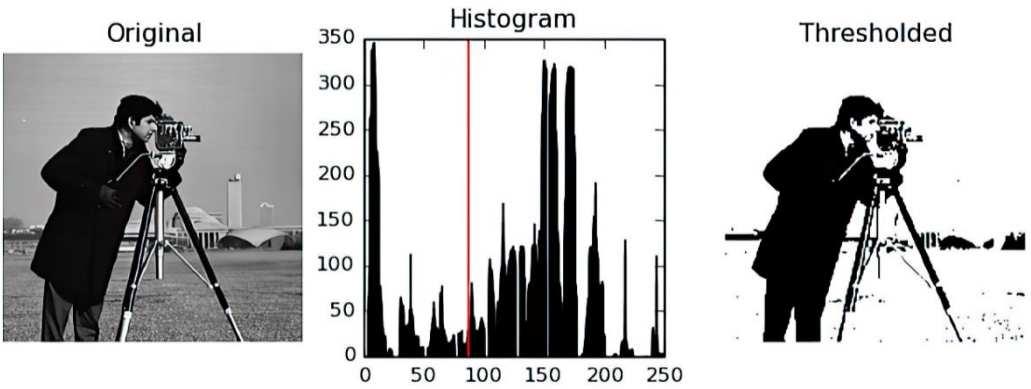

Рисунок 1.5 – Порогова обробка та гістограмний аналіз

Метод Оцу демонструє високу ефективність для зображень, у яких гістограма має бімодальний розподіл, тобто розділені піки інтенсивності [5].

Це характерно для багатьох типів зображень, зокрема у медичній діагностиці (аналіз МРТ, рентгенівських знімків), машинному зорі та системах контролю якості [11].

Формула для порога T, що мінімізує внутрішньокласову дисперсію:

$$\sigma_{\omega}^2(T) = \omega_1(T)\sigma_1^2(T) + \omega_2(T)\sigma_2^2(T), \quad (1.4)$$

де:

$\omega_1(T), \omega_2(T)$ – частки пікселів у класах (фон і об'єкт) [14],

$\sigma_1^2(T), \sigma_2^2(T)$ – дисперсії інтенсивностей у цих класах.

Оптимальне порогове значення T^* вибирається як таке, що мінімізує $\sigma_{\omega}^2(T)$.

Однак метод має деякі обмеження. Зокрема, його точність суттєво знижується в умовах неоднорідного освітлення, коли рівень яскравості змінюється по всьому зображенню, а також у випадках наявності значного рівня шуму.

У таких ситуаціях можуть застосовуватися адаптивні методи порогової сегментації, які враховують локальні особливості зображення, або методи попередньої обробки, такі як фільтрація Гауса чи вирівнювання гістограми.

Замість фіксованого порогу T використовується локальне значення T(x,y), яке розраховується для кожного пікселя в залежності від його оточення (наприклад, середнього значення інтенсивності в околицях N×N):

$$T(x, y) = \frac{1}{N^2}\sum_{(i,j)\in N(x,y)} I(i, j), \quad (1.5)$$

де N(x,y) – локальне вікно навколо пікселя (x,y).

Додатково, порогову сегментацію часто поєднують із іншими алгоритмами, методами кластеризації або нейронними мережами, що дозволяє покращити якість сегментації та зробити її стійкішою до складних умов освітлення.

Метод кластеризації k–means є одним із найпопулярніших алгоритмів для автоматичного групування пікселів у зображеннях на основі їхніх характеристик, таких як колір або текстурні особливості [6].

Його принцип роботи базується на розподілі всіх пікселів на k кластерів, де кожен кластер представлений своїм центроїдом – середнім значенням характеристик усіх пікселів у групі [6].

Процес кластеризації методом k–means складається з кількох основних етапів. Спочатку слід задати кількість кластерів (k) вручну або визначити автоматично через Elbow Method чи Silhouette Score.

Алгоритм випадково обирає початкові центри кластерів, а для підвищення стабільності часто використовують k–means++, що допомагає уникнути невдалого вибору. Далі кожен піксель призначається найближчому центроїду за евклідовою відстанню, після чого центроїди оновлюються як середнє значення всіх пікселів у кластері.

Набір даних Iris з використанням кластеризації k–means подано на рисунку 1.6:

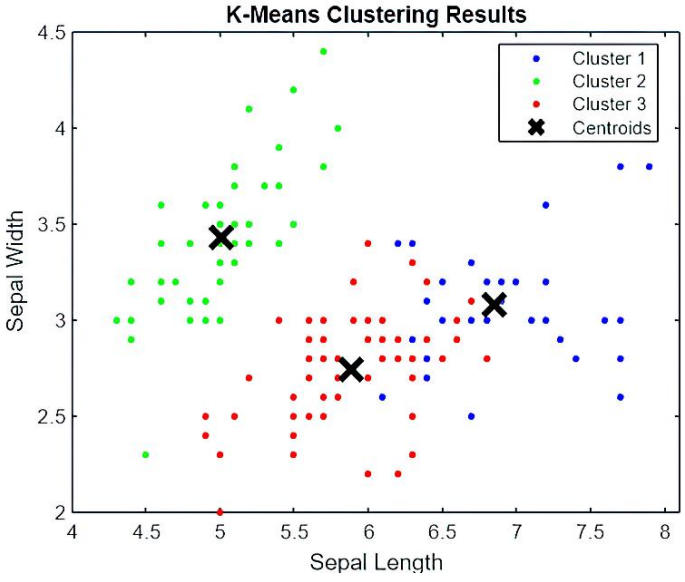

Рисунок 1.6 – Результати кластеризації методом K–Means

Метод k–means є ітеративним та мінімізує суму квадратів відстаней між точками й центроїдами. Він добре працює для сферичних кластерів, але менш ефективний для складних форм. Використання k–means++ покращує вибір початкових центроїдів, а альтернативні метрики, як–от мангеттенська відстань, можуть змінювати результати. Для точнішої кластеризації інколи застосовують комбіновані підходи: DBSCAN чи агломеративне групування.

Синтетичні дані з використанням кластеризації k–means зображено на рисунку 1.7:

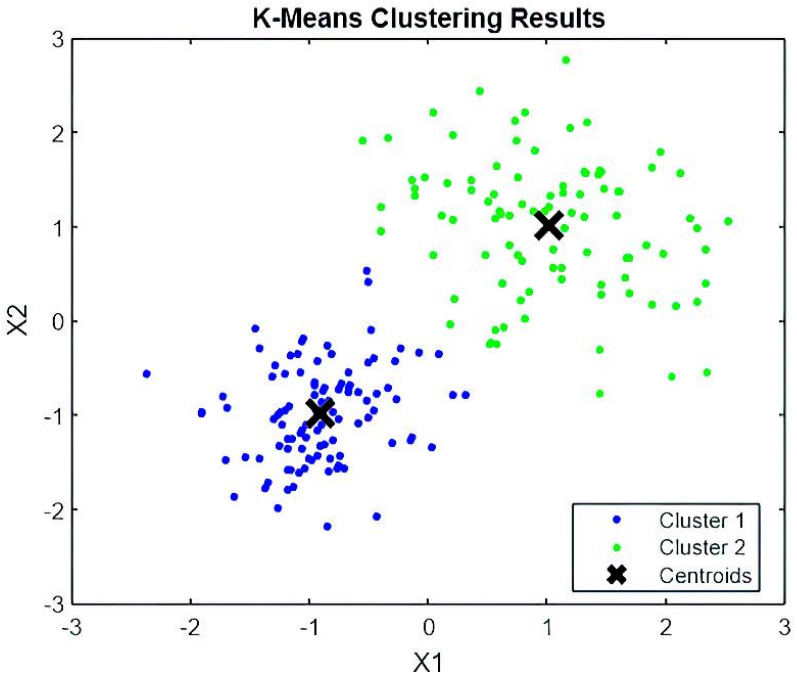

Рисунок 1.7 – K–Means: 2 кластери, центроїди

Переваги методу k–means:

Цей метод легко реалізувати, і він має невисоку обчислювальну складність, що дозволяє використовувати його навіть для обробки великих зображень. Він добре працює із зображеннями, на яких чітко видно компактні кластери.

Крім того, метод є досить універсальним і застосовується не тільки в обробці зображень, а й у стисненні даних, сегментації текстур, медичній діагностиці тощо. Також він знаходить застосування в розпізнаванні образів та аналізі біометричних даних. Завдяки своїй ефективності та гнучкості метод використовується у різних науково–технічних галузях.

Обмеження та недоліки:

Алгоритм чутливий до вибору початкових центрів кластерів — якщо їх визначити неправильно, це може призвести до некоректних результатів або збіжності в локальному мінімумі.

Він найкраще працює для сферичних кластерів, а з витягнутими або складними структурами може мати проблеми. Також метод не завжди правильно сегментує дані, якщо кластери мають різну щільність точок.

Вплив шуму та аномальних значень – у разі сильного шуму на зображенні алгоритм може створювати зайві кластери або неправильно визначати межі сегментації.

Для покращення результатів кластеризації k–means іноді комбінують із іншими методами, такими як Гаусові змішані моделі (GMM), які дозволяють працювати з кластерами довільної форми, або використовують ієрархічну кластеризацію для більш гнучкого аналізу даних.

Останнім часом у комп'ютерному зорі також активно використовуються глибокі нейронні мережі (Deep Learning) для покращення кластеризації та сегментації, що дозволяє адаптивно визначати нелінійні структури кластерів і зменшувати вплив шуму. Як приклад глибокого навчання в сегментації – можна згадати функцію втрат для сегментації, такий як Dice Loss:

$$\text{Loss} = 1 - \frac{2\sum y_{\text{true}} y_{\text{pred}}}{\sum y_{\text{true}} + \sum y_{\text{pred}}}, \tag{1.6}$$

де y_{true} – істинна маска, y_{pred} – передбачена мережею.

Метод нарощування областей (Region Growing) є одним із найбільш інтуїтивно зрозумілих підходів до сегментації зображень, який базується на пошаровому розширенні сегментованих ділянок шляхом поступового приєднання нових пікселів до вже сформованих областей.

Цей процес починається з вибору точок, які називаються насінням. Після, до них додаються сусідні пікселі, якщо вони відповідають критеріям схожості.

Схожість можна визначати за кількома параметрами. Наприклад, можна враховувати яскравість або інтенсивність – у такому разі додаються тільки ті пікселі, чия інтенсивність знаходиться в допустимому діапазоні.

Інший підхід – аналіз кольору, коли алгоритм оцінює кожен колірний канал окремо або використовує певний колірний простір.

Також враховуються особливості текстури, тобто додаються пікселі, які відповідають певним характеристикам поверхні зображення. Крім того, алгоритм може обмежувати розширення області, зважаючи на форму, розмір або орієнтацію об’єкта.

Процес складається з кількох основних етапів. Спочатку визначають початкові точки, які можуть бути обрані вручну або автоматично. Далі аналізуються сусідні пікселі, і якщо вони відповідають заданим критеріям, їх додають до області. Після цього перевіряються вже нові сусідні пікселі, і процедура повторюється. Процес триває доти, доки більше не залишається пікселів, які можна додати.

Метод Region Growing точно визначає межі об’єктів, адаптується до різних зображень завдяки налаштуванню критеріїв схожості, підтримує ручний вибір початкових точок для кращого контролю сегментації та ефективно працює з медичними та промисловими зображеннями [14-15].

Метод Region Growing має важливу особливість – його здатність локально адаптуватися до варіацій інтенсивності зображення, що особливо корисно при аналізі неоднорідних структур. Завдяки цьому він знаходить застосування в медичних дослідженнях для сегментації тканин, де традиційні методи можуть давати похибки через природні варіації щільності. Крім того, у промисловій візуальній інспекції він допомагає визначати дефекти, що мають складну форму або поступові зміни кольору, які складно розпізнати пороговими методами.

Ще однією перевагою є можливість інтеграції Region Growing із сучасними технологіями штучного інтелекту. Наприклад, алгоритми глибокого навчання можуть використовувати результати сегментації як вхідні дані для класифікації об'єктів або прогнозування їх стану.

Методи вирощування регіонів часто дають дуже хорошу сегментацію, яка добре відповідає спостережуваним краям, що можна побачити на рисунку 1.8, де відображено приклад зростання регіону:

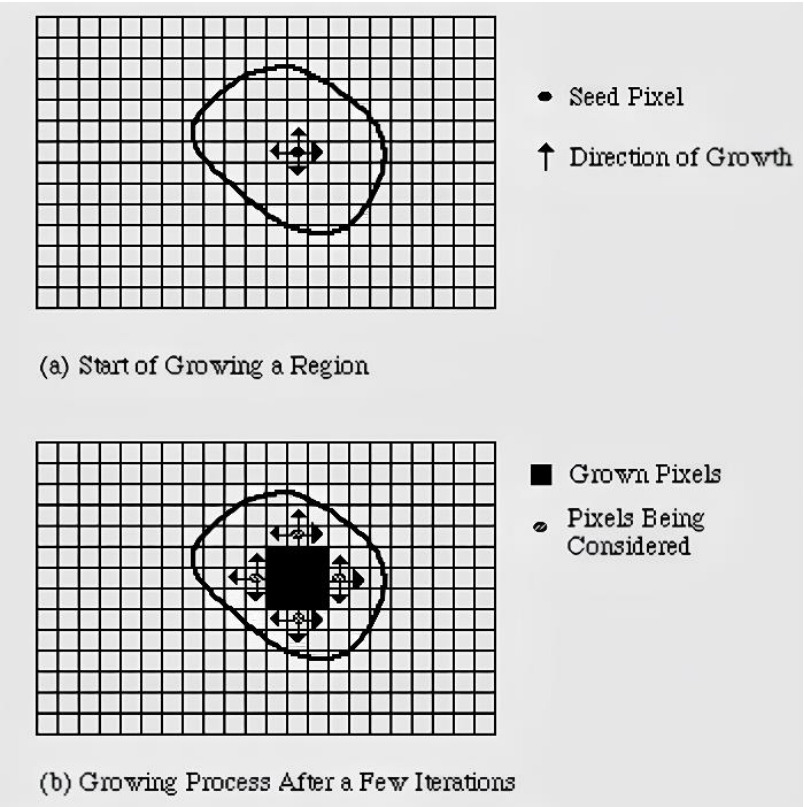

Рисунок 1.8 – Сегментація методом зростання областей

Однак у методу є й певні обмеження. Його точність сильно залежить від правильного вибору початкових точок: невдалий вибір може призвести до неповної або неправильної сегментації.

Також алгоритм чутливий до шуму – навіть невеликі зміни інтенсивності або текстури можуть спричинити помилки, що потребують додаткової фільтрації.

Крім того, якщо об’єкт має неоднорідну структуру або градієнти яскравості, метод може працювати неефективно.

Щоб покращити роботу Region Growing, застосовують кілька підходів. Спочатку зображення обробляють фільтрами, такими як Gaussian або Median, щоб зменшити рівень шуму. Для автоматичного вибору початкових точок можуть використовуватися методи машинного навчання або аналіз гістограми.

Також цей метод часто комбінують із іншими підходами, наприклад, алгоритмом Watershed, кластеризацією або нейромережами, що дозволяє отримати більш точні, стабільні та адаптивні результати в складних умовах, зокрема в медичних та аерокосмічних зображеннях, деяких промислових та наукових задачах.

Функція енергетичного рельєфу для сегментації:

$$E(x,y) = -|\nabla I(x,y)|, \tag{1.7}$$

де ∇I(x,y) — градієнт інтенсивності. Вододіл визначається як лінія між локальними мінімумами.

Завдяки високій точності та адаптивності, метод Region Growing широко використовується в медичній візуалізації (наприклад, для аналізу знімків КТ та МРТ), супутниковій обробці зображень, розпізнаванні об'єктів у відеоспостереженні та біометричних системах [14-15].

Метод Graph Cut є потужним підходом до сегментації зображень, заснованим на представленні зображення у вигляді зваженого неорієнтованого графа. У цій моделі кожен піксель інтерпретується як вершина графа, а зв’язки між ними формуються за допомогою ребер, вага яких визначається рівнем подібності між сусідніми пікселями. Метод дозволяє ефективно розділяти області зображення, враховуючи як локальні характеристики, так і глобальні залежності.

Формула для задачі мінімізації енергетичної функції:

$$E(L) = \sum_p R_p(L_p) + \sum_{p,q} B_{p,q} \cdot \delta(L_p, L_q), \tag{1.8}$$

де $R_p(L_p)$ – вартість присвоєння мітки L_p пікселю p, $B_{p,q}$ – вага ребра між пікселями p та q, $\delta(L_p, L_q)$ – індикаторна функція (дорівнює 1, якщо $L_p \neq L_q$, інакше 0).

Процес сегментації виконується шляхом знаходження мінімального розрізу (min–cut) у графі. Це означає, що алгоритм шукає таке розділення графа на дві підмножини (об’єкт та фон), при якому сума ваг ребер, що з’єднують ці підмножини, є мінімальною [14].

Метод Graph Cut використовується для сегментації зображень, враховуючи як локальні властивості пікселів (яскравість, колір, текстуру), так і загальні характеристики всього зображення, наприклад, положення об'єкта або різкість його меж [15].

Процес сегментації починається зі створення графа, де кожен піксель є окремим вузлом. Додаються два спеціальні вузли: джерело, що відповідає об’єкту, і сток, який представляє фон, забезпечуючи ефективний поділ областей зображення.

Потім обчислюються ваги зв’язків між пікселями. Якщо сусідні пікселі мають схожий колір або яскравість, їхній зв’язок буде сильнішим. Окремо визначаються зв’язки пікселів із джерелом і стоком, що допомагає алгоритму визначити, чи належить піксель до об’єкта чи до фону.

Далі застосовується алгоритм мінімального розрізу, наприклад, Ford–Fulkerson або Dinic’s algorithm. Він знаходить оптимальний спосіб розділення графа так, щоб мінімізувати сумарну вагу розірваних зв’язків.

На завершальному етапі формується остаточний результат. Пікселі, що залишилися у групі з джерелом, вважаються частиною об’єкта, а ті, що пов’язані зі стоком, належать до фону.

Метод Graph Cut має кілька переваг. Він дозволяє знаходити оптимальний поділ всього зображення, а не тільки окремих його частин.

Його можна налаштовувати, щоб краще розпізнавати об’єкти, використовуючи різні функції. Також цей метод добре працює навіть у складних умовах, наприклад, якщо зображення має нерівномірне освітлення або містить шум.

Проте у нього є й недоліки. Обчислення займає багато часу і ресурсів, особливо якщо зображення велике.

Якість сегментації залежить від правильно підібраних параметрів алгоритму. Щоб покращити роботу цього методу, можна спочатку обробляти зображення з меншою роздільною здатністю, а потім уточнювати результат.

Також його часто поєднують з іншими методами, наприклад, нейронними мережами або кластеризацією. Попередня обробка, така як згладжування або підсилення контурів, теж допомагає отримати кращий результат.

Метод Graph Cut застосовують у різних сферах. У медицині його використовують для виділення органів і пухлин на знімках КТ та МРТ [11]. У сфері комп'ютерного зору він допомагає розпізнавати об'єкти у відео та в робототехніці, а також у 3D–реконструкції та обробці зображень, сегментації сцен і відеоаналізі.

У програмах для редагування зображень, наприклад Adobe Photoshop, цей метод застосовують для виділення об'єктів. Також його використовують для аналізу супутникових знімків, наприклад, для виявлення змін у природних ландшафтах.

Метод Graph Cut є потужним та ефективним підходом до сегментації, особливо коли необхідно враховувати як локальні характеристики, так і глобальний контекст зображення.

Graph Cut дозволяє не лише розділяти об'єкти на зображенні, а й контролювати рівень деталізації сегментації. Він може використовувати енергетичні функції для балансування між точністю контуру та згладжуванням меж. Це робить його ефективним для задач, де важливо уникати зайвого поділу областей.

Ще одним важливим аспектом використання Graph Cut є його здатність працювати з багатоканальними та кольоровими зображеннями. Це особливо корисно у медичних дослідженнях, де необхідно обробляти томографічні дані з різних спектральних діапазонів. Поєднання цього методу з глибокими згортковими нейронними мережами дозволяє підвищити точність сегментації, автоматично адаптуючи модель до особливостей кожного набору даних.

Подання для алгоритма Graph Cut у вигляді графа на рисунку 1.9:

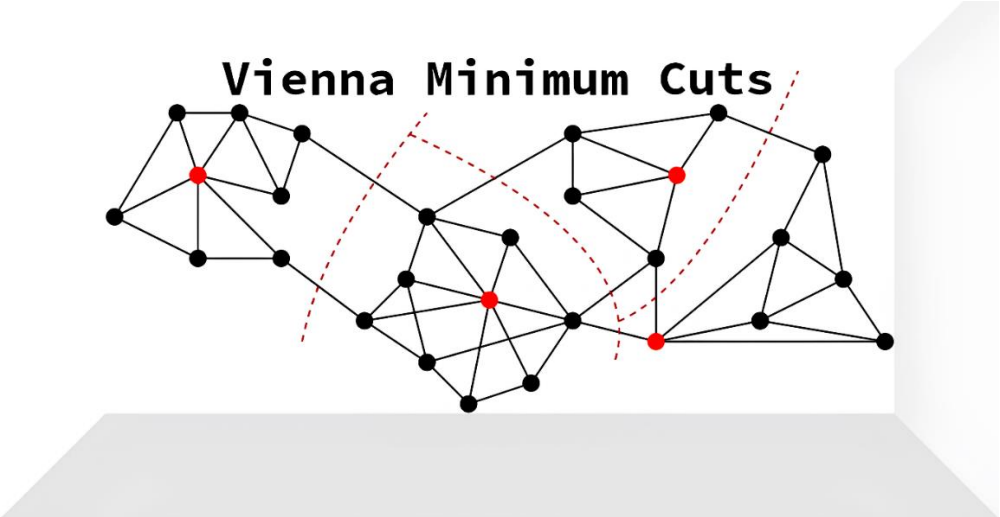

Рисунок 1.9 – Мінімальні розрізи у графах

Метод Watershed використовується для сегментації зображень, сприймаючи його як рельєфну карту. У цій інтерпретації яскраві пікселі відповідають височинам, темні – западинам.

Заглиблення сприймаються як водозбірні басейни, а найвищі точки утворюють гребені, які стають межами між областями.

Основний принцип роботи такий: вода ніби починає заповнювати заглиблення на зображенні. Коли два потоки зустрічаються, між ними формується вододіл – це і є межа між сегментами.

На першому етапі виконується попередня обробка зображення, що включає застосування фільтраційних методів для зменшення рівня шуму та покращення якості контурів об’єктів.

Використання згладжувальних та градієнтних операторів дозволяє підвищити точність подальшої сегментації та виділення ключових областей, особливо на складних зображеннях.

Наступним кроком є визначення маркерів, яке здійснюється шляхом застосування морфологічних операцій. На цьому етапі формується початковий розподіл об’єктів у зображенні, що передбачає вибір опорних точок (насіння), які задають початкові умови для подальшого поширення областей. Цей процес допомагає мінімізувати помилки сегментації.

Далі відбувається виконання основного алгоритму, в межах якого здійснюється розширення областей від маркерів.

Процес сегментації базується на аналізі локальних характеристик пікселів, що забезпечує формування чітких меж між сегментованими областями.

Завершальним етапом є постобробка отриманих результатів, яка включає усунення дрібних артефактів, корекцію меж об’єктів та накладання отриманих контурів сегментації на початкове зображення. Це дозволяє підвищити точність і наочність отриманих результатів.

Метод має кілька переваг. Він добре розділяє об’єкти з чіткими межами, не потребує попереднього навчання та ефективний для сегментації злитих об’єктів, наприклад, клітин у мікроскопічних зображеннях.

Проте є і недоліки. Watershed схильний до надмірного поділу зображення через шум, тому потребує якісної попередньої обробки. Також важливо правильно вибрати початкові маркери, інакше результат буде некоректним.

Щоб покращити точність, можна застосовувати морфологічні операції для очищення зображення, комбінувати Watershed з іншими методами (Graph Cut, кластеризація k–means, нейронні мережі) або використовувати маркерний варіант, де початкові області задаються вручну або алгоритмічно.

Даний метод знаходить широке застосування у різних галузях.

У медицині він використовується для ідентифікації клітинних структур та сегментації органів на знімках, отриманих за допомогою комп’ютерної та магнітно–резонансної томографії.

У сфері комп’ютерного зору цей підхід дозволяє ефективно виділяти об’єкти та розпізнавати дрібні деталі на зображеннях.

У мікроскопічних дослідженнях метод застосовується для аналізу біологічних структур, що сприяє точнішому вивченню клітин і тканин.

В обробці супутникових знімків він допомагає визначати кордони географічних об’єктів, таких як ліси, водойми, міські території та сільськогосподарські угіддя.

1.4 Порівняння реалізацій у бібліотеках OpenCV та scikit–image

OpenCV (Open Source Computer Vision Library) — це популярна бібліотека комп'ютерного зору, яка підтримує ефективні алгоритми обробки зображень і відео.

OpenCV забезпечує високу швидкість і продуктивність під час сегментації зображень, оскільки написаний на C++. Він містить багато вбудованих методів, таких як порогова сегментація, кластеризація та вододільний алгоритм, що спрощує обробку зображень. Бібліотека також підтримує апаратне прискорення, використовуючи OpenCL і обчислення на GPU, що значно підвищує швидкодію.

OpenCV широко використовується в реальних додатках, зокрема в системах відеоспостереження, автономному водінні, розпізнаванні облич та жестів. Завдяки гнучкості та підтримці різних мов програмування, таких як Python, Java та C++, бібліотека легко інтегрується в різні проєкти. Крім того, вона має вбудовані засоби для роботи з нейронними мережами, що дозволяє поєднувати класичні методи обробки зображень із сучасними алгоритмами глибокого навчання, значно покращуючи точність аналізу.

Крім того, OpenCV легко інтегрується з іншими інструментами, такими як NumPy і TensorFlow, і дозволяє працювати як у Python, так і в C++.

Основні функції для сегментації в OpenCV:

- Порогова сегментація (Thresholding):

```
import cv2
import numpy as np
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
ret, thresholded = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('Thresholded Image', thresholded)
cv2.waitKey(0)cv2.destroyAllWindows()
```

- Метод Оцу (Otsu's Thresholding):

```
ret, otsu_thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

- Сегментація методом k–means:

```
Z = image.reshape((–1,1))
Z = np.float32(Z)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 3  # кількість кластерів
ret, label, center = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
center = np.uint8(center)
res = center[label.flatten()]
segmented_image = res.reshape((image.shape))
cv2.imshow('K–means Segmentation', segmented_image)
cv2.waitKey(0)
```

- Метод Watershed:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
dist_transform = cv2.distanceTransform(thresh, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(thresh, sure_fg)
ret, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0
cv2.watershed(image, markers)
image[markers == –1] = [0, 0, 255]
cv2.imshow('Watershed Segmentation', image)
cv2.waitKey(0)
```

OpenCV має менш гнучкий інтерфейс у порівнянні зі scikit–image, оскільки містить менше параметрів для налаштування [19].

