Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.912

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: “Автоматизована система сегментації цифрових зображень МРТ головного мозку ”

Спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

БР.ІПЗ – __.__.__.___

Студентка

ІПЗ–44 _____/Олена КОЛЕСНИК/

Науковий керівник

к.ф.–м.н.,доц._____/Ірина ЮРЧУК/

Консультант

з питань нормоконтролю

фахівець_____/Тамара ЧАПОВСЬКА/

Допускається до захисту

Завідувач кафедри

д.т.н., проф. _____ /Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії
професор, доктор техн. наук Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО

Зав. кафедри програмних систем і технологій

_____(Олексій БИЧКОВ)

„_____”_____ 2021 р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
СТУДЕНТУ

Колесник Олені Сергіївні

(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної бакалаврської роботи “Автоматизована система сегментації цифрових зображень МРТ головного мозку”, керівник роботи доцент Юрчук І.А., затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від «11» листопада 2020р.

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до дипломної роботи: монографії, підручники, навчальні посібники, статті та тези конференцій вітчизняних і зарубіжних авторів, Інтернет–ресурси з питань сегментації цифрових зображень МРТ головного мозку.

4. Зміст пояснювальної записки:

1) Аналітична частина:

проаналізувати існуючі програмні рішення та засоби сегментації цифрових зображень МРТ головного мозку, обґрунтувати доцільність обраної теми дослідження, порівняти з вже існуючими рішеннями на українському та зарубіжному ринку.

2) Практична частина:

спроектувати архітектуру програмного забезпечення для сегментації цифрових зображень МРТ головного мозку, розробити програмне забезпечення, розробити інтерфейс користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1) Аналітична частина: 10 рисунків, 12 формул.
2) Практична частина: 20 рисунків, 1 таблиця, 5 листів додатку програмного коду, 6 листів додатку результуючої таблиці.

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1 розділ аналітична частина	Юрчук І.А.	17.01.2021	17.01.2021
2 розділ практична частина	Юрчук І.А.	17.01.2021	17.01.2021

7. Дата видачі завдання _____

Керівник _____ (Ірина ЮРЧУК)

Завдання прийняв до виконання _____ (Олена КОЛЕСНИК)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Уточнення постановки задачі	29.11.2020–05.12.2020	Виконано
2	Аналіз літератури	06.12.2020–04.01.2021	Виконано
3	Аналіз існуючих методів, концепцій та алгоритмів вирішення завдання	09.01.2021–16.01.2021	Виконано
4	Побудова алгоритмічної моделі основних процесів	28.01.2021–14.02.2021	Виконано
5	Опис розробленого алгоритму	15.02.2021–20.03.2021	Виконано
6	Розроблення програмного забезпечення	21.03.2021–30.04.2021	Виконано
7	Тестування розробленого програмного забезпечення	02.05.2021–15.05.2021	Виконано
8	Оформлення і друк пояснювальної записки	16.05.2021–26.05.2021	Виконано
9	Оформлення презентації	27.05.2021–03.06.2021	Виконано
10	Отримання рецензії		
11	Затвердження пояснювальної записки роботи завідувачем кафедри		
12	Захист дипломної роботи		

Студент _____ Олена КОЛЕСНИК
(підпис)

Керівник роботи _____ Ірина ЮРЧУК
(підпис)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 79 с., 30 рис., 1 табл., 9 додат., 12 джерел.

Тема: Автоматизована система сегментації цифрових зображень МРТ головного мозку.

Об'єкт дослідження: процес сегментації цифрових зображень.

Мета роботи: автоматизація процесу виявлення пухлини головного мозку на МРТ зображеннях за допомогою сегментації.

Предмет дослідження: метод ефективної сегментації МРТ головного мозку на основі водорозділу та порогового значення.

Результати дослідження:

Досліджено методи сегментації зображення та алгоритм сегментації на основі водорозділу та порогового значення. Реалізовано автоматизовану систему сегментації МРТ зображень головного мозку з користувацьким інтерфейсом. Запропоновано алгоритми удосконалення швидкості сегментації цифрових зображень та комбінування методів сегментації для покращення результатів сегментації.

Висновок

В результаті досліджень було отримано алгоритм, що застосовується для виділення пухлин на цифрових даних (МРТ); а також алгоритм для обробки масиву зображень головного мозку, що дозволяє спростити подальшу обробку великої кількості цифрових ресурсів і їх сегментацію зі збереженням у документ Microsoft Word, шляхом створення результуючого шаблону.

АЛГОРИТМ, ПОРОГ, ЗВ'ЯЗНІ КОМПОНЕНТИ, МОРФОЛОГІЧНІ ОПЕРАЦІЇ, ЕРОЗІЯ, ДИЛАТАЦІЯ, МАРКЕР, ВОДОДІЛ, АВТОМАТИЗОВАНА СИСТЕМА, ІНТЕРФЕЙС.

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 79 ст., 30 рис., 1 табл., 9 доп., 12 источников.

Тема: Автоматизированная система сегментации цифровых изображений МРТ головного мозга.

Объект исследования: процесс сегментации цифровых изображений.

Цель работы: автоматизация процесса выявления опухоли головного мозга на МРТ изображениях с помощью сегментации.

Предмет исследования: метод эффективной сегментации МРТ головного мозга на основе водораздела и порогового значения.

Результаты исследования:

Исследованы методы сегментации изображения и алгоритм сегментации на основе водораздела и порогового значения. Реализовано автоматизированную систему сегментации МРТ изображений головного мозга с пользовательским интерфейсом. Предложены алгоритмы усовершенствования скорости сегментации цифровых изображений и комбинирования методов сегментации для улучшения результатов сегментации.

Вывод

В результате исследований было получено алгоритм, применяемый для выделения опухолей на цифровых данных (МРТ); а также алгоритм для обработки массива изображений головного мозга, позволяющий упростить дальнейшую обработку большого количества цифровых ресурсов и их сегментацию с сохранением в документ Microsoft Word, путем создания результирующего шаблона.

АЛГОРИТМ, ПОРОГ, СВЯЗНЫЕ КОМПОНЕНТЫ, МОРФОЛОГИЧЕСКИЕ ОПЕРАЦИ, ЭРОЗИЯ, ДИЛАТАЦИЯ, МАРКЕР, ВОДОРАЗДЕЛ, АВТОМАТИЗИРОВАННАЯ СИСТЕМА, ИНТЕРФЕЙС.

ABSTRACT

Graduation qualifying bachelor's work: 79 p., 30 figs., 1 table, 9 appendices, 12 sources.

Theme: Automated system for segmentation of digital images of MRI of the brain.

Research object: digital image segmentation process.

The purpose of research is automation of the process of brain tumor detection on MRI images using segmentation.

Subject of research: method of effective segmentation of MRI of the brain based on watershed and threshold value.

Research results:

The methods of image segmentation and the segmentation algorithm based on the watershed and the threshold value have been investigated. An automated system for segmentation of MRI images of the brain with a user interface has been implemented. Algorithms for improving the speed of segmentation of digital images and combining segmentation methods to improve the segmentation results are proposed.

Conclusion

As a result of the research, an algorithm was obtained that is used to isolate tumors on digital data (MRI); as well as an algorithm for processing an array of brain images, which makes it possible to simplify the further processing of a large number of digital resources and their segmentation with saving in a Microsoft Word document by creating a resulting template.

ALGORITHM, THRESHOLD, CONNECTED COMPONENTS, MORPHOLOGICAL OPERATIONS, EROSION, DILATION, MARKER, WATERSHED, AUTOMATED SYSTEM, INTERFACE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ООП – об’єктно–орієнтоване програмування.

ОС – операційна система.

ПЗ – програмне забезпечення.

КТ – комп'ютерна томографія.

МРТ – магнітно–резонансна терапія.

МР – магнітно–резонансна.

ВСТУП

МРТ є одним з найбільш інформативних методів візуалізації, а для дослідження пухлин мозку має ряд переваг в порівнянні з іншими методами, зокрема, завдяки здатності отримувати тривимірні зображення з високою міжтканинною контрастністю.

Обсяг інформації про анатомічні структури головного мозку, представлений на МРТ-зображеннях, настільки великий, що візуальний аналіз, найбільш широко використовуваний в клінічній практиці, не дозволяє повністю використовувати отримані в ході дослідження дані, так як вони залишаються за межами можливостей людського ока і не враховуються.

Наявна кількість програмного забезпечення на ринку для сегментації цифрових зображень головного мозку обмежена, а саме ПЗ являє собою продукт, що важко налаштовується.

Актуальність теми: розпізнавання відхилень на зображеннях головного мозку комп'ютерною системою грає велике значення для полегшення роботи медичних закладів, адже спрощує процес виявлення непримітних ореолів пухлин на початковому етапі хвороби, коли людське око майже не здатне відрізнити невеликі відмінності у кольорах. Аналізуючи велику кількість даних (цифрових зображень) комп'ютер може з точністю до 99,9% визначити наявність пухлин головного мозку на цифрових зображення МРТ.

Метою роботи є автоматизація процесу виявлення пухлини головного мозку на МРТ зображеннях за допомогою сегментації.

Завдання:

1. Дослідити методи сегментації зображень МРТ головного мозку;
2. Розробити алгоритм сегментації цифрового зображення методом ефективної сегментації на основі водорозділу та порогового значення;
3. Створити автоматизовану систему на основі розробленого алгоритму;
4. Провести тестування автоматизованої системи на вибірці зображень;
5. Провести порівняльний аналіз з іншими методами сегментації.

Об’єкт дослідження – процес сегментації цифрових зображень.

Предметом дослідження є метод ефективної сегментації МРТ головного мозку на основі водорозділу та порогового значення.

Методи дослідження: Для розробки алгоритму були використані: пороговий метод сегментації Отцу, морфологічні операції (метод виділення зв'язних компонент), операції ерозії і дилатації, методи вододілу (з маркерами, шляхом дилатації). Для моделювання обробки проміжних даних сегментації було використано бібліотеку OpenCv. Для імплементації інтерфейсу користувача – PyQt5.

Новизна одержаних результатів. Удосконалено швидкість сегментації зображень (до декількох секунд), автоматизовано обробку масиву МРТ зображень, автоматизовано створення результуючої документації (у “.doc” файл).

Практичне значення одержаних результатів: Практична цінність отриманих результатів полягає в можливості виявлення пухлин головного мозку на перших етапах розвитку захворювання для полегшення та оптимізації роботи медичних працівників.

РОЗДІЛ 1
ОСНОВНІ ПОНЯТТЯ ТА МЕТОДИ СЕГМЕНТАЦІЇ ЦИФРОВИХ ЗОБРАЖЕНЬ

Сегментація зображення – це широко використовуваний метод обробки і аналізу цифрових зображень для поділу зображення на кілька частин або областей, часто на основі характеристик пікселів в зображенні. Сегментація зображення може включати відділення переднього плану від фону або кластеризацію областей пікселів на основі подібності кольору або форми. Наприклад, нормальним застосуванням сегментації зображення в медичній візуалізації є виявлення та маркування пікселів на зображенні або вокселей тривимірного об'єму, які представляють пухлину в головному мозку або інших органах пацієнта.

1.1. Огляд методів сегментації

Процес кластеризації зображень, тобто пошуку в них однорідних областей, називається сегментацією. Вона вважається першим етапом аналізу зображень, це – базова процедура практично у всіх завданнях обробки зображень за допомогою систем комп'ютерного зору. Як і інші завдання цієї області, сегментація не може бути повністю формалізована, вона включає в себе елементи фільтрації перешкод і виділення зображень [1].

Деякими практичними застосуваннями сегментації зображень є:

- Виявлення пухлин та інших патологій;
- Визначення обсягів тканин;
- Виділення об'єктів на супутникових світлинах;
- Розпізнавання об’єктів.

1.1.1. Морфологічні методи

Двійкові зображення можуть містити безліч недоліків. Зокрема, двійкові області, створені простою установкою порогових значень, спотворюються шумом і текстурою. Обробка морфологічного зображення переслідує мету усунення цих недоліків шляхом урахування форми і структури зображення.

Морфологічна обробка зображень – це набір нелінійних операцій, пов'язаних з формою або морфологією елементів зображення. Згідно Вікіпедії, морфологічні операції покладаються тільки на відносний порядок значень пікселів, а не на їх числові значення, і тому особливо підходять для обробки двійкових зображень [6].

Морфологічні методи досліджують зображення за допомогою невеликої форми або шаблону, званого структурують елементом. Елемент структурування розташовується у всіх можливих місцях зображення і порівнюється з відповідною околицею пікселів. Деякі операції перевіряють, «вписується» чи елемент в околиці, в той час як інші перевіряють, «потрапляє» він в околиці або перетинає їх за допомогою таких логічних операцій, як: NOT (логічне «НЕ»); AND (логічне «І»); OR (логічне «АБО»); XOR (виключає «АБО»).

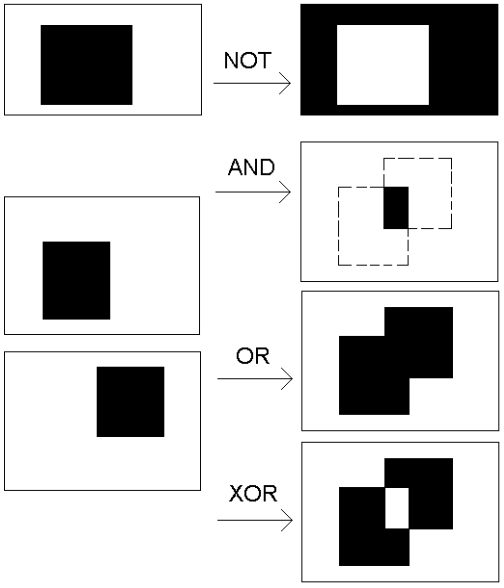


Рис. 1.1. Приклади найпростіших логічних операцій над зображенням

Приклади відповідних процедур представлені на рис. 1.1. Також в морфологічних методах використовуються ерозія і дилатація – операції, призначені в першу чергу для виявлення різних морфологічних особливостей зображень, з використанням різних структурних елементів.

<u>До морфологічних методів відносять:</u>

1. Метод виділення границь

Границя множини A, яку позначено через β(A), може бути отримана шляхом виповнення операції ерозії A по B, а потім отримання різничної множини між множиною A і результатом її ерозії.

$$\beta(A)=A\setminus(A\ominus B), \qquad (1.1)$$

де B – будь–який примітив операції.

На рис. 1.2. показаний механізм роботи процедури. Зліва показано вихідне зображення, праворуч – зображення, отримане відніманням з множини A результату ерозії множини по деякому примітиву B.

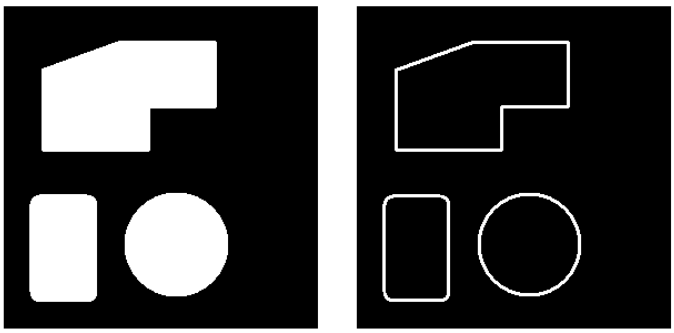


Рис. 1.2. Приклад отримання меж шляхом виділення границь

2. Метод виділення зв'язкових компонент

Ставлення зв'язності між елементами зображення є фундаментальним поняттям. Для того, щоб встановити, чи є два елементи зображення зв'язними, необхідно, щоб вони були сусідами і рівні їх яскравості задовольняли заданому критерію подібності.

Нехай S – деяка підмножина елементів зображення. Два елементи p і q називаються зв'язними в S, якщо між ними існує шлях, що цілком складається з

елементів множини S. Для кожного пікселя p з S множини усіх пікселів, пов'язаних з ним в S, називається зв'язковою компонентою (або компонентною пов'язаністю) S. Якщо множина S містить тільки одну компоненту зв'язності, вона називається зв'язковою множиною.

На практиці виділення компонент зв'язності на зображенні займає центральне місце в багатьох прикладних задачах аналізу зображень.

Нехай Y – деяка зв'язкова компонента з множини A. Припустимо, що нам відома точка p ∈ Y. Тоді всі елементи компоненти Y можуть бути отримані за допомогою рекурентного виразу:

$$Xk=(Xk-1\oplus B)\cap Ak=1,2,3,\ldots, \quad (1.2)$$

де $X_0=p$; B – будь–який примітив операції.

Використання примітиву A в якості обмежуючого обґрунтовано тим, що всі розшукувані пікселі мають значення 1. Взяття перетину з множиною A на кожному кроці ітерації виключає з результатів дилатації ті значення, які припадають на нульові елементи. Даний алгоритм застосовується в разі будь–якого кінцевого числа зв'язкових компонент, з яких складається множина A, за умови, що всередині кожної компоненти зв'язності відома хоча б одна точка.

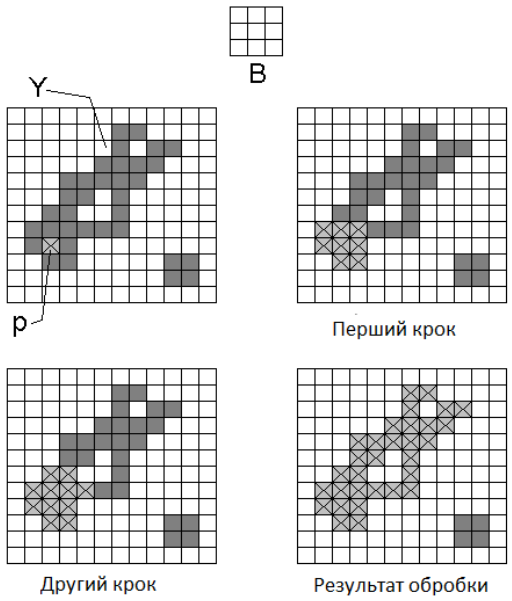


Рис. 1.3. Приклад виконання операції виділення зв'язкових компонент

На рис. 1.3. показана процедура обробки зображення, починаючи з деякої компоненти p. Процедура завершується тоді, коли на поточному кроці не було не виявлено жодної нової зв'язкової компоненти.

1.1.2. Порогові методи

Встановлення граничних значень – це процес створення чорно–білого зображення з зображення в градаціях сірого шляхом установки білого кольору саме тих пікселів, значення яких перевищує заданий поріг, і установки інших пікселів на чорний. Зображення в градаціях сірого з встановленням граничних значень може використовуватися для створення двійкових зображень. Прості методи порогової обробки включають заміну кожного пікселя в зображенні чорним пикселем, якщо інтенсивність зображення src (x, y) менше деякої фіксованої константи T (тобто src (x, y) <T), або білого пікселя, якщо інтенсивність зображення більше цієї константи.

На рис. 1.4. показані гістограми деяких зображень. В даному випадку такі гістограми відповідають зображенню зі світлими об'єктами на темному фоні. Можна бачити, що пікселі групуються навколо основних центрів. Для виділення цих областей, потрібно вибрати деяке значення T і визначити всі точки, які мають f (x, y) ≤T, як ті, що належать об'єкту, а в іншому випадку – належать фону. Тоді отримане на виході зображення визначається виразом:

$$g(x,y) = \begin{cases} 1, if\ f(x,y) > T \\ 0, if\ f(x,y) \leq T, \end{cases} \quad (1.3)$$

де 1 – значення для пікселя, відповідного об'єкту;

0 – значення для пікселя, відповідного фону.

Якщо значення T однаково для всіх точок зображення, то такий поріг називають глобальним. Якщо значення T залежить від просторових координат x

і y, то такий поріг називають динамічним. Якщо ж T залежить від значення f (x, y), то такий поріг називають адаптивним.

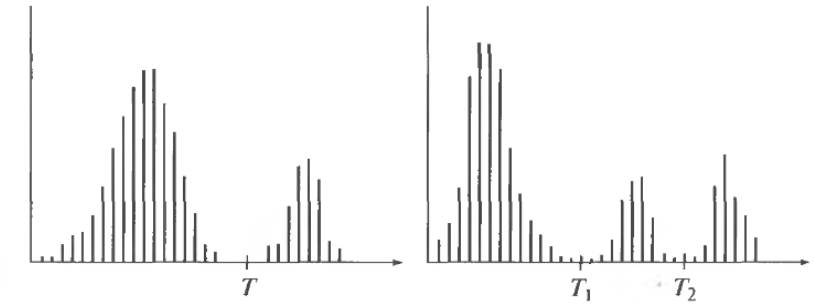


Рис. 1.4. Приклад гістограм з можливістю поділу одиночним (T) і множинним (T_1, T_2) порогами.

Основні порогові методи:

1. *Граничний метод з глобальним порогом*

Даний метод є найпростішим. Після вибору глобального порога відбувається поелементно перевірка всього зображення. Процедура передбачає поділ зображення на дві області: перша відноситься до об'єкту, друга – до фону. В даному випадку успішність цілком залежить від того, наскільки добре діаграма піддається поділу. Успішного застосування даного методу можна очікувати в умовах контрольованого освітлення.

На рис. 1.5. зліва наведено вихідне зображення, праворуч – отримане після обробки. Для даного методу можливий автоматичний вибір порога. Для цього застосовується наступний алгоритм:

1) вибирається деяка початкова оцінка порогу T;
2) використовуючи поріг T, сегментуємо зображення на дві області G1 і G2;
3) обчислюємо значення μ1 і μ2 середніх значень яскравості областей G1 і G2;
4) обчислюємо нове значення порога за формулою: T = 12 (μ1 + μ2);

5) обчислюємо кроки 2–4 до тих пір, поки різниця значень T при сусідніх ітераціях не опиниться менше або дорівнює деякому △ T

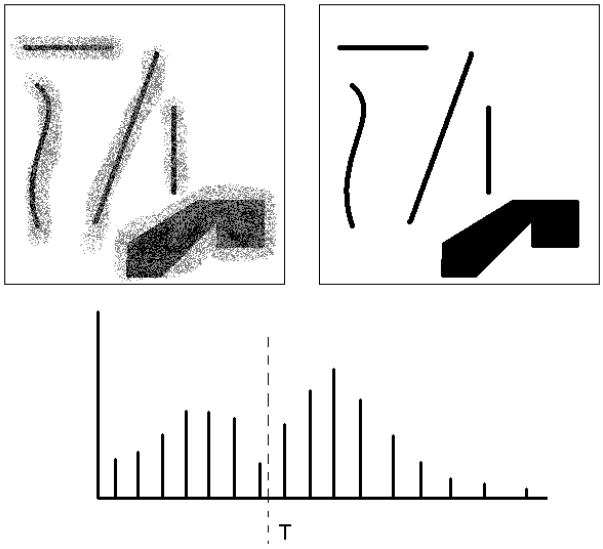


Рис. 1.5. Приклад поділу зображення з використанням глобального порога T.

Обчислення середніх значень яскравості має сенс, якщо апріорі відомо, що фон і об'єкт мають площі, які можна порівняти на зображенні. Якщо ж площа об'єкта мала, то домінуючим буде фон і в цьому випадку поділ по середньої яскравості області буде не дуже хорошим. У цьому випадку в якості обчислюваних критеріїв μ1 і μ2 можна використовувати половину суми мінімального і максимального значень яскравості.

2. Граничний метод з адаптивним порогом

У попередньому пункті було зазначено, що метод з глобальним порогом хороший до тих пір, поки ми маємо контрольоване освітлення. Як тільки освітлення стає нерівномірним, гістограма, що добре розділялась може перетворитися в гістограму, що погано розділяється, і метод не спрацює. В цьому випадку вихідне зображення слід розділити на підобласті, в кожній з яких для сегментації шукається і використовується свій поріг. Основною проблемою тут є завдання розбиття зображення на підобласті і вибір для кожної з них свого порога.

Оскільки поріг залежить від характеристик підобласті зображення, такий поріг називають адаптивним. На рис. 1.6. наведено приклад використання глобального і адаптивного порогів.

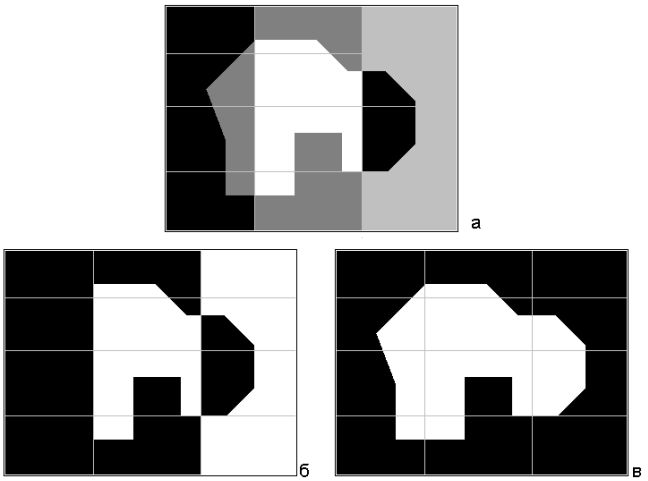


Рис. 1.6. Приклад обробки зображення глобальним (б) і адаптивним (в) порогами.

На рис. 1.6. (а) показано початкове зображення деякої області. На рис. 1.6. (б) представлений результат використання глобального порога. Права частина області, що шукається загубилася, так як значення пікселів фону і пікселів об'єкта злилися і були відсічені як фон. На рис. 1.6. (в) показаний результат використання адаптивного порога.

Як критерій розбиття зручно використовувати поняття дисперсії освітлення. Тобто зображення розбивається на області, освітленість яких приблизно однакова.

Дисперсія обчислюється за формулою:

$$\sigma^2(z) = \sum_{i=0}^{L-1}(z_i - m)^2 p(z_i), \qquad (1.4)$$

де z – величина, відповідна яскравості елементів зображення;

а p(zi), i = 0,1,2, ..., L–1 − гістограма z, де L позначає число різних рівнів яскравості.

1.1.3. Методи нарощування областей

Сегментація вододілів – це ще один регіональний метод, який бере свій початок в математичної морфології.

При сегментації вододілів зображення розглядається як топографічний ландшафт з гребенями і долинами. Значення висоти ландшафту зазвичай визначаються значеннями сірих відповідних пікселів або величиною їх градієнта. Ґрунтуючись на такому тривимірному поданні, перетворення вододілу розбиває зображення на водозбірні басейни. Для кожного локального мінімуму водозбірний басейн включає всі точки, шлях найбільш крутого спуску яких закінчується в мінімумі. Вододіли відокремлюють басейни один від одного. Перетворення вододілу повністю розкладає зображення і, таким чином, призначає кожен піксель або регіону, або вододілу. При наявності зашумленних даних медичних зображень виникає велика кількість невеликих ділянок. Це проблема відома як «надмірна сегментація».

Лінії, утворені крапками–гребенями, являють собою лінії вододілів, тому основним завданням даного методу є саме пошук ліній вододілів.

Алгоритм методу водорозділу (нарощування областей):

1. У місцях локального мінімуму утворюємо «дірки», через які вода почне заповнювати тривимірну поверхню.
2. Якщо вода з двох сторін гребеня готова об'єднатися в один басейн, встановлюємо перегородку.
3. Коли над водою залишаться тільки загородки, зупиняємо алгоритм.

Отримані таким чином перегородки і є необхідні лінії вододілів.

На рис. 1.7. наведено покрокове виконання алгоритму. На верхньому лівому малюнку показаний оригінальний «рельєф» деякого зображення, на нижньому правому – результат роботи алгоритму.

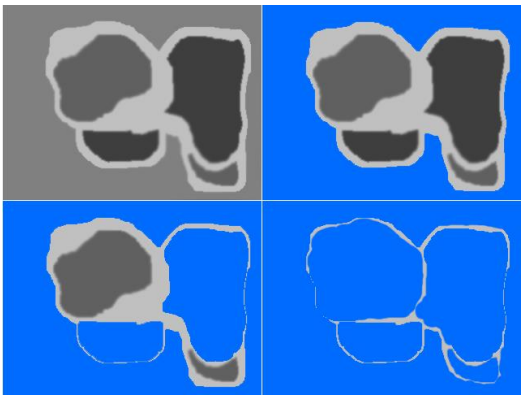


Рис.1.7. Приклад роботи алгоритму водорозділів

1. *Вододіл шляхом дилатації*

Інший найпростіший спосіб побудови перегородок – використання морфологічної дилатації. Приклад використання цього способу наведено на рис. 1.8. Як примітив в даному випадку використовується матриця 3 * 3 елементи.

Рис. 1.8. Побудова вододілів методом дилатації.

Під час роботи алгоритму використовуються два правила:

- застосування дилатації повинно обмежуватися тільки своєю областю;
- дилатація не повинна застосовуватися в тих точках, де можливе злиття двох областей.

На рис. 1.8. (А) показані дві початкові області. При застосуванні дилатації до областей (рис. 1.8. (Б)) їх площа збільшується. На наступному кроці (рис. 1.8.(В)) нарощувані області готові злитися в одну – значить необхідно встановити перегородку. Після того, як точки розділу знайдені, їм присвоюється

значення рівне максимальної яскравості + 1. Це запобігає злиття областей в ході роботи алгоритму. Важливо відзначити, що отримані лінії вододілу є зв'язковими компонентами, які не мають розривів в лініях сегментації.

Однак безпосереднє застосування методу вододілу може привести до надлишкової сегментації, викликаної локальними нерівностями і шумом в зображенні (рис. 1.8. (А)). Практичне розв'язання цієї проблеми полягає в тому, щоб обмежити допустиму кількість областей шляхом включення додаткового кроку попередньої обробки. Такий підхід заснований на ідеї маркерів.

2. Вододіл з маркерами

Маркер являє собою зв'язну компоненту, що належить зображенню. Розрізняють зовнішні (відповідають фону) і внутрішні (що відносяться до об'єкта) маркери.

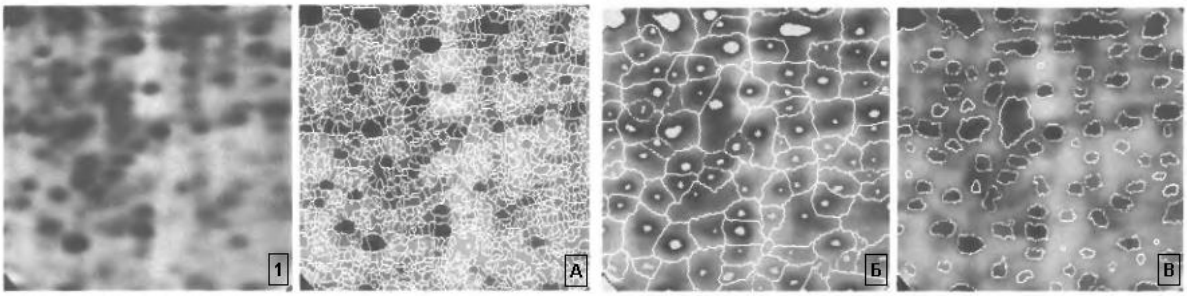


Рис. 1.9. Приклад вододілу з використанням маркерів.

Процедура вибору маркера складається з двох основних етапів:

1. передобробка;
2. вироблення критеріїв, яким повинні задовольняти маркери.

Нехай внутрішній маркер визначено як область з наступними критеріями:

- оточена точками з великим значенням яскравості;
- її точки утворюють компоненту зв'язності;
- всі точки мають однакове значення яскравості.

Внутрішні маркери, знайдені за цими 3 правилам, показані на рис. 1.9. (Б) світло–сірим кольором. Далі застосовуємо алгоритм сегментації по вододілах з тим обмеженням, що в якості локальних мінімумів розглядаються тільки локальні маркери. В результаті отримаємо зображення, що набагато краще читається в порівнянні з вихідним зображенням (рис. 1.9. (В)).

Наведений випадок – найпростіший. У загальному випадку маркери можуть мати більш складний опис, що включає розміри, форму, місце розташування, відстані, текстурні і інші ознаки. Найбільшою перевагою маркерів є те, що можна використовувати апріорні знання про завдання і ефективно використовувати їх при вирішенні задач сегментації.

1.2. Ефективний алгоритм виявлення пухлини мозку за допомогою сегментації на основі вододілу та порогів

Розробка методів автоматичної сегментації пухлин головного мозку залишається однією з найскладніших завдань обробки медичних даних. Точна сегментація може поліпшити діагностику, наприклад, оцінку обсягу пухлини в часі. Однак ручна сегментація даних магнітного резонансу – це трудомістке завдання.

У методі для запобігання надмірної сегментації використовується сегментація вододілу, контрольована маркерами. Попередня обробка зображення МРТ є основним етапом, який усуває шум і згладжує зображення. Щоб запобігти неправильній класифікації тканин мозку та немозкових тканин, проводиться видалення черепа. Сегментація зображень здійснюється за допомогою маркерованої сегментації вододілу. Потім з сегментованого зображення за допомогою морфологічної операції визначають область пухлини. Після цього визначається місце розташування пухлинної області.

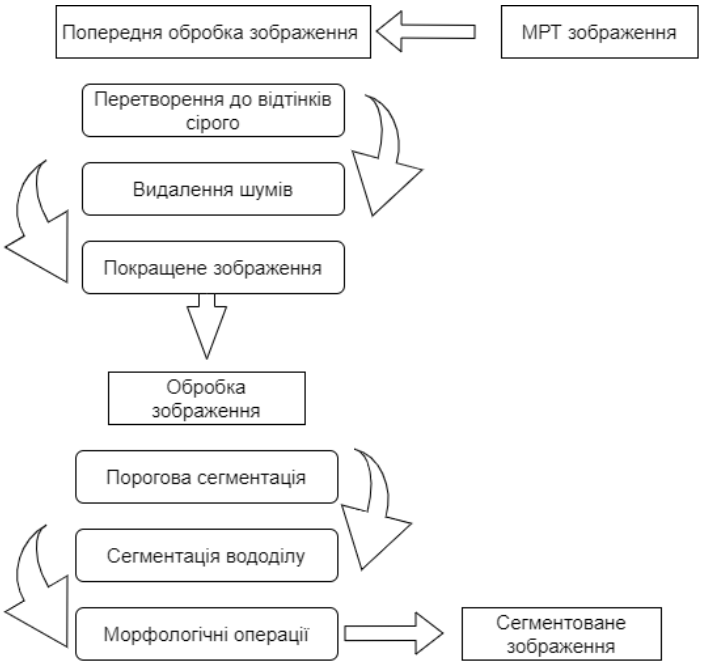


Рис. 1.10. Етапи виявлення пухлини

1.2.1. Фільтрація зашумленних зображень

Видалення шуму

Шум завжди присутній в цифрових зображеннях на етапах отримання, кодування, передачі і обробки зображень. Фільтрація даних зображення – стандартний процес, який використовується майже в кожній системі обробки зображень. Для цього використовуються фільтри. Вони видаляють шум з зображень, зберігаючи їх деталі. Вибір фільтра залежить від його поведінки і типу даних.

Різновиди фільтрів:

1. *Фільтр, заснований на обчисленні середнього арифметичного*

Такий фільтр є найпростішим. Він згладжує локальні варіації яскравості на зображенні, і видалення шуму відбувається за рахунок цього згладжування.

Нехай S_{xy} – деяка околиця розмірами m * n і з центром в точці (x, y). Необхідно обчислити середнє арифметичне по околиці S_{xy} . Таким чином, для довільної точки оброблюваної околиці маємо:

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(st)\in S_{xy}} g(s,t) \tag{1.5}$$

Дану операцію можна представити у вигляді маски, все коефіцієнти якої рівні 1/mn.

2. Медіанний фільтр

Найбільш ходовим з фільтрів, заснованих на статистиках, є медіанний фільтр. Дія цього фільтра зводиться до заміни значення в точці зображення на медіану значень яскравості в околиці цієї точки.

$$\hat{f}(x,y) = med_{(st)\in S_{xy}}\{g(s,t)\} \tag{1.6}$$

При обчисленні медіани, значення в самій точці також враховується. Широка популярність цих фільтрів обґрунтована тим, що вони прекрасно пристосовані до придушення випадкових шумів, і при цьому приводять до найменшого розмивання в порівнянні з іншими фільтрами. Медіанні фільтри особливо успішно працюють у випадках імпульсного шуму.

3. Гаусів шум

Гаусів шум є найбільш простим з математичної точки зору. Спектральні складові цього типу шуму рівномірно розподілені по всьому діапазону задіяних частот. Прикладами білого шуму є шум водоспаду або ефірні перешкоди. У природі і техніці «чисто» білий шум (тобто білий шум, який має однакову спектральну потужність на всіх частотах) не зустрічається (з огляду на те, що такий сигнал мав би нескінченну потужність), проте під категорію білих шумів

потрапляють будь–які шуми, спектральна щільність яких однакова (або слабо відрізняється) в розглянутому діапазоні частот. Функція щільності розподілу Гаусова шуму випадкової величини z має вигляд:

$$p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\mu)^2/2\sigma^2} \tag{1.7}$$

де z є значенням яскравості,

μ – середнє значення випадкової величини z,

σ – її середньоквадратичне відхилення.

1.2.2. Видалення черепної коробки з цифрового зображення

Для методів сегментації пухлини головного мозку першим кроком є видалення черепа з вихідного МРТ–зображення. Це важливий крок, тому що на багатьох МРТ–зображеннях череп виглядає як одна з найяскравіших областей зображення і різко контрастує з іншими областями мозку, такими як сіра речовина. У типах МРТ–зображень, використовуваних в цій дипломній роботі, пухлини також виглядають як яскраві області, і, таким чином, щоб обмежити виникнення хибнопозитивних сегментів, череп повинен бути вилучений з зображення [7].

Техніки видалення черепа можна розділити на 3 основні категорії:

- **Методи, засновані на інтенсивності**. Вони засновані на порогової класифікації. Основним недоліком цього підходу є його значна чутливість до коливань інтенсивності (в разі МРТ, викликаного, наприклад, неоднорідністю магнітного поля, зареєстрованим шумом або навіть властивостями пристрою).

- **Методи, засновані на морфології**. Основна ідея полягає в тому, щоб об'єднати використання морфологічних операцій, порогових значень і методів виявлення країв, щоб найбільш точно відокремити область мозку від навколишньої тканини.
- **Методи на основі деформованої моделі**, в якій застосовується деформація активної контури та підгонка для локалізації областей мозку та її ідентифікація за допомогою характеристик зображення.

Алгоритм, представлений у цій роботі, відноситься до груп методів, заснованих на морфології.

- порогова обробка зображень,
- заповнення прогалин в витягнутих об'єктах за допомогою морфологічних операторів,
- виявлення країв і поліпшення країв, якщо це необхідно,
- виділення найбільшої області зображення і створення бінарної маски,
- об'єднання двійковій маски і вхідного зображення в якості вихідного зображення.

Після проведення порогової обробки зображення, інформація про яку зазначена у підрозділі 1.1.2., наступним кроком є створення маски, яка, помножена на вхідне зображення, дозволяє виділити тільки область тканини мозку. Маска розробляється з використанням морфологічних операцій, які засновані на застосуванні так званого «структурного елементу». Це набір пікселів, які можуть мати різні форми і розміри і містити будь–яку комбінацію значень 0 і 1. Якщо значення пікселя не має значення, його можна помітити в структурному елементі як z. У пропонованому методі застосовуються такі морфологічні оператори:

- **Розширення (потовщення, dilatation).** Структурний елемент порівнюється з кожним пікселем зображення. Якщо хоча б один піксель околиці має значення, рівне «1», точка фокусування також отримує його (в іншому випадку присвоюється значення «0»). Типи структурних елементів сильно впливають на вихідне зображення.
- **Ерозія (витончення, erosion).** Ця операція застосовує повернений структурний елемент до кожного пікселя зображення. Якщо хоча б один піксель в околиці має значення, рівне «0», точка фокусування також отримує це значення. В іншому випадку його значення не змінюється. Це операція, протилежна дилатації. На ерозію істотно впливає вибір конструктивного елементу.
- **Відкриття (opening).** Накладення операції дилатації на результат ерозії вихідного зображення. Це викликає згладжування зображення (видалення деталей, чим більше використовується структурний елемент, тим сильніше згладжування зображення).
- **Закриття (closing).** Накладення ерозійних операцій на результат дилатації вихідного зображення. Вона видаляє всі отвори на зображенні і увігнутість нижче структурного елементу (чим більше структурний елемент, тим більше елементів заповнюється).