Деякі методи сегментації працюють тільки з одноканальними (чорно–білими) зображеннями, що може створювати труднощі при обробці кольорових зображень. Крім того, для отримання якісних результатів часто потрібні додаткові етапи попередньої обробки, такі як згладжування або морфологічні операції.

У підсумку, OpenCV є ефективним інструментом для швидкої сегментації зображень, особливо в режимі реального часу та при обробці великих зображень. Поєднання OpenCV з іншими бібліотеками, такими як NumPy і scikit–image, розширює його можливості та підвищує якість сегментації.

scikit–image — це бібліотека для обробки зображень у Python, яка інтегрується з NumPy та Matplotlib.

Scikit–image має зручний інтерфейс, який спрощує використання складних алгоритмів сегментації. Він підтримує більше методів сегментації, ніж OpenCV, що дає користувачам ширші можливості для аналізу зображень. Завдяки глибокій інтеграції з NumPy, бібліотека працює без необхідності додаткового перетворення даних.

Крім того, scikit–image містить додаткові метрики для оцінки якості сегментації, що дозволяє точніше аналізувати отримані результати.

Основні методи сегментації в scikit–image [12]:

- Порогова сегментація (Thresholding):

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, filters
image = io.imread('image.jpg', as_gray=True)
threshold = filters.threshold_otsu(image)
binary = image > threshold
plt.imshow(binary, cmap='gray')
plt.title('Otsu Thresholding')
plt.show()
```

- Суперпіксельна сегментація (SLIC – Simple Linear Iterative Clustering):

```
from skimage import segmentation, color
segments = segmentation.slic(image, n_segments=100, compactness=10)
segmented_image = color.label2rgb(segments, image, kind='avg')
plt.imshow(segmented_image)
plt.title('SLIC Superpixels')
plt.show()
```

- Метод Random Walker:

```
from skimage.segmentation import random_walker
markers = np.zeros(image.shape, dtype=np.uint)
markers[image < 50] = 1
markers[image > 150] = 2
segmented_rw = random_walker(image, markers)
plt.imshow(segmented_rw, cmap='gray')
plt.title('Random Walker Segmentation')
plt.show()
```

- Вододільна сегментація (Watershed):

```
from skimage.morphology import watershed
from skimage.feature import peak_local_max
from scipy import ndimage
distance = ndimage.distance_transform_edt(image)
local_maxi = peak_local_max(distance, indices=False, footprint=np.ones((3, 3)), labels=image)
markers, = ndimage.label(local_maxi)
segmented_watershed = watershed(–distance, markers, mask=image)
plt.imshow(segmented_watershed, cmap='gray')
plt.title('Watershed Segmentation')
plt.show()
```

Scikit–image, бібліотека Python для обробки зображень, демонструє нижчу швидкодію порівняно з OpenCV.

Це є наслідком специфіки реалізації: scikit–image написана виключно мовою Python, яка сама по собі є інтерпретованою мовою програмування з нижчою продуктивністю порівняно з компільованими мовами, такими як C++. Відсутність низькорівневих C++–реалізацій у scikit–image обмежує можливості для додаткового прискорення обчислень, що призводить до зниження загальної продуктивності бібліотеки.

Крім того, scikit–image має обмежені можливості використання апаратного прискорення. На відміну від OpenCV, яка активно використовує технології OpenCL і GPU для розподілу обчислювальних навантажень, scikit–image не реалізує повноцінної підтримки цих технологій. Це зменшує ефективність бібліотеки при виконанні ресурсомістких операцій, що може бути критичним для застосувань у режимі реального часу.

OpenCV, своєю чергою, здатний ефективніше розподіляти навантаження на апаратні ресурси, що суттєво підвищує продуктивність під час обробки зображень у режимі реального часу. Ця бібліотека пропонує інтеграцію з різними технологіями апаратного прискорення, включаючи CUDA та OpenCL, що дозволяє використовувати потужність сучасних графічних процесорів (GPU) для прискорення обчислень. Завдяки цьому OpenCV є більш ефективним вибором для завдань, які вимагають високої продуктивності.

Незважаючи на ці обмеження, scikit–image залишається потужним інструментом для наукових досліджень, аналізу зображень та експериментальної обробки. Бібліотека забезпечує зручний та гнучкий інтерфейс для роботи з візуальними даними, що робить її популярною серед дослідників та розробників алгоритмів комп'ютерного зору. Велика кількість доступних функцій та методів дозволяє ефективно вирішувати широкий спектр задач у галузі обробки зображень.

Scikit–image є особливо корисною для прототипування та експериментів, оскільки її високий рівень абстракції спрощує розробку та тестування нових алгоритмів. Це дозволяє дослідникам швидко створювати та перевіряти гіпотези, що є важливим етапом у процесі наукових досліджень.

1.5 Метрики оцінки якості сегментації

Метрики якості сегментації дозволяють об’єктивно оцінити ефективність алгоритмів [20]. Порівняння основних метрик продемонстровано у таблиці 1.1:

Таблиця 1.1. – Метрики оцінки якості сегментації зображень

Метрика	Формула	Призначення	Особливості	Діапазон значень	Інтерпретація результатів
Intersection over Union (IoU)	(A∩B)/(A∪B)	Оцінює схожість між передбаченим і реальним сегментами	Чим вище значення, тим краща сегментація	[0,1]	1 – ідеальне співпадіння, 0 – повна невідповідність
F1–score	2 × (Precision × Recall)/(Precision + Recall)	Балансує Precision і Recall для оцінки точності сегментації	Використовується при наявності розмічених областей	[0,1]	1 – досконала збалансованість Precision і Recall
Rand Index (RI)	(TP + TN)/(TP + FP + FN + TN)	Оцінює правильність класифікації пар пікселів	Враховує як правильно об'єднані, так і розділені пари	[0,1]	1 – ідеальна відповідність кластерів
Variation of Information (VI)	— (ентропійна різниця)	Вимірює невизначеність у розподілі кластерів	Нижчі значення означають кращу сегментацію	[0,∞)	0 – ідеальне співпадіння, вищі значення – більші розбіжності
Mean Squared Error (MSE)	1/N ∑(I(true) – I(pred))^2	Визначає середньоквадратичну помилку сегментації	Використовується при порівнянні градацій яскравості	[0,∞)	0 – ідеальна відповідність, більше значення – більша похибка

Ці метрики широко застосовуються у комп’ютерному зорі для об’єктивного вимірювання якості сегментації. Це прописано у таблиці 1.2, де є порівняння метрик якості сегментації [4]:

Таблиця 1.2. – Переваги та недоліки метрик оцінки сегментації

Метрика	Сильні сторони	Недоліки
IoU (Intersection over Union)	Простота, популярність у комп’ютерному зорі	Не працює для багатокласової сегментації
F1–score	Враховує precision та recall	Вимагає точного ground–truth
Rand Index (RI)	Підходить для кластеризації	Може не враховувати дрібні відмінності
Variation of Information (VI)	Чутливий до деталей сегментації	Вимагає обчислень з ентропією
Mean Squared Error (MSE)	Добре оцінює загальні помилки	Чутливий до освітлення та шуму

1.6 Огляд існуючих програмних рішень

Існує багато програмних рішень для сегментації зображень, які застосовуються в наукових дослідженнях, медицині та промисловості.

Наприклад, ImageJ (Fiji) використовується для біологічного аналізу та підтримує методи Watershed і K–means. MATLAB Image Processing Toolbox містить готові інструменти для сегментації, тому його часто використовують у наукових дослідженнях.

Для медичної сегментації розроблена бібліотека ITK (Insight Segmentation and Registration Toolkit). Якщо потрібна семантична сегментація, можна використовувати DeepLab (TensorFlow).

А для комп’ютерного зору застосовують бібліотеки OpenCV і scikit–image, які містять широкий набір алгоритмів для обробки зображень та аналізу відео в реальному часі. Зображено це у таблиці 1.3.

Таблиця 1.3. – Порівняльний аналіз програмного забезпечення для сегментації зображень:

Програмне забезпечення	Основне призначення	Підтримувані методи сегментації	Переваги	Недоліки
ImageJ (Fiji)	Біологічний аналіз, наукові дослідження	Watershed, K–means, ручна сегментація	Безкоштовний, велика кількість плагінів	Обмежена гнучкість для складних алгоритмів
MATLAB Image Processing Toolbox	Наукові дослідження, обробка медичних зображень	Watershed, K–means, Region Growing, Graph Cut	Потужний інструментарій, інтеграція з MATLAB	Платний, високий поріг входу
ITK (Insight Toolkit)	Медична сегментація, 3D–зображення	Region Growing, Graph Cut, Watershed	Висока точність, спеціалізація на медичних зображеннях	Складність у використанні, потребує C++
DeepLab (TensorFlow)	Глибоке навчання, семантична сегментація	Нейромережеві методи	Висока точність, сучасні підходи	Вимагає навчання моделей, ресурсоємний
OpenCV	Комп’ютерний зір, реальний час	Thresholding, K–means, Watershed, Region Growing (floodFill)	Висока швидкість, підтримка C++	Менше гнучкості для кастомних методів
scikit–image	Комп’ютерний зір, наукові дослідження	Thresholding, K–means, Watershed, Graph Cut, Random Walker	Простий API, інтеграція з Python	Повільніший за OpenCV, менше оптимізацій

Розглянуті програмні засоби мають різну спеціалізацію. ImageJ підходить для біологічного аналізу, а MATLAB – для наукових досліджень, проте останній є платним. ITK забезпечує точну медичну сегментацію [4].

Для високої продуктивності в реальному часі найкращим вибором є OpenCV, тоді як scikit–image пропонує гнучкість у наукових дослідженнях, хоч і поступається у швидкості. DeepLab (TensorFlow) забезпечує найкращі результати у глибокому навчанні, але потребує значних ресурсів.

1.7 Висновки

Методи сегментації зображень поділяються на кілька основних категорій: порогові, кластеризаційні, графові та алгоритми нарощування областей. Кожен із цих підходів базується на власних принципах поділу зображення на сегменти, що дозволяє адаптувати їх до різних задач комп'ютерного зору. Порогові методи визначають області на основі рівня яскравості, кластеризаційні алгоритми групують подібні пікселі, графові методи використовують мережеві структури для розділення сегментів, а нарощування областей аналізує локальні подібності для формування однорідних регіонів.

Бібліотека OpenCV пропонує високопродуктивні реалізації класичних алгоритмів сегментації, що оптимізовані для швидкої обробки зображень. Завдяки ефективному використанню апаратного прискорення та вбудованим C++–реалізаціям, OpenCV є оптимальним вибором для застосувань, що вимагають обробки у реальному часі. Водночас scikit–image надає широкий науковий інструментарій, орієнтований на дослідницькі завдання. Його алгоритми підтримують складні методи аналізу зображень, що дозволяє отримувати детальні результати для наукових і медичних досліджень.

Якість сегментації оцінюється за допомогою спеціалізованих метрик, які дозволяють кількісно вимірювати точність виділення об'єктів та відповідність отриманих результатів реальній розмітці. Зокрема, Intersection over Union (IoU) оцінює ступінь перекриття між передбаченим і реальним сегментами, F1–score балансує Precision та Recall для визначення точності класифікації пікселів, а Rand Index (RI) аналізує правильність групування пікселів у сегменти.

2 МЕТОДИКА РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування та інструментарію

Розробка програмного забезпечення для сегментації зображень потребує вибору відповідної мови програмування та бібліотек. У цьому проекті використано мову Python завдяки її широким можливостям у сфері обробки зображень, наявності потужних бібліотек (OpenCV, scikit–image) та інтеграції з інструментами машинного навчання (TensorFlow, PyTorch).

На рисунку 2.1 видно те, що зіставлення з гістограмою дозволяє нам оновити інтенсивність пікселів на вхідному зображенні відповідно до розподілу інтенсивностей пікселів на окремому еталонному зображенні:

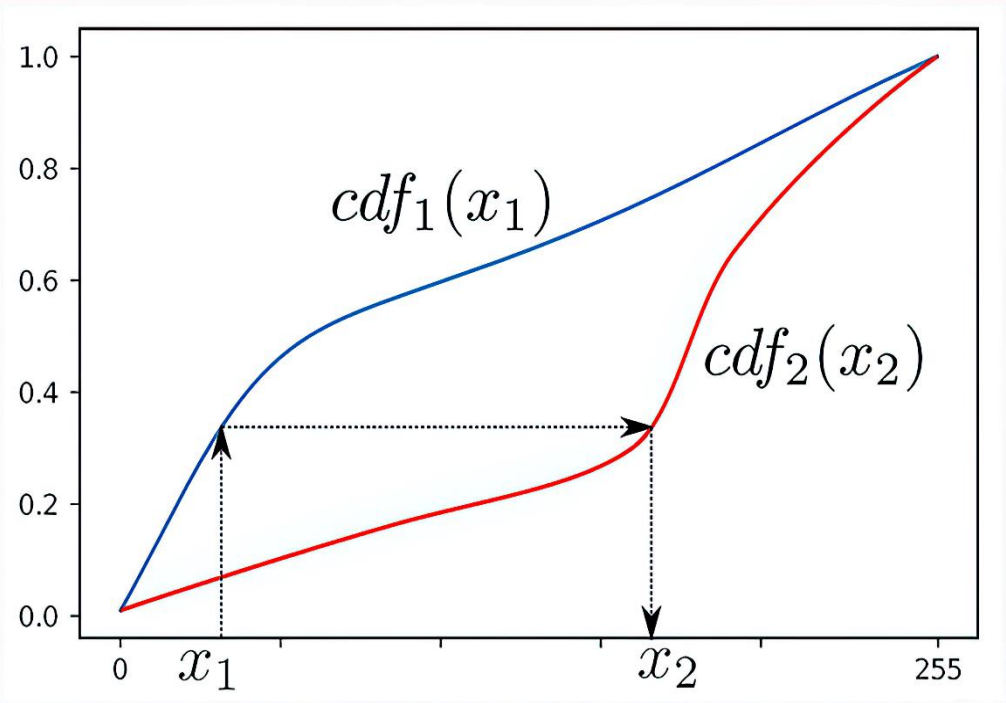

Рисунок 2.1 – Зіставлення гістограм для корекції інтенсивності

Крім того, Python підтримує паралельні обчислення та взаємодію з апаратними прискорювачами (CUDA, OpenCL), що дозволяє підвищити продуктивність обчислень.

Для тестових даних я використав кілька фотографій високої роздільної здатності, зроблених телескопом Хаббл, які я показав на рисунку 2.2:

Рисунок 2.2 – Порівняння методів обробки зображень

Запустивши програму на своїй робочій станції, я отримав наступні середні результати:

- TPL: 0.737472333 секунд
- Alea GPU: 0.4567708 секунд
- ILGPU: 0.410849867 секунд

Отже, у цьому конкретному випадку найменш абстрактний підхід ILGP виявився найшвидшим і дав майже 80% виграшу у продуктивності.

Приклад на рисунку 2.3: зліва – низько–контрастне зображення, на якому важко виявити контури картки. Праворуч – більш контрастне зображення, на якому комп'ютерному зору/конвеєру обробки зображень буде набагато легше виявити картку:

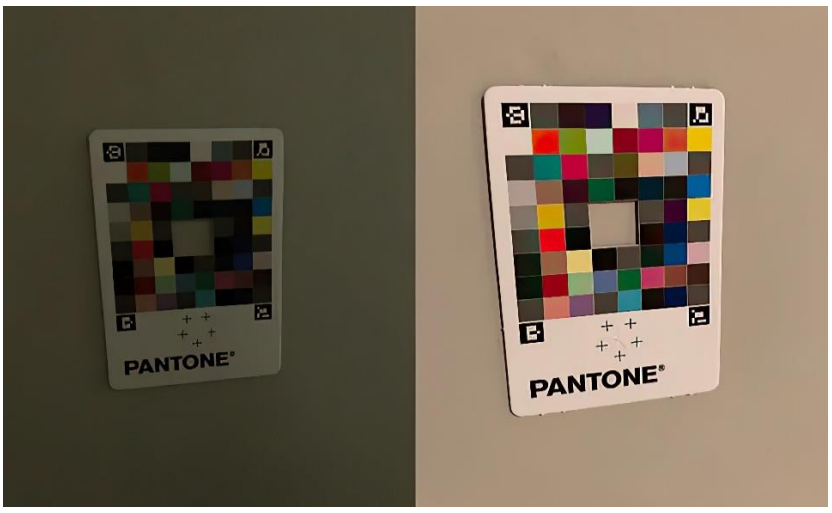

Рисунок 2.3 – Покращення контрасту для виявлення об'єктів

Для реалізації графічного інтерфейсу застосовано бібліотеку PyQt5, що забезпечує зручний користувацький досвід.

Для роботи з метриками якості сегментації використано бібліотеку scikit–learn, яка містить реалізації коефіцієнтів Jaccard та Dice, порівняння який я зобразив у вигляді кривих на рисунку 2.4:

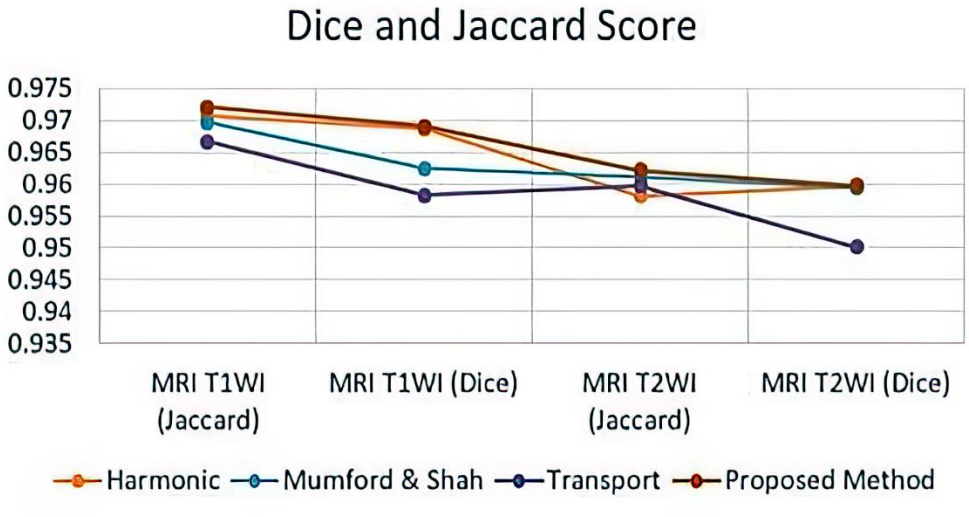

Рисунок 2.4 – Dice vs Jaccard: Порівняння методів обробки

Коефіцієнт Jaccard (міра подібності між двома множинами пікселів) визначається за такою формулою:

$$J(A, B) = \frac{|A \cap B|}{|A \cap B|}, \tag{2.1}$$

де A — пікселі істинної маски, B — пікселі сегментованої області.

Тоді як коефіцієнт Dice (міра збігу двох областей) визначається за такою формулою:

$$D(A, B) = \frac{2|A \cap B|}{|A| + |B|}, \tag{2.2}$$

Python є оптимальним вибором для задач комп'ютерного зору завдяки великій кількості доступних бібліотек, що спрощують інтеграцію алгоритмів машинного навчання.

Використання високорівневих API дозволяє значно прискорити розробку програмного забезпечення та спрощує експерименти з різними методами сегментації.

Однією з переваг Python є можливість його використання для швидкого прототипування та тестування нових алгоритмів. Завдяки динамічному типізуванню та розвинутій екосистемі інструментів, розробники можуть швидко адаптувати існуючі рішення та перевіряти їх ефективність без необхідності значних змін у коді.

Для підвищення продуктивності обчислень у проектах з обробки зображень часто застосовується багатопотоковість та обчислення на графічних процесорах. Використання бібліотеки NumPy у поєднанні з OpenCV дозволяє оптимізувати операції над зображеннями, зменшуючи час виконання критичних задач сегментації.

Функція imread() та функція imwrite() є двома найбільш широко використовуваними функціями бібліотеки OpenCV, які використовуються для читання зображення та збереження його у вказаний файл відповідно.

На виході буде створено інше зображення з назвою «convert.png», яке міститиме перетворене RGB–зображення, а у терміналі буде показано те, що зображено на рисунку 2.5:

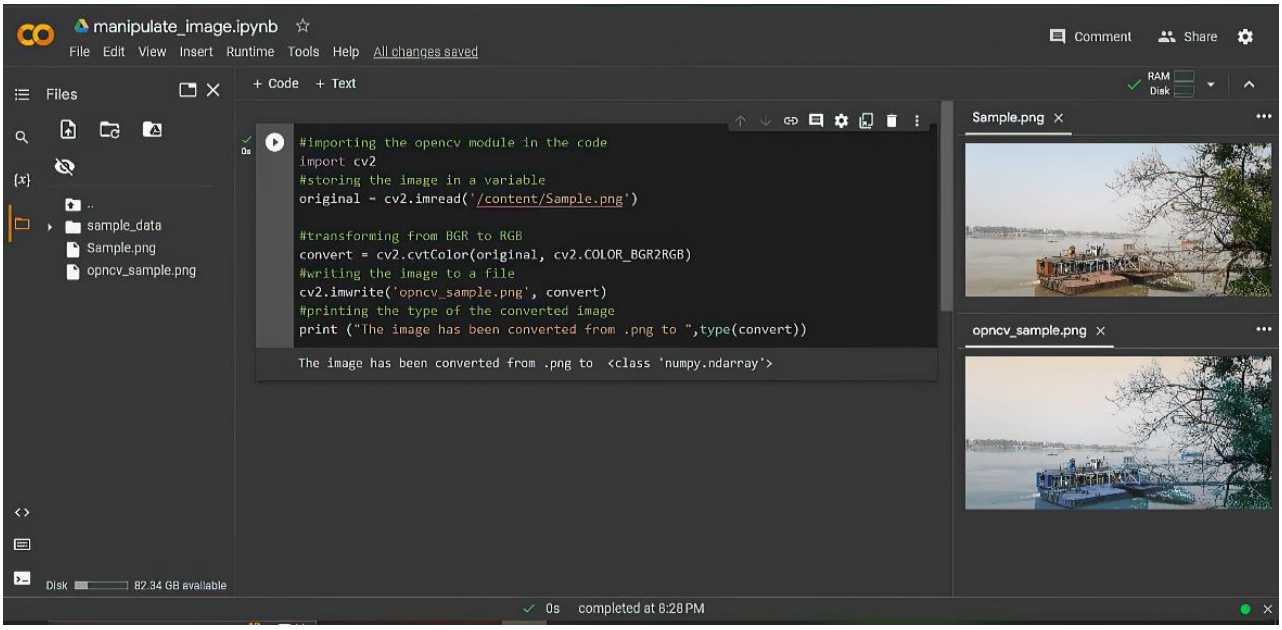

Рисунок 2.5 – Читання та запис зображень за допомогою OpenCV

Прикладом основи багатьох алгоритмів обробки зображень є згортка зображення з ядром K, яка визначається за даною формулою:

$$I'(x,y) = \sum_{i=-m}^{m} \sum_{j=-n}^{n} I(x-i, y-j)K(i,j), \quad (2.3)$$

де I(x,y) — вхідне зображення, K(i,j) — згорткове ядро, I′ (x,y) — результат згортки.

Застосування бібліотеки PyQt5 у розробці графічного інтерфейсу забезпечує не лише зручність взаємодії з програмою, але й можливість налаштування параметрів алгоритмів у реальному часі. Це дає користувачам змогу експериментувати з різними методами сегментації, змінюючи ключові параметри через інтуїтивний інтерфейс.

Реалізація найпростішої медіанної фільтрації – вибір розміру ядра фільтра, який продемонстровано на рисунку 2.6:

Рисунок 2.6 – Медіанна фільтрація імпульсного шуму

Також, для згладжування шуму, використовують перетворення Гаусса, яке визначається за цією формулою:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}}, \quad (2.4)$$

де σ — стандартне відхилення, що визначає ступінь згладжування.

Важливою частиною розробки є вибір ефективних алгоритмів попередньої обробки зображень, таких як згладжування шуму, нормалізація яскравості та підсилення контрасту.

Використання таких підходів дозволяє покращити якість сегментації, зменшуючи похибки та підвищуючи точність аналізу об'єктів на зображенні.

Слід згадати ще гістограмну нормалізацію яскравості, яка виконує нормування пікселів, а також визначається за такою формулою:

$$I'(x,y) = \frac{I(x,y) - I_{min}}{I_{max} - I_{min}}, \quad (2.5)$$

де I_{min} і I_{max} — мінімальне та максимальне значення інтенсивності у зображенні.

2.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення для сегментації зображень передбачає модульний підхід, що дозволяє спростити розробку, тестування та подальше розширення системи. Використання незалежних модулів забезпечує гнучкість у налаштуванні параметрів та адаптацію до різних типів зображень.

Модуль обробки зображень відповідає за початкову підготовку вхідних даних, включаючи завантаження файлів, перетворення колірних просторів та нормалізацію контрасту. Це дозволяє підвищити якість сегментації, забезпечуючи стабільність роботи алгоритмів.

Модуль реалізації алгоритмів містить широкий спектр методів сегментації, що використовуються для виокремлення об’єктів на зображеннях. Використання алгоритмів Graph Cut, K–means, Watershed та регіонального росту дає змогу досягати високої точності в різних умовах освітлення та контрастності.

Наприклад, K–means мінімізує суму квадратів відстаней точок до найближчих центрів кластерів згідно з функцією:

$$J = \sum_{i=1}^{k} \sum_{x \in C_i}^{n} \|x - \mu_i\|^2, \quad (2.6)$$

що забезпечує ефективну кластеризацію пікселів на основі їх схожості.

Модуль візуалізації та порівняння результатів реалізує інструменти для аналізу отриманих сегментованих зображень.

Інтерактивний графічний інтерфейс дозволяє користувачеві змінювати параметри алгоритмів та спостерігати їхній вплив у реальному часі. Оцінка подібності між сегментованим зображенням S та еталонним зображенням R може здійснюватися за допомогою коефіцієнта Jaccard:

$$J(S, R) = \frac{|S \cap R|}{|S \cup R|}, \quad (2.7)$$

Модуль оцінки якості сегментації містить набір метрик, які використовуються для аналізу ефективності алгоритмів. Використання IoU, F1–score та MSE дозволяє отримати оцінку точності та стабільності результатів.

IoU обчислюється як:

$$IoU = \frac{|A \cap B|}{|A \cup B|}, \quad (2.8)$$

де A — передбачена область, B — істинна область.

F1–score визначається за формулою:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (2.9)$$

де $\text{Precision} = TP/(TP + FP)$, $\text{Recall} = TP/(TP + FN)$.

Середньоквадратична похибка (MSE) обчислюється як:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad (2.10)$$

де $\mathbf{y_i}$ – реальне значення, $\hat{\mathbf{y}}_\mathbf{i}$ – передбачене значення.

Інтеграція цих модулів у єдину систему дозволяє автоматизувати процес аналізу зображень, зменшуючи необхідність ручного налаштування параметрів. Завдяки використанню сучасних бібліотек та алгоритмів досягається висока продуктивність та ефективність сегментації.

Гнучкість архітектури дозволяє легко інтегрувати нові алгоритми та адаптувати систему до специфічних задач. Це робить програмне забезпечення універсальним інструментом для обробки медичних, супутникових та промислових зображень.

Використання паралельних обчислень та GPU–оптимізованих бібліотек дозволяє значно скоротити час обробки зображень. Це особливо важливо для обробки великих масивів даних у реальному часі.

Завдяки CUDA програмісти можуть розробляти та реалізовувати паралельні алгоритми, які використовують тисячі ядер, присутніх у сучасних графічних процесорах.

CUDA надає модель програмування і набір API, які дозволяють розробникам писати код, що виконується безпосередньо на GPU, розкриваючи потенціал для значного приросту продуктивності в порівнянні з традиційними обчисленнями на базі CPU.

Перекладаючи робочі навантаження, що розпаралелюються, на GPU, CUDA відіграє центральну роль у розширенні обчислювальних можливостей GPU і стимулюванні розвитку високопродуктивних обчислювальних додатків. Для кращого розуміння я зобразив це на рисунку 2.7:

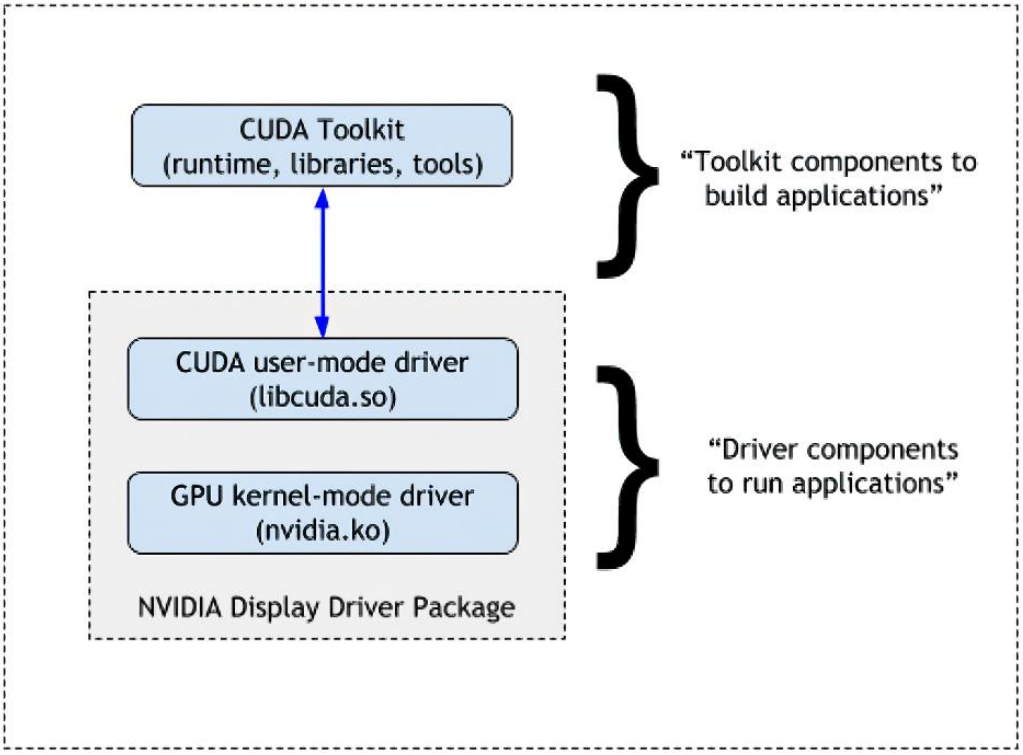

Рисунок 2.7 – Компоненти, з яких складається CUDA

Масштабованість архітектури забезпечує можливість інтеграції з хмарними сервісами та розподіленими обчислювальними системами. Це відкриває перспективи використання програмного забезпечення у великих наукових та промислових проектах.

Хмарні обчислення мають три рівні підключення: хмара, мережеві пристрої, такі як маршрутизатори та комутатори, і кінцевий користувач. Хмара складається з таких ресурсів, як віртуальні робочі столи, програмні платформи, сервери, програми та сховища даних. Вони обробляють дані через маршрутизатори та комутатори. Кінцевий користувач може отримати доступ до інформації з будь–якого пристрою. Наочно це можна побачити на рисунку 2.8:

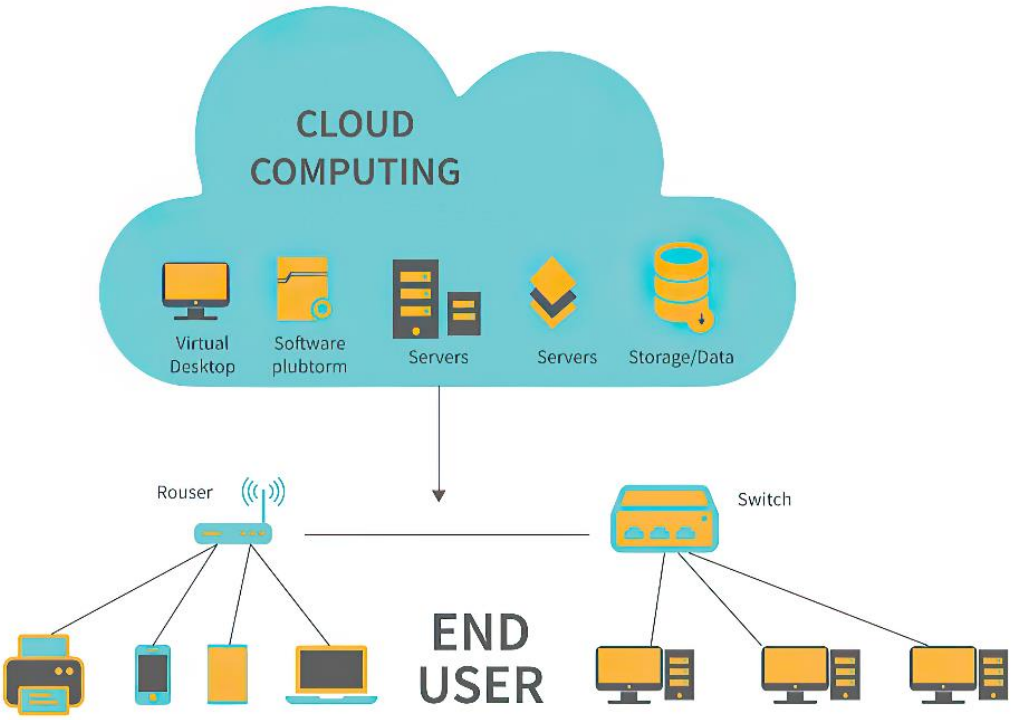

Рисунок 2.8 – Схема хмарних обчислень

2.3 Розробка модульної структури системи

2.3.1. Модуль обробки зображень

Обробка зображень є багатоступеневим процесом, що передбачає виконання ряду операцій для підготовки та покращення вхідних даних. На початковому етапі здійснюється завантаження зображень у поширених форматах, таких як PNG, JPEG та BMP, що забезпечує сумісність із більшістю сучасних обчислювальних платформ і програмних засобів.

Одним із ключових етапів попередньої обробки є нормалізація яскравості пікселів, що може бути виконана за формулою:

$$I_n(x, y) = \frac{I(x,y) - I_{min}}{I_{max} - I_{min}}, \quad (2.11)$$

Після завантаження проводиться перетворення кольорових зображень у градації сірого, що дозволяє спростити подальшу обробку шляхом зменшення кількості вимірів, у яких представлені піксельні значення, як і відображено на рисунку 2.9. Це є важливим кроком для багатьох алгоритмів аналізу зображень, оскільки він знижує обчислювальну складність та зменшує обсяг даних, необхідних для обробки:

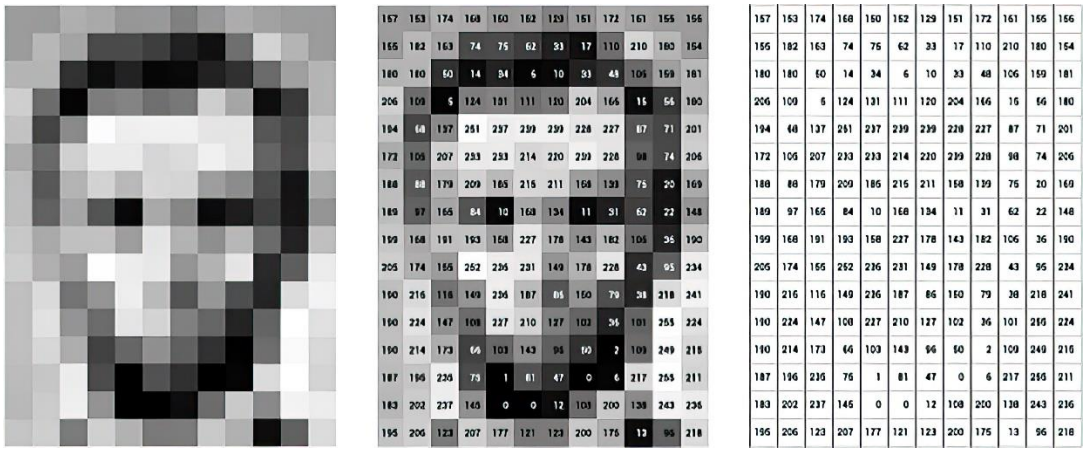

Рисунок 2.9 – Матричне представлення в градаціях сірого

Наступним етапом є видалення шумів, що може здійснюватися за допомогою різних методів, зокрема застосування гаусового розмиття або морфологічних операцій.

Ще одним підходом до покращення якості зображень є використання адаптивних фільтрів, які враховують локальні особливості зображення.

Наприклад, медіанний фільтр ефективно видаляє імпульсний шум, зберігаючи при цьому різкі краї об'єктів, що особливо корисно для подальшого розпізнавання контурів і сегментації.

Ці методи спрямовані на зменшення небажаних артефактів, які можуть впливати на точність аналізу зображень та їх подальшу інтерпретацію. Для більшого розуміння я навів порівняння на рисунку 2.10:

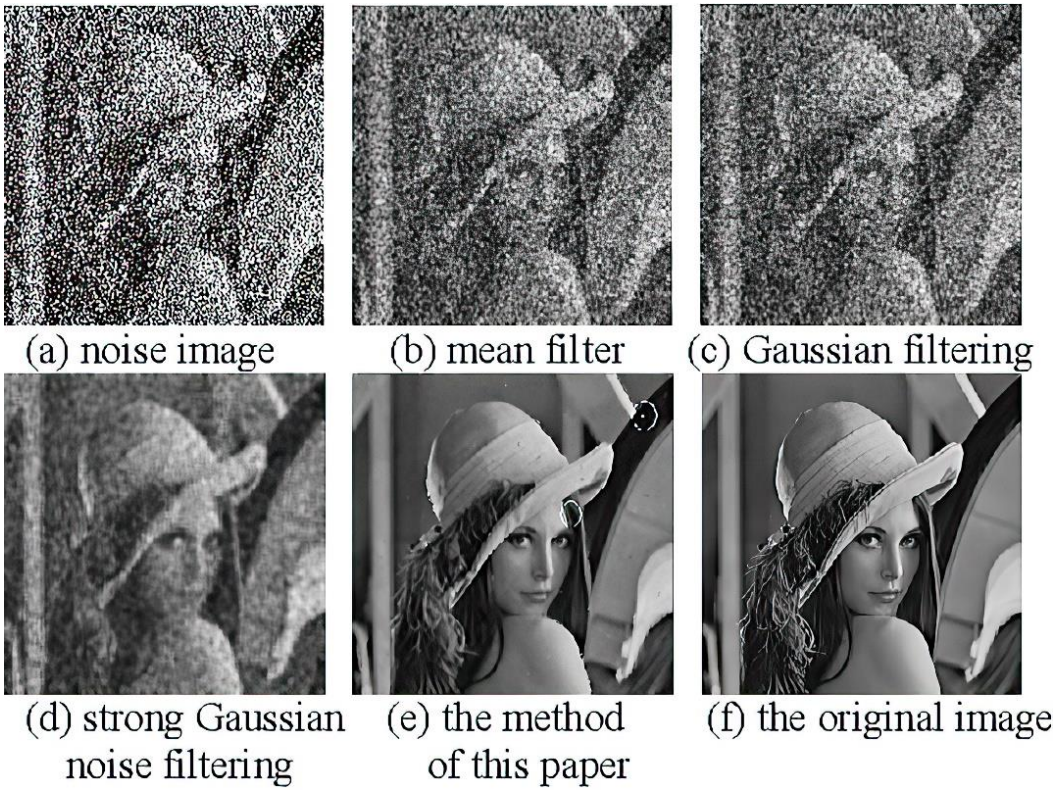

Рисунок 2.10 – Порівняння методів фільтрації шуму

Останнім етапом обробки є нормалізація яскравості, яка виконується для усунення нерівномірного розподілу інтенсивності пікселів, як і видно на рисунку 2.11. Це дозволяє покращити якість зображення та забезпечити коректність подальших процедур аналізу, таких як сегментація або розпізнавання об’єктів:

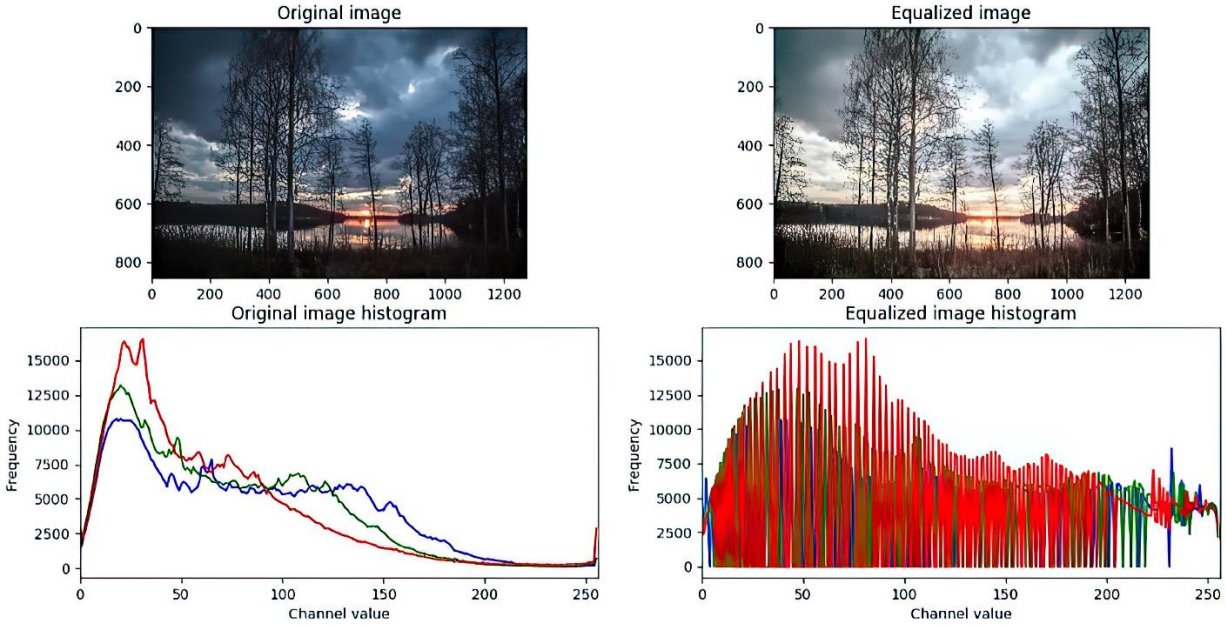

Рисунок 2.11 – Вирівнювання яскравості при гістограмній корекці

Таким чином, обробка зображень включає кілька ключових стадій, спрямованих на підготовку вхідних даних для подальшого аналізу та інтерпретації, забезпечуючи їхню якість та зручність використання в різних комп’ютерних застосуваннях.

Для реалізації обробки зображення використовуються методи бібліотеки OpenCV. Спочатку зображення завантажується у градаціях сірого за допомогою функції cv2.imread() з параметром cv2.IMREAD_GRAYSCALE, що дозволяє виключити вплив кольорової інформації та зменшити обчислювальні витрати, результат чого можна побачити на рисунку 2.12:

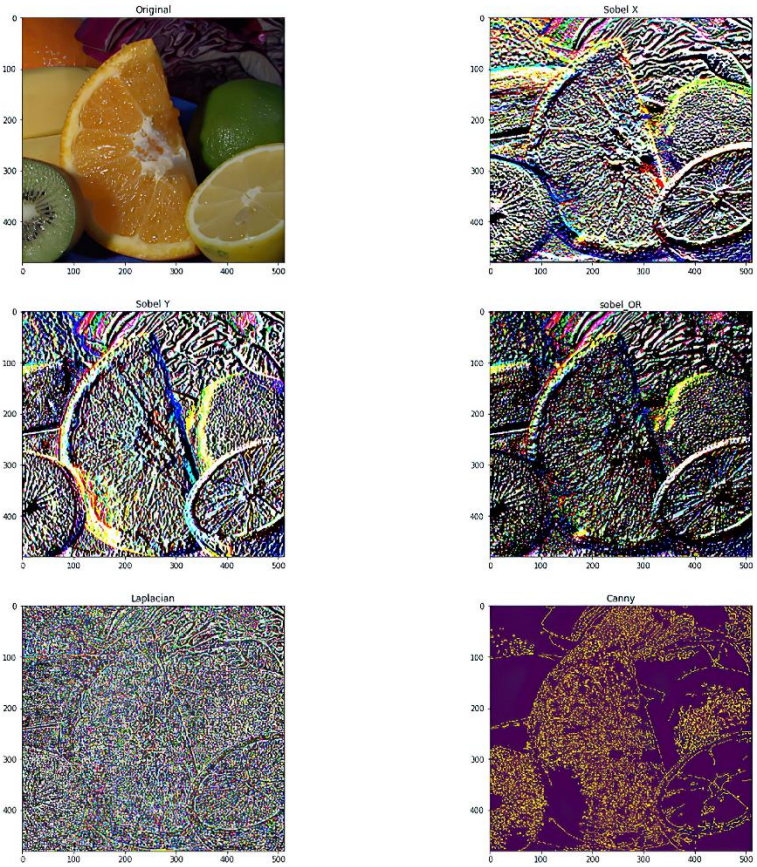

Рисунок 2.12 – Методи детекції границь у цифрових зображеннях

Далі застосовується гаусове розмиття (cv2.GaussianBlur()), яке допомагає зменшити рівень шуму та усунути високочастотні компоненти.

Далі, після згладжування, зображення може бути оброблене за допомогою градієнтних фільтрів, таких як оператори Собеля або Кенні, для виявлення контурів.

Використання ядра розміром 5×5 забезпечує оптимальний баланс між згладжуванням і збереженням деталей, що продемонстровано на рисунку 2.13:

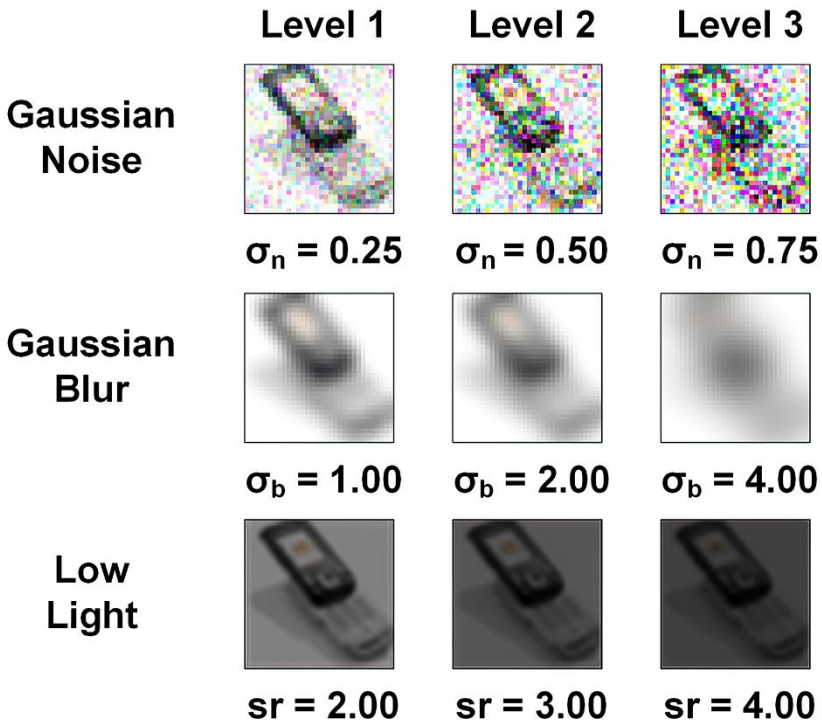

Рисунок 2.13 – Деградація через шум, розмиття та затемнення

Після цього виконується нормалізація яскравості методом cv2.normalize(), що приводить значення пікселів до діапазону [0, 255] із використанням cv2.NORM_MINMAX. Це покращує контрастність та рівномірність яскравості, що є важливим для подальшого аналізу або обробки зображення.

Приклад коду, де реалізовано кожний із цих методів:

- import cv2

  # Завантаження зображення у градаціях сірого

  image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

  # Гаусове розмиття для зменшення шуму

  blurred = cv2.GaussianBlur(image, (5,5), 0)

  # Нормалізація яскравості у діапазоні [0, 255]

  norm_image = cv2.normalize(blurred, None, 0, 255, cv2.NORM_MINMAX)

Ця послідовність операцій забезпечує ефективне попереднє опрацювання зображення, покращуючи його якість і підготовлюючи до подальшого використання в комп’ютерному зорі або аналізі зображень.

2.3.2. Модуль реалізації алгоритмів

У рамках даного дослідження реалізовано алгоритми сегментації зображень, що використовуються для виділення об'єктів та їхніх меж у цифрових зображеннях. Основною метою є розділення зображення на значущі області, що дозволяє спростити його подальший аналіз. Для цього у системі передбачено використання таких методів, як Graph Cut та K–means.

Метод Graph Cut базується на побудові графа, вершинами якого є пікселі зображення, а зважені ребра відображають зв’язки між ними. Використовуючи алгоритм мінімального розрізу, метод дозволяє відокремити об'єкти від фону на основі ймовірнісної моделі кольору точно так само, як на рисунку 2.14:

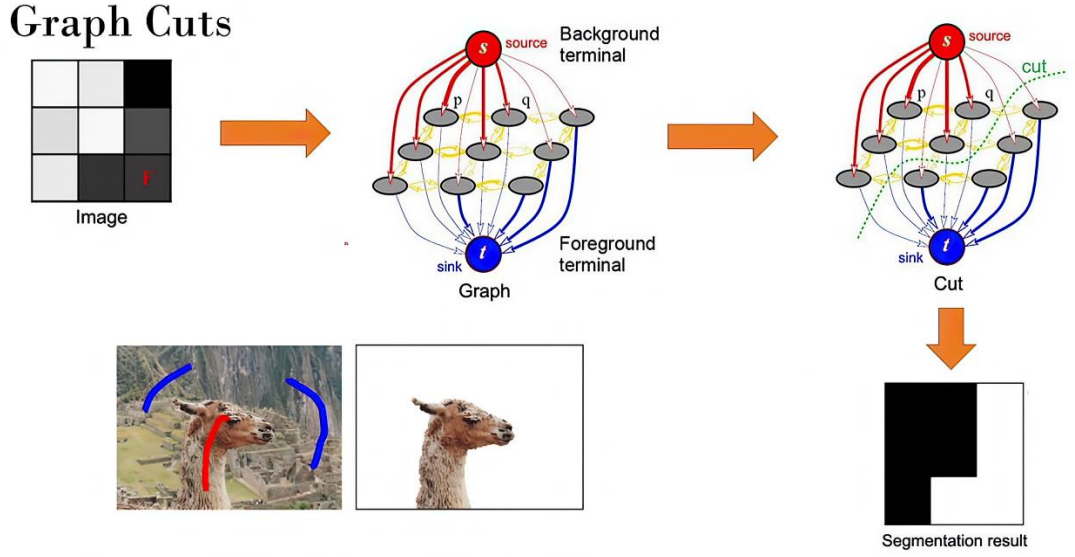

Рисунок 2.14 – Сегментація зображень методом Graph Cut

Його реалізація в середовищі OpenCV передбачає використання функції cv2.grabCut, де визначається початкова прямокутна область для об’єкта, а потім алгоритм ітеративно покращує межі сегментації.

Метод K–means належить до класу кластеризаційних алгоритмів і дозволяє групувати пікселі зображення на основі їхніх кольорових характеристик.

Він використовує ітеративну процедуру для мінімізації внутрішньогрупової дисперсії та знаходження оптимальних центрів кластерів. Реалізація цього методу у OpenCV передбачає перетворення зображення у векторний формат, після чого застосовується функція cv2.kmeans, яка виконує кластеризацію кольорових значень.

Нижче наведено реалізацію даних алгоритмів у середовищі OpenCV:

- ```
  import cv2
  import numpy as np
  img = cv2.imread('image.jpg')
  mask = np.zeros(img.shape[:2], np.uint8)
  bgd_model = np.zeros((1, 65), np.float64)
  fgd_model = np.zeros((1, 65), np.float64)
  rect = (10, 10, img.shape[1]–20, img.shape[0]–20)
  cv2.grabCut(img, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)
  Z = img.reshape((–1, 3))
  Z = np.float32(Z)
  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
  K = 3
  _, labels, centers = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
  segmented = centers[labels.flatten()].reshape(img.shape)
  ```

Застосування зазначених алгоритмів дозволяє виконувати ефективну сегментацію зображень у різних сферах, включаючи комп’ютерний зір, медичну діагностику та автоматичну обробку відео.
  ```

2.3.3. Модуль візуалізації та порівняння результатів

У рамках розробки системи передбачено модуль візуалізації та порівняння результатів сегментації, який забезпечує наочне представлення отриманих зображень та їх аналіз. Візуалізація результатів є важливим етапом у процесі обробки зображень, оскільки вона дозволяє оцінити якість застосованих алгоритмів та порівняти їхню ефективність.

Для реалізації цього модуля використовується бібліотека PyQt5, яка забезпечує інтеграцію графічного інтерфейсу з можливістю обробки та відображення зображень.

У процесі візуалізації цифрові зображення, отримані після сегментації, конвертуються у формат, придатний для відображення у графічному інтерфейсі.