Виявлення країв дозволяє перевірити, чи є кореляція між краями маски (застосовується для вилучення мозку) і анатомічними краями вхідного зображення. Перекриття кордонів мозку на вхідному зображенні з кордонами, встановленими пропонованим методом видалення черепа, доводить, що алгоритм вірний. Для кожної площині зображення застосовується фільтр виявлення границь Кенні. Потім зображення, що представляє краї, об'єднується з відповідним вхідним зображенням.

Алгоритм виявлення границь Кенні:

- *Згладжування*. Розмивання зображення для видалення шуму.

- *Пошук градієнтів*. Межі відзначаються там, де градієнт зображення набуває максимальне значення.
- *Подавлення НЕ–максимумів.* Тільки локальні максимуми відзначаються як кордони.
- *Подвійна порогова фільтрація.* Потенційні межі визначаються порогами.
- *Трасування області неоднозначності*. Підсумкові межі визначаються шляхом придушення всіх країв, незв'язаних з певними (сильними) межами.

1.2.3. Сегментація зображення методом Отцу та маркерованого водорозділу

У комп'ютерному зорі і обробці зображень метод Оцу використовується для автоматичного визначення порогових значень зображення. У простій формі алгоритм повертає єдиний поріг інтенсивності, який розділяє пікселі на два класи: передній план і фон. Цей поріг визначається шляхом мінімізації дисперсії інтенсивності всередині класу або, що еквівалентно, шляхом максимізації дисперсії між класами. Метод Оцу є одномірним дискретним аналогом дискримінантного аналізу Фішера, пов'язаний з методом оптимізації Дженкса і еквівалентний глобальному оптимальному k–середньому, виконаному на гістограмі інтенсивності [5].

Метод Оцу шукає поріг, що зменшує дисперсію всередині класу, яка визначається як зважена сума дисперсій двох класів:

$$\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t), \quad (1.8)$$

де ваги ω_i – це ймовірності двох класів, розділених порогом t,

σ_i^2 – дисперсія цих класів.

Алгоритм методу Отцу:

Нехай дано монохромне зображення $G(i,j), i = \overline{1, Heıght}, j = \overline{1, Wıdth}$. Лічильник повторів k=0.

1. Обчислити гістограму ***p(l)*** зображення та частоту ***N(l)*** для кожного рівня інтенсивності зображення G.
2. Обчислити початкові значення для $\omega_1(0)$, $\omega_2(0)$ та $\mu_1(0)$, $\mu_2(0)$.
3. Для кожного значення $t = \overline{1, \max(G)}$ – полутону – горизонталь вісь гістограми:
    1. Оновлюємо ω_1, ω_2 та μ_1, μ_2.
    2. Обчислюємо $\sigma_b^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$.
    3. Якщо $\sigma_b^2(t)$ більше, ніж наявне, то запам'ятовуємо σ_b^2 і значення порогу t.
4. Шуканий поріг відповідає максимуму $\sigma_b^2(t)$

$$N_T = \sum_{i=0}^{max(G)} p(i), \tag{1.9}$$

$$\omega_1(t) = \frac{\sum_{i=0}^{t-1} p(i)}{N_T} = \sum_{i=0}^{t-1} N(i), \ \omega_2(t) = 1 - \omega_1(t) \tag{1.10}$$

$$\mu_T = \frac{\sum_{i=0}^{max(G)} i*p(i)}{N_T} = \sum_{i=0}^{\max(G)} i * N(i), \tag{1.11}$$

$$\mu_1(t) = \frac{\sum_{i=0}^{t-1} i*p(i)}{N_T*\omega_1(t)} = \frac{\sum_{i=0}^{t-1} i*N(i)}{\omega_1(t)}, \ \mu_2(t) = \frac{\mu_T - \mu_1(t)*\omega_1(t)}{\omega_2(t)} \tag{1.12}$$

Будь–яке зображення у відтінках сірого можна розглядати як топографічну поверхню, де висока інтенсивність означає піки і пагорби, а низька інтенсивність позначає западини. Починаємо заповнювати кожну ізольовану западину (локальні мінімуми) водою різного кольору (мітки). У міру підйому води, в залежності від найближчих піків (градієнтів), вода з різних западин, очевидно, різного кольору, почне зливатися. Щоб цього уникнути, будуємо перешкоди в місцях злиття води. Продовжуємо заповнювати водою і будувати перепони, поки

всі вершини не опиняться під водою. Тоді створені бар'єри дають результат сегментації.

Але такий підхід дає сверхсегментований результат через шум або будь–які інші нерівності зображення. Отже, реалізувавши алгоритм вододілу на основі маркерів, в якому вказуємо, які точки западини повинні бути об'єднані, а які ні. Надаємо об'єкту різні ярлики. Позначаємо область, яка, є переднім планом або об'єктом, одним кольором (або інтенсивністю), позначаємо область, яка є фоном або не є об'єктом, іншим кольором і, нарешті, область, в якій ми не впевнені, позначаємо її нулем. Це маркер. Потім застосовуємо алгоритм вододілу. Потім маркер буде оновлено присвоєними мітками, а межі об'єктів матимуть значення –1.

Алгоритм:

1. Починаємо з знаходження приблизної оцінки пухлини. Для цього використовуємо бінаризацію Отцу.
2. Тепер потрібно видалити невеликі білі шуми на зображенні. Для цього використовується морфологічне відкриття. Щоб видалити невеликі отвори в об'єкті, використовується морфологічне закриття. Ітак, визначаємо: область поруч із центром об'єкта є переднім планом, а область далеко від об'єкта – фоном. Єдина невідома область – це погранична область пухлини.
3. Потрібно витягти область, в якій знаходитися пухлина за допомогою ерозії, яка видаляє граничні пікселі.

Висновки до розділу 1

У даному розділі презентовано найпоширініші методи сегментації зображень, які використовуються для формування якісної сегментації головного мозку. Розглянуто та виділено основні етапи сегментації зображень МРТ

головногу мозку, такі як: фільтрація зашумлених зображень за допомогою Гаусового шуму, видалення черепної коробки морфологічними операціями, та саме сегментування за допомогою комбінації порогового методу Отцу та маркерованого водорозділу.

РОЗДІЛ 2

АВТОМАТИЗОВАНА СИСТЕМА СЕГМЕНТАЦІЇ ЦИФРОВИХ ЗОБРАЖЕНЬ МРТ ГОЛОВНОГО МОЗКУ

2.1. Технічні засоби та структури

Мова програмування Python

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією і автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення переносимості написаних на ньому програм. Мова є повністю об'єктно–орієнтованою – все є об'єктами. Незвичайною особливістю мови є виділення блоків коду пробільними відступами. Синтаксис ядра мови мінімалістичний, за рахунок чого на практиці рідко виникає необхідність звертатися до документації. Сама же мова відома як інтерпретована і використовується в тому числі для написання скриптів. Недоліками мови є часто більш низька швидкість роботи і більш високе споживання пам'яті написаних на ній програм в порівнянні з аналогічним кодом, написаним на компільованих мовах, таких як C або C ++.

Python є мультипарадигмальною мовою програмування, що підтримує імперативне, процедурне, структурне, об'єктно–орієнтоване програмування, метапрограмування і функціональне програмування. Завдання узагальненого програмування вирішуються за рахунок динамічної типізації. Аспектно–орієнтоване програмування частково підтримується через декоратори, більш повноцінна підтримка забезпечується додатковими фреймворками. Такі методики як контрактне і логічне програмування можна реалізувати за допомогою бібліотек або розширень. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм

обробки виключень, підтримка багатопоточних обчислень з глобальним блокуванням інтерпретатора (GIL), високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

Середовище розробки PyCharm

PyCharm – інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладчик, інструмент для запуску юніт–тестів і підтримує веб–розробку на Django.

PyCharm пропонує великий набір інструментів з коробки: вбудований відладчик і інструмент запуску тестів, профілювальник Python, повнофункціональний вбудований термінал, інструменти для роботи з базами даних. IDE інтегрована з популярними системами контролю версій, містить вбудований SSH–термінал, підтримує можливості віддаленої розробки та віддалені інтерпретатори, а також інтеграцію з Docker і Vagrant.

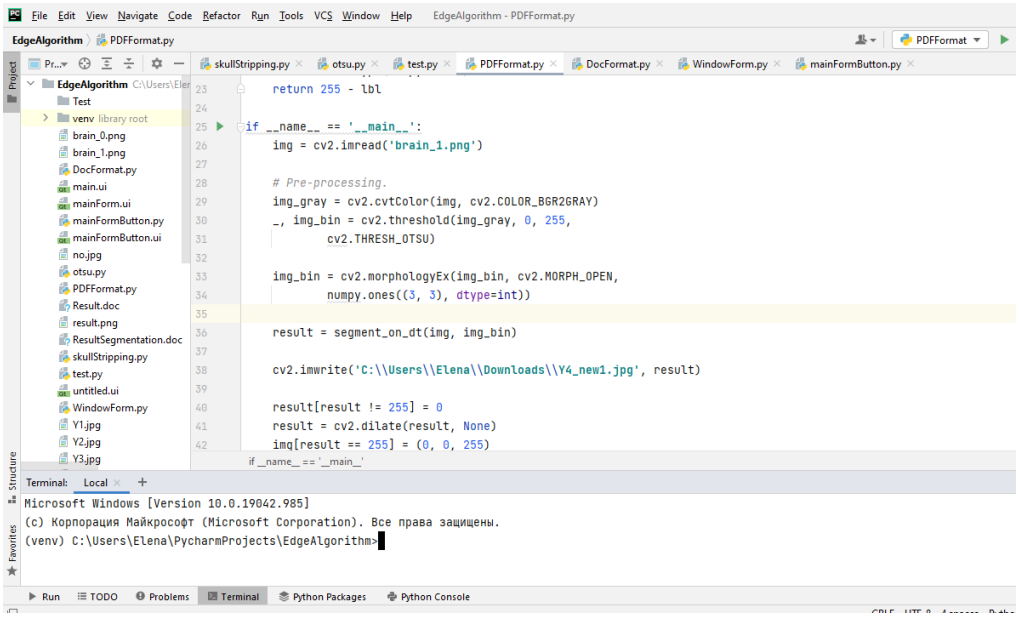


Рис. 2.1. Редактор файлу коду PyCharm

Розумний редактор PyCharm призначений для максимально продуктивної розробки на Python, JavaScript, CoffeeScript, TypeScript, CSS і популярних мовах

шаблонів. Функції автодоповнення, виявлення помилок і швидкі виправлення враховують особливості кожної з підтримуваних мов.

PyCharm надає широкі можливості реорганізації коду за допомогою рефакторингов Rename і Delete, Extract Method, Introduce Variable, Inline Variable, Inline Method і багатьох інших. Рефакторинги враховують особливості конкретної мови або фреймворка, допомагаючи вносити зміни по всьому проекту.

Фреймворк Qt

Qt – це кроссплатформна середа розробки додатків для настільних, вбудованих і мобільних пристроїв. Підтримувані платформи включають Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS і інші.

Qt сам по собі не є мовою програмування. Це фреймворк, написаний на C ++. Препроцесор, MOC (компілятор метаоб'єктів), використовується для розширення мови Python такими функціями, як сигнали і слоти. Перед етапом компіляції MOC аналізує вихідні файли, написані на розширеному Qt Python, і генерує з них вихідні коди Python, що відповідають стандарту. Таким чином, сам фреймворк і додатки / бібліотеки, що його використовують, можуть бути скомпільовані будь–яким стандартним компілятором C ++/Python.

Компоненти:

- QtCore – класи ядра бібліотеки, які використовуються іншими модулями;
- QtGui – компоненти графічного інтерфейсу;
- QtWidgets – містить класи для класичних додатків на основі віджетів, модуль виділений з QtGui в Qt 5;
- Qt QML – модуль для підтримки QML;
- QtNetwork – набір класів для мережевого програмування. Підтримка різних високорівневих протоколів може змінюватися від версії до версії. У версії 4.2.x присутні класи для роботи з протоколами FTP і HTTP. Для роботи з протоколами TCP / IP призначені такі класи, як QTcpServer, QTcpSocket для TCP і QUdpSocket для UDP;

- QtOpenGL – набір класів для роботи з OpenGL;
- QtSql – набір класів для роботи з базами даних з використанням SQL. Основні класи даного модуля в версії 4.2.x: QSqlDatabase – клас для надання з'єднання з базою, для роботи з якою–небудь конкретною базою даних вимагає об'єкт, успадкований від класу QSqlDriver – абстрактного класу, який реалізується для конкретної бази даних і може вимагати для компіляції SDK бази даних. Наприклад, для складання драйвера під СУБД Firebird або InterBase потрібні .h–файли і бібліотеки статичного компонування, що входять в комплект поставки даної СУБД;
- QtScript – класи для роботи з Qt Scripts;
- QtSvg – класи для відображення і роботи з даними Scalable Vector Graphics (SVG);
- QtXml – модуль для роботи з XML, підтримуються моделі SAX і DOM;
- QtDesigner – класи створення розширень для своїх власних віджетів;

Середовище Qt Designer

Qt Designer – це інструмент програмного забезпечення Qt для проектування та побудови графічних інтерфейсів користувача (GUI) з компонентів Qt. Дозволяє складати та налаштовувати свої віджети чи діалоги візуально–інтерпретовано, і тестувати їх, використовуючи різні стилі та роздільну здатність.

Віджети та форми, створені за допомогою Qt Designer, легко інтегруються із запрограмованим кодом, використовуючи механізми сигналів і слотів Qt, що дозволяє легко призначати поведінку графічним елементам. Усі властивості, встановлені в Qt Designer, можна динамічно змінювати в коді. Крім того, такі функції, як просування віджетів та спеціальні плагіни, дозволяють використовувати власні компоненти з Qt Designer.

Кожен віджет має свій набір властивостей, що визначається відповідним йому класом бібліотеки Qt. Властивості віджета можуть бути змінені за допомогою «Редактора властивостей». Характерною особливістю Qt Designer є

підтримка візуального редагування сигналів і слотів. Так, наприклад, можна зв'язати сигнал, що генерується з переключення стану віджета CheckBox зі слотом, що відповідає за доступність іншого віджета.

Розроблений інтерфейс зберігається в файл з розширенням ui, який підключається до створюваної програмі за допомогою спеціальних методів бібліотеки Qt. Цей файл має xml–формат, і може, в разі необхідності, редагуватися в будь–якому текстовому редакторі.

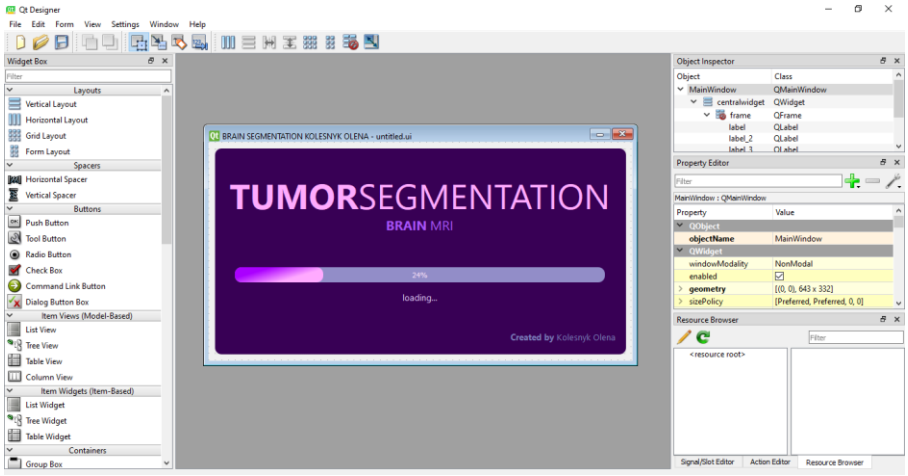


Рис. 2.2. Головне вікно Qt Designer

2.2. Опис програмного забезпечення

Для виконання поставленої задачі було розроблено програмне забезпечення для обробки цифрових зображень МРТ мозку.

Основні характеристики додатку:

Мова програмування: Python

Середовище розробки: PyCharm 2019

Мінімальні системні вимоги:

- 64–бітна операційна система Widows;
- Python 3.5 або вище;
- 36 КБ доступного простору на жорсткому диску.

Вхідні дані:

- цифрові зображення форматів JPEG (.jpg) та PNG (.png);
- параметри: шляхи збереження файла/файлів;

Вихідні дані: сегментовані цифрові зображення МРТ головного мозку.

Після зчитування зображення з файлу, програма проводить перетворення від цифрового зображення до тривимірного масиву. На основі інтенсивності трьох каналів R, G, B зображення розмивається за допомогою формули визначення інтенсивності пікселя. Результатом є зображення в градаціях сірого.

Далі програма визначає поріг та перетворює зображення на бінарне за допомогою порогового методу Отцу. Після проведення порогової обробки зображення наступним кроком є створення маски, яка, помножена на вхідне зображення, дозволяє виділити тільки область тканини мозку. Використовуючи фільтр Кенні для виявлення границь, програма за допомогою методу зв’язних компонент виділяє найбільшу область зображення і робить маску, яку поєднує з початковим зображенням для отримання мозку без черепної коробки.

Зображення тканини мозку проходить фільтрацію шуму Гаусса, і за допомогою морфологічних операцій розширення та відкриття виділяє області фону(тканини мозку) та об’єкту (пухлини) для проведення сегментації методом маркерованого водорозділу. Області, що не належать ні до множини фону, ні до множини об’єкту визначаються, як маркери. Вони використовуються для проведення остаточної морфологічної операції – ерозії, що дозволяє виділити пухлину як окремий об’єкт.

Результатом роботи є виведений на екран масив зображень, що відображають поетапну обробку зображення, з фінальним результатом – сегментованим зображенням, де область пухлини позначена іншим кольором. Програма дозволяє обрати користувачу необмежну кількість зображень для проведення сегментації, а також записати результати сегментації у документ Microsoft Word.

2.2.1. Реалізація програмного забезпечення

На першому етапі розробки було спроектовано інтерфейс користувача. Для реалізації інтерфейсу було обрано фреймворк Qt та середу QtDesigner, інформація про них зазначена у підрозділі 2.1. QtDesigner був використаний для створення певного макету форми з віджетами у форматі файлу .ui. Щоб відредагувати код та динамічно створювати об'єкти, файл форми з розширенням ui був переформатований за допомогою влаштованого термінала PyCharm та команди: *pyuic5 –x example.ui –o example.py.*

Після отримання доступу до коду форми, було створено деякі функції для оптимізації та більшої читабельності коду. На рис. 2.3. наведено приклад реалізації функції динамічного створення віджетів QLabel.

```
def createLabel(self, nameLabel, x1, x2, y1, y2, textSize ):
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(x1, x2, y1, y2))
    font = QtGui.QFont()
    font.setPointSize(textSize)
    font.setUnderline(False)
    self.label.setFont(font)
    self.label.setFrameShape(QtWidgets.QFrame.WinPanel)
    self.label.setTextFormat(QtCore.Qt.AutoText)
    self.label.setScaledContents(True)
    self.label.setAlignment(QtCore.Qt.AlignCenter)
    self.label.setObjectName(nameLabel)
    self.widgets.append(self.label)
```

Рис. 2.3. Функція створення екземплярів віджету QLabel.

```
self.widgets = []
self.nameLabels = [["img_1", "60", "20", "211", "191","20"],
             ["img_2", "60", "300", "211", "191","20"],
             ["img_3", "390", "20", "211", "191","20"]]
for n in range(len(self.nameLabels)):
        self.createLabel(str(self.nameLabels[n][0]), int(self.nameLabels[n][1]), int(self.nameLabels[n][2]),
                         int(self.nameLabels[n][3]), int(self.nameLabels[n][4]), int(self.nameLabels[n][5]))
```

Рис. 2.4. Ініціалізація масиву параметрів та виклик функції

Для використання функції createLabel() потрібно прописати код, в якому ініціалізувати масив основних параметрів функції, як показано на рис. 2.4.

Таким чином було створено інтерфейсну частину автоматизованої системи. Для того, щоб реалізувати функціонал було створено класи ImageBrain, ImageBrainArray, Otsu, Watershed, ConnectedComponents, Ui_MainWindow, MainFunctional та інтерфейс ISegmentable (див. Додаток Е).

Клас MainFunctional відповідає за функцкіональну частину інтерфейсу, тобто має подієво–орієнтовані функції. Для обробки подій панелі меню та подій при створенні головної форми було створено такі функції: openFile(), openFiles(), setOriginalImage(), viewLabels(), saveFileDoc(), preparePathes(), createPathes(), ShowInLabels().

```
def preparePathes(self):
    splitPath = self.fnameOutput.split('/')
    nameFileInput = (self.fnameInput.split('/'))
    nameFileArray = nameFileInput[-1].split('.')
    nameFile = nameFileArray[0]
    expFile = nameFileArray[1]
    del splitPath[-1]
    outputPath = "\\".join(splitPath) + "\\"
    return outputPath, nameFile, expFile
```

Рис. 2.5. Функція обробки та підготовки шляху місцерозташування файлу

З функції вказаної на рис. 2.5. ми отримуємо шлях для зберігання відсегментованого зображення, ім’я файлу та розширення. Далі ці дані використовуються у функції створення нових шляхів для збереження всіх проміжних зображень сегментації (Рис .2.6.). До шляху директорії додається назва оригінального зображення, порядковий номер та розширення. Наприклад оригінальне зображення має назву “brain1.png”, тоді вихідним масивом буде [“brain1_0.png”, “brain1_1.png”, “brain1_2.png”, “brain1_3.png”,“brain1_4.png”], де перший елемент масиву буде містити оригінальне зображення, другий елемент – зображення в градаціях сірого, третій – зображення оброблене

пороговим методом Отцу, четвертий – зображення мозку без черепної коробки, п’ятий – результат сегментації.

```
def createPathes(self):
    outputPath, nameFile, expFile = self.preparePathes()
    imagePathes = list()
    for i in range(len(self.segmentedImages)):
        path =outputPath+nameFile+"_"+str(i)+"."+expFile
        cvToImage(self.segmentedImages[i], path)
        imagePathes.append(path)
    return imagePathes
```

Рис. 2.6. Функція створення нових шляхів для збереження файлів

Розглянемо клас Otsu, що відповідає за знаходження порогів методом Отцу. Основними функціями якого є знаходження нормалізованої гістограми, функції кумулятивного розподілу та саме порогу.

```
class Otsu:
    def findNormalizedHistogram(self, blurImage):
        hist = cv2.calcHist([blurImage], [0], None, [256], [0, 256])
        hist_norm = hist.ravel() / hist.sum()
        return hist_norm
    def findCumulativeDistributionFunction(self, blurImage):
        hist_norm = self.findNormalizedHistogram(blurImage)
        Q = hist_norm.cumsum()
        return Q
```

Рис. 2.7. Шаблон класу Otsu

Функція tresholdOtsu(self, blurImage) на рис. 2.8. демонструє реалізацію методу Отцу, формули якого можна побачити у підрозділі 1.2.3. (1.8–1.12).

Наступним, не менш важливим класом, є клас ConnectedComponents, що реалізує маркерування зв’язних компонент. Алгоритм роботи основного методу класу представлений на рис. 2.9.

```
def tresholdOtsu(self, blurImage):
    bins = np.arange(256)
    fn_min = np.inf
    thresh = -1
    hist_norm = self.findNormalizedHistogram(blurImage)
    Q = self.findCumulativeDistributionFunction(blurImage)
    for i in range(1, 256):
        p1, p2 = np.hsplit(hist_norm, [i])  # probabilities
        q1, q2 = Q[i], Q[255] - Q[i]  # cum sum of classes
        if q1 < 1.e-6 or q2 < 1.e-6:
            continue
        b1, b2 = np.hsplit(bins, [i])  # weights
        # finding means and variances
        m1, m2 = np.sum(p1 * b1) / q1, np.sum(p2 * b2) / q2
        v1, v2 = np.sum(((b1 - m1) ** 2) * p1) / q1, np.sum(((b2 - m2) ** 2) * p2) / q2
        # calculates the minimization function
        fn = v1 * q1 + v2 * q2
        if fn < fn_min:
            fn_min = fn
            thresh = i
    return thresh
```

Рис. 2.8. Функція знаходження порогу методом Отцу

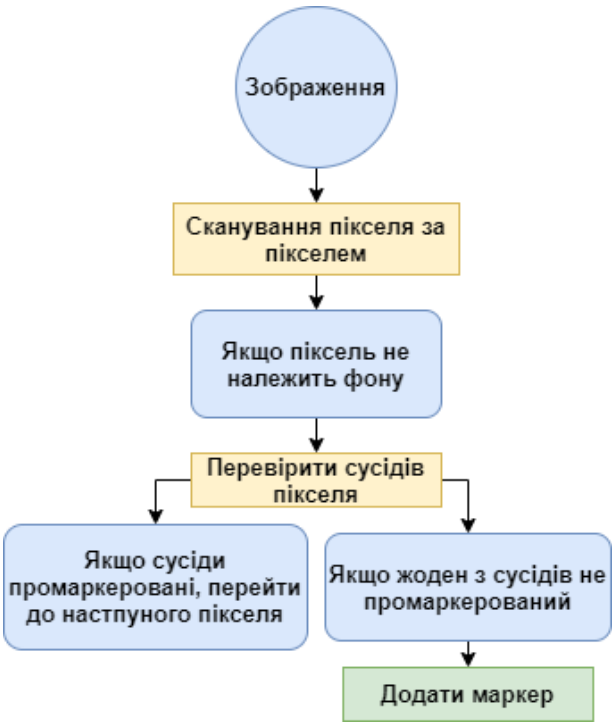


Рис. 2. 9. Алгоритм роботи методу getConnectedComponents()

```
def getConnectedComponents(bin_image):
    im = Image.open(bin_image)
    (h, w) = im.size
    yc, xc = np.where(bin_image != 0)
    queue = []
    connected_array = np.zeros((h, w))  # позначення масиву
    counter = 1
    for elem in range(len(xc)):
        # перебирати всі ненульові елементи
        i = yc[elem]
        j = xc[elem]
        if connected_array[i, j] == 0:
            # ще не позначені продовжувати
            connected_array[i, j] = counter
            queue.append((i, j))
            while len(queue) != 0:
                # робота через чергу
                current = queue.pop(0)
                i, j = current
```

Рис. 2.10. Частина реалізації методу getConnectedComponents()

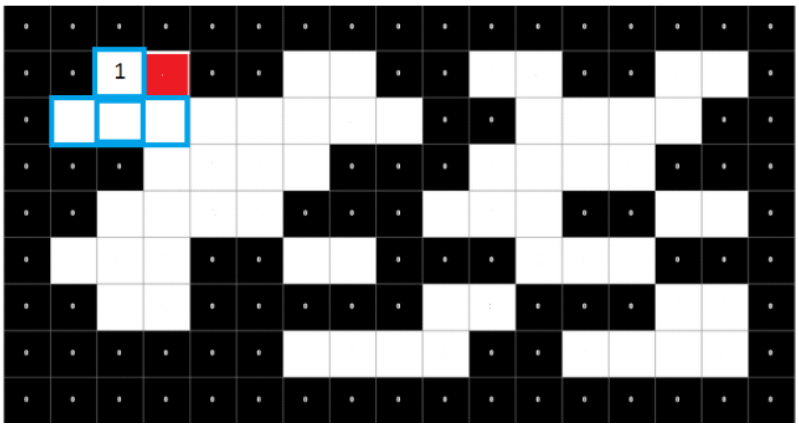


Рис. 2.11. Перевірка сусідів пікселя на наявність маркера

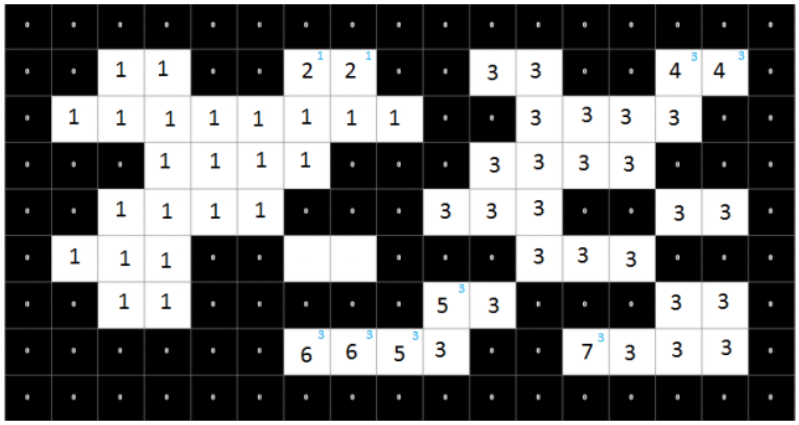


Рис. 2.12. Результат пошуку зв’язних компонент

2.2.2. Інструкція користувача

Коротка інструкція:

1. Запустити програмне забезпечення;
2. В меню обрати пункт «Відкрити» та натиснути на підпункт меню «Відкрити файл» або «Відкрити файли»
3. З діалогового вікна обрати файл формату jpg або png;
4. В наступному діалоговому вікні обрати місце для збереження результатів сегментації;
5. Натиснути на кнопку «Відкрити»;
6. Зачекати поки зображення з'являться на екрані.
7. У випадку обраного підпункту меню «Відкрити файли» натиснути на кнопку «Наступне зображення» для відображення результатів сегментації іншого зображення з обраних;
8. При потребі створення результуючої документації обрати в меню пункт «Зберегти»;
9. В діалоговому вікні обрати місце та ім'я файлу для збереження документу;
10. Після цього відкриється файл Microsoft Word, за бажанням його можна відредагувати та зберегти. Нередагований файл автоматично зберігається за обраним користувачем в попередньому пункті шляхом.
11. Користувач має можливість скористатися довідкою обравши у меню пункт «Довідка»;
12. По закінченню роботи в програмі натиснути кнопку «Завершити програму».

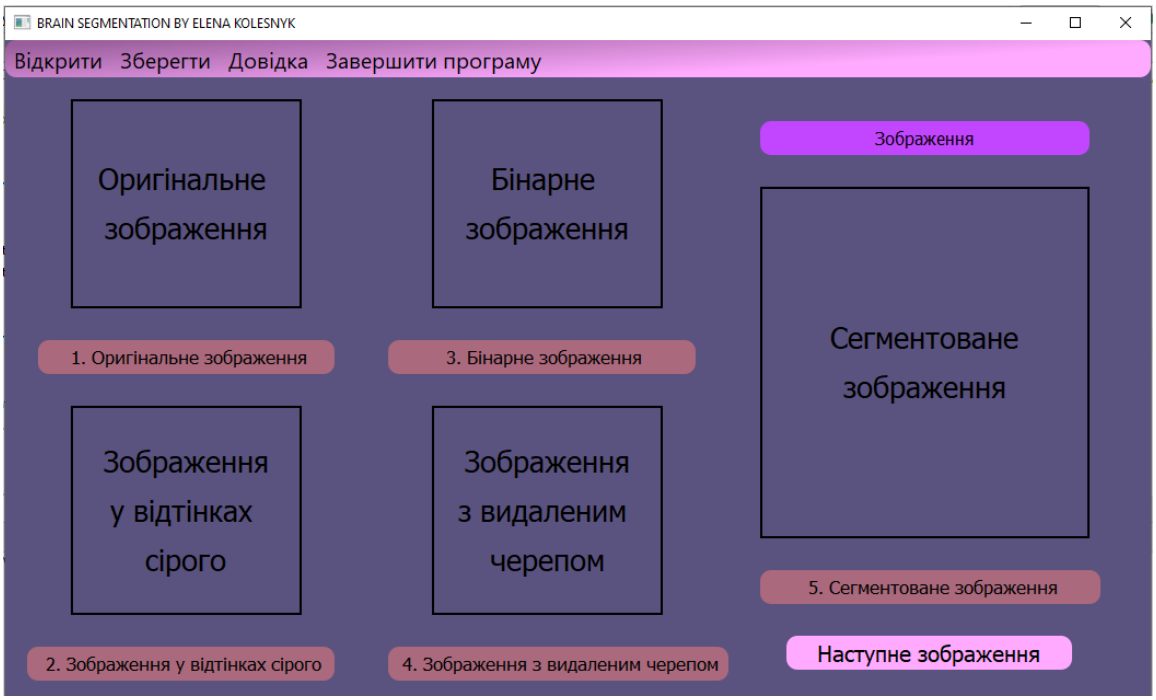


Рис. 2.13. Головне вікно автоматизованої системи сегментації

Вікно складається з 2 основних областей:

1. Область меню;
2. Область відображення результатів.

Область відображення складається з 5 віджетів для демонстрації зображень, кнопки–перемикача, та віджету для відслідковування порядкового номеру зображення в обраній послідовності. Для початку роботи потрібно в меню обрати пункт «Відкрити файл» для зчитування файлу зображення формату «JPG» або «PNG». У випадку коректного зчитування у області відображення, кожний віджет буде заповнений відповідними даними (перший – оригінальне зображення, другий – зображення у градаціях сірого, третій – бінарне зображення, четвертий – зображення з видаленим черепом, п’ятий віджет – відсегментоване зображення). Зображення у градаціях сірого, бінарне зображення відображаються для показання проміжних етапів сегментації.

На рис. 2.15. показані результати обробки вхідного зображення поетапно. Спочатку на основі інтенсивності трьох каналів R, G, B зображення розмивається за допомогою формули визначення інтенсивності пікселя. Зображення під

номером 3 показує результат порогової сегментації методом Отцу, наступним кроком є результат поєднання морфологічних операції і методу знаходження зв'язних компонент у вигляді зображення з видаленим черепом. На зображені під номером 5 відображений результат сегментації, пухлина мозку виділена іншим кольором.

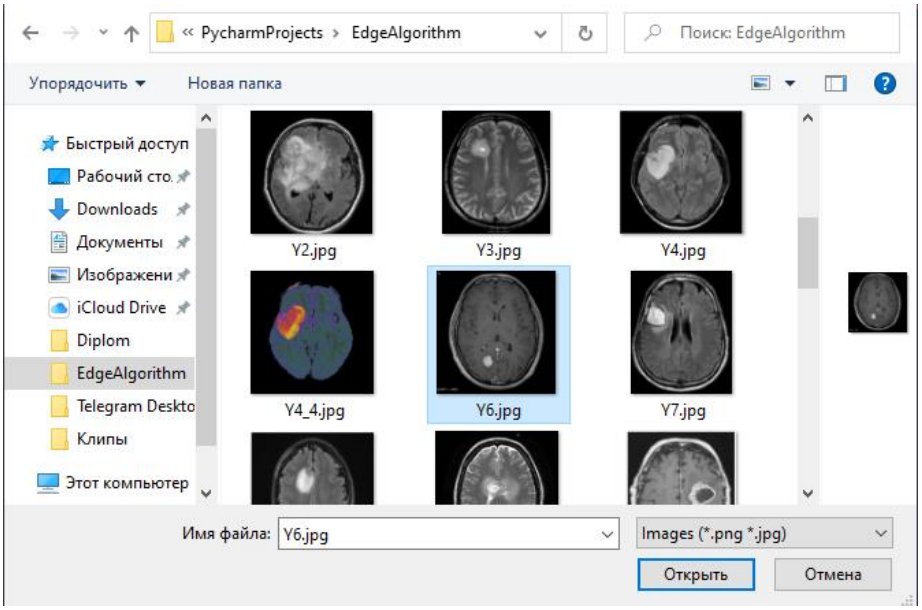


Рис. 2.14. Діалогове вікно вибору файлу для сегментації

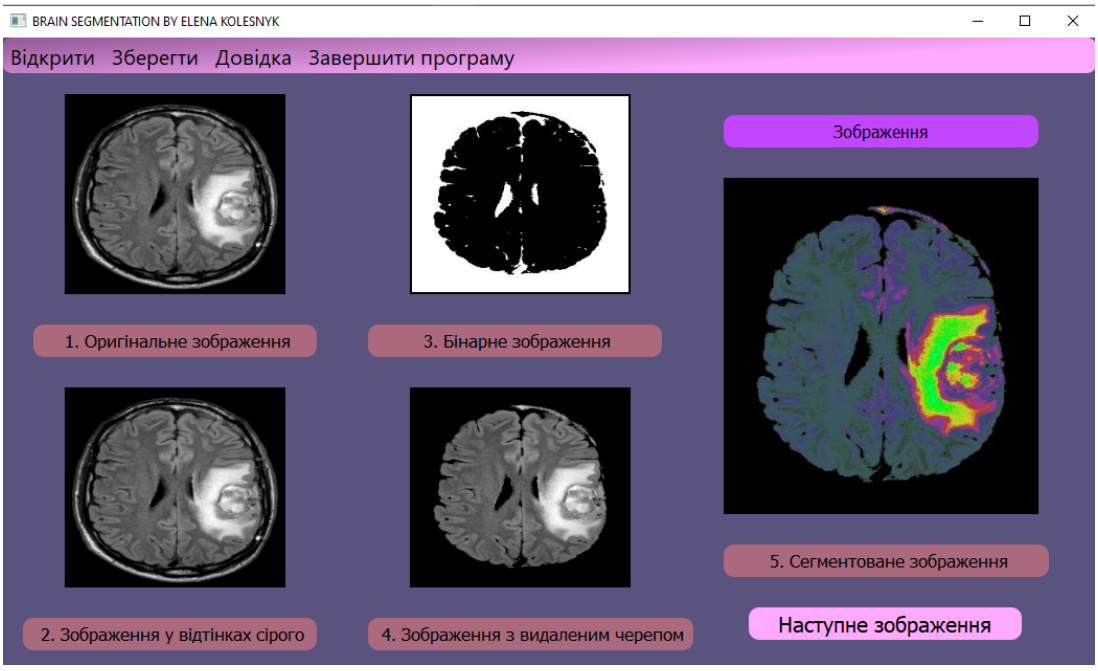


Рис. 2.15. Результат обробки МРТ зображення

Користувач має можливість обрати масив файлі для сегментації. Для цього в меню потрібно обрати «Відкрити файли» та лівою клавішою миші виділити одразу декілька зображень.

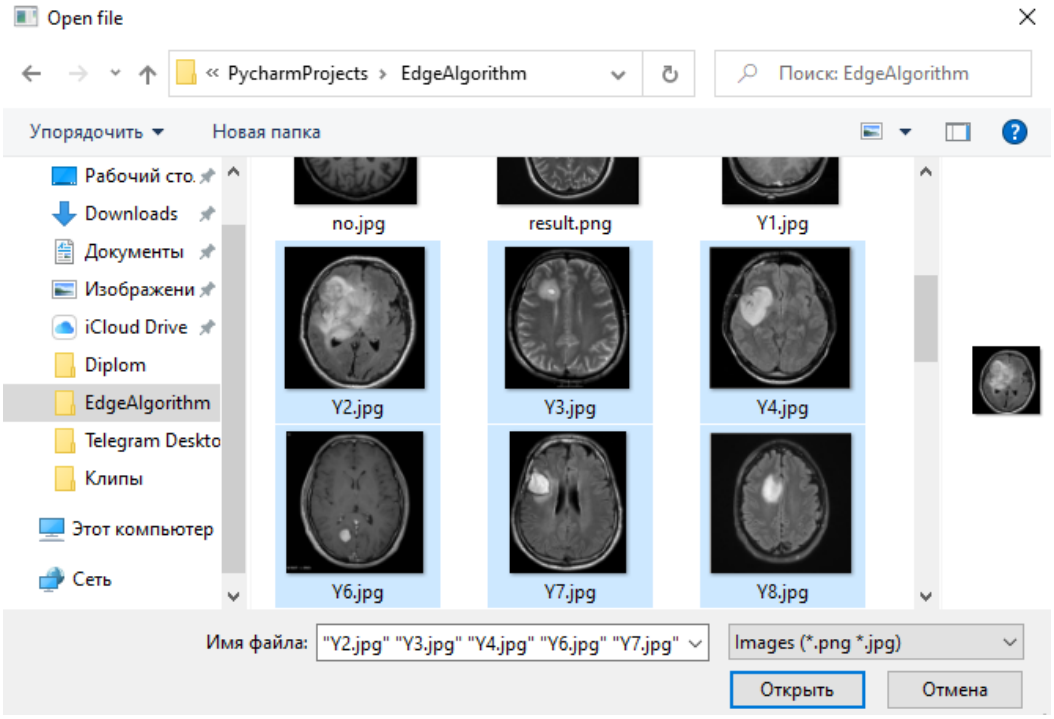


Рис. 2.16. Діалогове вікно для вибору декількох файлів

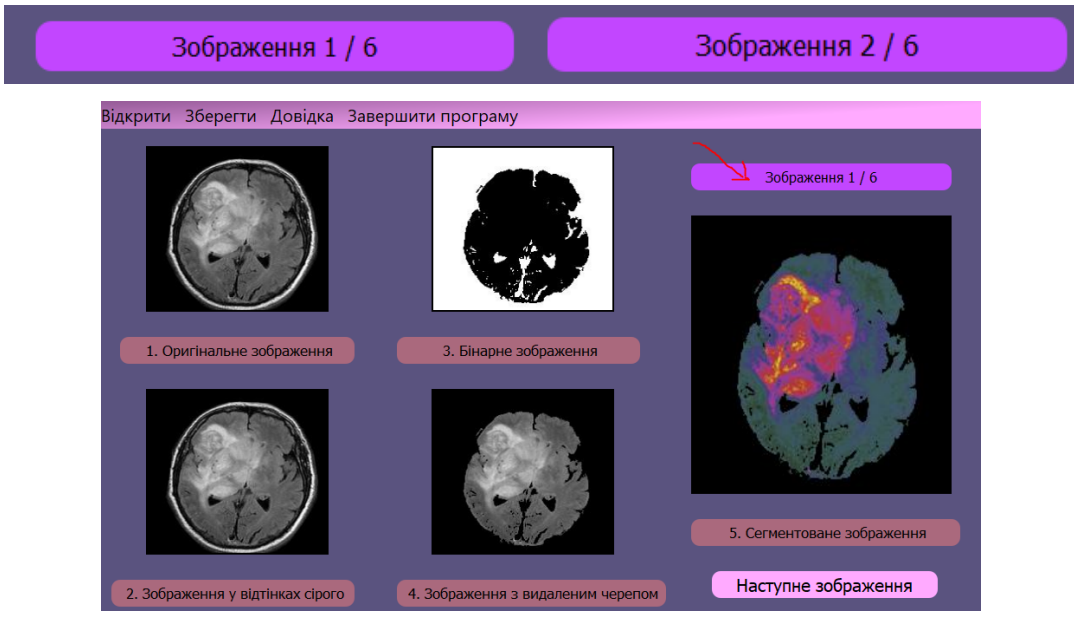


Рис. 2.17. Вікно обробки масиву зображень

Таким чином користувач натиснувши кнопку «Наступне зображення» може переключатися від одного зображення до наступного. На віджеті «Зображення» відображається порядковий номер зображення, що переглядається, та кількість завантажених користувачем файлів.