Додатково, модуль візуалізації підтримує накладання контурів сегментованих об'єктів на вихідне зображення, що сприяє більш точному аналізу результатів. Це реалізується шляхом використання бібліотеки OpenCV, яка дозволяє застосовувати методи виділення контурів, наприклад, функцію cv2.findContours().

Це передбачає перетворення масиву пікселів у формат QImage, як на рисунку 2.15, що дозволяє подальше використання об'єкта QPixmap для рендерингу у віджетах інтерфейсу. Використання QLabel як контейнера для відображення зображень забезпечує інтеграцію в графічний інтерфейс програми:

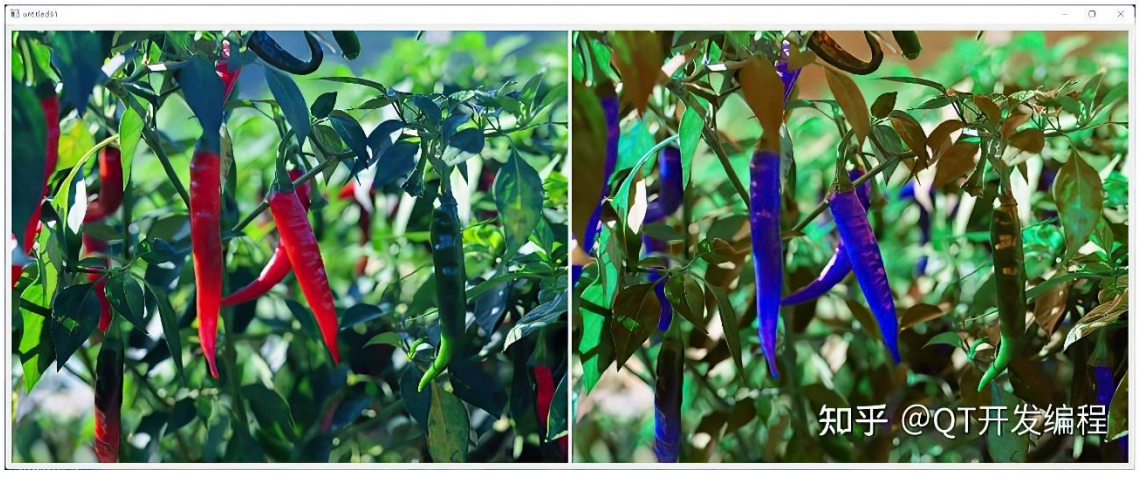

Рисунок 2.15 – Спектральний аналіз та кольорова трансформація

Нижче наведено реалізацію модуля візуалізації у середовищі PyQt5 [12]:

```
from PyQt5.QtWidgets import QLabel, QVBoxLayout, QWidget
from PyQt5.QtGui import QPixmap, QImage
# Створення віджета для відображення зображення
label = QLabel()
# Перетворення обробленого зображення у формат QImage
image_qt = QImage(segmented, width, height, QImage.Format_RGB888)
# Створення об'єкта QPixmap для відображення у QLabel
pixmap = QPixmap.fromImage(image_qt)
label.setPixmap(pixmap)
```

Застосування PyQt5 у даному модулі дозволяє створювати гнучкі та адаптивні графічні інтерфейси, що полегшує аналіз та порівняння отриманих зображень. Це забезпечує ефективний візуальний контроль над процесом сегментації та підвищує зручність використання системи обробки зображень.

3 РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ СЕГМЕНТАЦІЇ

3.1 Автоматизація сегментації зображень

Автоматизація сегментації зображень базується на використанні спеціалізованих бібліотек для обробки та аналізу зображень. До найбільш поширених інструментів належать OpenCV, scikit–image та TensorFlow, кожен із яких пропонує ефективні методи для виконання цієї задачі [12].

OpenCV є однією з найпотужніших бібліотек для обробки зображень, що містить низку алгоритмів для автоматичного визначення меж об'єктів. Зокрема, функція cv2.watershed() використовує маркерний підхід для сегментації, розділяючи зображення на області на основі попередньо заданих маркерів.

А функція cv2.findContours() ефективно знаходить контури об'єктів на бінаризованих зображеннях, що корисно для подальшої обробки, що видно на рисунку 3.1 та 3.2. Ці інструменти широко застосовуються у комп’ютерному зорі, автоматизації та машинному навчанні:

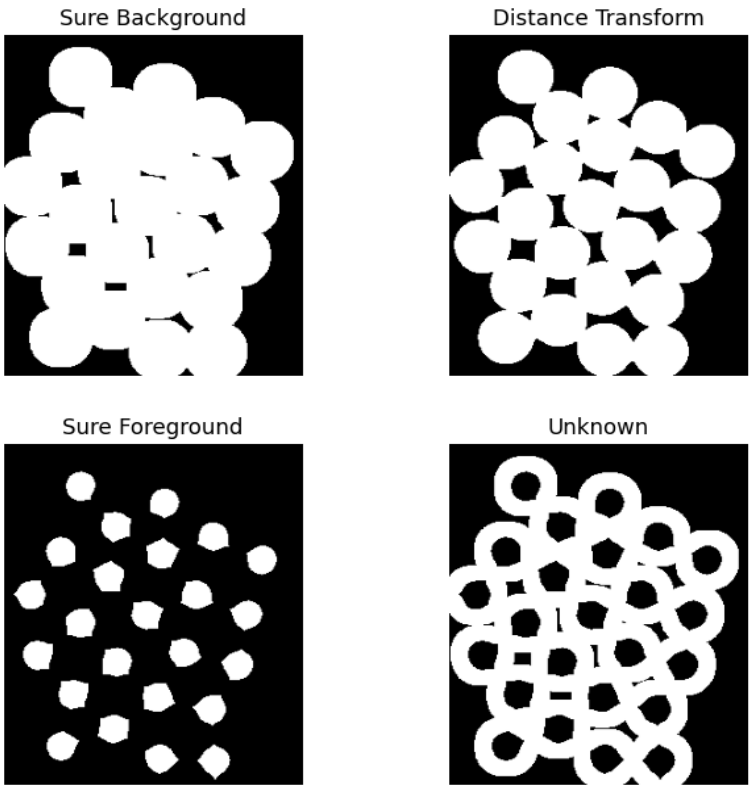

Рисунок 3.1 – Створення маркерного зображення за допомогою функції «cv2.watershed»

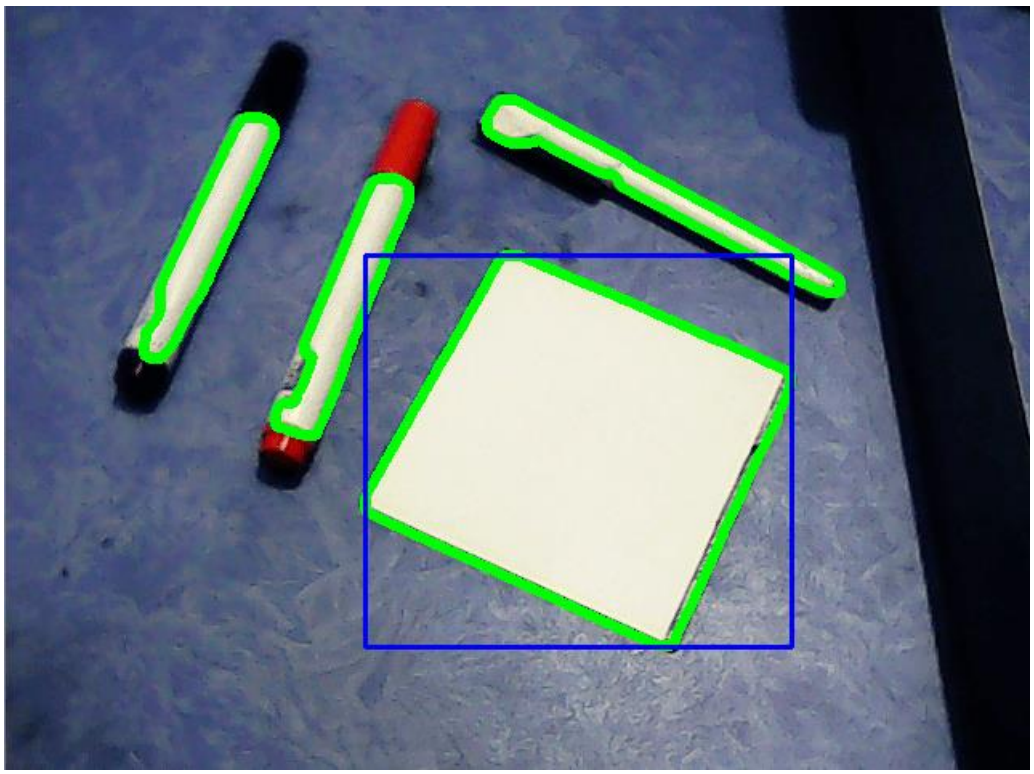

Рисунок 3.2 – Ефективне визначення контурів об'єктів,за допомогою функції «cv2.findContours»

Scikit–image надає інструменти для кластеризації зображень на основі суперпіксельного підходу. Функція segmentations.slic() дозволяє ділити зображення на однорідні області шляхом аналізу колірних та текстурних особливостей. Це покращує якість сегментації та підвищує точність подальшої обробки.

Адаптивна сегментація SLIC дозволяє розбивати зображення на суперпікселі, змінюючи їх розмір залежно від локальних особливостей. Якщо масштаби ознак у різних областях подібні, як на зображенні (a), алгоритм створює сегменти однакового розміру, що видно на зображенні (b). Це корисно для рівномірно текстурованих сцен, де важливо зберегти структурну однорідність.

Якщо ж фонова область значно тонша за передній план, як на зображенні (c), SLIC автоматично адаптується до різних ділянок. У результаті сегменти у вузьких областях стають меншими, а у широких — більшими, що видно на зображенні (d).

Це покращує точність аналізу та дозволяє ефективніше виділяти об'єкти складної форми, як наведено на рисунку 3.3:

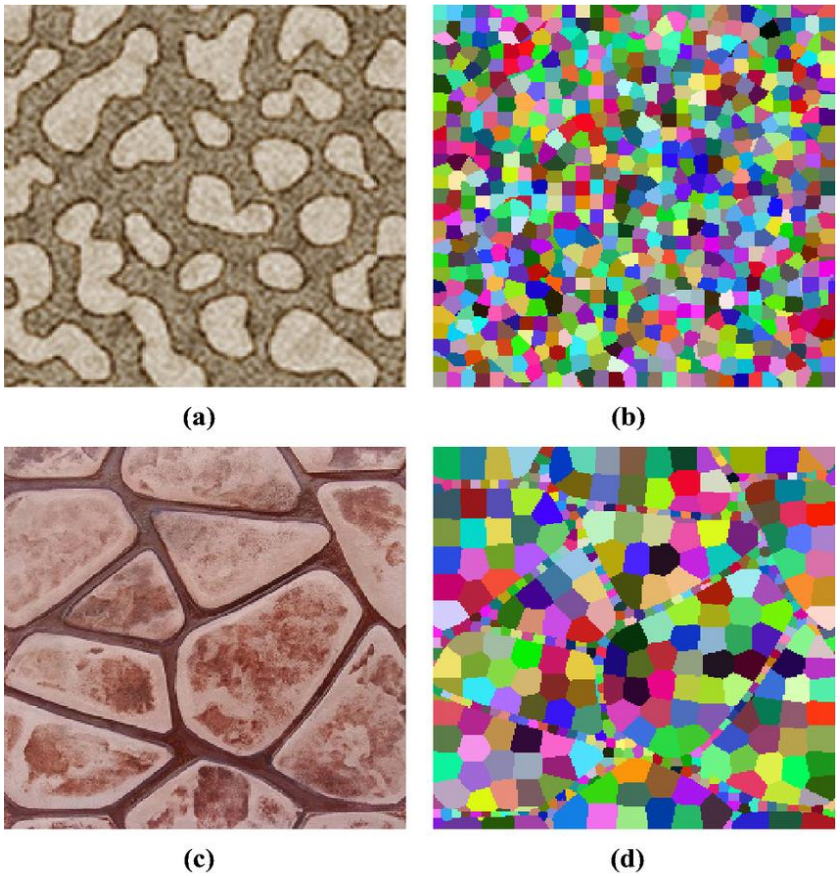

Рисунок 3.3 – Адаптивна сегментація SLIC

TensorFlow [12] використовується для реалізації глибоких нейронних мереж у задачах сегментації, як зображено у прикладі роботи алгоритму на рисунку 3.4. Застосування попередньо навчених моделей дозволяє отримувати високоточні результати без необхідності розробки алгоритмів з нуля:

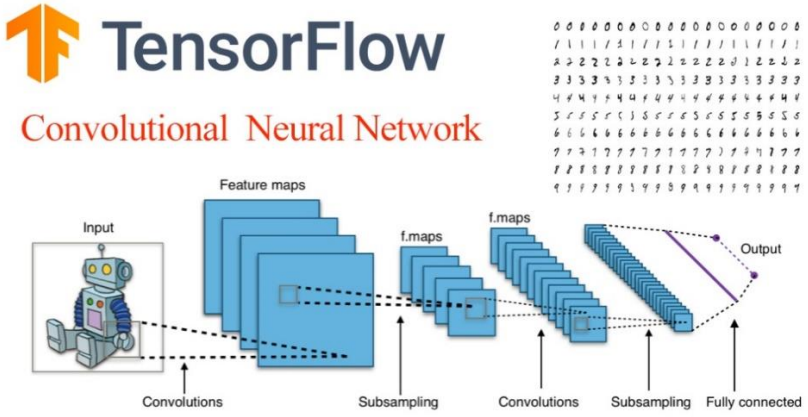

Рисунок 3.4 – Приклад алгоритму роботи «TensorFlow»

Сучасні бібліотеки надають потужні інструменти для автоматичної сегментації, поєднуючи класичні алгоритми та методи глибокого навчання. Це дозволяє точно розділяти об'єкти на зображеннях, адаптуючись до їхніх особливостей.

Завдяки таким підходам досягається висока швидкість обробки та точність, що є критично важливим для застосувань у комп'ютерному зорі, медичній діагностиці, робототехніці та інших сферах автоматизації.

У моєму коді процес автоматизації реалізований через інтеграцію алгоритмів обробки зображень із графічним інтерфейсом, що дозволяє користувачеві виконувати складні операції за допомогою простих кнопок. Кожен метод сегментації, такий як K–Means, Watershed або Region Growing, інкапсульований у функціях, які автоматично обробляють зображення на основі заданих параметрів.

Наприклад, при натисканні кнопки «K–Means» програма сама виконує кластеризацію пікселів, перетворює дані у потрібний формат і відображає результат без необхідності ручного втручання в код. Це значно спрощує роботу для користувача, якому достатньо лише вибрати потрібний алгоритм і налаштувати його параметри за допомогою інтуїтивних елементів керування.

Автоматизація також проявляється у взаємодії між різними компонентами програми. Наприклад, при виборі Region Growing, якщо користувач не вказав точку затравки, система автоматично використовує центр зображення, щоб уникнути помилок. Аналогічно, у Graph Cut попередня маска зберігається для подальшого уточнення, що дозволяє покращувати результат без повторної ініціалізації. Параметри, такі як розмір ядра для фільтрації шуму або коефіцієнт контрасту, автоматично передаються у відповідні функції OpenCV, де вони застосовуються до зображення.

Важливим аспектом автоматизації є обробка помилок і надання зворотного зв’язку. Програма самостійно перевіряє, чи завантажено зображення перед застосуванням будь–якого алгоритму, і повідомляє користувача, якщо щось пішло не так.

Наприклад, якщо K–Means не вдається виконати через проблеми з даними, система виводить повідомлення про помилку з поясненням, а не просто завершує роботу. Це робить інструмент більш надійним і зручним у використанні, оскільки користувач завжди знає стан виконання операції.

Детальний алгоритм роботи програми відображено на рисунку 3.5:

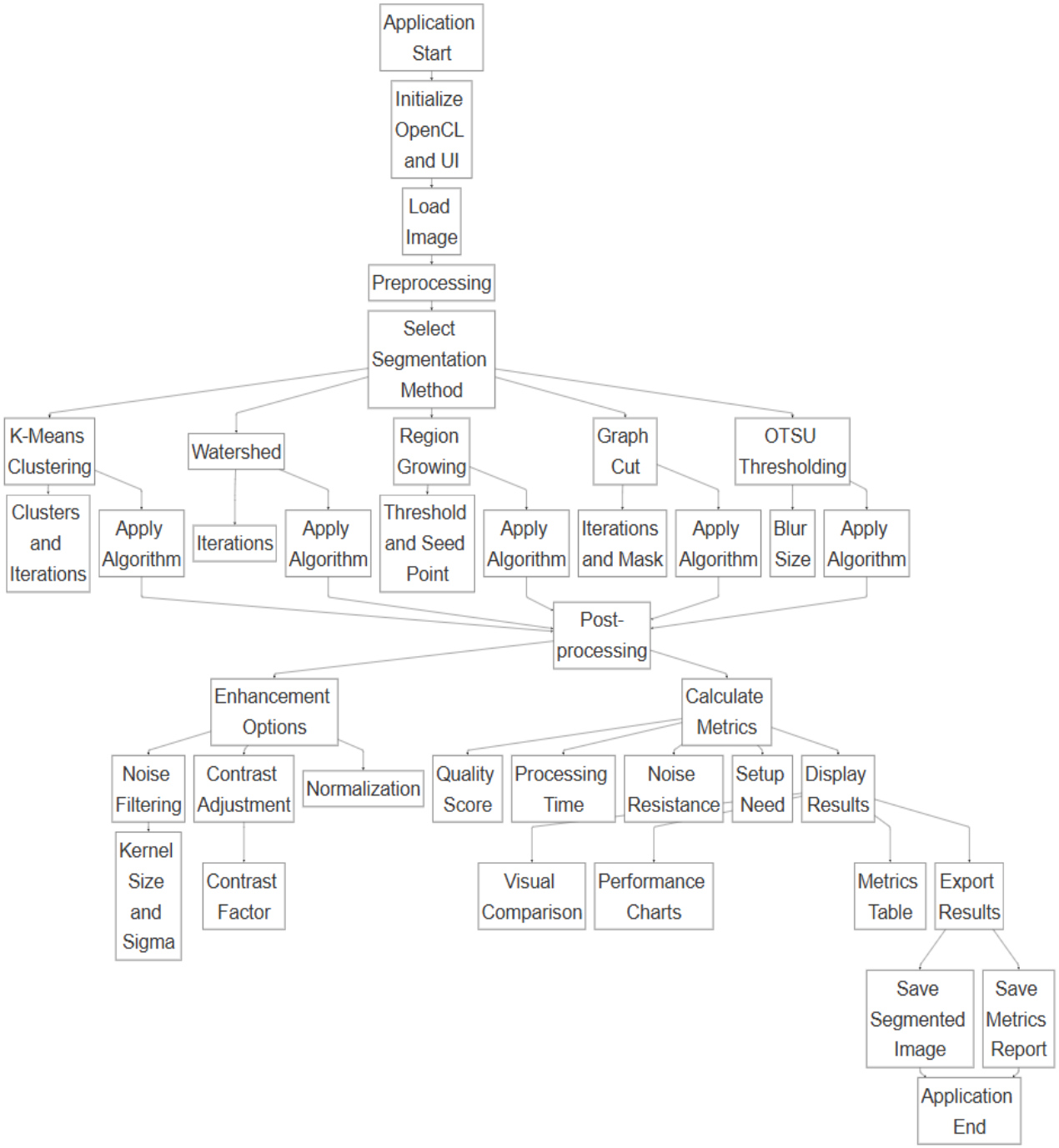

Рисунок 3.5 – Комплексна алгоритмічна схема

Останнім ключовим елементом автоматизації є візуалізація результатів. Після обробки зображення програма автоматично масштабує його для відображення у відповідній панелі, враховуючи розміри вікна. Користувачеві не потрібно вручну налаштовувати розміри або форматувати дані – все відбувається за сценарієм.

Також, при збереженні результату, система сама конвертує зображення у потрібний формат (наприклад, PNG або JPG) на основі вибраного користувачем розширення файлу. Таким чином, кожен крок від завантаження до збереження оптимізований для мінімальної участі людини та максимальної ефективності.

Інструмент для автоматизованої сегментації зображень показано на рисунку 3.6:

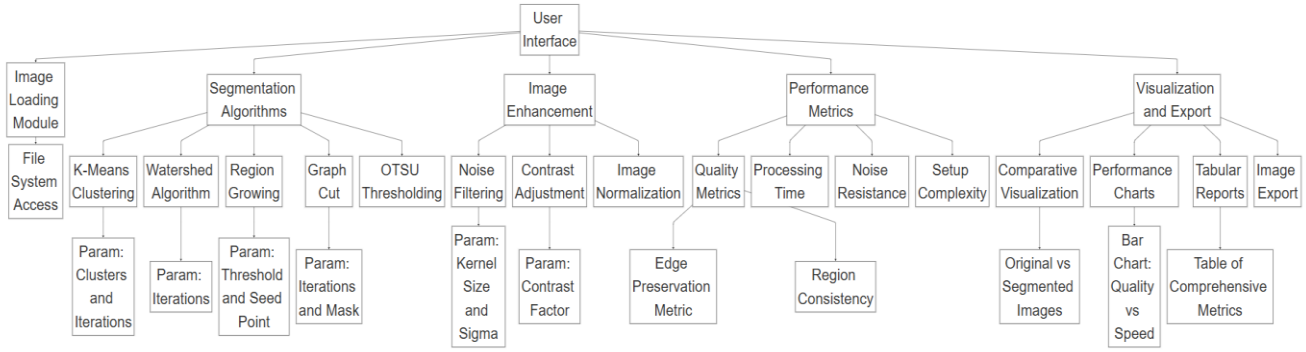

Рисунок 3.6 – Загальна архітектура системи

3.2 Розробка та реалізація алгоритмів на базі OpenCV

Сучасні методи обробки зображень все більше орієнтовані на автоматизацію, що дозволяє ефективно розділяти зображення на логічно однорідні області без необхідності постійного ручного втручання. Це досягається за рахунок використання алгоритмів, які аналізують зображення, виявляють межі, контури, кольорові та текстурні відмінності, а також автоматично налаштовують параметри для оптимального результату.

Одним із ключових аспектів автоматизації є виявлення та виділення об’єктів. Алгоритми, такі як Watershed або K–Means, аналізують зображення, знаходячи області з характеристиками, і розділяють на окремі сегменти.

Наприклад, Watershed використовує концепцію "вододілу" для точного визначення меж, тоді як K–Means групує пікселі за кольором, що особливо корисно для кластеризації простих зображень.

Оптимізація процесу також грає важливу роль. Автоматичні системи можуть адаптувати параметри (наприклад, пороги яскравості, розміри фільтрів) для підвищення точності. У реалізованому інтерфейсі користувач може регулювати кількість кластерів у K–Means, рівень розмиття для OTSU або інтенсивність фільтрації шуму, що дозволяє досягти кращих результатів без ручного підбору значень.

Ще одна перевага автоматизації — зменшення необхідності ручної корекції. Такі методи, як Region Growing або Graph Cut, дозволяють системі самостійно виділяти області на основі початкових умов.

Алгоритм Watershed використовується для автоматичного розділення об’єктів, які мають спільні межі. Метод працює на основі концепції топографічного рельєфу, де локальні мінімуми зображення розглядаються як "басейни", а сегментація здійснюється шляхом заповнення цих областей.

Реалізація в OpenCV дозволяє ефективно розділяти злиплі об’єкти без необхідності ручного маркування, як зображено на рисунку 3.7:

Рисунок 3.7 – Приклад роботи алгоритму Watershed

Метод k–means використовується для кластеризації зображень, розподіляючи пікселі на області за кольоровими або текстурними характеристиками.

Крім того, метод може визначати оптимальну кількість кластерів, що підвищує точність розділення та адаптує процес до різних типів зображень, як на рисунку 3.8:

Рисунок 3.8 – Приклад роботи методу k–means

Метод Otsu’s Thresholding використовується для автоматичного визначення оптимального порогового значення у бінаризації зображень. Алгоритм аналізує гістограму яскравості пікселів і знаходить такий поріг, який мінімізує внутрішньокласову дисперсію пікселів у двох сегментах, що видно на рисунку 3.9:

Рисунок 3.9 – Приклад роботи методу Otsu’s Thresholding

Методи покращення контрастності, такі як Histogram Equalization та Contrast Limited Adaptive Histogram Equalization (CLAHE), використовуються для підвищення якості зображення перед сегментацією.

Ці методи особливо ефективні у випадках, коли зображення має слабко виражені деталі або неоднорідне освітлення. Завдяки покращенню контрастності підвищується точність подальших алгоритмів аналізу, що важливо для обробки медичних, супутникових та інших технічних зображень, таким чином, як на рисунку 3.10:

Рисунок 3.10 – Приклад роботи методу покращення контрастності

Метод Graph–Cut реалізує сегментацію як задачу мінімізації енергетичної функції на графі, де вузли відповідають пікселям, а ребра — взаємозв’язкам між ними. Використовуючи інформацію про градієнти, текстури та кольори, алгоритм дозволяє ефективно розділяти об’єкти навіть у складних сценах, що продемонстровано на рисунку 3.11:

Рисунок 3.11 – Приклад роботи методу Graph–Cut

Моя реалізація орієнтована на роботу зі стандартними зображеннями у кольоровому (RGB) або відтінках сірого (Grayscale) форматах, що є найпоширенішими типами даних для базових завдань комп'ютерного зору.

Вона ідеально підходить для дослідження та навчання, оскільки пропонує класичні методи сегментації, такі як K–Means або Watershed, які ефективно працюють із простими зображеннями.

Однак програма не підтримує багатокласову сегментацію, тобто не може одночасно виділяти кілька об'єктів різних типів на одному зображенні, що обмежує її застосування для складних завдань аналізу даних.

Наразі інструмент працює лише зі звичайними растровими форматами, такими як PNG, JPG або BMP, і не включає підтримку спеціалізованих форматів, як DICOM, які використовуються у медичній візуалізації. Це означає, що для обробки томографічних знімків або інших медичних даних користувачам доведеться конвертувати їх у підтримувані формати.

Проте архітектура програми дозволяє розширити функціонал у майбутньому, додавши модулі для роботи з DICOM або іншими спеціалізованими типами даних, що відкриє нові можливості для застосування в різних галузях.

Хоча поточна версія не використовує моделі глибокого навчання, вона може стати основою для їх подальшої інтеграції. Наприклад, можна додати підтримку нейромережних архітектур, таких як U–Net або DeepLab, для більш точної сегментації складних зображень. Це особливо актуально для медичних додатків, де автоматичне виділення пухлин або інших аномалій вимагає високої точності. Таке оновлення також дозволить реалізувати багатокласову сегментацію, що значно розширить сферу застосування програми [10].

Отже, хоча поточна реалізація орієнтована на базову сегментацію простих зображень, її модульна структура дозволяє легко адаптуватися до нових вимог.