Результуючою дією є створення документації у форматі Word документу з розширенням .doc. Для цього користувач має натиснути на панелі меню вкладку «Зберегти» та обрати підпункт «Зберегти у форматі doc». Після відкриття діалогового вікна та вибору місцерозташування майбутнього результуючого документу у системі користувача відкриється документ Microsoft Word (Рис. 2.18.). Після відкриття користувач має змогу відредагувати файл та зберегти нову версію документа.

Рис. 2.18. Приклад результуючого документу сегментації декількох зображень

2.3. Порівняння з методом сегментації бібліотеки OpenCv

- Cv2.morphologyEx(cv2.MORPH_OPEN) – відкриття, ще одна назва ерозії, за якою слідує розширення. Це корисно для видалення шуму.
- cv2.dilate() – протилежність ерозії. Елемент пікселя дорівнює «1», якщо хоча б один піксель під ядром дорівнює «1». Таким чином, збільшується біла область на зображенні або збільшується розмір об'єкта переднього плану.

Зазвичай в таких випадках, як видалення шуму, за ерозією слідує розширення. Тому що ерозія прибирає білі шуми, але вона також стискає і об'єкт.

- Cv2.threshold(cv.THRESH_OTSU) – алгоритм знаходить оптимальне порогове значення, яке повертається в якості першого виводу.

```
def segment_on_dt(a, img):
    border = cv2.dilate(img, None, iterations=5)
    border = border - cv2.erode(border, None)

    dt = cv2.distanceTransform(img, 2, 3)
    dt = ((dt - dt.min()) / (dt.max() - dt.min()) * 255).astype(numpy.uint8)
    _, dt = cv2.threshold(dt, 180, 255, cv2.THRESH_BINARY)
    lbl, ncc = label(dt)
    lbl = lbl * (255 / (ncc + 1))
    # Заповнення маркерів
    lbl[border == 255] = 255

    lbl = lbl.astype(numpy.int32)
    cv2.watershed(a, lbl)

    lbl[lbl == -1] = 0
    lbl = lbl.astype(numpy.uint8)
    return 255 - lbl
```

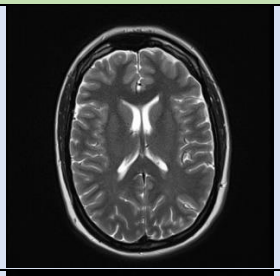
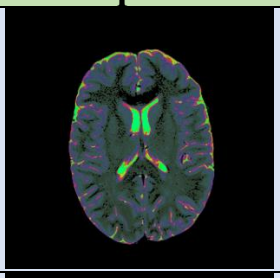
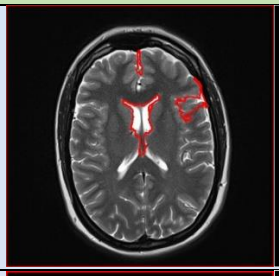
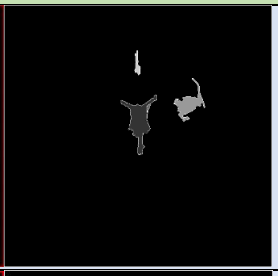
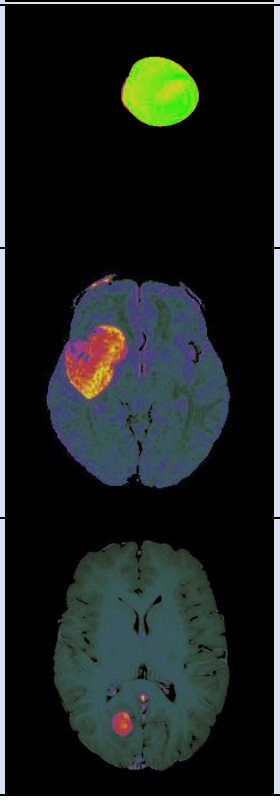
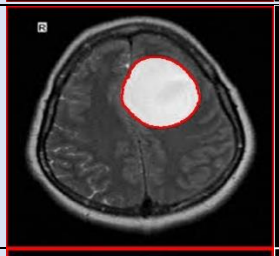
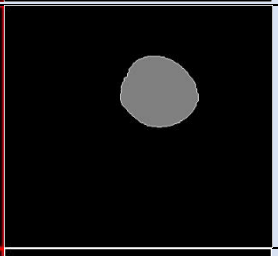
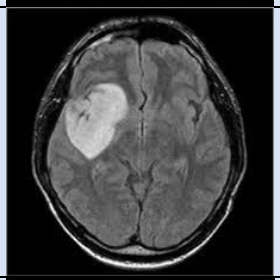
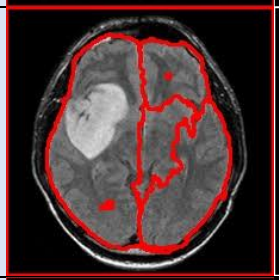
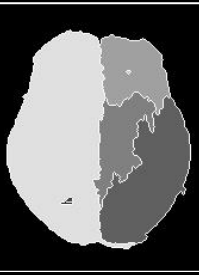
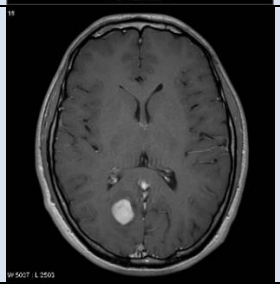
Рис. 2.19. Приклад реалізації сегментації методами бібліотеки OpenCv

```
if __name__ == '__main__':
    img = cv2.imread('Y6_3.jpg')

    # передобробка
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, img_bin = cv2.threshold(img_gray, 0, 255,
            cv2.THRESH_OTSU)

    img_bin = cv2.morphologyEx(img_bin, cv2.MORPH_OPEN,
            numpy.ones((3, 3), dtype=int))

    result = segment_on_dt(img, img_bin)

    cv2.imwrite('C:\\Users\\Elena\\Downloads\\Y2_new1.jpg', result)

    result[result != 255] = 0
    result = cv2.dilate(result, None)
    img[result == 255] = (0, 0, 255)
    cv2.imwrite('C:\\Users\\Elena\\Downloads\\Y2_new2.jpg', img)
```

Рис. 2.20. Приклад передобробки зображення та виклику функції сегментації

Таблиця 2.1

Результати сегментації різними методами

№	Оригінальне зображення	Сегментоване зображення на основі методу вододілу та порогів	Сегментовані зображення методами з бібліотеки OpenCv	
1				
2				
3				
4				

Аналізуючи результати сегментації обох програм, можна зробити такий висновок: не дивлячись на обширність бібліотеки OpenCv та легкість використання, сегментація деяких типів зображень видає некоректні результати. У випадку методу, реалізованому у дипломній роботі, комбінація методів порогової сегментації та вододілу в певній послідовності та повторюваності (з різними вхідними масивами) дає кращий результат. Методи бібліотеки OpenCv

хоча і створені для обробки зображень, але не повністю пристосовані до сегментації саме МРТ зображень мозку.

Висновки до розділу 2

У даному розділі була представлена реалізація автоматизованої системи сегментації цифрових зображень МРТ головного мозку. Використавши фреймворк Qt було створено інтерфейс програми, що полегшує та покращує роботу користувача із системою. На даний момент система являє собою цінний продукт, яким може скористуватися будь–яка людина не маючи спеціальної медичної освіти або спеціальної підготовки. Система розроблена таким чином, щоб уникнути максимум помилок при обробці та сегментації зображення, тому всі методи та функції приймають автоматичну кількість ітерацій, а поріг знаходиться без втручання користувача (як це буває з глобальним порогом).

ВИСНОВКИ

1. Проведено аналіз зарубіжної літератури та розглянуто сучасні підходи до сегментації зображення та його попередньої обробки.

2. Розглянуто метод сегментації МРТ зображень головного мозку на основі вододілу та порогів. Визначено основні складові алгоритму методу та комбінації для кращої роботи системи.

3. Розроблено автоматизовану систему сегментації МРТ зображень головного мозку.

4. Проведено дослідження залежності результатів сегментації від типу зображення (див. Додаток Д). В результаті аналізу отриманих даних було виявлено, що в залежності від форми черепа, або кута нахилу черепної коробки на МРТ зображенні може змінюватися якість сегментації. Також, чим контрастніше оригінальне зображення (залежить від МРТ апарату) тим більша ймовірність правильної сегментації пухлини. В залежності від кількості об’єктів у масиві зображень змінюється і час сегментації, але методи, використані у роботі, дозволяють сегментувати 50 зображень менше ніж за хвилину, що є одним з найкращих результатів цієї предметної області.

5. Найкращі результати по сегментації пухлин дали групи зображень, черепна коробка яких не перевищує певну ширину та знаходиться на деякій відстані від тканини самого мозку. Метод, що було розглянуто у дипломній роботі не є універсальним, але однозначно може використовуватися для початкового аналізу пухлин головного мозку .

6. Перспективою даної роботи є удосконалення алгоритму видалення черепа, проведення автоматичного аналізу (виду, розміру та стадії) пухлини за допомогою нейронних мереж та автоматизація оцінювання якості сегментації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gonzalez, R.C. Digital Image Processing, 2nd ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2002.

2. H.K. Hahn and H.–O. Peitgen, “The Skull Stripping Problem in MRI Solved by a Single 3D Watershed Transform”, Lecture Notes in Computer Science, Medical Image Computing and Computer–Assisted Intervention (MICCAI), Vol. 1935/2000, pp. 134–143, 2000.

3. J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, (6) (1986) 679– 698.

4. J.G. Park and C. Lee, “Skull Stripping Based On Region Growing For Magnetic Resonance Brain Images”, NeuroImage, 47, 1394–1407, 2009.

5. N. Otsu, “A Threshold Selection Method from Gray–Level Histograms”, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, No. 1, pp. 62–66, Jan 1979.

6. Pal, N.R.; Pal, S.K. A review on image segmentation techniques. Pattern Recognit. 1993, 26, 1277–1294.

7. R, Roslan, N. Jamil and R. Mahmud, “Skull Stripping of MRI Brain Images using Mathematical Morphology”, IEEE–EBMS Conference on Biomedical Engineering and Sciences (IECBES 2010). Nov 2010.

8. R.B. Dubey, M. Hanmandlu and S.K. Gupta, “Region Growing for MRI Brain Tumor Volume Analysis”, Indian Journal of Science and Technology, Volume 2, No. 9, 2009

9. T. Kapur, W.E.L. Grimson, W.M. Wells and R. Kikinis, “Segmentation of Brain Tissue from Magnetic Resonance Images”, Medical Image Analysis, Volume 1, Number 2, pp 109–127, 1996.

10. V. Grau, A.U.J. Mewes, M. Alcaniz, R. Kikinis and S.K. Warfield, “Improved watershed transform for medical image segmentation using prior information”, IEEE Trans. Med. Imag., 23(4), 447–458, 2004

11. X. Lin, T. Qiu, F. Morain–Nicolier and S. Ruan Automatic Hippocampus Segmentation from Brain MRI Images", International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM), Vol.1 (2009), pp.239–248

12. Y. J. Zhang, A survey on evaluation methods for image segmentation, Pattern recognition 29 (8) (1996) 1335–1346.

ДОДАТКИ

ДОДАТОК А

Приклад реалізації функціональної частини інтерфейсу користувача

```
class MainFunctional(object):

# sgmIm – масив cv2 результатів, fIn – шлях до обраного файла, fOut – шлях для збереження
global segmentedImages, fnameInput, fnameOutput

# sgmArrImg – масив cv2 результатів, OutF – шлях для збереження,
# InF – масив шляхів до обраних зображень, imgPth – масив шляхів відсегментованих зображень
global segmentedArrayImages, outputFiles, inputFiles, imagePathes

     def createLabel(self, nameLabel, x1, x2, y1, y2, textSize ):
         self.label = QtWidgets.QLabel(self.centralwidget)
         self.label.setGeometry(QtCore.QRect(x1, x2, y1, y2))
         font = QtGui.QFont()
         font.setPointSize(textSize)
         font.setUnderline(False)
         self.label.setFont(font)
         self.label.setFrameShape(QtWidgets.QFrame.WinPanel)
         self.label.setTextFormat(QtCore.Qt.AutoText)
         self.label.setScaledContents(True)
         self.label.setAlignment(QtCore.Qt.AlignCenter)
         self.label.setObjectName(nameLabel)
         self.widgets.append(self.label)

     def setup(self, MainWindow):
         MainWindow.setObjectName("MainWindow")
         MainWindow.resize(1047, 600)
         MainWindow.setStyleSheet("background-color: rgb(90, 83, 127);")
         self.centralwidget = QtWidgets.QWidget(MainWindow)
         self.centralwidget.setObjectName("centralwidget")

         self.widgets = []
         self.nameLabels = [["img_1", "60", "20", "211", "191","20"],
                            ["img_2", "60", "300", "211", "191","20"],
                            ["img_3", "390", "20", "211", "191","20"]]
         for n in range(len(self.nameLabels)):
                 self.createLabel(str(self.nameLabels[n][0]),
                  int(self.nameLabels[n][1]), int(self.nameLabels[n][2]),
                   int(self.nameLabels[n][3]), int(self.nameLabels[n][4]),
                   int(self.nameLabels[n][5]))

          self.retranslateUi(MainWindow)
          QtCore.QMetaObject.connectSlotsByName(MainWindow)

     def preparePathes(self):
         splitPath = self.fnameOutput.split('/')
         nameFileInput = (self.fnameInput.split('/'))
         nameFileArray = nameFileInput[-1].split('.')
         nameFile = nameFileArray[0]
         expFile = nameFileArray[1]
         del splitPath[-1]
         outputPath = "\\".join(splitPath) + "\\"
         return outputPath, nameFile, expFile
```

```
def createPathes(self):
    outputPath, nameFile, expFile = self.preparePathes()
    imagePathes = list()
    for i in range(len(self.segmentedImages)):
        path =outputPath+nameFile+"_"+str(i)+"."+expFile
        cvToImage(self.segmentedImages[i], path)
        imagePathes.append(path)
    return imagePathes

def createPathesForArray(self):
    pathFileNames = []
    imagePathes = [[0] * 5 for i in
range((len(self.segmentedArrayImages)))]
    pathes = []

    for j in range(len(self.inputFiles)):
        outputPath, nameFile, expFile =
self.preparePathesToArray(r""+self.inputFiles[j])
        pathFileNames.append([outputPath, nameFile, expFile])
    for i in range(len(self.segmentedArrayImages)):
        for y in range(len(self.segmentedArrayImages[i])):
            path = pathFileNames[i][0] + pathFileNames[i][1] + "_" +
str(y) + "." + pathFileNames[i][2]
            cvToImage(self.segmentedArrayImages[i][y], path)
            imagePathes[i][y] = path

    return imagePathes

def ShowInLabels(self):
    imagePathes = self.createPathes()
    for i in range(len(imagePathes)):
        self.setOriginalImage(imagePathes[i], i)

def saveFileDoc(self):
    resultDoc = ResultDoc()

    options = QFileDialog.Options()
    fnameOutputDoc, filt = QtWidgets.QFileDialog.getSaveFileName(self,
'Save file', '/desktop', "*.doc", options=options)
    resultDoc.createDoc(self.imagePathes,  r""+fnameOutputDoc)
    os.startfile(r""+fnameOutputDoc)
def setOriginalImage(self, image, numLabel):
    if numLabel == 0:
        self.label.setPixmap(QtGui.QPixmap(image))
    elif numLabel == 1:
        self.label_2.setPixmap(QtGui.QPixmap(image))
    elif numLabel == 2:
        self.label_3.setPixmap(QtGui.QPixmap(image))
    elif numLabel == 3:
        self.label_4.setPixmap(QtGui.QPixmap(image))
    else:
        self.label_9.setPixmap(QtGui.QPixmap(image))

def viewLabels(self):

    if self.counter<len(self.imagePathes):

        for y in range(5):
            self.label_11.setText("Зображення " + str(self.counter+1) + "
/ " + str(len(self.imagePathes)))
            self.setOriginalImage(self.imagePathes[self.counter][y], y)
        self.counter += 1
```

ДОДАТОК Б

Приклад реалізації класу для знаходження зв’язних компонент

```
class ConnectedComponents:
    def getConnectedComponents(bin_image):
        im = Image.open(bin_image)
        (h, w) = im.size
        yc, xc = np.where(bin_image != 0)
        queue = []
        connected_array = np.zeros((h, w))  # позначення масиву
        counter = 1
        for elem in range(len(xc)):
            # перебирати всі ненульові елементи
            i = yc[elem]
            j = xc[elem]
            if connected_array[i, j] == 0:
                # ще не позначені продовжувати
                connected_array[i, j] = counter
                queue.append((i, j))
                while len(queue) != 0:
                    # робота через чергу
                    current = queue.pop(0)
                    i, j = current
                    if i == 0 and j == 0:
                        coords = np.array([[i, i + 1], [j + 1, j]])
                    elif i == h - 1 and j == w - 1:
                        coords = np.array([[i, i - 1], [j - 1, j]])
                    elif i == 0 and j == w - 1:
                        coords = np.array([[i, i + 1], [j - 1, j]])
                    elif i == h - 1 and j == 0:
                        coords = np.array([[i, i - 1], [j + 1, j]])
                    elif i == 0:
                        coords = np.array([[i, i, i + 1], [j - 1, j + 1, j]])
                    elif i == h - 1:
                        coords = np.array([[i, i, i - 1], [j - 1, j + 1, j]])
                    elif j == 0:
                        coords = np.array([[i, i + 1, i - 1], [j + 1, j, j]])
                    elif j == w - 1:
                        coords = np.array([[i, i + 1, i - 1], [j - 1, j, j]])
                    else:
                     coords = np.array([[i, i, i + 1, i - 1], [j - 1, j + 1, j, j]])

                    for k in range(len(coords[0])):
                        # перебирати пікселі сусідів,
                        # якщо вони не позначені та не дорівнюють нулю,
                        # тоді присвоюється поточна мітка
                        if connected_array[coords[0, k], coords[1, k]] == 0 and
                               bin_image
                               coords[0, k], coords[1, k]] != 0:
                            connected_array[coords[0, k], coords[1, k]] =
                               counter
                               queue.append((coords[0, k], coords[1, k]))
                               counter += 1

        return connected_array, counter - 1
```