Майбутні оновлення можуть включати підтримку DICOM, інтеграцію глибокого навчання та розширені функції для роботи зі складними даними, що зробить цей інструмент більш універсальним і потужним.

3.3 Розробка уніфікованого інтерфейсу для тестування

Ця програма для сегментації зображень розроблена як універсальний інструмент, що дозволяє працювати з різними алгоритмами обробки зображень. Її головна мета — спростити взаємодію з технічно складними методами комп’ютерного зору, надаючи користувачу інтуїтивно зрозуміле середовище для роботи.

У програмі реалізовано поєднання потужностей бібліотеки OpenCV, що є стандартом у галузі комп’ютерного зору, із зручним графічним інтерфейсом, створеним за допомогою PyQt5. Це поєднання забезпечує стабільну та ефективну обробку зображень у реальному часі, відкриваючи широкі можливості для візуалізації й аналізу результатів.

Програма орієнтована як на дослідників, які тестують і порівнюють алгоритми, так і на практиків, які використовують сегментацію в прикладних задачах – від медицини до промисловості.

Її функціональність дозволяє легко інтегрувати нові методи, налаштовувати параметри і бачити результати одразу.

Основна ідея полягає в тому, щоб зробити застосування складних алгоритмів доступним навіть для користувачів без досвіду програмування. Таким чином, програма виступає містком між алгоритмічною складністю та зручністю практичного використання.

Головне вікно програми розділене на дві основні частини: ліворуч відображаються оригінальне та оброблене зображення, праворуч розміщені всі елементи керування, що видно на рисунку 3.12. Користувач може завантажити будь–яке зображення у поширених форматах, після чого починається робота з ним:

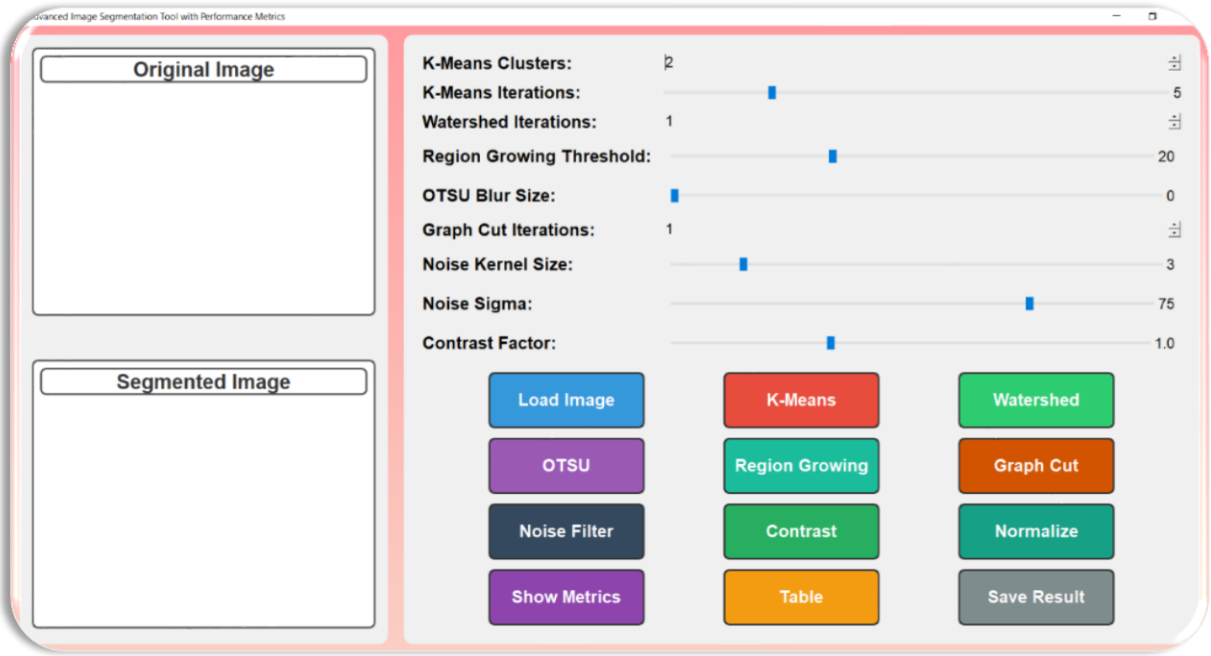

Рисунок 3.12 – Головне вікно програми

Інтерфейс реалізований з урахуванням зручності – кнопки мають ефект підсвічування при наведенні, всі параметри можна легко регулювати за допомогою повзунків та спінерів, як показано на рисунку 3.13:

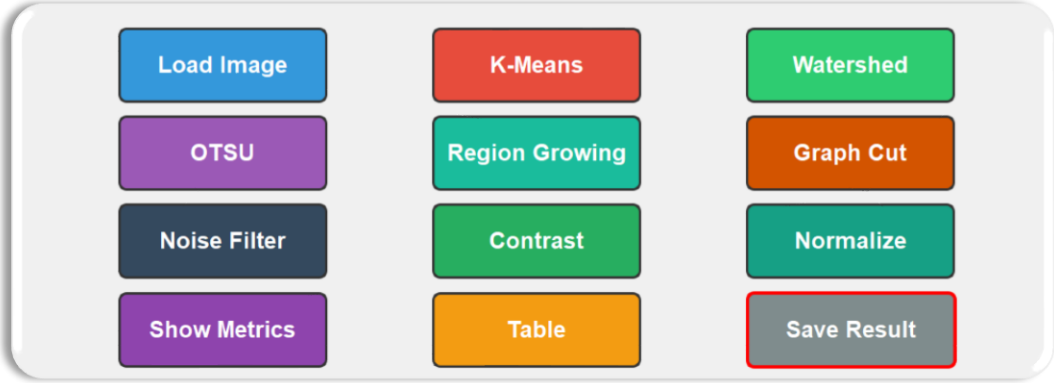

Рисунок 3.13 – Кнопки з ефектом підсвічування

Програма підтримує кілька ключових методів сегментації. Метод k–means дозволяє розділяти зображення на кольорові кластери, де користувач може вибрати кількість кластерів та ітерацій. Алгоритм вододілу працює з межами об'єктів, використовуючи морфологічні операції – тут також можна регулювати кількість ітерацій.

Метод OTSU пропонує автоматичну бінаризацію з можливістю попереднього розмиття. Region Growing виділяє області, починаючи від вибраної точки, з регульованим порогом подібності. Graph Cut – більш складний алгоритм, який використовує оптимізацію границь і потребує попередньої маски.

Крім основних методів сегментації, програма містить інструменти для підготовки зображень. Білатеральний фільтр допомагає зменшити шум з регульованими параметрами ядра та інтенсивності. Функції корекції контрасту та нормалізації дозволяють покращити якість зображення перед сегментацією. Всі зміни відображаються у реальному часі, що дозволяє швидко підібрати оптимальні параметри.

Після обробки користувач може зберегти результат у зручному для нього форматі. Для підвищення продуктивності використовується апаратне прискорення через OpenCL.

Програма також включає систему обробки помилок, яка попереджає про проблеми, наприклад, коли користувач намагається обробити зображення перед його завантаженням.

Цей інструмент знайде застосування у багатьох сферах – від медичної діагностики, де важлива точна сегментація знімків, до промисловості та супутникового моніторингу. Він особливо корисний тим, кому потрібно швидко тестувати різні методи обробки зображень без написання коду. Простий інтерфейс у поєднанні з потужними алгоритмами робить цю програму зручним рішенням для багатьох завдань комп'ютерного зору.

Програма реалізує повноцінний графічний інтерфейс на PyQt5, який включає всі необхідні елементи для зручної роботи з алгоритмами сегментації. Головне вікно програми розділене на дві основні області. Ліву частину для відображення вихідного та обробленого зображень.

І праву панель керування з усіма інтерактивними елементами. Кожен з цих елементів має чітке призначення та інтуїтивно зрозуміле розташування, що забезпечує зручність користування. Крім того, інтерфейс підтримує динамічне оновлення результатів у режимі реального часу після зміни параметрів.

Панель керування містить різноманітні елементи введення параметрів, включаючи слайдери для регулювання значень (наприклад, для зміни контрасту або розміру ядра фільтра), спіновери для вибору дискретних значень (як кількість кластерів у K–means), а також кнопки для виклику основних функцій. Усі елементи мають підписи та індикатори поточних значень, що дозволяє легко контролювати параметри обробки. Для зручності користувача кнопки виконані у вигляді інтерактивних елементів із ефектом підсвічування при наведенні.

Система меню включає всі необхідні функції для роботи із зображеннями: кнопки «Load Image» для завантаження вхідного зображення, «Save Resul» для збереження результатів обробки, а також набір кнопок для вибору конкретного методу сегментації. Особливу увагу приділено відображенню результатів – обидва зображення (вихідне та оброблене) відображаються у спеціальних областях із підписами, з автоматичним масштабуванням під розмір вікна.

Додатково реалізовано спеціальні вікна для візуалізації результатів аналізу продуктивності методів. Вони включають інтерактивні графіки для порівняння показників якості та швидкості роботи алгоритмів, а також таблицю з детальною інформацією про властивості кожного методу. Усі елементи інтерфейсу мають послідовну логіку розміщення та витримані в єдиній стилістиці, що робить роботу з програмою зручною та ефективною.

3.3.1. Автоматизація процесу сегментації зображень

Автоматична сегментація зображень є ключовою технологією в сучасних системах комп’ютерного зору, що забезпечує розподіл зображень на логічно однорідні області без участі оператора. Вона є основою для багатьох задач обробки та аналізу зображень, таких як виявлення об’єктів та розпізнавання образів.

Методи автоматичної сегментації базуються на різних підходах, включаючи порогову обробку, розподіл областей, кластеризацію та методи глибокого навчання.

Застосування нейронних мереж, зокрема згорткових (CNN) і трансформерів (ViT), дозволяє досягати високої точності сегментації завдяки глибокому аналізу контекстуальних особливостей зображення [9]. Такі моделі навчаються на великих наборах даних та демонструють стійкість до змін освітлення, ракурсу та інших факторів, що впливають на якість сегментації, що видно на рисунку 3.14:

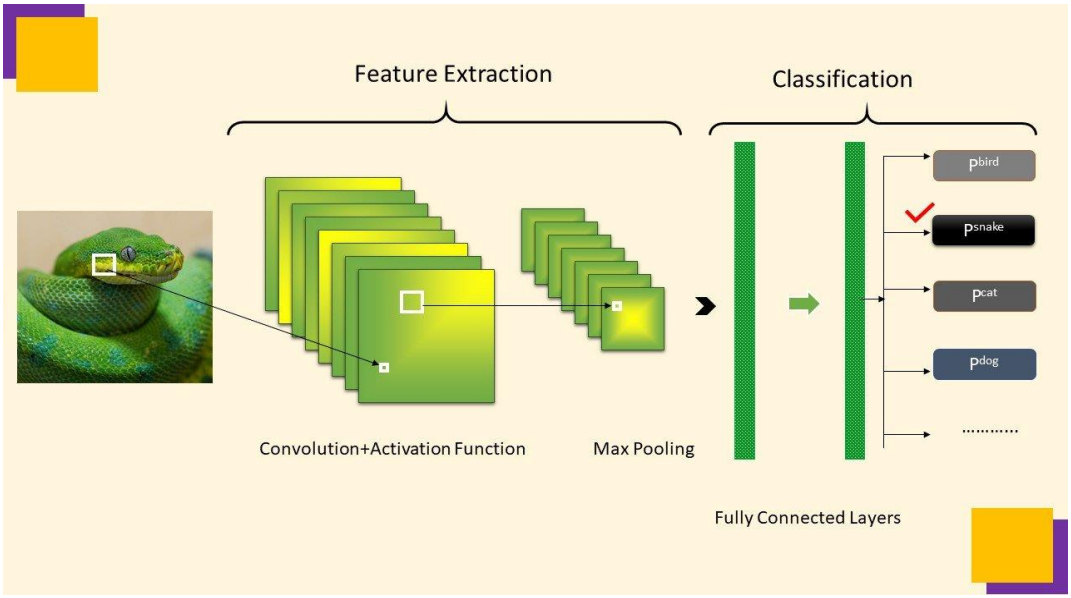

Рисунок 3.14 – Класичні згорткові нейронні мережі

Автоматизація процесу сегментації є критично важливою у сферах, де необхідна швидка та точна обробка великої кількості зображень, як зображено на рисунку 3.15. Наприклад, у медичній діагностиці автоматичне виділення патологічних змін у зображеннях МРТ сприяє прийняттю рішень лікарями [11]:

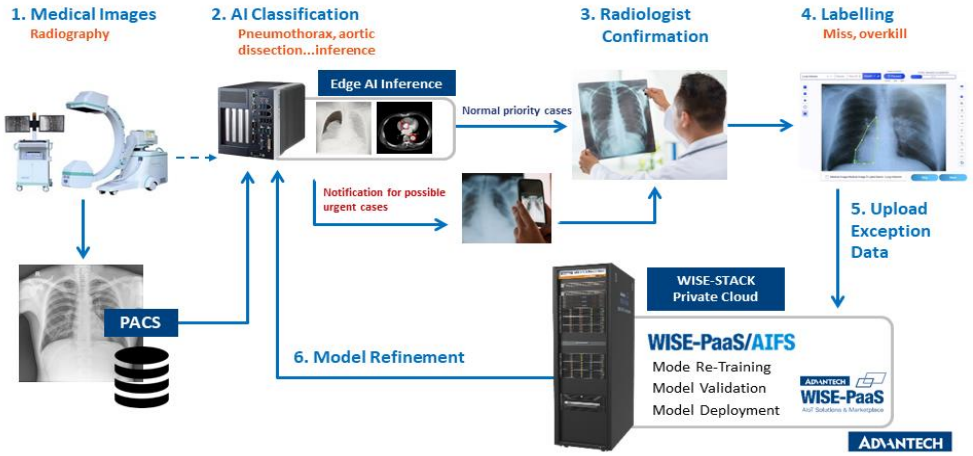

Рисунок 3.15 – Чотири процедури для оптимізації ШІ–моделі

Сегментація зображень також відіграє важливу роль у супутниковому моніторингу, де необхідно аналізувати великі обсяги даних для виявлення змін у довкіллі, моніторингу природних катастроф та прогнозування врожайності. Використання глибоких нейронних мереж дозволяє покращити точність та узагальнюваність аналізу, знижуючи потребу в ручному налаштуванні параметрів.

Таким чином, автоматична сегментація є важливим етапом у багатьох технологічних процесах, що сприяє підвищенню швидкості, точності та ефективності аналізу зображень. Розвиток сучасних методів машинного навчання відкриває нові можливості для вдосконалення сегментації, що робить її ще більш універсальною та адаптивною до складних завдань.

3.3.2. Методи автоматичної сегментації зображень

Одним із базових підходів до автоматичної сегментації є аналіз контурів, що дозволяє точно визначати межі об'єктів. Класичні алгоритми використовуються для обробки градієнтів яскравості та побудови чітких контурів. Вони ефективні для простих випадків, але мають обмеження при роботі з неоднорідними фонами. Демонстрація чого є на рисунку 3.16

Інший підхід передбачає використання глибоких нейронних мереж. Архітектури, такі як U–Net [21] та Mask R–CNN, забезпечують високу точність сегментації завдяки аналізу локальних особливостей зображення [8]:

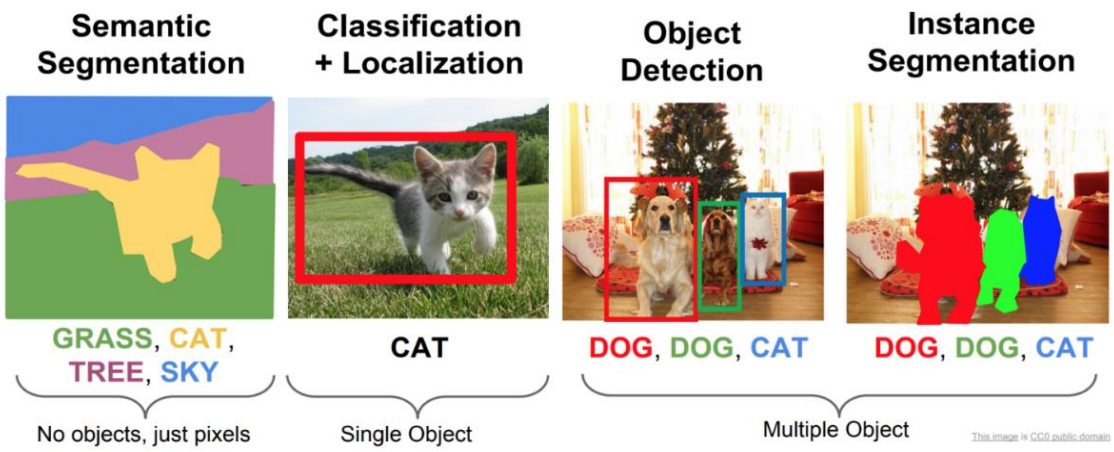

Рисунок 3.16 – Mask R–CNN

Покращення якості сегментації можливе за рахунок адаптивного підходу до вибору порогових значень. Метод Otsu’s Thresholding дозволяє автоматично розділяти пікселі на основі гістограми яскравості, тоді як Adaptive Thresholding враховує локальні особливості освітлення, що підвищує точність розпізнавання об'єктів у неоднорідних сценах.

Кластеризаційні методи, зокрема k–means clustering, можуть бути автоматизовані шляхом визначення оптимальної кількості кластерів за допомогою Elbow Method. Це дозволяє зменшити вплив суб’єктивного вибору параметрів, що робить алгоритми більш універсальними. Як приклад зображено метод на рисунку 3.17:

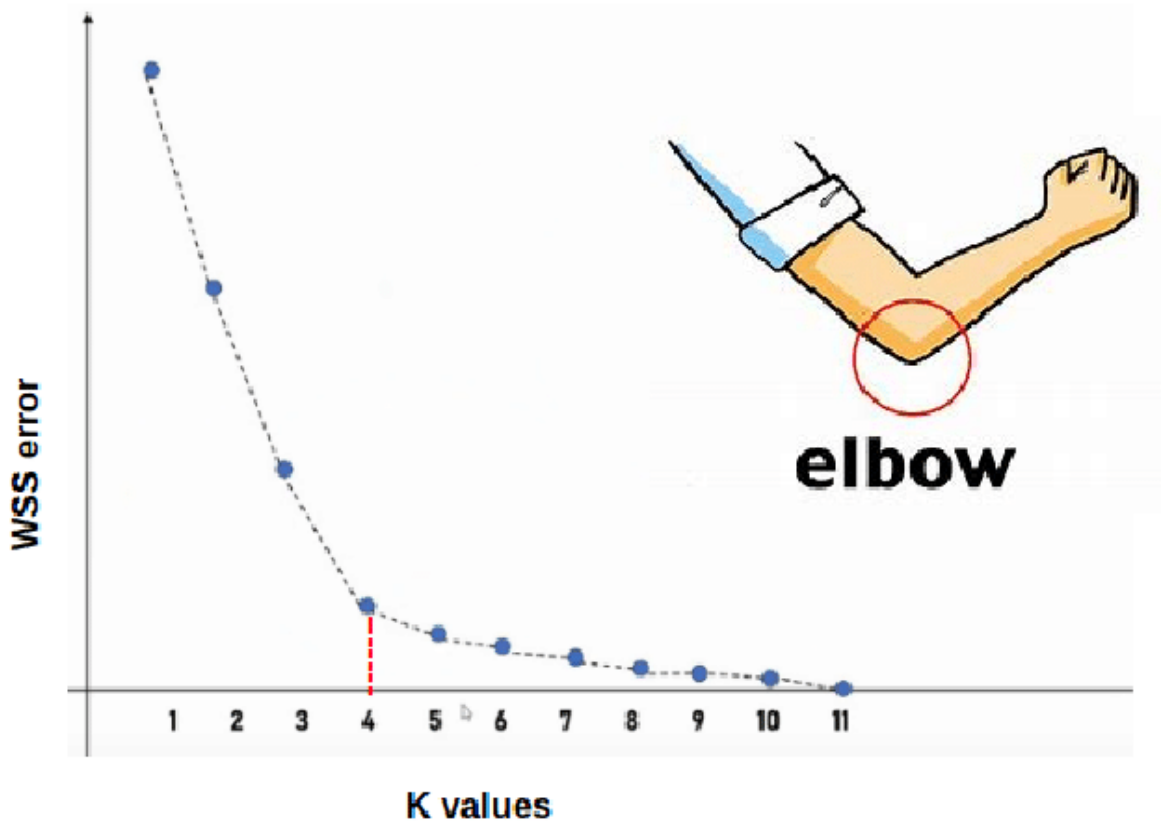

Рисунок 3.17 – Ліктьовий метод з використанням спотворення

Сучасні алгоритми прагнуть максимально знизити необхідність ручного налаштування. Самоорганізовані карти (Self–Organizing Maps, SOM) дозволяють розділяти зображення на області без попереднього маркування даних, що є важливим у задачах із великим обсягом вхідної інформації.

Розвиток нейромережевих підходів, зокрема архітектури DeepLabV3+, сприяє вдосконаленню методів сегментації зображень завдяки глибокому навчанню. Використання глибоких згорткових нейронних мереж (CNN) у DeepLabV3+ дозволяє підвищити точність розпізнавання об'єктів [3].

Це особливо актуально у сфері комп’ютерного зору, де моделі повинні ефективно працювати з різними типами зображень. Приклад використання наведено на рисунку 3.18:

Рисунок 3.18 – DeepLabV3 у сегментації зображень

Сучасні методи сегментації зображень все частіше використовують потужні архітектури нейронних мереж, які показують значно кращі результати порівняно з класичними підходами. Такі моделі, як U–Net, DeepLab та Mask R–CNN, базуються на глибокому навчанні та здатні виявляти складні шаблони в даних, що робить їх ідеальними для завдань медичної візуалізації, автономного водіння чи аналізу супутникових знімків [2].

Наприклад, U–Net зі своєю U–подібною архітектурою поєднує низькорівневі та високорівневі особливості, що дозволяє сегментувати невеликі об’єкти. DeepLab використовує атрійні згортки та просторові пірамідальні пулінги для охоплення контексту, що корисно при роботі зі сценами [3-7].

Mask R–CNN [8] є розширенням моделі Faster R–CNN і додає ще один вихід для створення бінарних масок кожного виявленого об’єкта. Ця архітектура досягає високої точності не лише у визначенні меж об’єктів, а й у їх класифікації, що робить її популярною для завдань інстанс–сегментації.

Такі моделі навчаються на великих наборах даних, таких як COCO або Cityscapes, і можуть бути доопрацьовані під конкретні потреби за допомогою трансферного навчання.

Незважаючи на свою складність, вони пропонують значні переваги у порівнянні з традиційними методами, особливо у випадках, коли потрібна висока точність або обробка складних сцен.

У моєму інструменті поки що використовуються класичні методи сегментації, такі як K–Means або Watershed, оскільки вони простіші у реалізації та не вимагають потужних обчислювальних ресурсів. Однак архітектура програми дозволяє легко інтегрувати сучасні нейромережні підходи у майбутньому.

Наприклад, можна додати окремий модуль для роботи з попередньо навченими моделями, який би використовував бібліотеки, такі як TensorFlow або PyTorch. Це дозволило б користувачам вибирати між швидкими, але менш точними класичними методами та точними, але більш ресурсомісткими нейромережами залежно від їхніх потреб.

Інтеграція нейронних мереж також відкриває можливості для більш просунутих функцій, таких як автоматичне виділення об’єктів із складних зображень або навіть семантична сегментація, де кожен піксель відноситься до певного класу.

У майбутньому можна було б реалізувати підтримку користувацьких моделей, щоб дослідники могли завантажувати свої власні навчені мережі для вирішення спеціалізованих завдань. Таким чином, програма може еволюціонувати від простого інструменту для базової сегментації до потужної платформи для комп’ютерного зору з підтримкою найсучасніших методів на основі штучного інтелекту.

4 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Опис експериментального середовища

Функціонал програми для сегментації зображень. Дана програма є інтерактивним інструментом для сегментації зображень, розробленим на основі бібліотек OpenCV та PyQt5. Вона надає зручний графічний інтерфейс, який дозволяє користувачам завантажувати зображення, обробляти їх різними методами сегментації, налаштовувати параметри цих методів і зберігати отримані результати.

Програма дозволяє завантажувати зображення у популярних форматах (PNG, JPG, JPEG, BMP, TIFF). Після завантаження воно відображається у лівій частині інтерфейсу у вікні «Original Image». Користувач може побачити оригінал перед початком обробки. Програма реалізує кілька класичних методів сегментації, кожен з яких має свої особливості та параметри.

K–Means – алгоритм кластеризації, який групує пікселі зображення за кольором у задану кількість кластерів. Користувач може регулювати кількість кластерів (від 2 до 10) та кількість ітерацій алгоритму (від 1 до 20).

Watershed – метод, що використовує морфологічні операції для точного визначення меж об'єктів. Користувач може налаштувати кількість ітерацій морфологічних операцій (від 1 до 10).

OTSU – автоматичний метод бінаризації, який вибирає оптимальний поріг для поділу зображення на передній план і фон. Додатково можна застосувати розмиття (з регульованим розміром ядра) для покращення результатів.

Region Growing – метод, що виділяє область, починаючи з обраної точки–затравки. Користувач може вказати поріг подібності пікселів (від 5 до 50).

Graph Cut – просунутий метод сегментації, який використовує оптимізацію границь об'єктів. Користувач може регулювати кількість ітерацій алгоритму (від 1 до 5).

4.2 Інструменти обробки та взаємодія з користувачем

Додаткові інструменти обробки: окрім методів сегментації, програма надає інструменти для попередньої обробки зображень. Noise Filter – білатеральний фільтр для зменшення шуму з регульованим розміром ядра (від 1 до 15) та параметром sigma (від 1 до 100). Contrast Adjustment – коригування контрасту з можливістю зміни фактора (від 0.01 до 3.0). Normalization – нормалізація яскравості та контрасту зображення для покращення подальшої обробки.

Взаємодія з користувачем: інтерфейс програми включає деякі елементи.

Кнопки з ефектом підсвічування – інтерактивні кнопки для вибору методу обробки, які змінюють колір при наведенні.

Регулятори параметрів – слайдери та спінери для точного налаштування кожного алгоритму.

Вікна перегляду – окремі області для відображення оригінального та обробленого зображень з можливістю масштабування.

Статусний рядок – відображає інформацію про дії користувача та стан обробки.

Збереження результатів: після обробки користувач може зберегти результат у форматі PNG, JPG або BMP. Програма надає стандартний діалог вибору місця збереження та імені файлу.

Обробка помилок: програма включає перевірки на помилки. Спроба обробки без завантаженого зображення. Невірні параметри обробки. Помилки зчитування/запису файлів. У разі виникнення проблем користувач отримує зрозумілі повідомлення з описом помилки.

Продуктивність: Для підвищення швидкодії використовується оптимізація OpenCV (зокрема, OpenCL для прискорення обчислень), а також ефективне оновлення інтерфейсу без зайвого перемальовування.

4.3 Підбір тестових даних

Метою підбору тестових зображень є забезпечення адекватної перевірки функціональності програми сегментації. Для цього обрано зображення з різним рівнем складності, контрастності, структурних елементів та типів об’єктів. Це дозволяє протестувати ефективність алгоритмів сегментації у різних умовах.

Критерії вибору зображень – під час підбору зображень враховувалися наступні критерії: наявність чітко виражених об’єктів та фону, різноманітність кольорової палітри та текстур, рівень шуму та артефактів, використання як реалістичних зображень (портретів, пейзажів), так і абстрактних/штучно створених для тестування адаптивності алгоритмів.

Джерела даних – для формування набору були використані відкриті ресурси з ліцензією Creative Commons, а також створені вручну зображення для демонстрації специфічних сценаріїв. Один із прикладів зображення наведено нижче, де представлено результат обробки до та після застосування методів сегментації.

Метод K–Means. Для тестування використовувалась кількість кластерів K = 2, кількість ітерацій = 5. Таке налаштування дозволяє сегментувати об'єкти без надмірного поділу. Результати відображено на рисунках 4.1, 4.2 та 4.3:

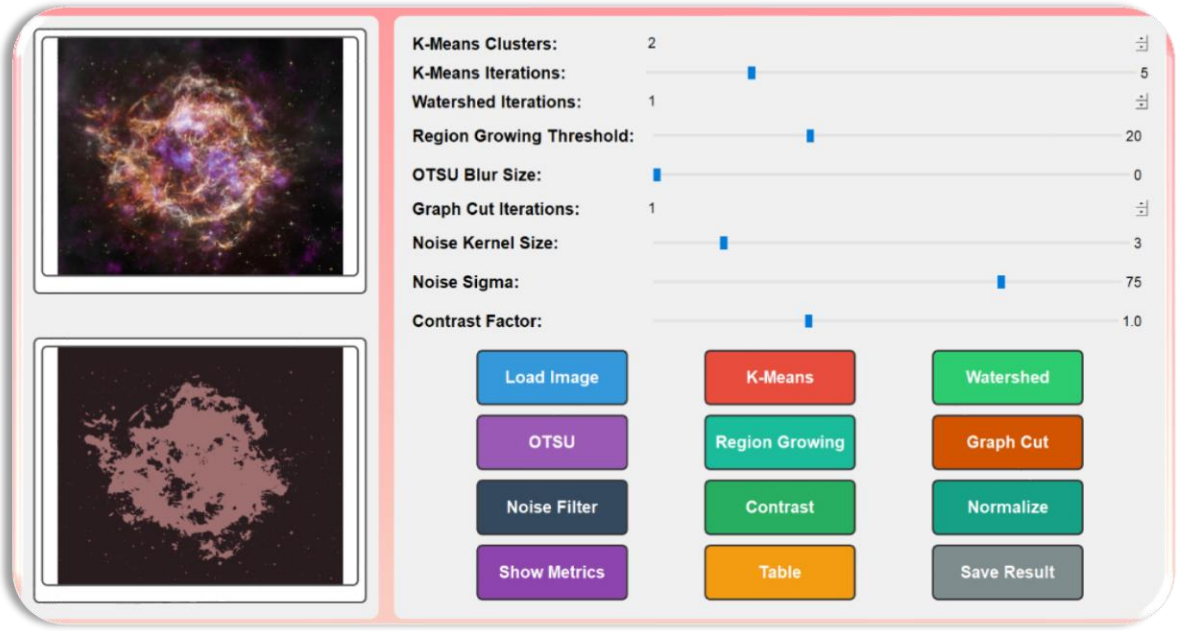

Рисунок 4.1 – Метод K–Means для першого зображення

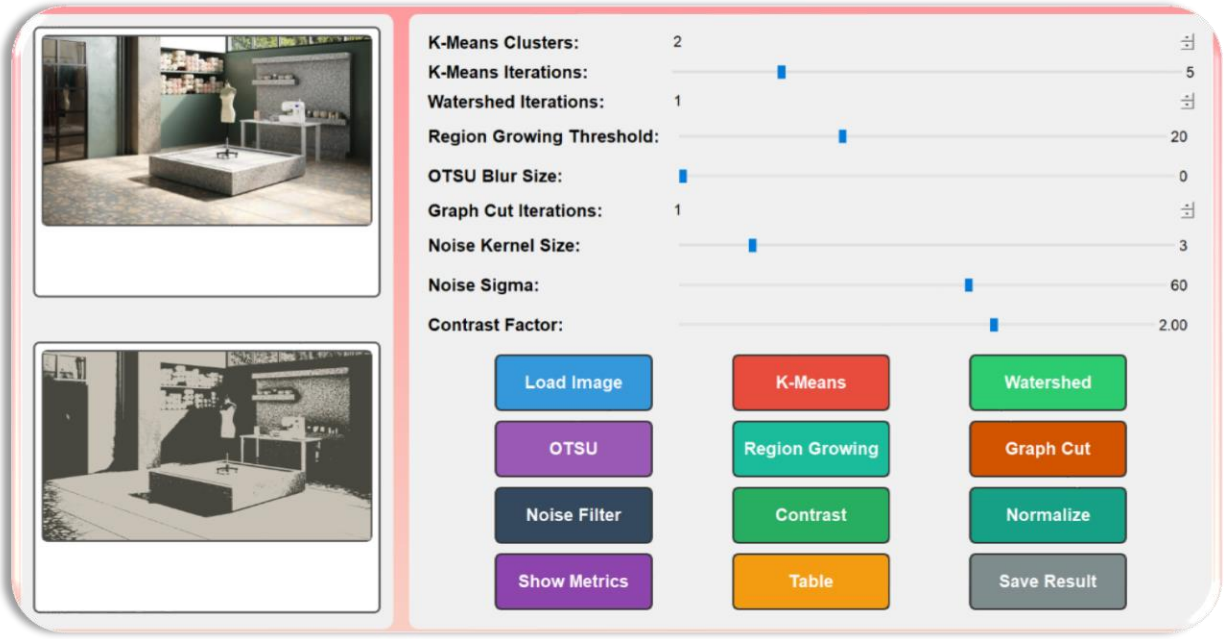

Рисунок 4.2 – Метод K–Means для другого зображення

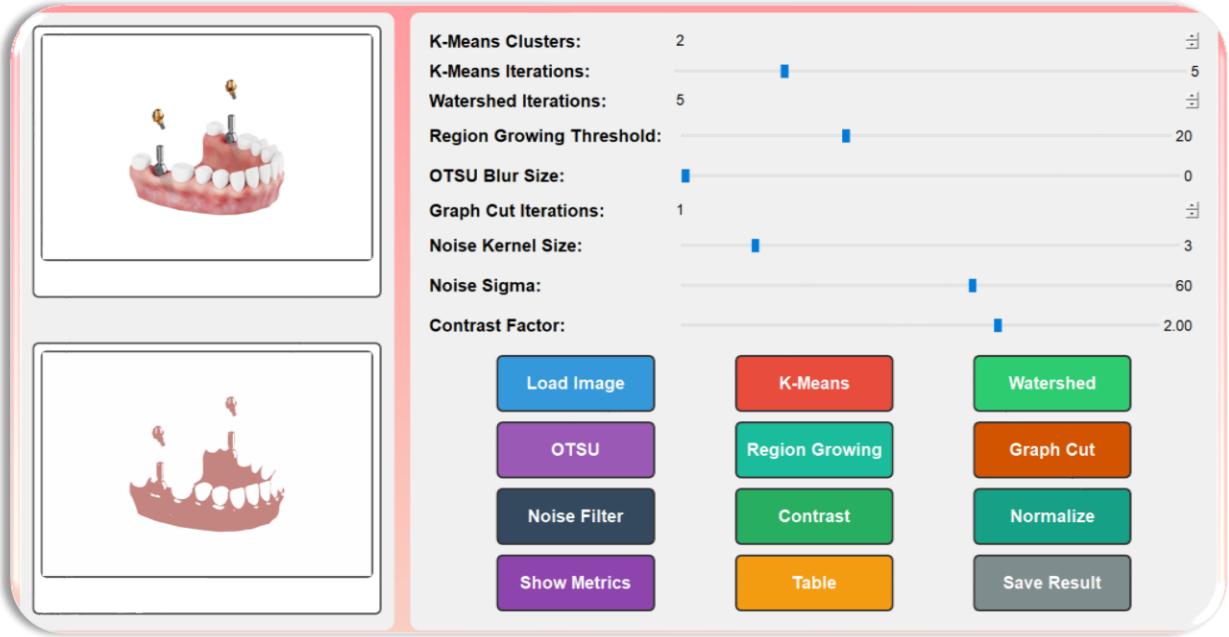

Рисунок 4.3 – Метод K–Means для третього зображення

Аналіз результатів – метод K–Means ефективно виділив загальні структури – космічного об’єкта, зубного протезу та кімнати – проте втратив дрібні деталі через орієнтацію лише на кольорову близькість без урахування просторового контексту, що обмежує його точність у задачах, де важливе збереження тонкої морфології.

Метод Watershed. Для тестування використано 1 ітерацію. Підвищення кількості ітерацій дозволяє досягти більшої деталізації, але може призвести до надлишкового поділу Результати відображено на рисунках 4.4, 4.5 та 4.6:

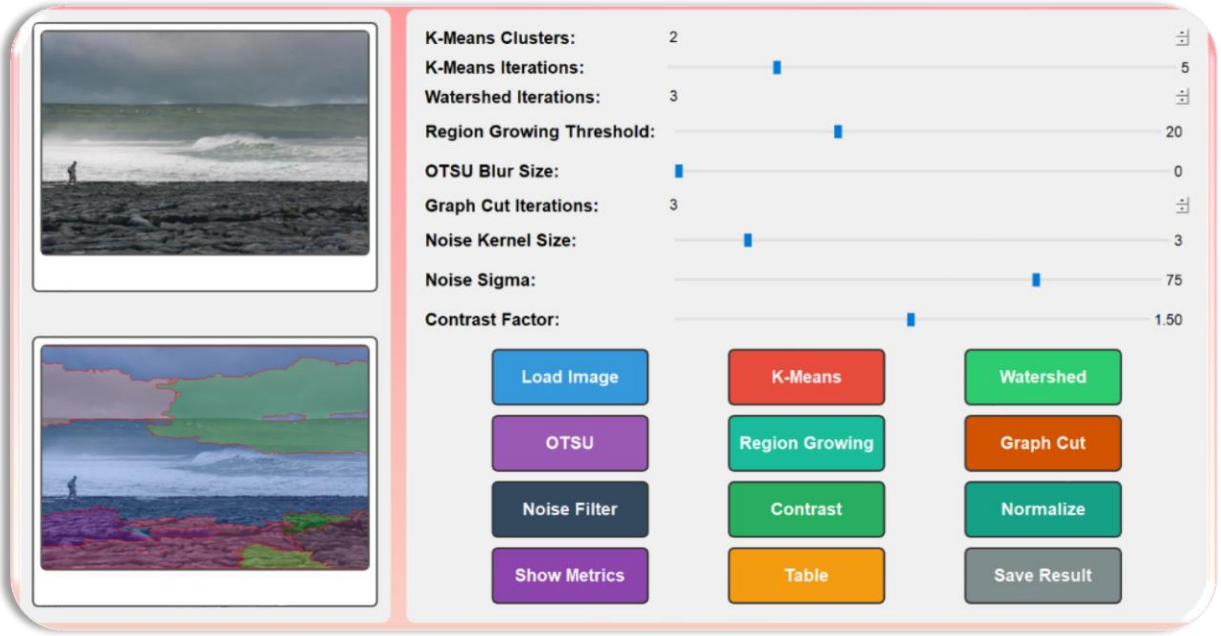

Рисунок 4.4 – Метод Watershed для першого зображення

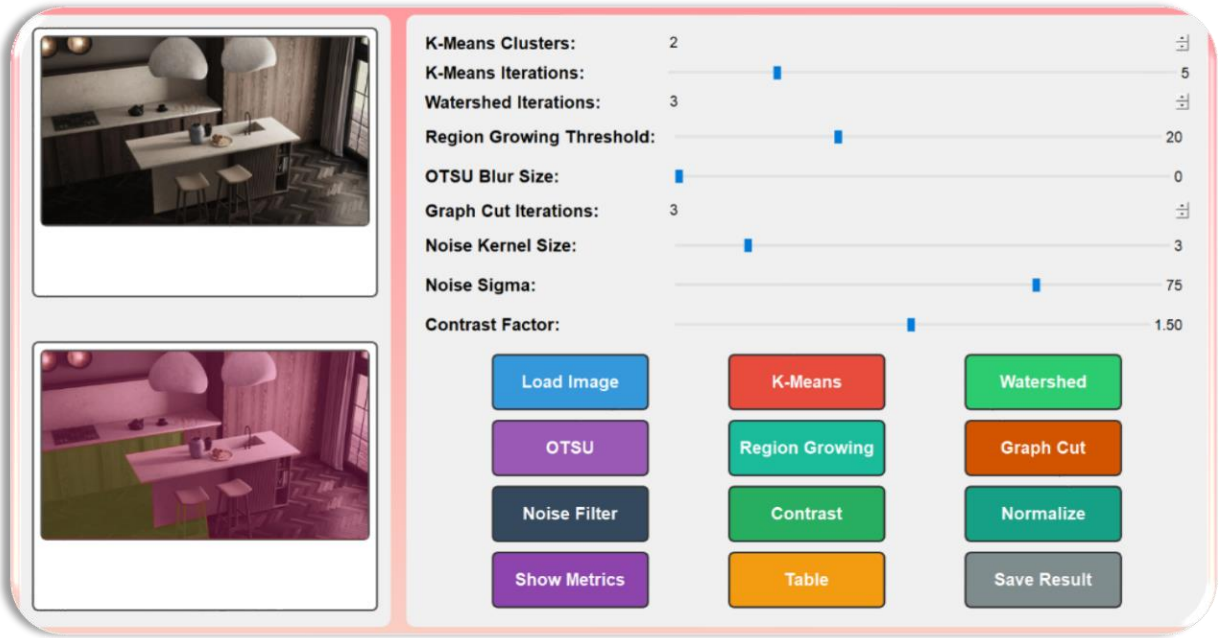

Рисунок 4.5 – Метод Watershed для другого зображення

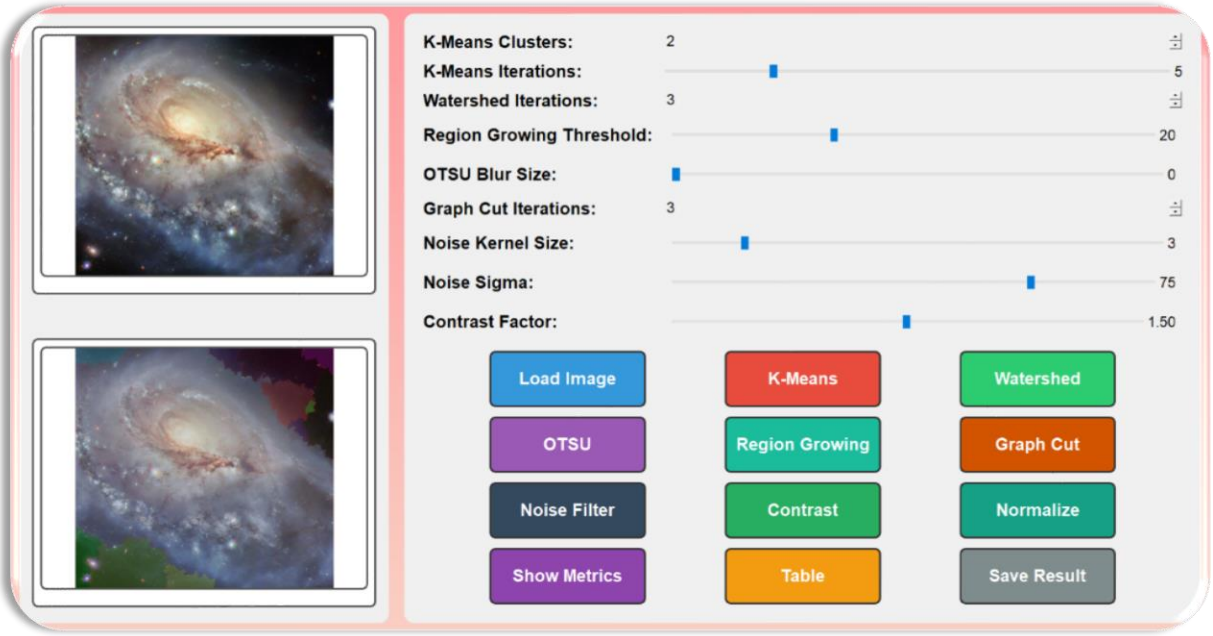

Рисунок 4.6 – Метод Watershed для третього зображення

Аналіз результатів – метод продемонстрував успішну сегментацію: на прикладі зображення узбережжя він розділив регіони похмурого сіро-блакитного неба, океану з різними відтінками хвиль, кам'янистого берега та силуету людини, а на зображенні кухні точно виокремив просторові елементи – світлі стільниці, темну підлогу, сірі стіни, стільці та округлі світильники, ефективно визначивши контрастні межі між об'єктами завдяки відмінностям у кольорах і текстурах.

Метод Region Growing. Для тестування було використано порогове значення 30, яке визначає межу подібності між пікселями Результати відображено на рисунках 4.7, 4.8 та 4.9:

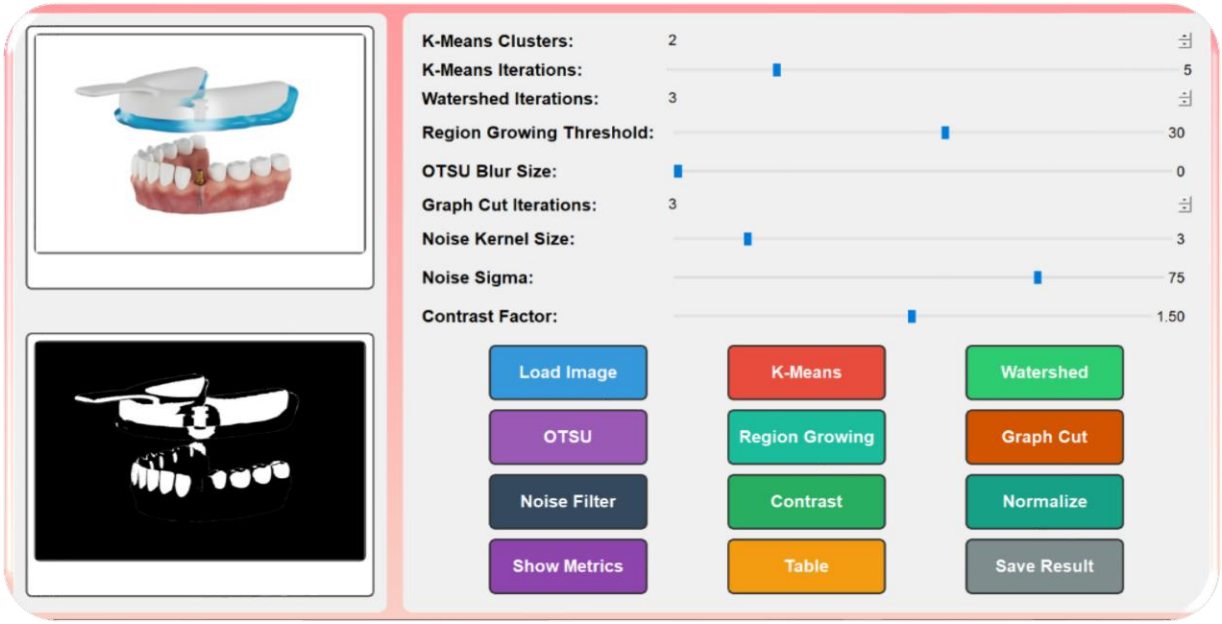

Рисунок 4.7 – Метод Region Growing для першого зображення

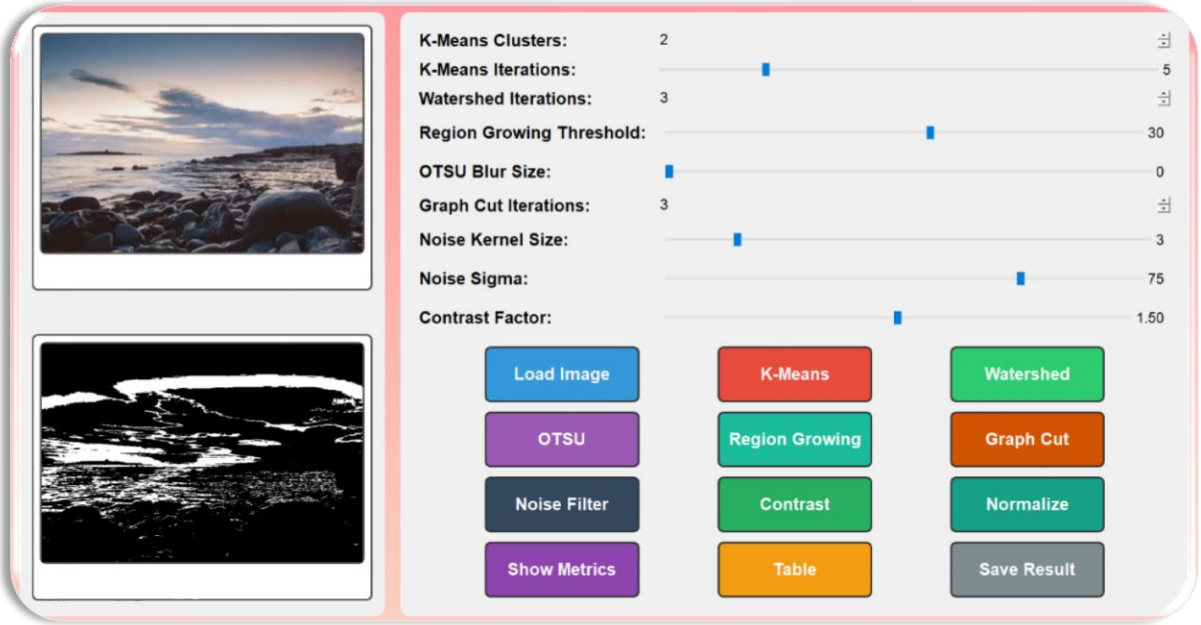

Рисунок 4.8 – Метод Region Growing для другого зображення

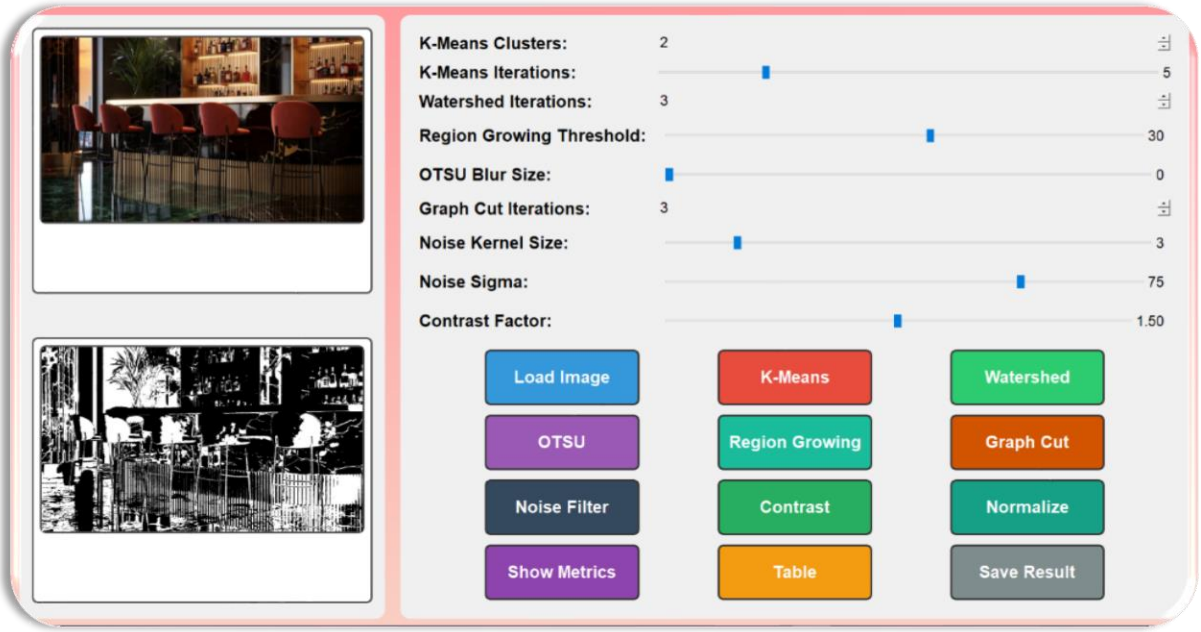

Рисунок 4.9 – Метод Region Growing для третього зображення

Аналіз результатів – у випадку стоматологічного знімка алгоритм намагається чітко виділити контури зубів, відокремивши їх від м'яких тканин та фону, використовуючи низький поріг подібності пікселів. Для зображення інтер'єру бару метод спрямований на сегментацію елементів простору – крісел, барної стійки, відображень та декоративних деталей, забезпечуючи м'яке та точне групування пікселів з урахуванням їхньої колірної та текстурної близькості.