ДОДАТОК В

Приклад реалізації класу знаходження порога методом Отцу

```
class Otsu:
    def findNormalizedHistogram(self, blurImage):
        hist = cv2.calcHist([blurImage], [0], None, [256], [0, 256])
        hist_norm = hist.ravel() / hist.sum()
        return hist_norm
    def findCumulativeDistributionFunction(self, blurImage):
        hist_norm = self.findNormalizedHistogram(blurImage)
        Q = hist_norm.cumsum()
        return Q
    def tresholdOtsu(self, blurImage):
        bins = np.arange(256)
        fn_min = np.inf
        thresh = -1
        hist_norm = self.findNormalizedHistogram(blurImage)
        Q = self.findCumulativeDistributionFunction(blurImage)
        for i in range(1, 256):
            p1, p2 = np.hsplit(hist_norm, [i])  # ймовірності
            q1, q2 = Q[i], Q[255] - Q[i]  # совокупна сумма класів
            if q1 < 1.e-6 or q2 < 1.e-6:
                continue
            b1, b2 = np.hsplit(bins, [i])  # ваги
            # пошук середніх та відхилень
            m1, m2 = np.sum(p1 * b1) / q1, np.sum(p2 * b2) / q2
            v1, v2 = np.sum(((b1 - m1) ** 2) * p1) / q1, np.sum(((b2 - m2) ** 2)
               * p2) / q2
            # обчислення функцію мінімізації
            fn = v1 * q1 + v2 * q2
            if fn < fn_min:
                fn_min = fn
                thresh = i
        return thresh
```

ДОДАТОК Г

Приклад реалізації створення результуючого документа

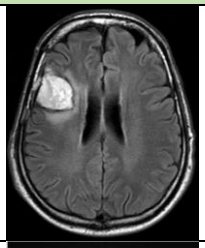
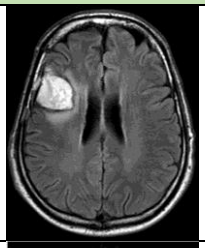
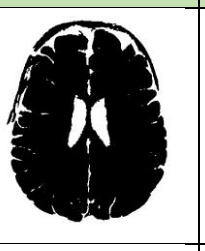
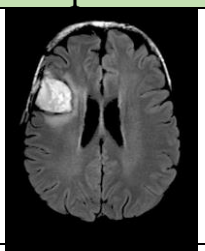
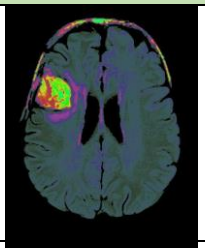
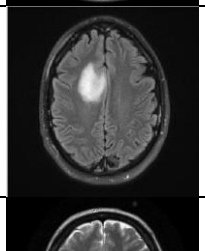
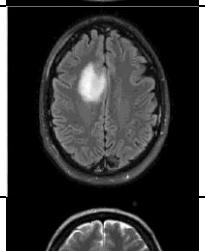
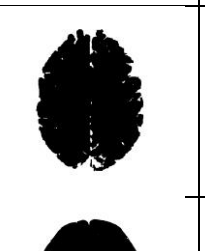
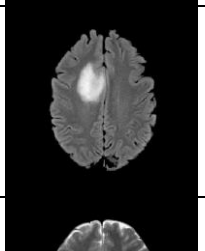
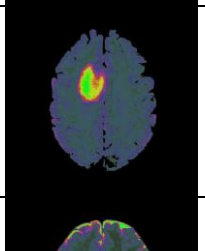
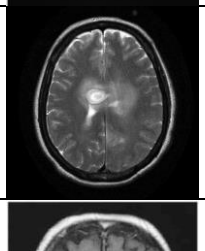
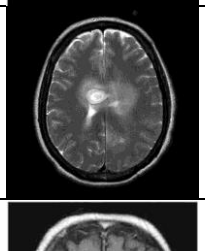
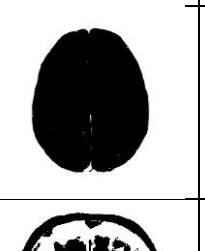
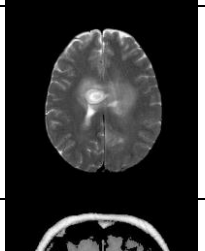
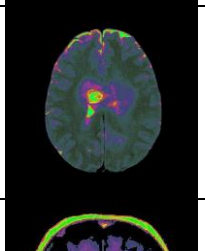
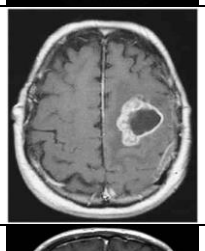
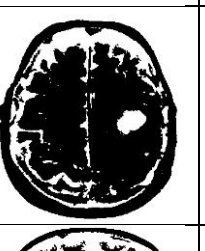
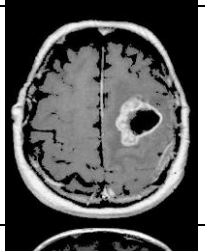
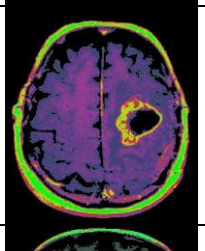
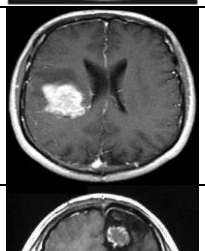
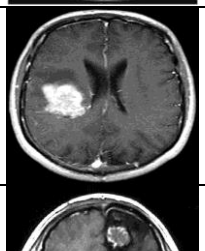
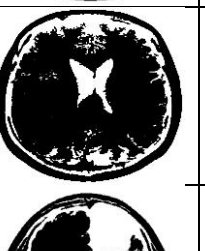
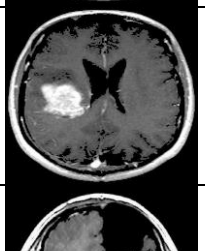
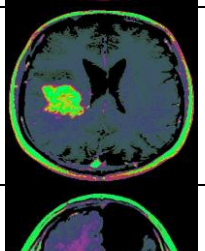
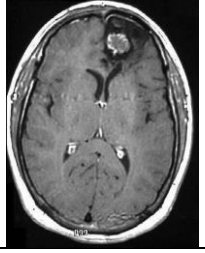
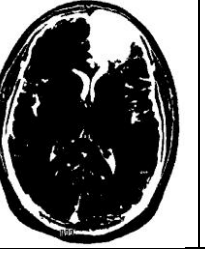
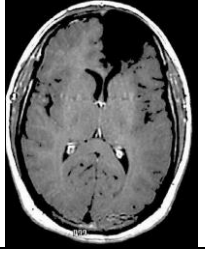
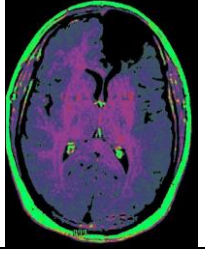
```
class ResultDoc:

    def cvToImage(self, cvArray):
        img = Image.fromarray(cvArray)
        return img
    def createDoc(self, images,  path):
        doc = Document()
        path1 = r'C:/Users/Elena/Downloads'
        os.chdir(path1)
        p = os.getcwd()
        head = doc.add_heading()
        head.paragraph_format.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
        run1 = head.add_run('Результати сегментації', 0)
        run1.font.size = Pt(30)

        numberImages = len(images)
        table = doc.add_table(rows=numberImages+1, cols=6,style="Table Grid")

        for row in range(numberImages+1):
            cells = table.rows[row].cells
            for col in range(6):
                if row == 0 and col==0:
                    cells[col].text = '№'
                if row == 0 and col == 1:
                    cells[col].text = 'Оригінальне зображення'
                if row == 0 and col == 2:
                    cells[col].text = 'Зображення у відтінках сірого'
                if row == 0 and col == 3:
                    cells[col].text = 'Бінарне зображення'
                if row == 0 and col == 4:
                    cells[col].text = 'Зображення з видаленим черепом'
                if row == 0 and col == 5:
                    cells[col].text = 'Сегментоване зображення'
                if row!=0 and col==0:
                    p = cells[col].paragraphs[0]
                    r = p.add_run()
                    r.text = str(row)
                if row!=0 and col!=0:
                    p = cells[col].paragraphs[0]
                    r = p.add_run()

                    count = r""+images[row-1][col-1]
                    r.add_picture(r""+images[row-1][col-1], width=Inches(1))

        self.changeFont(table)
        doc.save(path)


    def changeFont(self, table):
        for row in table.rows:
            for cell in row.cells:
                paragraphs = cell.paragraphs
                for paragraph in paragraphs:
                    for run in paragraph.runs:
                        font = run.font
                        font.size = Pt(10)
```

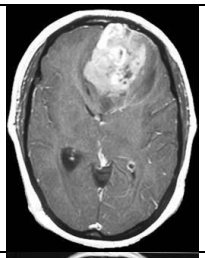
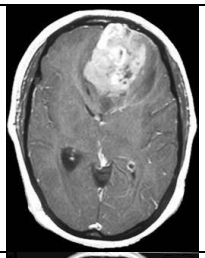
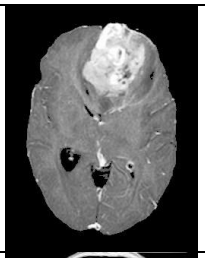
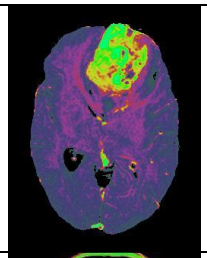
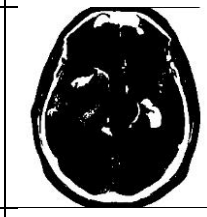
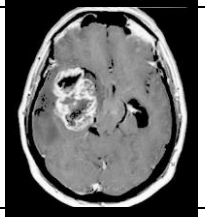
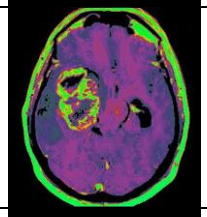
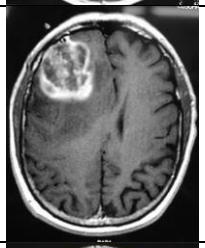
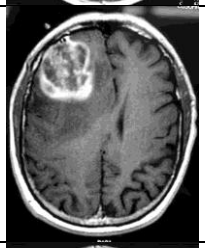
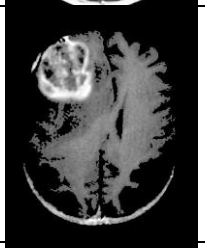
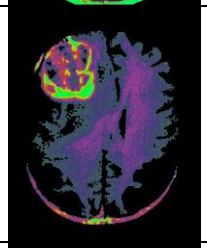
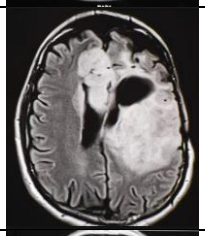
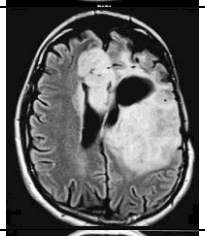
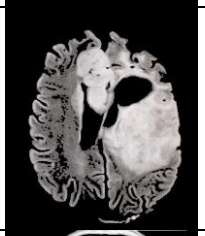
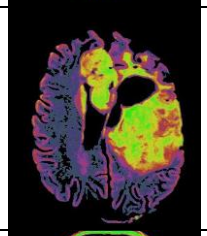
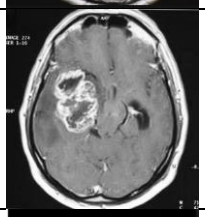
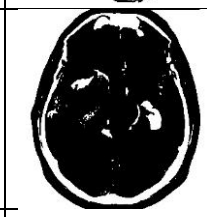
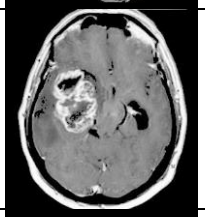
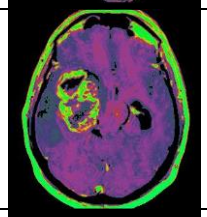
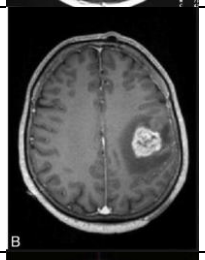
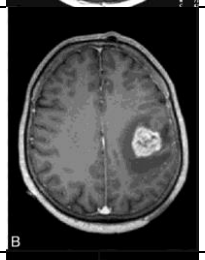
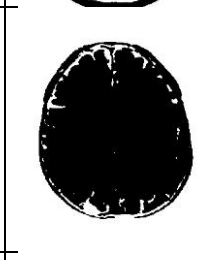
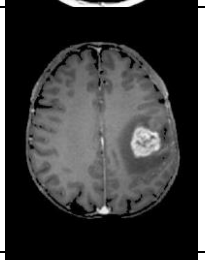
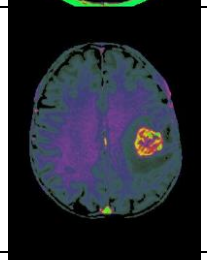
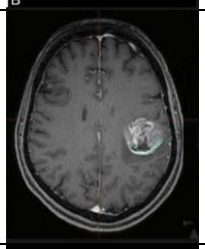
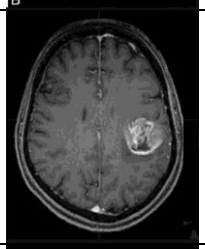
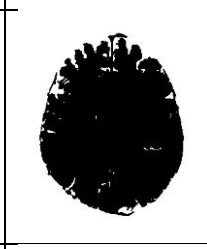
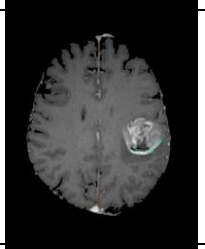
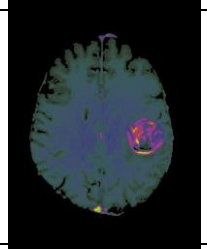
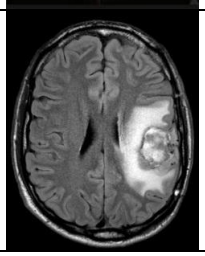
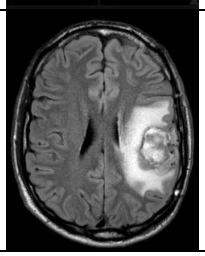
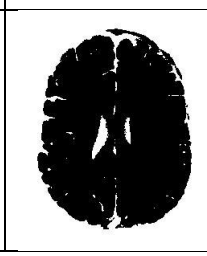
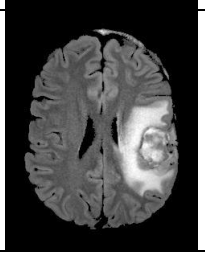
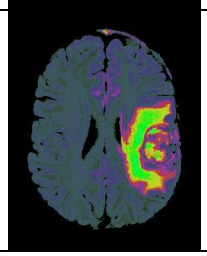
ДОДАТОК Д

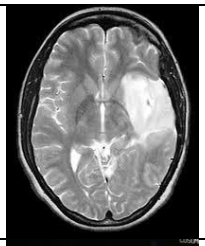
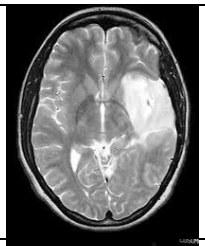
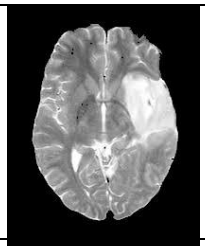
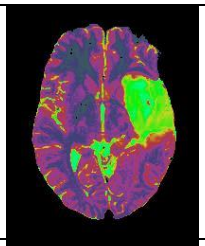
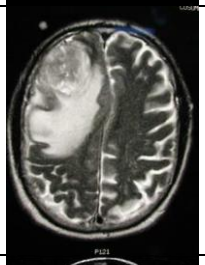
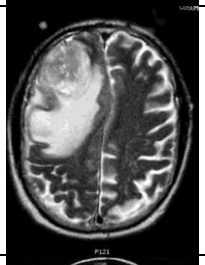
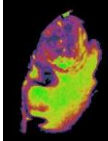
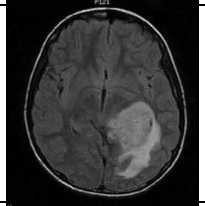
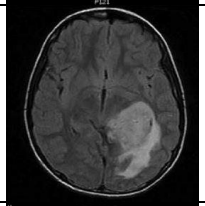
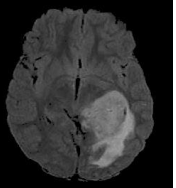
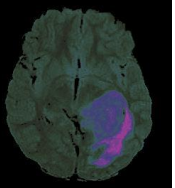
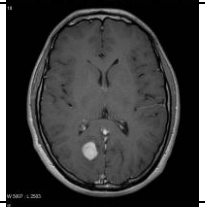
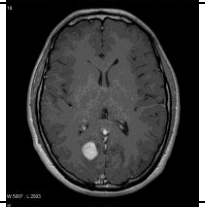
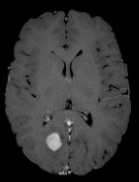
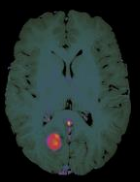
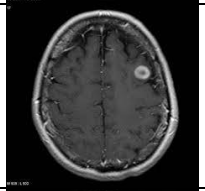
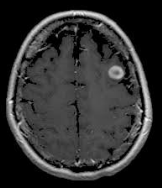
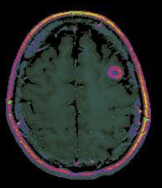
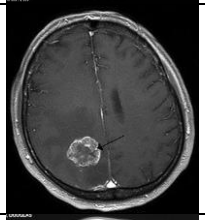
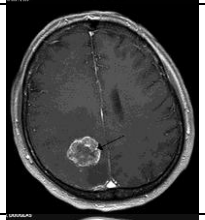
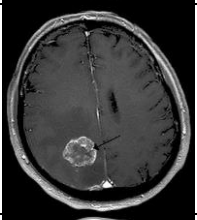
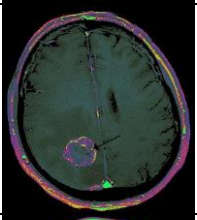
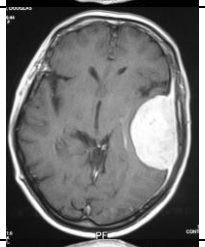
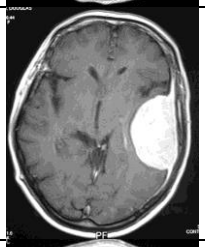
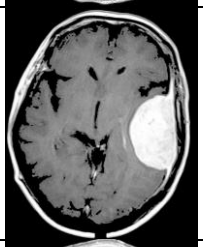
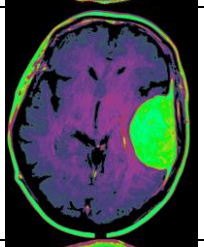
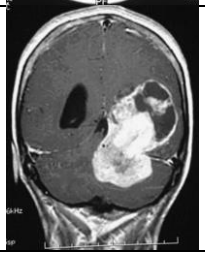
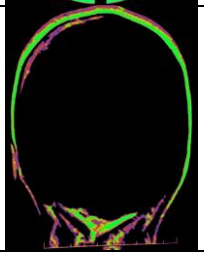
Результати сегментації

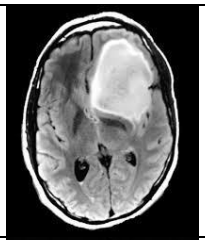
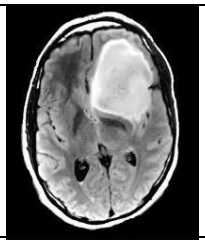
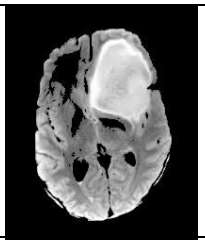
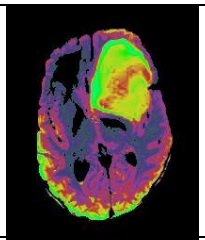
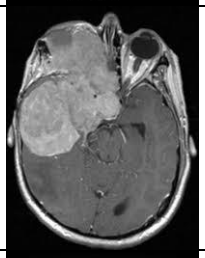
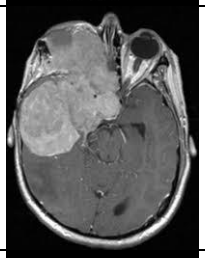
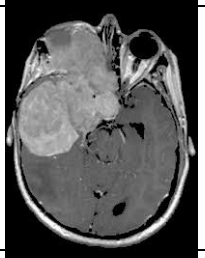
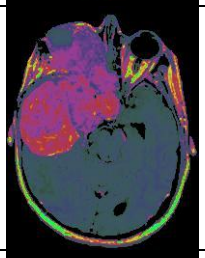
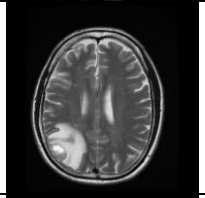
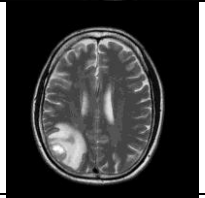
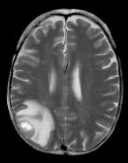
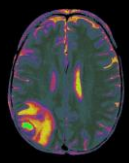
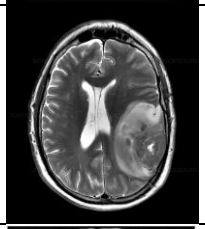
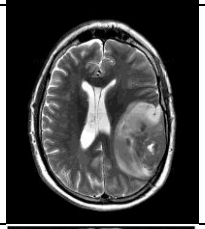
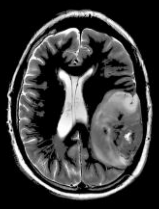
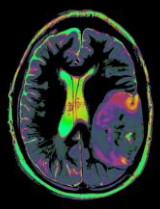
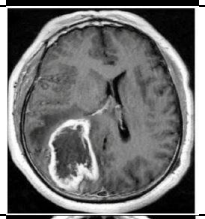
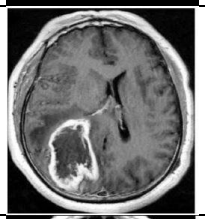
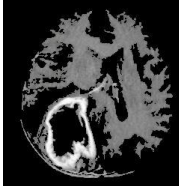
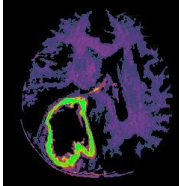
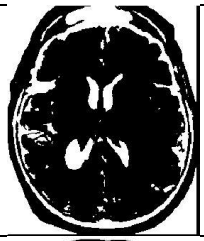
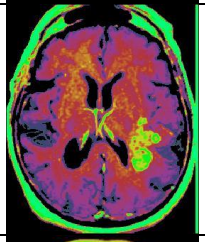
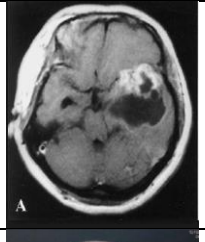
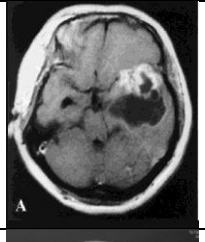
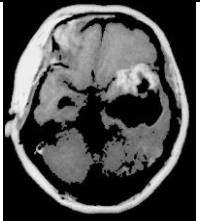
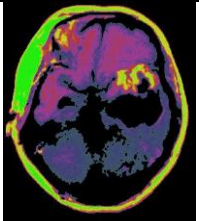
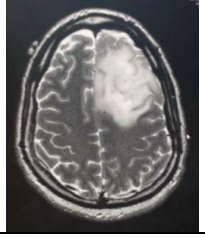
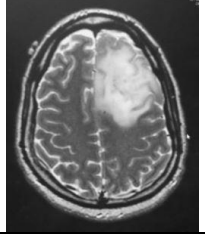
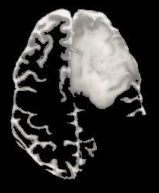
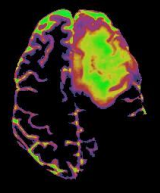
Таблиця Д.1

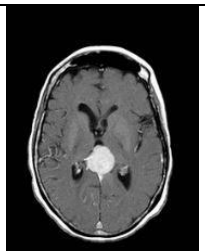
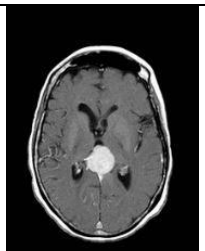
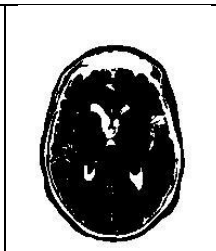
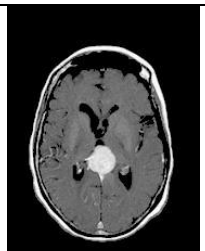
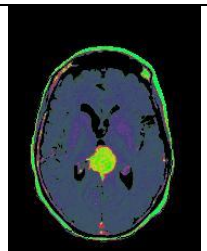
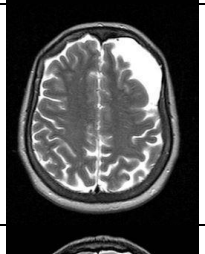
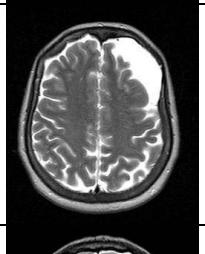
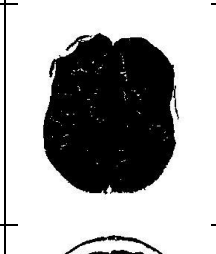
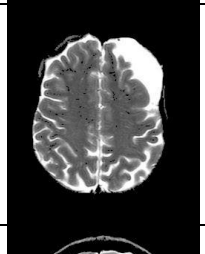
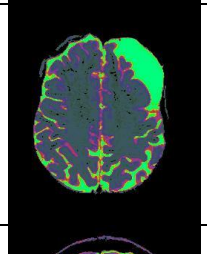
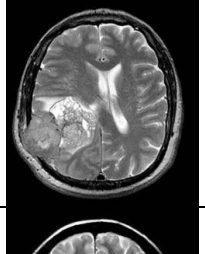
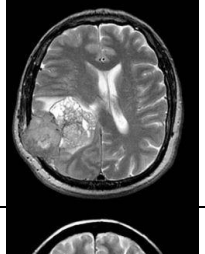
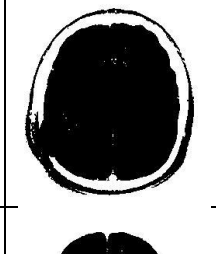
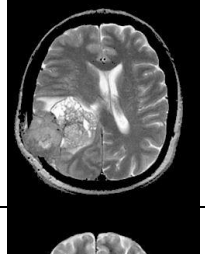
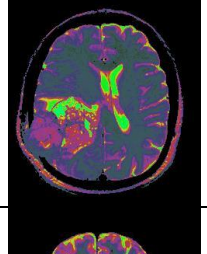
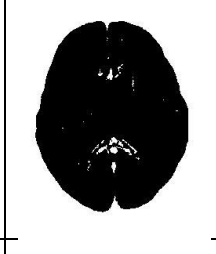
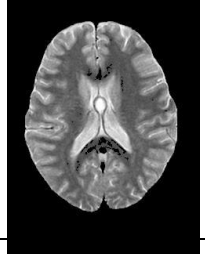
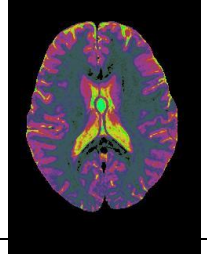
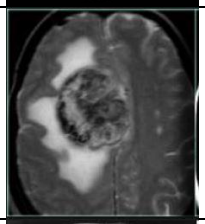
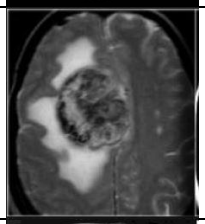
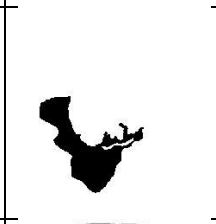
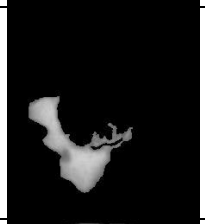
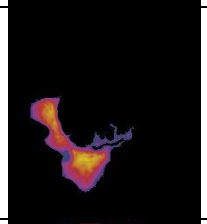
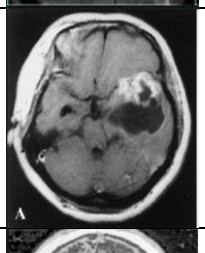
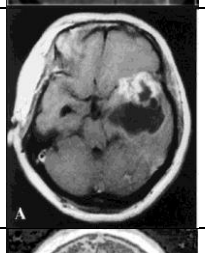
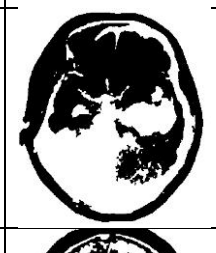
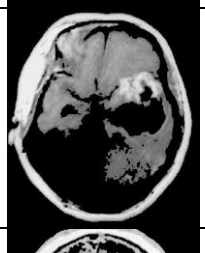
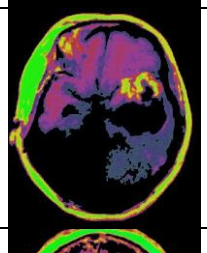
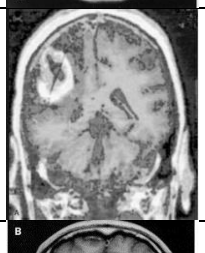
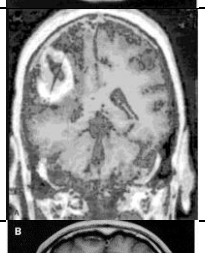
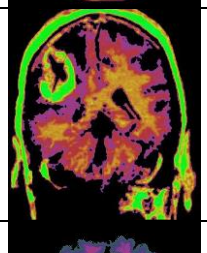
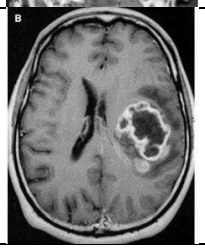
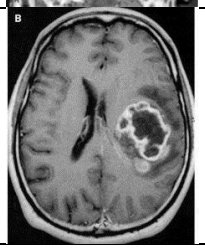
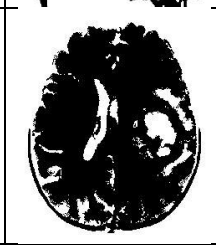
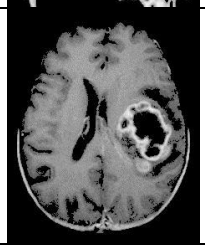
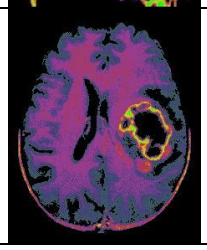
Результуючий документ

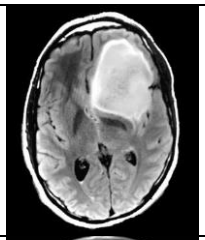
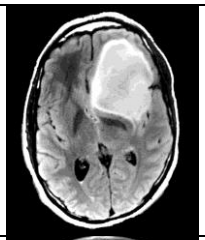
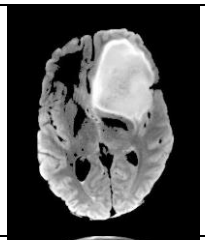
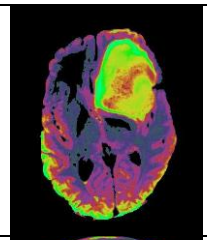
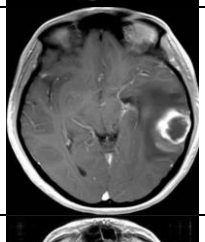
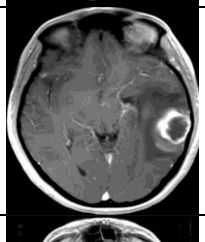
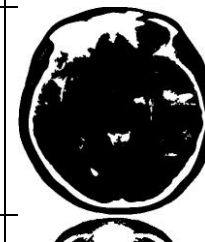
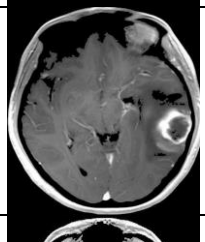
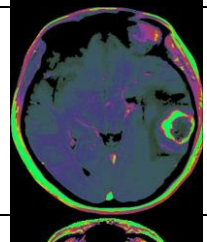
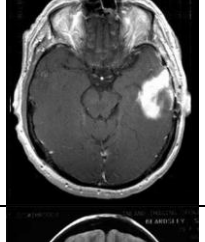
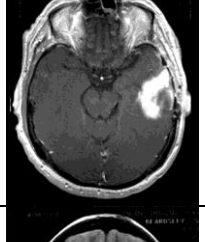
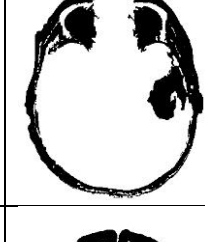
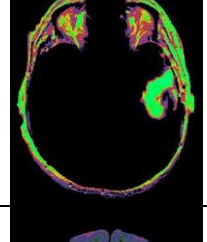
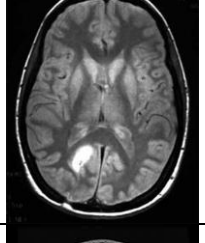
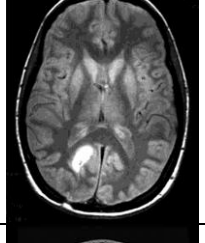
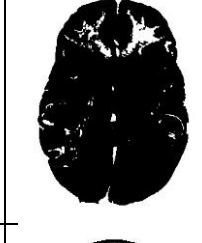
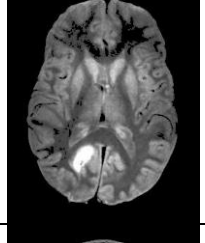
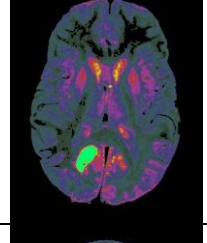
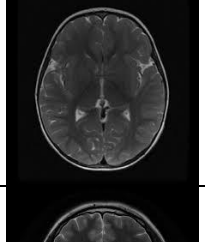
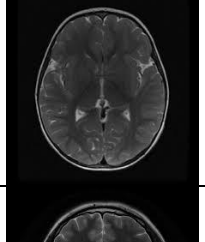
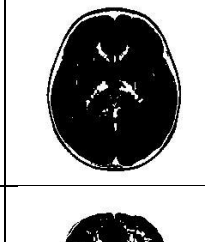
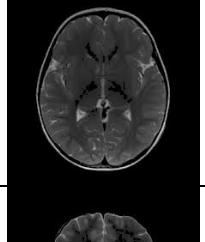
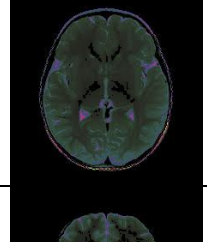
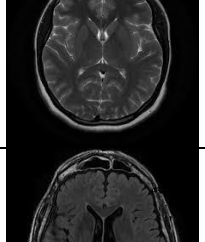
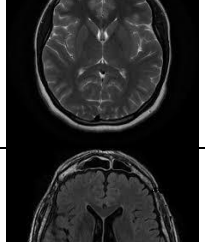
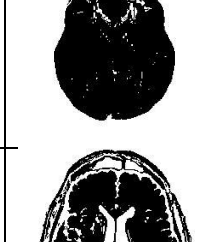
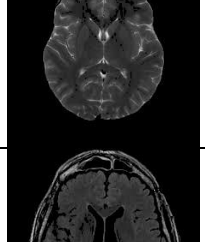
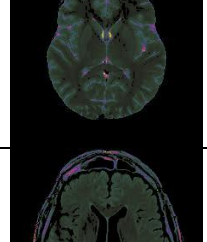
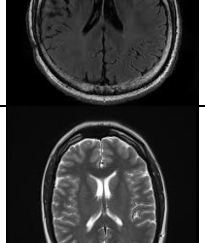
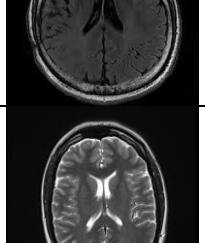
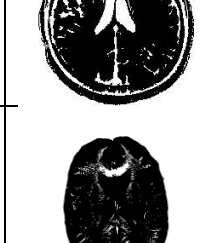
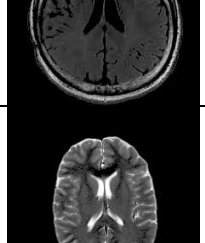
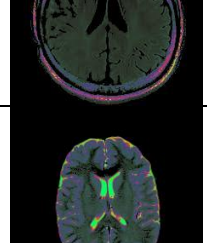
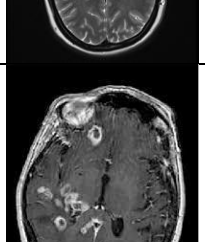
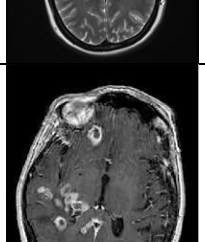
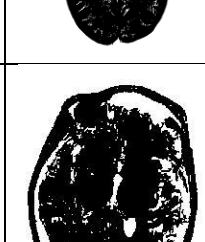
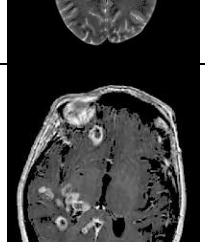
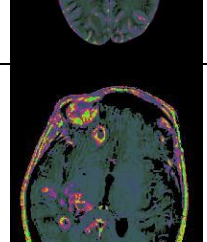
№	Оригінальне зображення	Зображення у відтінках сірого	Бінарне зображення	Зображення з видаленим черепом	Сегментоване зображення
1					
2					
3					
4					
5					
6					

7					
8					
9					
10					
11					
12					
13					
14					

15					
16					
17					
18					
19					
20					
21					
22					

23					
24					
25					
26					
27					
28					
29					
30					

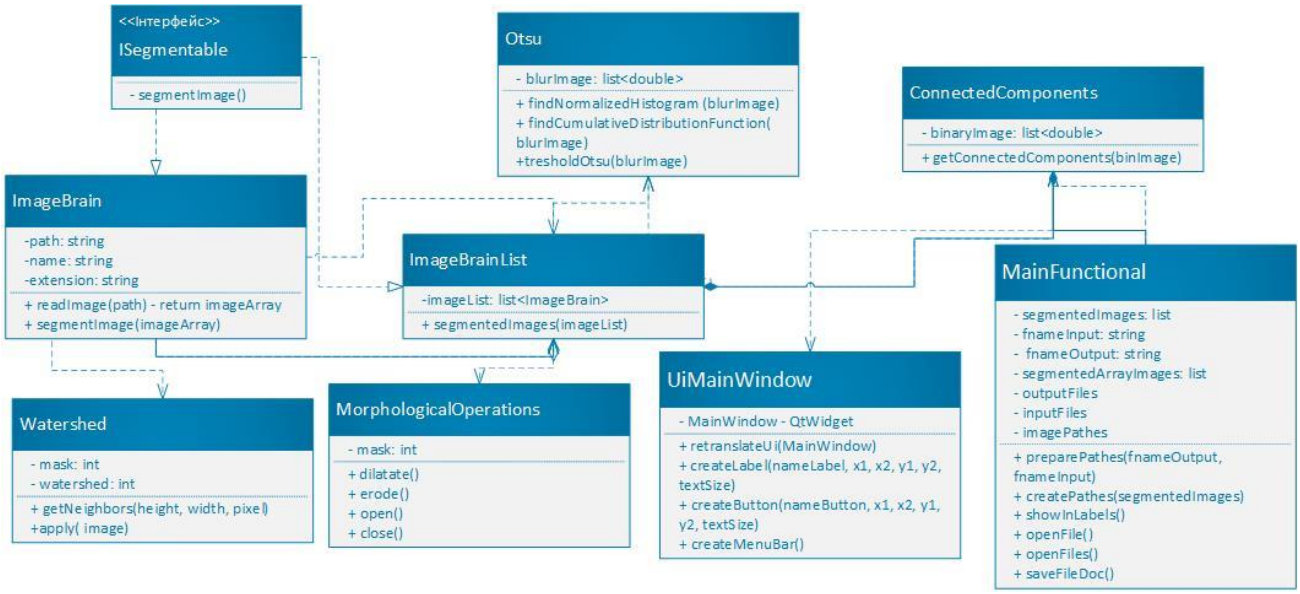
31					
32					
33					
34					
35					
36					
37					
38					