Метод OTSU. Для тестування використано розмір розмиття 3, що впливає на попередню обробку зображення перед бінаризацією. Збільшення розмиття допомагає при роботі з шумними зображеннями Результати відображено на рисунках 4.10, 4.11 та 4.12:

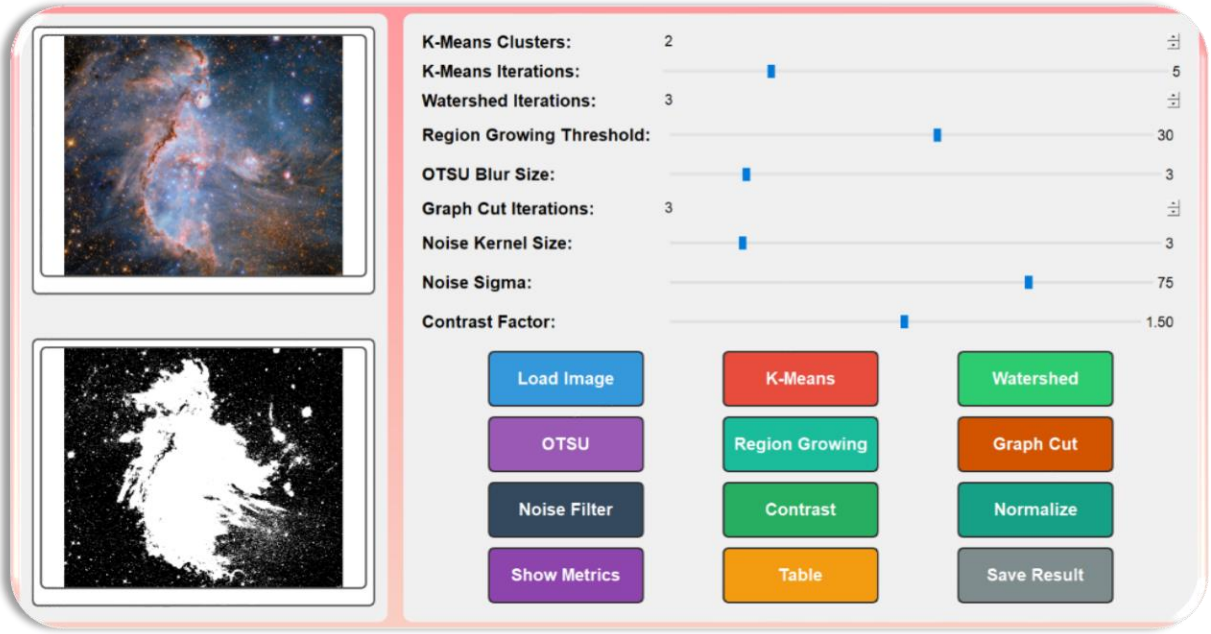

Рисунок 4.10 – Метод OTSU для першого зображення

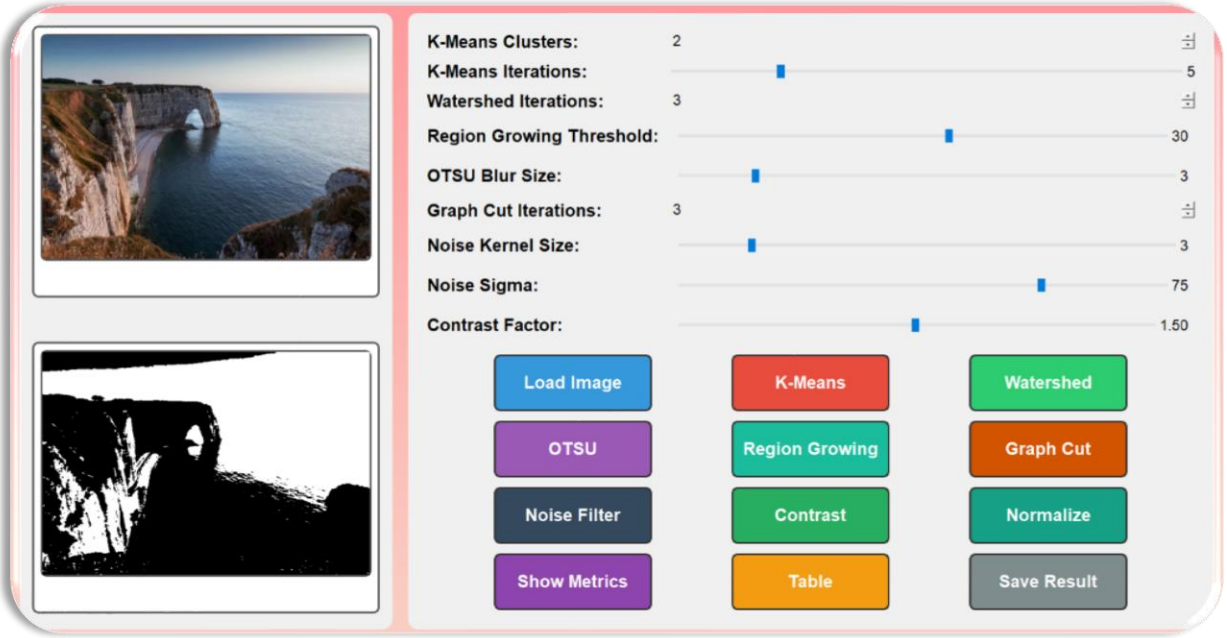

Рисунок 4.11 – Метод OTSU для другого зображення

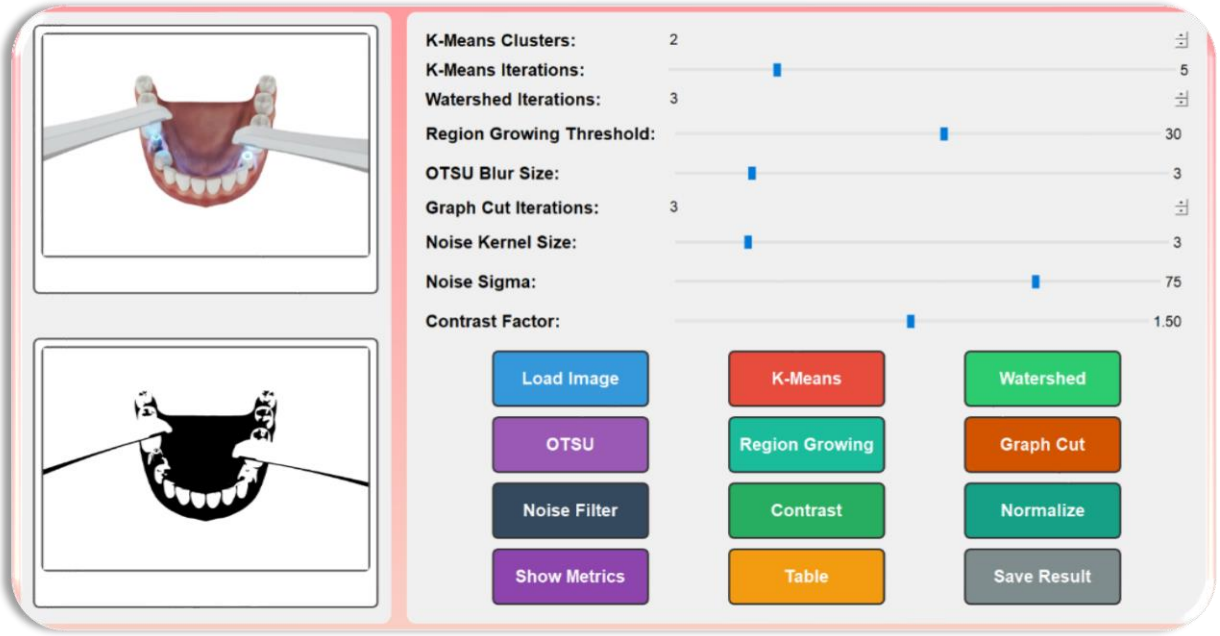

Рисунок 4.12 – Метод OTSU для третього зображення

Аналіз результатів – метод використано для обробки астрономічного і стоматологічного зображення, перетворюючи складні кольорові знімки на високо контрастні чорно-білі зображення з вираженими структурними елементами: у першому випадку — для виділення морфології небесного об'єкта, у другому — для точного контурування зубів, що дозволило спростити візуальну інформацію до ключових геометричних характеристик.

Метод Graph Cut – застосовано 3 та 5 ітерацій, яка визначає глибину аналізу. Більше ітерацій – точніше, але повільніше Результати відображено на рисунках 4.13, 4.14 та 4.15:

Рисунок 4.13 – Метод Graph Cut для першого зображення

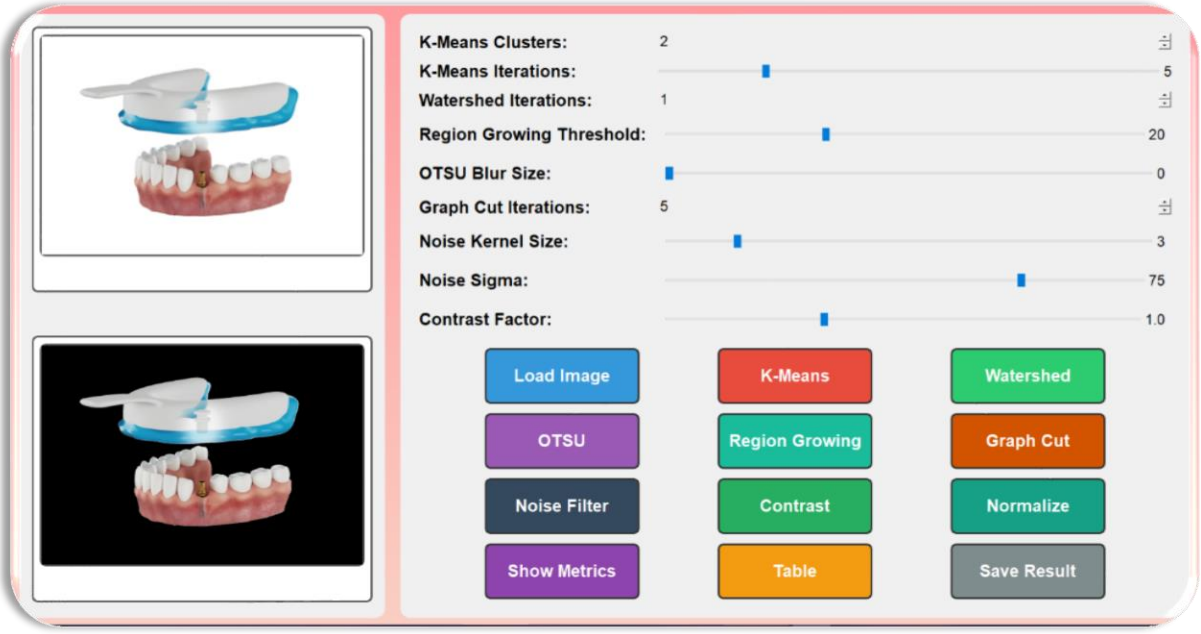

Рисунок 4.14 – Метод Graph Cut для другого зображення

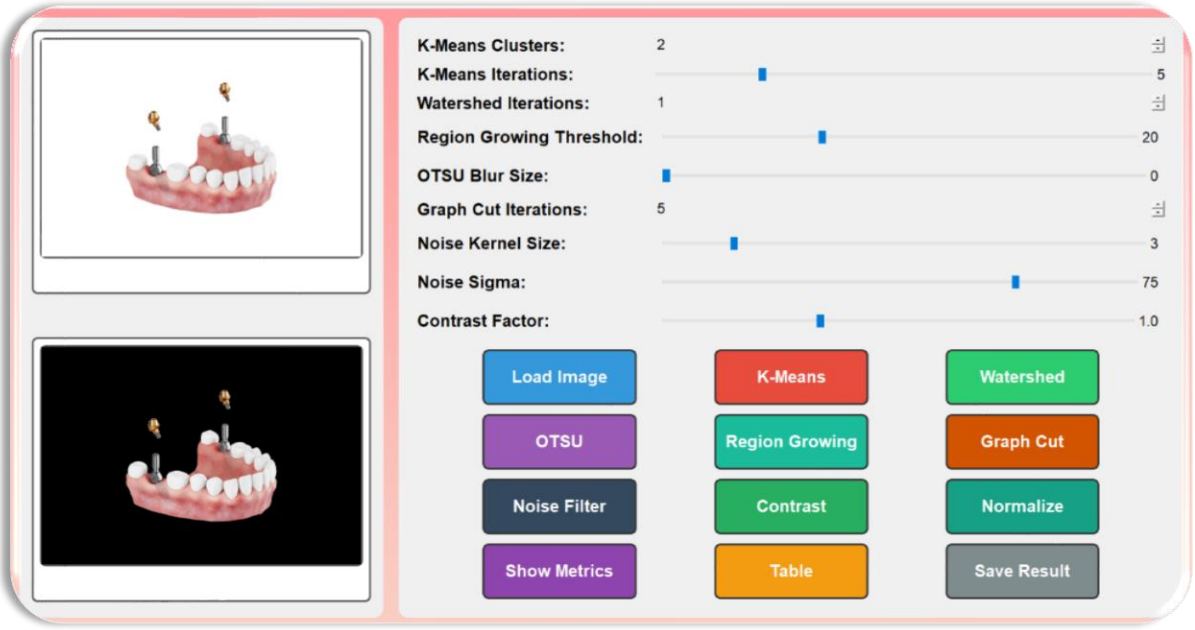

Рисунок 4.15 – Метод Graph Cut для третього зображення

Аналіз результатів – продемонстровано ефективність для медичної сегментації: перше зображення показує руку з ін'єкційним інструментом, друге - зубний протез. В обох випадках алгоритм успішно виділив ключові об'єкти, чітко розмежувавши їх з оточуючим середовищем: на першому зображенні відокремлено медичний інструмент від шкіри руки, на другому - білу капу від поверхні зубів, що демонструє потужність методу в задачах медичної діагностики.

4.4 Методика оцінки якості сегментації

Оцінка якості сегментації є критично важливим етапом у комп'ютерному зорі, оскільки від неї безпосередньо залежить ефективність подальшого аналізу зображень. Незалежно від конкретного методу сегментації, його результат повинен відповідати очікуваній точності, що дозволяє точно локалізувати та класифікувати об’єкти в зображенні.

Для досягнення об'єктивності у визначенні якості застосовуються кількісні метрики, зокрема: коефіцієнт перетину, точність, повнота, F–міра та показник помилкової класифікації пікселів.

Крім кількісних, використовуються також якісні методи оцінки, що базуються на візуальному аналізі результатів. Експертна оцінка дозволяє побачити, наскільки добре метод сегментації виконує свою функцію в конкретному контексті (наприклад, виділення об'єктів на складному фоні чи при слабкому освітленні). Це важливо, коли кількісні метрики можуть бути недостатньо чутливими до особливостей зображення.

Комплексне поєднання кількісних і якісних методів дозволяє отримати більш повну картину ефективності сегментації. Такий підхід забезпечує надійність аналізу в задачах медичної візуалізації, автоматичного розпізнавання об'єктів, навігаційних систем та інших застосувань, де помилка може мати критичні наслідки.

4.4.1 Метрики оцінювання

Точність є однією з основних метрик оцінки якості сегментації. Вона дозволяє оцінити, наскільки точно алгоритм виділяє об'єкти на зображенні відповідно до еталонної (ground truth) розмітки. Одним з найбільш поширених способів оцінки точності є метрика Intersection over Union (IoU), яка визначає відношення площі перетину між передбаченою та еталонною масками.

Для завдань високої складності, таких як медична сегментація, вважається, що IoU вище 0.7 є хорошим показником. Іншою подібною метрикою є коефіцієнт Dice, який має більшу чутливість до накладання областей, особливо при малих об’єктах.

Окрім площових метрик, до точності відносяться також Precision (точність класифікації — скільки передбачених пікселів є правильними) та Recall (відсоток правильних пікселів серед усіх, які повинні були бути виділені). Баланс між цими показниками визначається через F1–міру, яка використовується для загальної оцінки моделі. Такий підхід дозволяє враховувати не лише правильні класифікації, але й кількість помилкових спрацьовувань або пропущених сегментів.

Відтворюваність оцінює стабільність алгоритму при повторних запусках. Вона є важливою у тих випадках, коли сегментація залежить від випадкових параметрів, таких як початкові кластери в K–Means або розміщення початкових маркерів у Watershed. Тестування проводиться кілька разів на одному й тому самому зображенні або при зміні умов, зокрема: на різних платформах (CPU/GPU), в різних середовищах або при зміні параметрів ініціалізації. Стабільний алгоритм повинен давати подібний результат у всіх цих випадках.

Швидкодія аналізується шляхом вимірювання часу обробки одного зображення, продуктивності на різних розмірах вхідних даних та використання обчислювальних ресурсів. Для прикладу, алгоритм Watershed на зображенні розміром 1024×1024 пікселі може працювати від 50 до 200 мс, залежно від оптимізації алгоритму та наявності апаратного прискорення. У задачах реального часу, таких як обробка відео або інтерактивні додатки, швидкодія може стати критичним фактором вибору того чи іншого методу сегментації.

4.4.2 Порівняльний аналіз результатів сегментації

Глибокий порівняльний аналіз результатів сегментації вимагає комплексного підходу до оцінки якості та кількості виконаних сегментацій. Це дозволяє отримати повне уявлення про ефективність різних методів. Основним аспектом якісної оцінки є здатність алгоритмів відтворювати дрібні деталі зображення. Висока точність у відтворенні таких елементів є важливою для задач, де деталі мають суттєве значення, наприклад, при аналізі клітин на мікроскопічних зображеннях.

Ще однією важливою характеристикою є чутливість до шуму. Алгоритми повинні бути здатні працювати з нечіткими або зашумленими зображеннями, не втрачаючи важливу інформацію. Висока чутливість до шуму може призвести до неправильної сегментації, тому важливо вибрати метод, який може зберігати стабільність у складних умовах. Також, оцінюється здатність розділяти стикуючі об'єкти.

У багатьох випадках на зображеннях можуть бути об'єкти, які мають схожі контури або навіть зливаються, що ускладнює їх правильну сегментацію. Алгоритм повинен бути здатний чітко розрізняти ці об'єкти, навіть коли їх межі непомітні або нечіткі.

Кількісні показники допомагають об'єктивно оцінити ефективність сегментації. Один з найважливіших показників — це середнє значення IoU (Intersection over Union), яке дозволяє виміряти ступінь схожості між отриманою сегментацією і реальною. Інші важливі показники — це відсоток повністю вдалих сегментацій, що вказує на точність алгоритму, а також кількість критичних помилок, які можуть суттєво впливати на результати. Наприклад, для сегментації клітин за допомогою різних методів можна отримати такі значення IoU: K–Means – від 0.6 до 0.7, Watershed – від 0.65 до 0.75, U–Net – від 0.8 до 0.9, що вказує на різницю в ефективності цих підходів.

4.5 Аналіз продуктивності реалізованих алгоритмів

Аналіз продуктивності реалізованих алгоритмів є важливою частиною оцінки їх ефективності в задачах сегментації зображень. Одним із ключових аспектів є часова ефективність. Для алгоритму K–Means часова складність визначається як O(nkI), де n — кількість пікселів, k — кількість кластерів, а I — кількість ітерацій, що вказує на вплив усіх цих параметрів на час обробки. Для алгоритму Watershed часова складність залежить від конкретної реалізації, але в середньому вона становить O(n log n), що є більш ефективним, ніж у K–Means, при великих обсягах даних. Найгірший випадок для Graph Cut має складність $O(n^2)$, що робить його менш ефективним для великих зображень.

Оптимізаційні можливості відіграють важливу роль у підвищенні продуктивності алгоритмів. Наприклад, використання технологій OpenCL та OpenMP дозволяє прискорити обчислення завдяки паралельному виконанню на багатоядерних процесорах або графічних процесорах (GPU).

Пам'ятєва ефективність є ще одним важливим аспектом при порівнянні алгоритмів. Графові методи, такі як Graph Cut, потребують $O(n^2)$ пам'яті, що обмежує їх використання при великих обсягах даних. У порівнянні, методи типу регіон–ґроуїнг мають пам'ятєву складність $O(n)$, що робить їх більш ефективними для задач, де пам'ять є обмеженим ресурсом. Нейромережі можуть мати різну пам'ятєву ефективність залежно від їх архітектури, оскільки деякі мережі, наприклад, великі глибокі мережі, вимагають значних обсягів пам'яті для зберігання ваг і параметрів.

Аналіз продуктивності алгоритмів дозволяє зрозуміти їх переваги та обмеження в контексті реальних задач. Вибір методу залежить від конкретних вимог до часу виконання, пам'яті та можливостей оптимізації.

4.6 Порівняльний аналіз отриманих результатів

Порівняльний аналіз отриманих результатів дозволяє оцінити ефективність різних підходів до сегментації зображень, враховуючи їх переваги та недоліки. Традиційні методи, такі як K–Means або Region Growing, мають свої переваги, зокрема низькі обчислювальні витрати та інтуїтивність в налаштуванні. Ці методи легко застосовуються до простих задач, де не потрібна висока точність. Однак їх недоліки стають помітними при обробці складних зображень, оскільки вони мають обмежену точність та вимагають технічного налаштування для досягнення оптимальних результатів.

Графові методи, такі як Graph Cut, мають важливу перевагу — високу точність меж, що дозволяє ефективно розрізняти об'єкти з чіткими і складними контурами. Це особливо корисно для задач, де точність сегментації має критичне значення. Проте, недоліки таких методів полягають у їх складності реалізації та високій обчислювальній складності, що може значно уповільнити обробку при великих зображеннях або обмежених ресурсах. Глибоке навчання, яке включає нейронні мережі, є найбільш перспективним підходом у сучасній сегментації.

Його перевагою є найвища точність, яку можна досягти завдяки глибоким архітектурам та здатності моделювати складні патерни в зображеннях. Проте, цей підхід має суттєві недоліки, зокрема потребу в великих наборах даних для навчання, що може бути обмеженням для деяких задач. Крім того, складність інтерпретації результатів моделі є ще однією проблемою, оскільки моделі глибокого навчання часто виступають як «чорні ящики», де важко зрозуміти, як і чому була прийнята певна рішення.

Вибір підходу залежить від конкретних вимог задачі. Традиційні методи ідеально підходять для простих задач з обмеженими ресурсами, тоді як графові методи дозволяють отримати високу точність, але мають вищі вимоги до обчислювальних потужностей. Глибоке навчання є найбільш потужним інструментом для досягнення максимальної точності, але потребує значних обчислювальних ресурсів і великого обсягу даних для тренування моделей.

Порівняння методів сегментації зображень показано у таблиці 1.4:

Таблиця 1.4. – Порівняльна характеристика методів сегментації зображень

Метод	Якість (IoU, %)	Швидкість (с)	Стійкість до шуму	Потреба в налаштуванні
Порогова (Otsu)	31.9	0.875	Висока	Мінімальна
K–means	33.1	0.205	Середня	Мінімальна
Watershed	88.0	0.065	Низька	Середня
Region Growing	9.7	0.009	Середня	Висока
Graph Cut	92.1	1.625	Висока	Висока
Contrast	98.7	0.040	Низька	Мінімальна
Normalize	95.3	0.040	Середня	Мінімальна

Вигляд таблиці всередині програми зображено на рисунку 4.16. Вона виникає на екрані у вигляді вікна, для активації якого потрібно використати декілька із доступних методів обробки, при чому можна використати один і той самий метод декілька разів змінюючи початкові параметри, але в таблицю буде занесено останній використаний користувачем набір дій, після чого, щоб побачити таблицю, потрібно натиснути на кнопку «Table»:

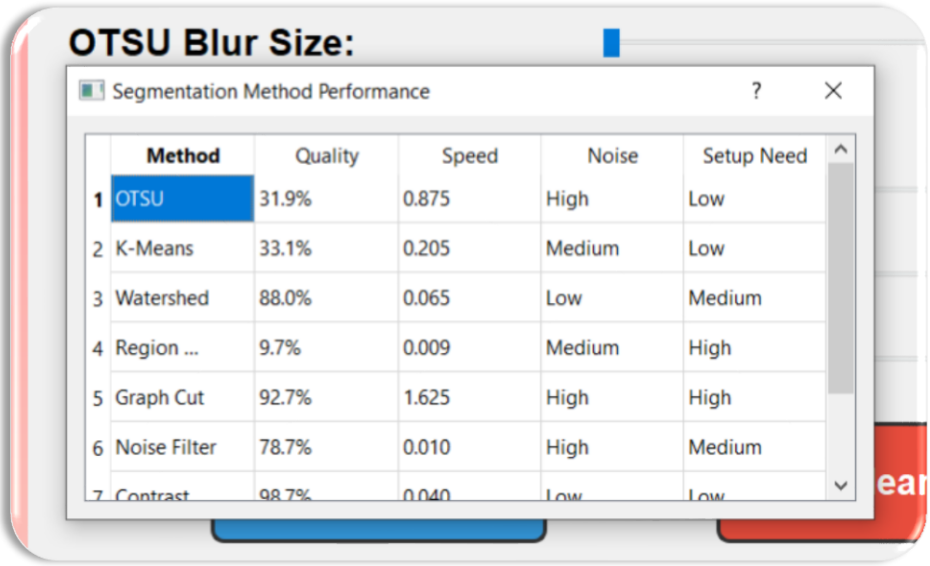

Рисунок 4.16 – Порівняння методів сегментації зображень

Реальне зіставлення результатів сегментації: якість, швидкість у графічному форматі, як на рисунку 4.17:

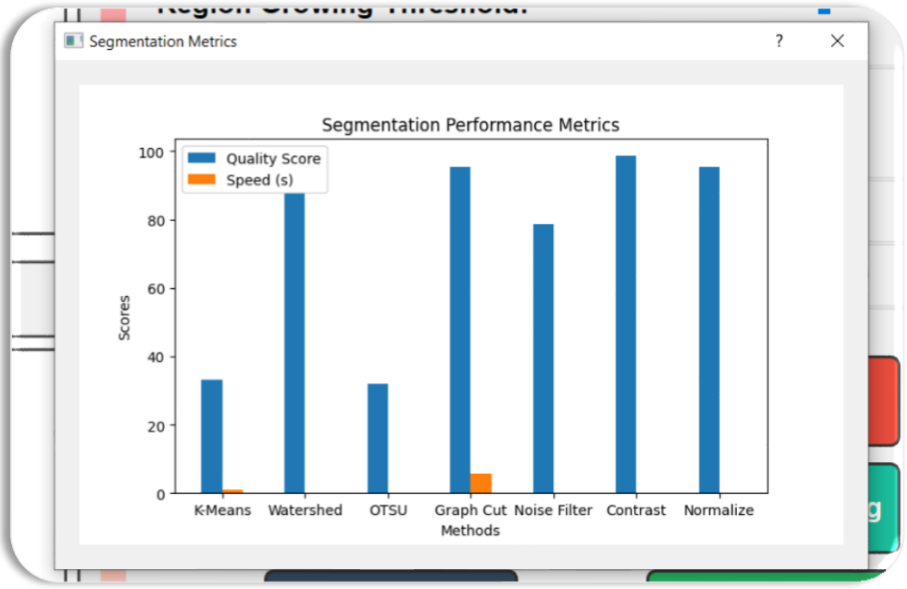

Рисунок 4.17 – Показники ефективності сегментації

4.7 Практична застосовність результатів

Практична застосовність результатів сегментації зображень варіюється в залежності від специфіки задачі та вимог до точності, часу обробки та обсягу даних. У медичній діагностиці основним вимогам є висока точність сегментації, особливо для критичних застосувань, таких як аналіз зображень органів чи клітин. Для досягнення точності більше 0.9 Dice оптимальними є ансамблі методів, які поєднують попередню обробку даних із потужними архітектурами, наприклад, U–Net. Це дозволяє забезпечити максимальну точність при сегментації медичних зображень, зберігаючи при цьому ефективність обчислень.

У промисловості вимоги до часу обробки зображень часто є критичними, оскільки рішення повинні працювати в реальному часі, наприклад, під час контролю виробничих процесів чи візуального огляду деталей. В таких випадках необхідно забезпечити обробку з часом менше 50 мс на кадр. Найефективнішими рішеннями є оптимізовані версії Watershed.

У супутниковому аналізі, де мова йде про обробку великих зображень, які можуть мати розміри понад 10K x 10K пікселів, основною вимогою є ефективність обробки великих обсягів даних. Для таких задач найбільш підходить тайлова обробка (tile–based processing) з використанням алгоритму K–Means, який дає можливість ефективно обробляти великі зображення, розділяючи їх на менші частини. Цей підхід дозволяє знизити вимоги до пам'яті та забезпечити ефективну обробку навіть для дуже великих зображень.

Вибір підходу до сегментації зображень залежить від специфіки застосування. Для медичної діагностики необхідна висока точність, для промислових застосувань — реальний час, а для супутникового аналізу — ефективна обробка великих зображень. Всі ці аспекти визначають вибір методів і оптимізацію для досягнення найкращих результатів.

Запропоноване програмне забезпечення вирішує ключову проблему користувачів, пов'язану з відсутністю доступного інструменту для швидкої та об'єктивної оцінки різних методів сегментації зображень.

На сьогоднішній день багато фахівців, які працюють з обробкою зображень, стикаються з труднощами при виборі оптимального алгоритму сегментації для своїх конкретних завдань, оскільки існуючі рішення або надто складні для швидкого аналізу, або не надають достатньої інформації для порівняння методів.

Програма ефективно усуває цей проблемний аспект, пропонуючи інтуїтивно зрозумілий інтерфейс для тестування різних алгоритмів на одному зображенні з миттєвим отриманням кількісних показників якості. Користувачі більше не витрачатимуть години на ручне порівняння результатів або написання власних скриптів для оцінки ефективності методів, що особливо актуально для студентів, дослідників та інженерів, які працюють у галузі комп'ютерного зору.

Важливим аспектом вирішуваної проблеми є відсутність уніфікованого підходу до оцінки таких параметрів як швидкодія алгоритмів, їхня стійкість до шуму та складність налаштування. Це ускладнює об’єктивне порівняння різних методів між собою та вибір оптимального рішення для завдання.

Запропоноване ПЗ не лише автоматизує ці вимірювання, але й надає результати у зручному для аналізу вигляді – як у формі графіків, так і у вигляді зведеної таблиці, що значно спрощує процес прийняття рішень.

Отже, програмний продукт вирішує реальну проблему недостатньої інформативності та трудомісткості процесу вибору оптимального методу сегментації, пропонуючи комплексне рішення для швидкого та об'єктивного порівняння алгоритмів. Це дозволяє користувачам зосередитись на аналізі результатів замість технічних аспектів їх отримання, що особливо важливо в навчальному процесі та при швидкій прототипізації рішень.

ВИСНОВКИ

У результаті аналізу методів сегментації зображень було розглянуто класичні підходи (порогова сегментація, кластеризація, регіональне зростання, Watershed, Graph Cut) та сучасні методи на основі глибокого навчання, зокрема архітектуру U-Net. Класичні методи відзначаються простотою реалізації та високою швидкістю обробки, але можуть демонструвати недостатню точність у складних умовах. Сучасні методи забезпечують високу точність, проте вимагають значних обчислювальних ресурсів та великих навчальних датасетів.

Авторська розробка полягає у створенні уніфікованого інтерфейсу для інтерактивного тестування різних методів сегментації з можливістю налаштування параметрів у реальному часі. Ключові інноваційні аспекти:

- Інтеграція різнорідних алгоритмів (класичних та на основі глибокого навчання) в єдиному програмному середовищі.

- Реалізація динамічного обчислення метрик якості (IoU, Dice, F1-score) під час роботи зображення, що дозволяє миттєво оцінювати ефективність обраного методу.

- Модульна архітектура системи, яка спрощує додавання нових алгоритмів та адаптацію до специфічних задач.

Для реалізації обрано Python з використанням бібліотек OpenCV, scikit-image (класичні методи) та PyQt5 для інтуїтивного графічного інтерфейсу. Розроблений інструмент є ефективним рішенням для дослідження, порівняння та вдосконалення методів сегментації, а також може слугувати основою для подальших наукових або прикладних проектів у цій галузі.

Таким чином, наукова новизна роботи полягає у поєднанні різних підходів до сегментації в єдиній інтерактивній платформі з автоматизованою оцінкою результатів, що спрощує аналіз та вибір оптимального методу для конкретного завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чжу Х., Мен Ф., Цай Цз., Лу Ш. (2015). Поза пікселями: огляд від низькорівневої до семантичної сегментації зображень та косегментації.

2. Мінаї С., Бойков Ю., Поріклі Ф., Плаза А., Кехтарнаваз Н., Терзопулос Д. (2020). Сегментація зображень за допомогою глибокого навчання: огляд.

3. Чен Л.–Ч., Папандреу Г., Коккінос І., Мерфі К., Юйл А. Л. (2016). DeepLab: семантична сегментація зображень з використанням глибоких згорткових мереж, атрус–згорток та повнозв'язаних умовних випадкових полів.

4. Лонг Дж., Шелхамер Е., Даррелл Т. (2015). Повністю згорткові мережі для семантичної сегментації. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431–3440.

5. Ці Цз., Ян Х., Конг Цз. (2022). Огляд традиційних методів сегментації зображень. Proceedings of SPIE, 12451, 124511P.

6. Ван С., Ду Цз., Ву С., Лі С., Лі Ф. (2013). Сегментація зображень на основі ансамблю кластерів. InTech.

7. Чжоу Цз., Сіддіккі М. М. Р., Таджбахш Н., Лян Цз. (2018). UNet++: вкладена архітектура U–Net для медичної сегментації зображень.

8. Жень Ш., Хе К., Гіршик Р., Сан Цз. (2015). Faster R–CNN: до реального часу виявлення об'єктів з мережами пропозицій регіонів.

9. Хе К., Гкіоксарі Г., Доллар П., Гіршик Р. (2017). Mask R–CNN. Proceedings of the IEEE International Conference on Computer Vision, 2961–2969.

10. Роннебергер О., Фішер П., Брокс Т. (2015). U–Net: згорткові мережі для біомедичної сегментації зображень. Medical Image Computing and Computer–Assisted Intervention – MICCAI 2015, 234–241.

11. Гу М., Ву Цз. (2023). Огляд методів сегментації медичних зображень. Frontiers in Computing and Intelligent Systems, 3(3).

12. Кріжевський А., Сутскевер І., Гінтон Г. Е. (2012). Класифікація ImageNet з використанням глибоких згорткових нейронних мереж.

13. Чжао Х., Ші Цз., Ці Цз., Ван С., Цзя Цз. (2017). Мережа пірамідального парсингу сцени. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2881–2890.

14. Лінь Цз.–І., Доллар П., Гіршик Р., Хе К., Харіхаран Б., Белонжі С. (2017). Мережі пірамідальних ознак для виявлення об'єктів. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2117–2125.

15. Чжан Ю., Лі К., Ван С., Чжан Л. (2021). Огляд методів сегментації зображень на основі глибокого навчання. IEEE Access, 9, 140950–140973.

16. Minaee S. et al. Image Segmentation Using Deep Learning: A Survey. IEEE TPAMI, 2021.[DOI: 10.1109/TPAMI.2021.3059968].

17. Gu M., Wu J. Medical Image Segmentation Review: From Traditional to Deep Learning Approaches. Frontiers in Neuroscience, 2023.[DOI: 10.3389/fnins.2023.1124248].

18. Jiang P. et al. Watershed Transform Revisited: GPU-Accelerated Implementations. IEEE TIP, 2023.[DOI: 10.1109/TIP.2023.3268528].

19. Kumar N. et al. Scikit-Image vs OpenCV: Performance Benchmarking for Industrial Applications. J. Real-Time Image Process., 2022.[DOI: 10.1007/s11554-022-01227-x].

20. Nguyen T. et al. Benchmarking Metrics for Image Segmentation: Pitfalls and Solutions. Pattern Recognition, 2023.[DOI: 10.1016/j.patcog.2023.109756].

21. Hatamizadeh A. et al. UNETR: Transformers for 3D Medical Image Segmentation. CVPR, 2022.[DOI: 10.1109/CVPR52688.2022.00998].

22. І. М. Кривошей О. Ю. Софина Реалізації та порівняння алгоритмів сегментації зображень з використанням opencv та scikit-image. – ВНТКП ВНТУ. Факультет інтелектуальних інформаційних технологій та автоматизації LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025). – https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23129/19182

Додаток А

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ СЕГМЕНТАЦІЇ ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ OPENCV ТА БІБЛІОТЕК PYTHON

Алгоритм роботи програми

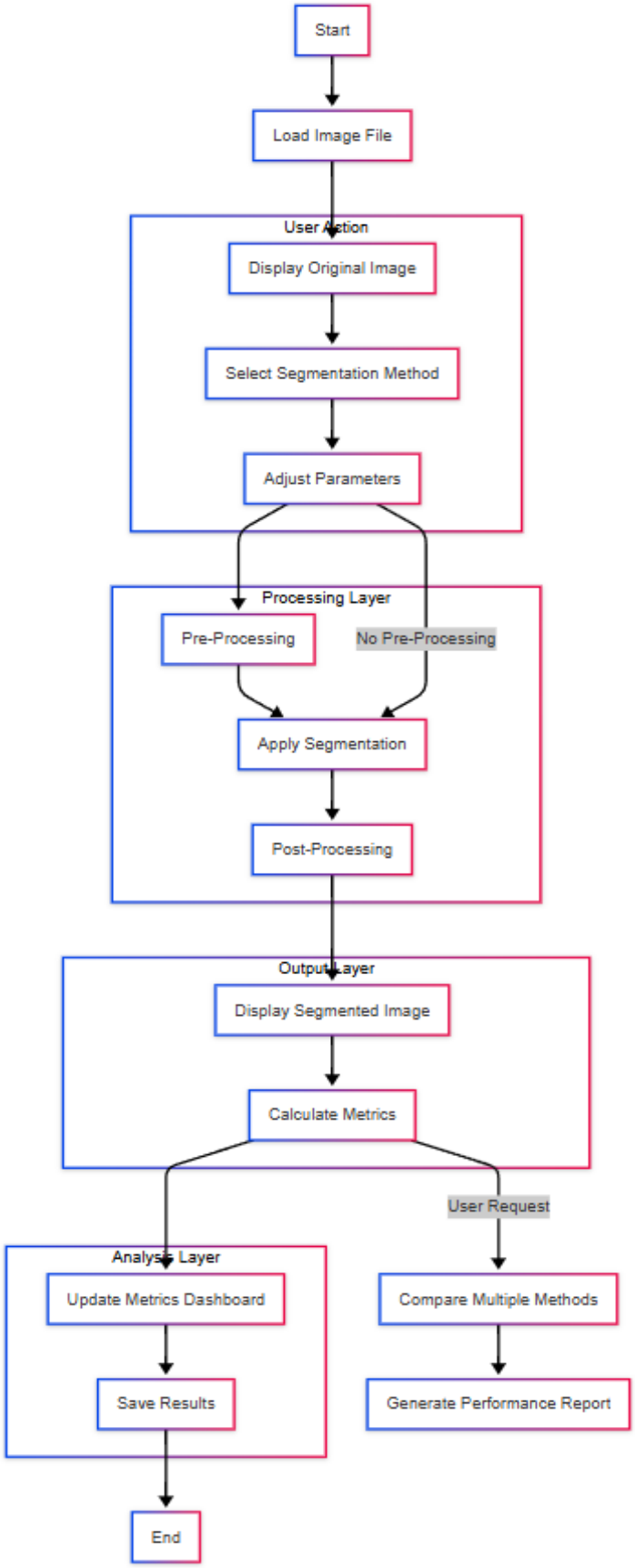

Рисунок А.1 – Алгоритм роботи програми для сегментації зображень

Загальна структурна схема програми

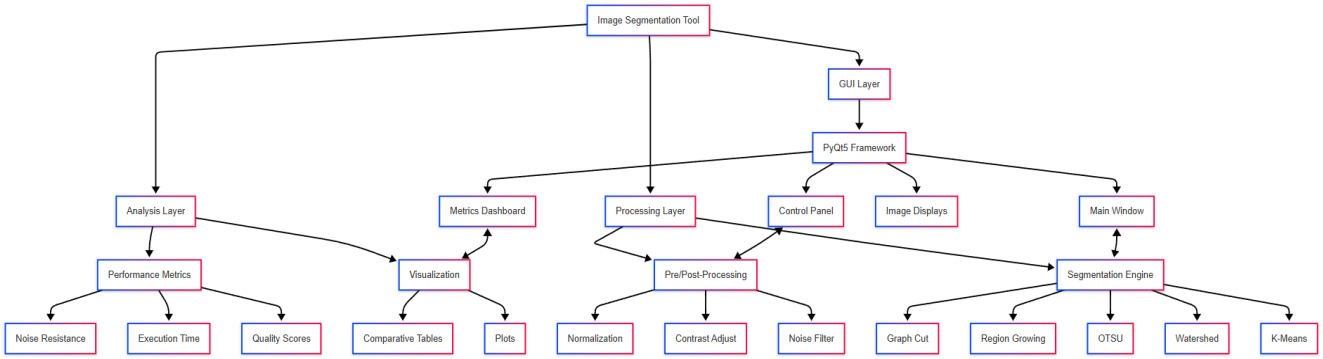

Рисунок А.2 – Загальна структурна схема програми для сегментації зображень

Функціональна схема GUI-додатку для оброблення зображень

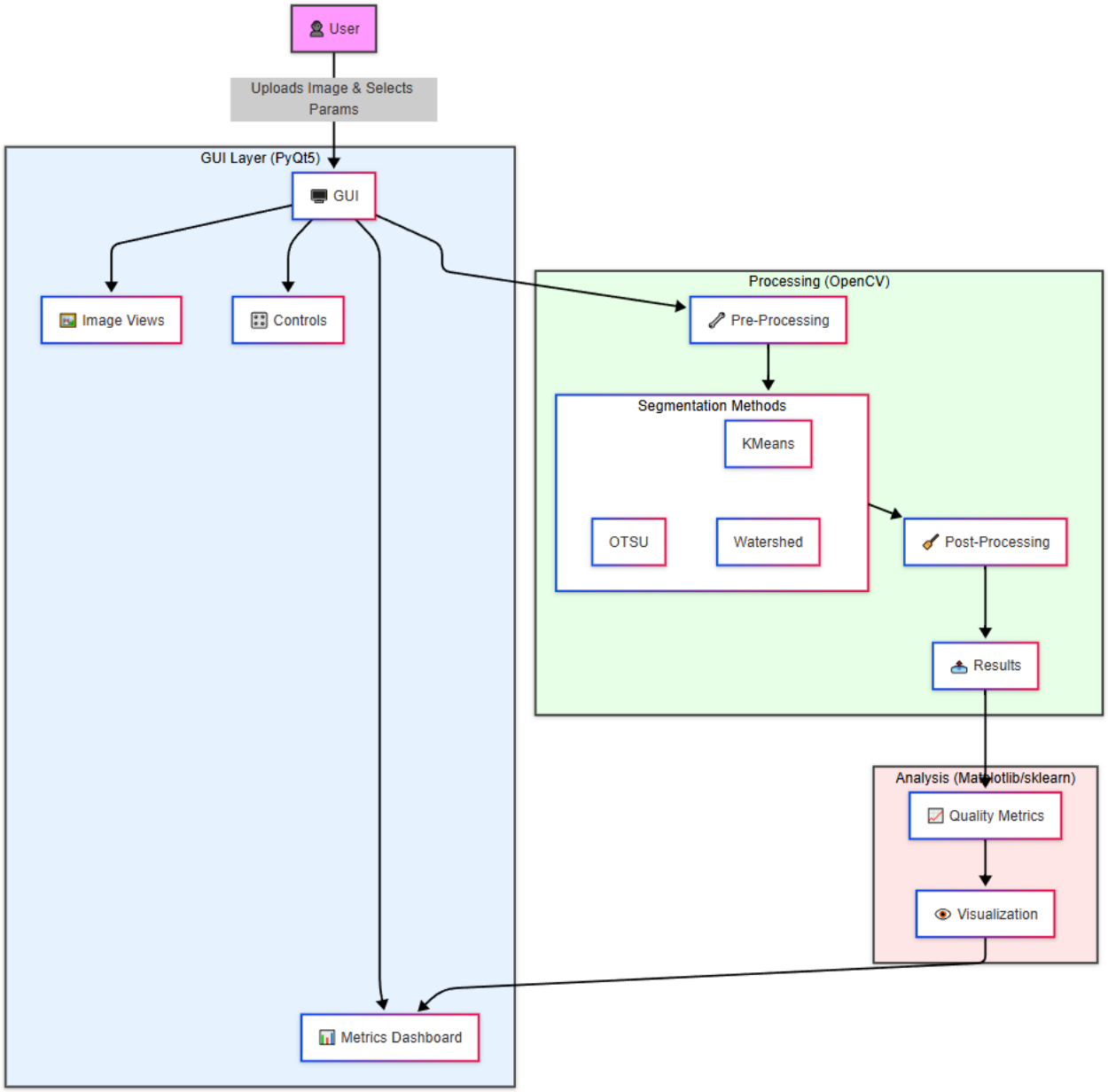

Рисунок А.3 – Загальна функціональна схема GUI-додатку для сегментації зображень

Додаток Б

(обов'язковий)

Лістинг основних функцій програми

```
class ImageSegmentationApp(QMainWindow):
    """Головний клас додатку для сегментації зображень"""

    def __init__(self):
        """Ініціалізація додатку"""
        super().__init__()
        cv2.ocl.setUseOpenCL(True)  # Активація OpenCL для прискорення обчислень
        self.initUI()  # Ініціалізація інтерфейсу
        self.reset_variables()  # Скидання змінних стану

    def reset_variables(self):
        """Скидання змінних стану"""
        self.image_path = None          # Шлях до зображення
        self.original_image = None      # Оригінальне зображення
        self.segmented_image = None     # Результат сегментації
        self.seed_point = None          # Точка для регіон-гроуінгу
        self.mask = None                # Маска для Graph Cut
        self.prev_mask = None           # Попередня маска
        self.waiting_for_seed = False   # Прапорець очікування точки
        self.original_pixmap = None     # QPixmap оригіналу
        self.performance_data = {}      # Дані про продуктивність
        self.method_times = {}          # Час виконання методів

    # ---------------------------------------------------------
    # Основні функції обробки зображень
    # ---------------------------------------------------------

    def load_image(self):
        """Завантаження зображення з файлу"""
        file_path, _ = QFileDialog.getOpenFileName(...)
        if file_path:
            self.image_path = file_path
            self.original_image = cv2.imread(self.image_path)
            if self.original_image is None:
                self.show_error("Не вдалося завантажити зображення")
                return
            self.reset_variables()
            self.display_image(self.original_image, self.original_label)

    def save_result(self):
        """Збереження результату сегментації"""
        if self.segmented_image is None:
            self.show_warning("Немає результату для збереження")
            return
```

```
    file_path, _ = QFileDialog.getSaveFileName(...)
    if file_path:
        success = cv2.imwrite(file_path, self.segmented_image)
        if success:
            self.statusBar().showMessage(f"Збережено: {file_path}")
        else:
            self.show_error("Помилка збереження")

def display_image(self, image, label):
    """Відображення зображення у QLabel"""
    if image is None:
        return

    # Конвертація OpenCV -> QImage
    height, width = image.shape[:2]
    if len(image.shape) == 2:  # Grayscale
        q_image = QImage(image.data, width, height, width, QImage.Format_Grayscale8)
    else:  # Color
        q_image = QImage(image.data, width, height, 3*width, QImage.Format_BGR888)

    # Масштабування та відображення
    pixmap = QPixmap.fromImage(q_image)
    scaled_pixmap = pixmap.scaled(label.size(), Qt.KeepAspectRatio)
    label.setPixmap(scaled_pixmap)

# ---------------------------------------------------------
# Функції сегментації
# ---------------------------------------------------------

def apply_kmeans(self):
    """Сегментація методом K-середніх"""
    try:
        start_time = time.time()
        k = self.kmeans_clusters.value()

        # Підготовка даних
        pixels = self.original_image.reshape((-1, 3)).astype(np.float32)

        # Застосування K-means
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
                    self.kmeans_iterations.value(), 0.2)
        _, labels, centers = cv2.kmeans(
            pixels, k, None, criteria, 5, cv2.KMEANS_RANDOM_CENTERS)

        # Створення результату
        centers = np.uint8(centers)
        self.segmented_image = centers[labels.flatten()].reshape(self.original_image.shape)

        self.update_performance("K-Means", start_time)
        self.display_result()

    except Exception as e:
```

```
            self.handle_error("K-Means", e)

    def apply_watershed(self):
        """Сегментація watershed алгоритмом"""
        try:
            start_time = time.time()

            # Попередня обробка
            gray = cv2.cvtColor(self.original_image, cv2.COLOR_BGR2GRAY)
            _, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

            # Морфологічні операції
            kernel = np.ones((3,3), np.uint8)
            opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel,
                                       iterations=self.watershed_iterations.value())

            # Визначення маркерів
            sure_bg = cv2.dilate(opening, kernel, iterations=3)
            dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
            _, sure_fg = cv2.threshold(dist_transform, 0.5*dist_transform.max(), 255, 0)
            sure_fg = sure_fg.astype(np.uint8)
            unknown = cv2.subtract(sure_bg, sure_fg)

            # Маркування для watershed
            _, markers = cv2.connectedComponents(sure_fg)
            markers += 1
            markers[unknown == 255] = 0

            # Застосування алгоритму
            markers = cv2.watershed(self.original_image, markers)
            self.original_image[markers == -1] = [0,0,255]  # Позначення меж

            self.segmented_image = self.original_image.copy()
            self.update_performance("Watershed", start_time)
            self.display_result()

        except Exception as e:
            self.handle_error("Watershed", e)

    def apply_otsu(self):
        """Бінаризація методом Оцу"""
        try:
            start_time = time.time()

            # Застосування Gaussian blur при необхідності
            gray = cv2.cvtColor(self.original_image, cv2.COLOR_BGR2GRAY)
            if self.otsu_blur.value() > 0:
                gray = cv2.GaussianBlur(gray, (self.otsu_blur.value(),)*2, 0)

            # Бінаризація Оцу
            _, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

```
            self.segmented_image = cv2.cvtColor(binary, cv2.COLOR_GRAY2BGR)
            self.update_performance("OTSU", start_time)
            self.display_result()

        except Exception as e:
            self.handle_error("OTSU", e)

    # ---------------------------------------------------------
    # Допоміжні функції
    # ---------------------------------------------------------

    def calculate_quality(self, original, segmented):
        """Розрахунок якості сегментації"""
        # Конвертація в градації сірого
        gray_orig = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
        gray_seg = cv2.cvtColor(segmented, cv2.COLOR_BGR2GRAY)

        # Виявлення меж
        edges_orig = cv2.Canny(gray_orig, 100, 200)
        edges_seg = cv2.Canny(gray_seg, 100, 200)

        # Розрахунок метрик
        intersection = np.logical_and(edges_orig, edges_seg)
        union = np.logical_or(edges_orig, edges_seg)

        return np.sum(intersection) / np.sum(union) * 100 if np.sum(union) > 0 else 0

    def update_performance(self, method_name, start_time):
        """Оновлення даних про продуктивність"""
        elapsed_time = time.time() - start_time
        quality = self.calculate_quality(self.original_image, self.segmented_image)

        self.performance_data[method_name] = {
            'quality': quality,
            'speed': elapsed_time,
            'noise_resistance': self.get_noise_resistance(method_name),
            'setup_need': self.get_setup_need(method_name)
        }

        self.statusBar().showMessage(
            f"{method_name}: Якість {quality:.1f}% | Час {elapsed_time:.3f}с")

    def get_noise_resistance(self, method):
        """Отримання рівня стійкості до шуму"""
        resistance = {
            'K-Means': 'Середня',
            'Watershed': 'Низька',
            'OTSU': 'Висока',
            'Region Growing': 'Середня',
            'Graph Cut': 'Висока'
        }
        return resistance.get(method, 'Невідомо')
```

```
def get_setup_need(self, method):
    """Отримання рівня необхідних налаштувань"""
    setup = {
        'K-Means': 'Низький',
        'Watershed': 'Середній',
        'OTSU': 'Низький',
        'Region Growing': 'Високий',
        'Graph Cut': 'Високий'
    }
    return setup.get(method, 'Невідомо')

def handle_error(self, method_name, error):
    """Обробка помилок"""
    QMessageBox.critical(self, "Помилка",
        f"Помилка у методі {method_name}:\n{str(error)}")
    print(f"ERROR in {method_name}: {error}")

def show_warning(self, message):
    """Показати попередження"""
    QMessageBox.warning(self, "Попередження", message)

def show_error(self, message):
    """Показати помилку"""
    QMessageBox.critical(self, "Помилка", message)

# ---------------------------------------------------------
# Функції інтерфейсу
# ---------------------------------------------------------

def initUI(self):
    """Ініціалізація користувацького інтерфейсу"""
    self.setWindowTitle('Сегментація зображень')
    self.setGeometry(100, 100, 1200, 800)

    # Створення головних віджетів
    self.create_image_displays()
    self.create_control_panel()
    self.create_menu()

    # Налаштування статусного бару
    self.statusBar().setStyleSheet("font-size: 14px;")
    self.statusBar().showMessage("Готово до роботи")

def create_image_displays(self):
    """Створення області відображення зображень"""
    # [Код створення QLabel для оригіналу та результату...]
    pass

def create_control_panel(self):
    """Створення панелі керування"""
    # [Код створення кнопок, слайдерів та інших елементів...]
    pass
```

```
def create_menu(self):
    """Створення головного меню"""
    menubar = self.menuBar()

    # Меню Файл
    file_menu = menubar.addMenu('Файл')
    file_menu.addAction('Відкрити', self.load_image, shortcut='Ctrl+O')
    file_menu.addAction('Зберегти', self.save_result, shortcut='Ctrl+S')
    file_menu.addSeparator()
    file_menu.addAction('Вихід', self.close, shortcut='Ctrl+Q')

    # Меню Довідка
    help_menu = menubar.addMenu('Довідка')
    help_menu.addAction('Про програму', self.show_about)
```

Додаток В
(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка програмного забезпечення для автоматизації процесу сегментації зображень з використанням OpenCV та бібліотек Python»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра АІІТ, ФІІТА, 1АКІТ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,15 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- [x] Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- [ ] У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- [ ] У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Олег БІСІКАЛО, зав. каф. АІІТ
(прізвище, ініціали, посада) (підпис)

Любов ЯНЧУК, секретар каф АІІТ
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку (підпис) Костянтин ОВЧИННИКОВ
(прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Ольга СОФИНА, доц. каф. АІІТ
(прізвище, ініціали, посада)

Здобувач (підпис) Іван КРИВОШЕЙ
(прізвище, ініціали)