39					
40					
41					
42					
43					
44					
45					
46					
47					

ДОДАТОК Е

Діаграма класів

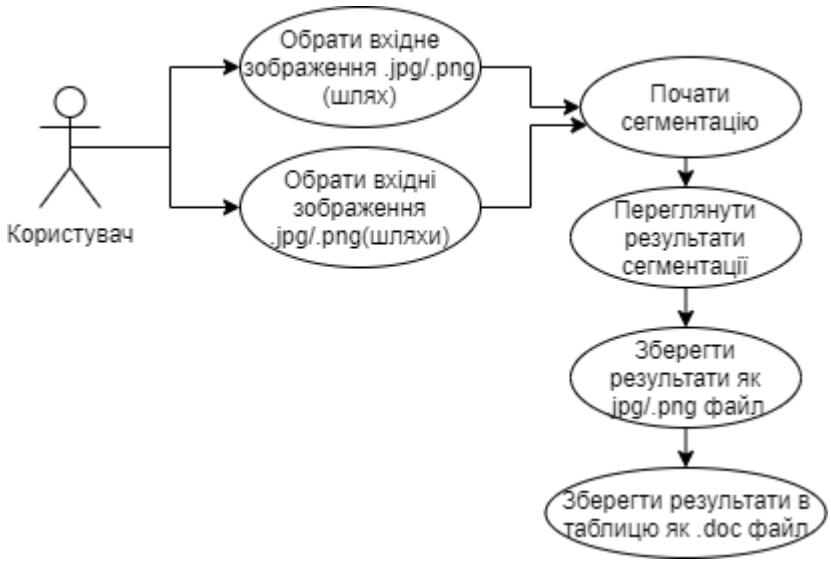
Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно–орієнтованого програмування. На діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини



ДОДАТОК Ж

Діаграма прецедентів

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

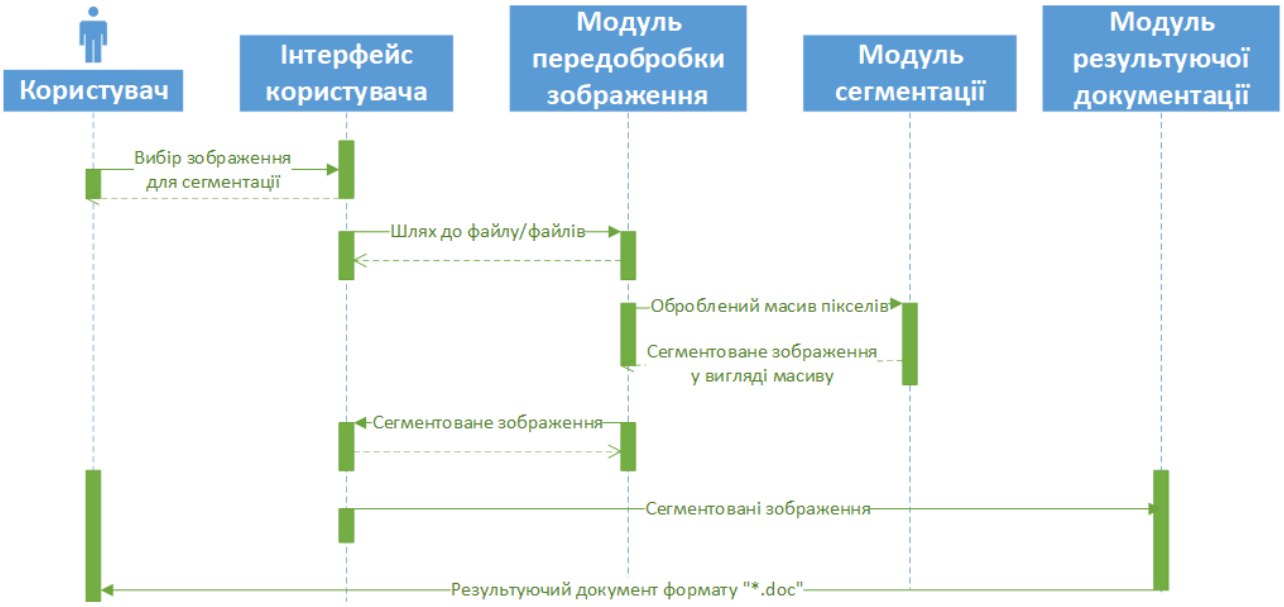


В даній системі актором є будь–який користувач, інших ролей не передбачено. А прецедентами є ті функції, що можливо виконати над системою.

ДОДАТОК З

Діаграма послідовностей

Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. На діаграмі послідовностей показано у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надіслані повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення.



На данній діаграмі зображена послідовність процесів автоматизованої системи сегментації цифрових зображень МРТ головного мозку. Вертикальними лініями позначені модулі обробки інформації та об’єкт «Коритсувач». Модулі відповідають за:

- «Інтерфес користувача» – відповідає за обробку подій викликаних користувачем (дозволяє обрати зображення через діалогові вікна, а також відображає результати сегментації);
- «Модуль передобробки зображення» – відповідає за попередню оброюку зображення перед початком сегментації, тобто переведення зображення у тривимірний масив, у градації сірого, та видалення шумів;
- «Модуль сегментації» – відповідає за виділення пухлини на МРТ зображенні та повертає оброблений масив пікселів;
- «Модуль результуючої документації» – відповідає за створення таблиці з відсегментованими зображеннями у документі Microsoft Word.

ДОДАТОК И

Software Architecture Document

Software Architecture Document

Brain Segmentation Tool (Automated system for segmentation of digital images of MRI of the brain)

Kolesnyk Olena

Version 1.2

May 2021

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Architecture Document has been completed.*

Date	Version	Description	Author
14/05/2021	1.0	Initial version of SAD	Kolesnyk Olena
25/05/2021	1.1	Some changes after first review	Kolesnyk Olena
10/06/2021	1.2	Ready for the next review	Kolesnyk Olena

Table of Contents

Software Architecture Document

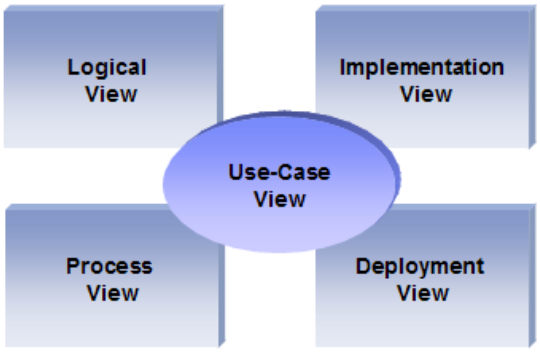
1. Introduction

This document provides a high level overview and explains the whole architecture of Brain Segmentation Tool (BS tool). It is explains how an online user will be able to create and maintain software development process definitions and includes the underlying architecture of the tool.
The document provides a high–level description of the goals of the architecture, the use cases support by the system and architectural styles and components that have been selected to best achieve the use cases. This framework then allows for the development of the design criteria and documents that define the technical and domain standards in detail.

1.1. Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of Brain Segmentation Tool (BS tool). It presents a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

In order to depict the software as accurately as possible, the structure of this document is based on the “4+1” model view of architecture [KRU41].



The “4+1” View Model allows various stakeholders to find what they need in the software architecture.

1.2. Scope

The scope of this SAD is to depict the architecture of Brain Segmentation Tool (BS tool) online application created by the student of Taras Shevchenko National University of Kyiv.
This document describes the aspects of Brain Segmentation Tool (BS tool) design that are considered to be architecturally significant; that is, those elements and behaviors that are most fundamental for guiding the construction Brain Segmentation Tool and for understanding this project as a whole. Stakeholders who require a technical understanding of Brain Segmentation Tool are encouraged to start by reading this document, then reviewing the Brain Segmentation Tool UML model, and then by reviewing the source code.

1.3. Definitions, Acronyms, and Abbreviations

- Windows – Operation System
- SAD – Software Architecture Document
- RUP – Rational Unified Process
- UML – Unified Modeling Language
- User – This is any user who use Brain Segmentation Tool

1.4. References

[SRS]: Software Requirements Specification

[MedBiquitous]: Sample SAD, http://medbiq.org/std_specs/techguidelines/softwarearchitecture.pdf

[KRU41]: The “4+1” view model of software architecture, Philippe Kruchten, November 1995, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

1.5. Overview

In order to fully document all the aspects of the architecture, the Software Architecture Document contains the following subsections.

Section 2: describes the use of each view

Section 3: describes the architectural constraints of the system

Section 4: describes the functional requirements with a significant impact on the architecture

Section 5: describes the most important use–case realization.

Section 6: describes design's concurrency aspects

Section 7: describes how the system will be deployed.

Section 8: describes any significant persistent element.

Section 9: describes any performance issues and constraints

Section 10: describes any aspects related to the quality of service (QoS) attributes

2. Architectural Representation

This document details the architecture using the views defined in the "4+1" model [KRU41], but using the RUP naming convention. The views used to document Brain Segmentation Tool application are:

Use Case view

Audience: all the stakeholders of the system, including the end–users.
Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).
Related Artifacts : Use–Case Model, Use–Case documents

Logical view

Audience: Designers.
Area: Functional Requirements: describes the design's object model. Also describes the most important use–case realizations and business requirements of the system.
Related Artifacts: Design model

Process view

Audience: Integrators.
Area: Non–functional requirements: describes the design's concurrency and synchronization aspects.
Related Artifacts: (no specific artifact).

Module Decomposition view

Audience: Programmers.
Area: Software components: describes the modules and subsystems of the application.
Related Artifacts: Implementation model, components

Deployment view

Audience: Deployment managers.
Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects. Describes potential deployment structures, by including

known and anticipated deployment scenarios in the architecture we allow the implementers to make certain assumptions on network performance, system interaction and so forth.
Related Artifacts: Deployment model.

3. Architectural Goals and Constraints

Server side
No server side.

Client Side
Users will be use Windows devices to access Brain Segmentation Tool. Users are requiring using modern Windows devices with Windows 7+ operation system version.

3.1. Security

User right will be grated to scan area and recognize objects. No other roles provided. User data won't be shared in any way.

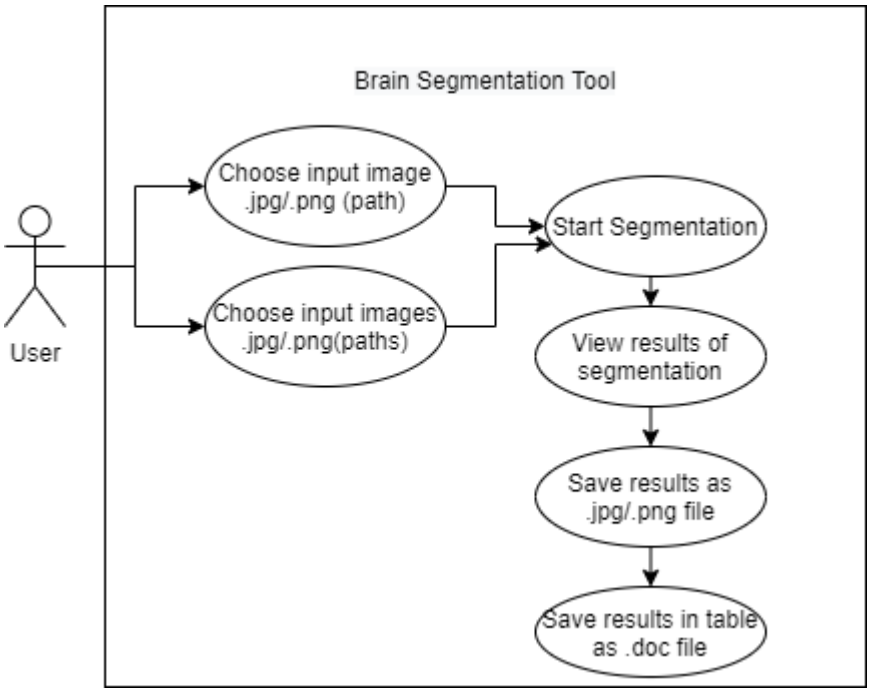
3.2. Performance

There is no particular constrains related to system performance.
It is anticipated that the system should respond to any request well under standard timeouts (20 seconds), also system performance can depend on available hardware. Therefore, actual performance can be determined only after system deployment and testing.

4. Use–Case View

This is a list of use–cases that represent major functionality of the final system [SRS]:

- Choose input image (.png/.jpg format, path)
- Choose input images (.png/.jpg format, paths)
- Start segmentation
- View results of segmentation
- Save results as .jpg/.png file
- Save results in table as .doc file



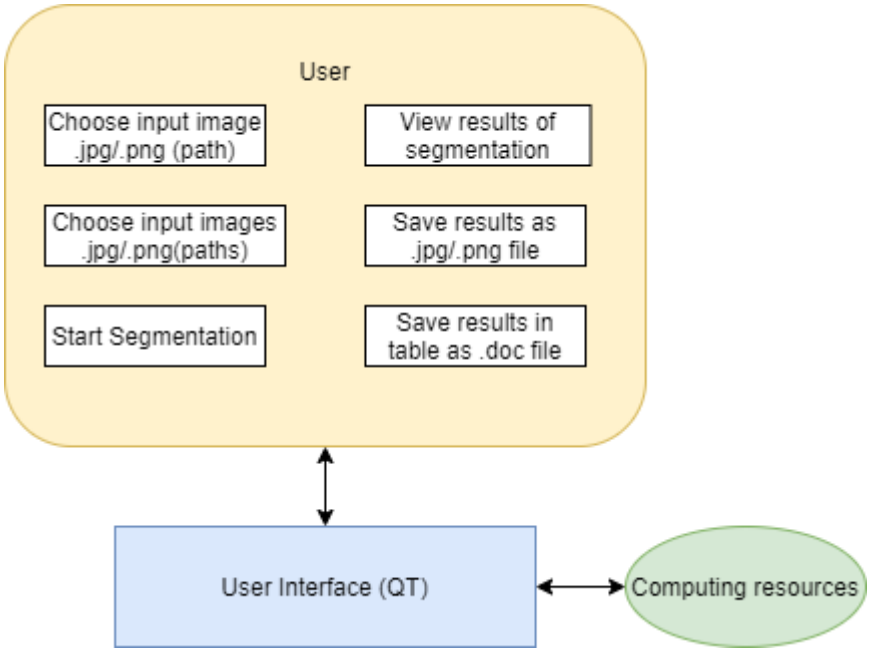
4.1. Actors

As described in the actors' correspondence diagram below, user could be one type:

1. User – default actor. He can choose, segment images and create result document.

4.2. Use–Case Realizations

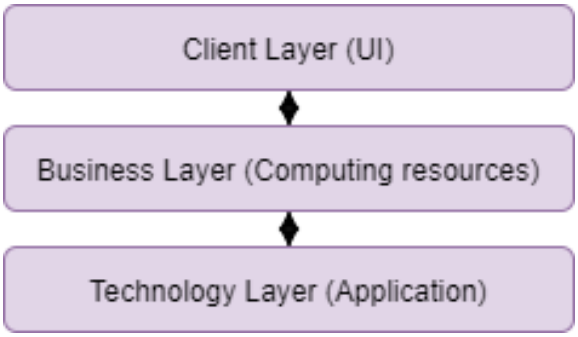
Use case functionality diagram below describes how design elements provide the functionalities identified in the significant use–cases. Use cases are displayed as functionalities for the system. Functionality may enclose more than one use–case.



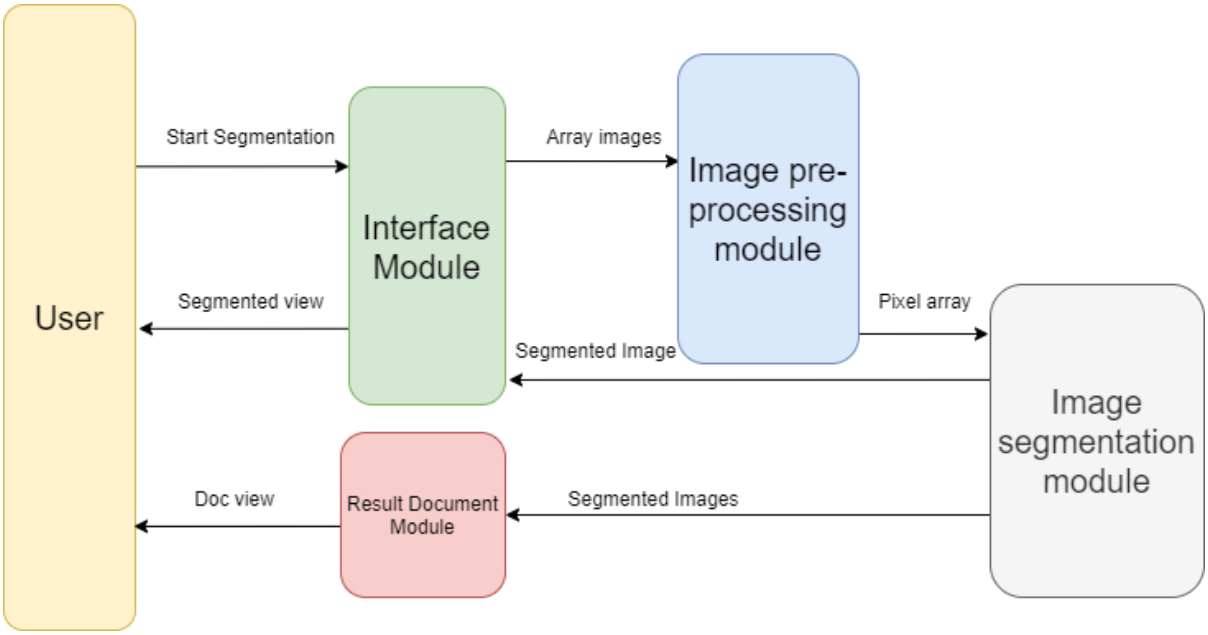
5. Logical View

5.1. Overview

Brain Segmentation Tool is divided into layers based on the N–tier architecture [KRU41].



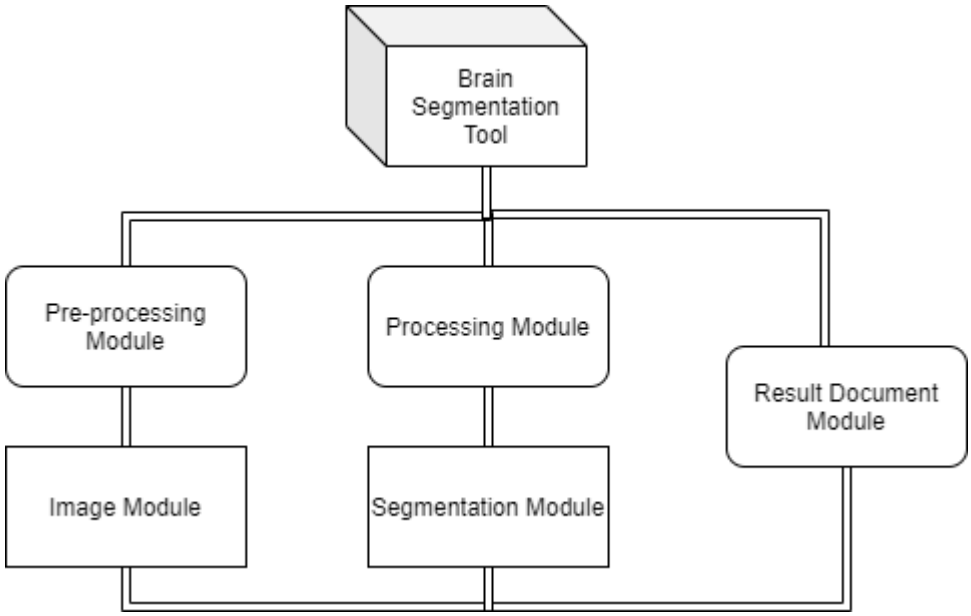
6. Process View



- User – default actor
- Image pre–processing module – preparation of the image for segmentation (in grayscale, noise)
- Image segmentation module – the classes that are responsible for the segmentation methods
- Result document module –

7. Module Decomposition View

Module decomposition view based on principles separation of concerns and abstraction and supports goals of modifiability and usability.



8. Deployment View

Brain Segmentation tool deployment has not been considered yet. All future implementation details will be included in this section.

9. Size and Performance

Volumes

- One (1) user per session
- App will occupy ~35–40 MB on device

Performance

- With maximum load all transactions well under standard timeout – 20 seconds.

10. Issues and concerns

- More UI customization.
- Improving the skull removal algorithm.
- Automatic analysis (type, size and stage) of the tumor using neural networks.
- Automation of segmentation quality assessment.